### REVISION HISTORY

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>DATE</th>
<th>DESCRIPTION</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.00.00</td>
<td>2019-12-11</td>
<td>Initial release.</td>
<td>TRON Forum</td>
</tr>
</tbody>
</table>
# Contents

API Notation ................................................. 1

Index of μT-Kernel/OS APIs ............................... 3

Index of μT-Kernel/SM APIs ............................... 7

Index of μT-Kernel/DS APIs ............................... 10

1 Overview of μT-Kernel 3.0 ............................... 12
   1.1 TRON Project and μT-Kernel 3.0 .................... 13
   1.2 Design Policy of μT-Kernel 3.0 .................... 14
   1.3 Structure of μT-Kernel 3.0 ......................... 15
   1.4 Reference Code .................................. 17
   1.5 Adaptability and Service Profile .................. 18
   1.6 Implementation Specification Document .......... 19
   1.7 Relation with Existing RTOS Specifications .... 20
      1.7.1 Relation with μT-Kernel 2.0 ................ 20
      1.7.2 Relation with T-Kernel 2.0 .................. 20
      1.7.3 Relation with IEEE 2050-2018 ............... 21

2 μT-Kernel Concepts ...................................... 22
   2.1 Meaning of Basic Terminology ...................... 23
   2.2 Task States and Scheduling Rules ................. 25
      2.2.1 Task States .................................. 25
      2.2.2 Task Scheduling Rules ....................... 28
   2.3 Interrupt Handling ................................ 31
   2.4 Task Exception Handling ........................... 32
   2.5 System States ................................... 33
      2.5.1 System States While Non-task Portion Is Executing 33
      2.5.2 Task-Independent Portion and Quasi-Task Portion 34
   2.6 Objects .......................................... 36
   2.7 Protection Levels .................................. 37
   2.8 Service Profile ................................... 38
4 μT-Kernel/OS Functions

4.1 Task Management Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_cre_tsk</td>
<td>59</td>
</tr>
<tr>
<td>tk_del_tsk</td>
<td>60</td>
</tr>
<tr>
<td>tk_sta_tsk</td>
<td>64</td>
</tr>
<tr>
<td>tk_ext_tsk</td>
<td>65</td>
</tr>
<tr>
<td>tk_exd_tsk</td>
<td>66</td>
</tr>
<tr>
<td>tk_ter_tsk</td>
<td>68</td>
</tr>
<tr>
<td>tk_chg_pri</td>
<td>69</td>
</tr>
<tr>
<td>tk_get_reg</td>
<td>71</td>
</tr>
<tr>
<td>tk_set_reg</td>
<td>73</td>
</tr>
<tr>
<td>tk_get_cpr</td>
<td>75</td>
</tr>
<tr>
<td>tk_set_cpr</td>
<td>77</td>
</tr>
<tr>
<td>tk_ref_tsk</td>
<td>79</td>
</tr>
</tbody>
</table>

4.2 Task Synchronization Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_slp_tsk</td>
<td>84</td>
</tr>
<tr>
<td>tk_slp_tsk_u</td>
<td>85</td>
</tr>
<tr>
<td>tk_wup_tsk</td>
<td>87</td>
</tr>
<tr>
<td>tk_can_wup</td>
<td>88</td>
</tr>
<tr>
<td>tk_rel_wai</td>
<td>90</td>
</tr>
<tr>
<td>tk_sus_tsk</td>
<td>91</td>
</tr>
<tr>
<td>tk_rsm_tsk</td>
<td>93</td>
</tr>
<tr>
<td>tk_frs_tsk</td>
<td>95</td>
</tr>
<tr>
<td>tk_slp_tsk</td>
<td>99</td>
</tr>
<tr>
<td>tk_dly_tsk</td>
<td>100</td>
</tr>
<tr>
<td>tk_can_dly_tsk_u</td>
<td>101</td>
</tr>
<tr>
<td>tk_rej_tev</td>
<td>103</td>
</tr>
<tr>
<td>tk_wai_tev</td>
<td>105</td>
</tr>
<tr>
<td>tk_dis_tev</td>
<td>106</td>
</tr>
<tr>
<td>tk_ena_tev</td>
<td>109</td>
</tr>
</tbody>
</table>

4.3 Task Exception Handling Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_def_tex</td>
<td>110</td>
</tr>
<tr>
<td>tk_ena_tex</td>
<td>111</td>
</tr>
<tr>
<td>tk_dis_tex</td>
<td>113</td>
</tr>
<tr>
<td>tk_ras_tex</td>
<td>115</td>
</tr>
<tr>
<td>tk_end_tex</td>
<td>117</td>
</tr>
<tr>
<td>tk_ref_tex</td>
<td>119</td>
</tr>
</tbody>
</table>

4.4 Synchronization and Communication Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semaphore</td>
<td>123</td>
</tr>
<tr>
<td>tk_slp_tsk</td>
<td>124</td>
</tr>
</tbody>
</table>
4.4.1.1 tk_cre_sem - Create Semaphore .................................................. 125
4.4.1.2 tk_del_sem - Delete Semaphore ................................................... 127
4.4.1.3 tk_sig_sem - Signal Semaphore .................................................... 128
4.4.1.4 tk_wai_sem - Wait on Semaphore ................................................. 129
4.4.1.5 tk_wai_sem_u - Wait on Semaphore (Microseconds) ....................... 131
4.4.1.6 tk_ref_sem - Reference Semaphore Status ...................................... 132

4.4.2 Event Flag ................................................................. 133
4.4.2.1 tk_cre_flg - Create Event Flag .................................................. 134
4.4.2.2 tk_del_flg - Delete Event Flag ................................................... 136
4.4.2.3 tk_set_flg - Set Event Flag ....................................................... 137
4.4.2.4 tk_clr_flg - Clear Event Flag ..................................................... 138
4.4.2.5 tk_wai_flg - Wait Event Flag ..................................................... 139
4.4.2.6 tk_wai_flg_u - Wait Event Flag (Microseconds) ............................ 142
4.4.2.7 tk_ref_flg - Reference Event Flag Status ..................................... 144

4.4.3 Mailbox ................................................................. 145
4.4.3.1 tk_cre_mbx - Create Mailbox ..................................................... 147
4.4.3.2 tk_del_mbx - Delete Mailbox ..................................................... 149
4.4.3.3 tk_snd_mbx - Send Message to Mailbox ...................................... 150
4.4.3.4 tk_rcv_mbx - Receive Message from Mailbox ................................ 152
4.4.3.5 tk_rcv_mbx_u - Receive Message from Mailbox (Microseconds) ....... 154
4.4.3.6 tk_ref_mbx - Reference Mailbox Status ...................................... 155

4.5 Extended Synchronization and Communication Functions .................. 157
4.5.1 Mutex ................................................................. 158
4.5.1.1 tk_cre_mtx - Create Mutex ....................................................... 160
4.5.1.2 tk_del_mtx - Delete Mutex ....................................................... 162
4.5.1.3 tk_loc_mtx - Lock Mutex .......................................................... 163
4.5.1.4 tk_loc_mtx_u - Lock Mutex (Microseconds) ................................ 165
4.5.1.5 tk_unl_mtx - Unlock Mutex ....................................................... 166
4.5.1.6 tk_ref_mtx - Refer Mutex Status .............................................. 168

4.5.2 Message Buffer ......................................................... 169
4.5.2.1 tk_cre_mbf - Create Message Buffer ......................................... 171
4.5.2.2 tk_del_mbf - Delete Message Buffer .......................................... 174
4.5.2.3 tk_snd_mbf - Send Message to Message Buffer .......................... 175
4.5.2.4 tk_snd_mbf_u - Send Message to Message Buffer (Microseconds) .... 177
4.5.2.5 tk_rcv_mbf - Receive Message from Message Buffer .................... 179
4.5.2.6 tk_rcv_mbf_u - Receive Message from Message Buffer (Microseconds) 181
4.5.2.7 tk_ref_mbf - Reference Message Buffer Status .......................... 182

4.6 Memory Pool Management Functions ....................................... 184
4.6.1 Fixed-size Memory Pool ....................................................... 185
4.6.1.1  tk_cre_mpf - Create Fixed-size Memory Pool ........................................... 186
4.6.1.2  tk_del_mpf - Delete Fixed-size Memory Pool ........................................... 189
4.6.1.3  tk_get_mpf - Get Fixed-size Memory Block .................................................. 190
4.6.1.4  tk_get_mpf_u - Get Fixed-size Memory Block (Microseconds) .......................... 192
4.6.1.5  tk_rel_mpf - Release Fixed-size Memory Block .......................................... 193
4.6.1.6  tk_ref_mpf - Reference Fixed-size Memory Pool Status ................................. 194

4.6.2  Variable-size Memory Pool .............................................................................. 196
4.6.2.1  tk_cre_mpl - Create Variable-size Memory Pool ........................................... 197
4.6.2.2  tk_del_mpl - Delete Variable-size Memory Pool ........................................... 200
4.6.2.3  tk_get_mpl - Get Variable-size Memory Block ............................................. 201
4.6.2.4  tk_get_mpl_u - Get Variable-size Memory Block (Microseconds) .................... 203
4.6.2.5  tk_rel_mpl - Release Variable-size Memory Block ....................................... 204
4.6.2.6  tk_ref_mpl - Reference Variable-size Memory Pool Status ........................... 205

4.7  Time Management Functions ............................................................................ 207
4.7.1  System Time Management .............................................................................. 208
4.7.1.1  tk_set_utc - Set System Time ................................................................. 209
4.7.1.2  tk_set_utc_u - Set Time (Microseconds) ...................................................... 211
4.7.1.3  tk_set_tim - Set System Time (TRON) ....................................................... 212
4.7.1.4  tk_set_tim_u - Set Time (TRON, Microseconds) ......................................... 213
4.7.1.5  tk_get_utc - Get System Time ................................................................. 214
4.7.1.6  tk_get_utc_u - Get System Time (Microseconds) ........................................ 216
4.7.1.7  tk_get_tim - Get System Time (TRON) ....................................................... 218
4.7.1.8  tk_get_tim_u - Get System Time (TRON, Microseconds) .............................. 220
4.7.1.9  tk_get_otm - Get Operating Time ............................................................. 222
4.7.1.10 tk_get_otm_u - Get Operating Time (Microseconds) ................................. 223

4.7.2  Cyclic Handler ............................................................................................... 224
4.7.2.1  tk_cre_cyc - Create Cyclic Handler ............................................................ 225
4.7.2.2  tk_cre_cyc_u - Create Cyclic Handler (Microseconds) ................................. 228
4.7.2.3  tk_del_cyc - Delete Cyclic Handler ............................................................ 230
4.7.2.4  tk_sta_cyc - Start Cyclic Handler .............................................................. 231
4.7.2.5  tk_stp_cyc - Stop Cyclic Handler .............................................................. 232
4.7.2.6  tk_ref_cyc - Reference Cyclic Handler Status ............................................ 233
4.7.2.7  tk_ref_cyc_u - Reference Cyclic Handler Status (Microseconds) .................. 235

4.7.3  Alarm Handler ............................................................................................... 236
4.7.3.1  tk_cre_alm - Create Alarm Handler ............................................................ 237
4.7.3.2  tk_del_alm - Delete Alarm Handler ............................................................. 239
4.7.3.3  tk_sta_alm - Start Alarm Handler ............................................................... 240
4.7.3.4  tk_sta_alm_u - Start Alarm Handler (Microseconds) .................................... 241
4.7.3.5  tk_stp_alm - Stop Alarm Handler ............................................................... 242
4.7.3.6 tk_ref_alm - Reference Alarm Handler Status .................................................. 243
4.7.3.7 tk_ref_alm_u - Reference Alarm Handler Status (Microseconds) ..................... 245

4.8 Interrupt Management Functions ........................................................................... 246
4.8.1 tk_def_int - Define Interrupt Handler ................................................................. 247
4.8.2 tk_ret_int - Return from Interrupt Handler ......................................................... 250

4.9 System Management Functions ............................................................................. 252
4.9.1 tk_rot_rdq - Rotate Ready Queue .................................................................. 253
4.9.2 tk_get_tid - Get Task Identifier ........................................................................ 255
4.9.3 tk_dis dsp - Disable Dispatch .............................................................................. 256
4.9.4 tk_ena dsp - Enable Dispatch .............................................................................. 258
4.9.5 tk_ref sys - Reference System Status ................................................................. 259
4.9.6 tk_set_pow - Set Power Mode .............................................................................. 261
4.9.7 tk_ref ver - Reference Version Information ........................................................ 263

4.10 Subsystem Management Functions ........................................................................ 266
4.10.1 tk_def ssy - Define Subsystem ............................................................................. 267
4.10.2 tk_evt ssy - Call Event Function ........................................................................ 272
4.10.3 tk_ref ssy - Reference Subsystem Status ............................................................ 274

5 \mu T-Kernel/SM Functions ......................................................................................... 276
5.1 System Memory Management Functions ................................................................. 277
5.1.1 Memory Allocation Library Functions ................................................................. 278
5.1.1.1 Kmalloc - Allocate Memory ............................................................................. 279
5.1.1.2 Kcalloc - Allocate Memory and Clear ............................................................. 280
5.1.1.3 Krealloc - Reallocate Memory .......................................................................... 281
5.1.1.4 Kfree - Release Memory ................................................................................... 283

5.2 Device Management Functions .............................................................................. 284
5.2.1 Common Notes Related to Device Drivers ............................................................ 286
5.2.1.1 Basic Concepts ................................................................................................. 286
5.2.1.1.1 Device Name (UB* type) ............................................................................. 286
5.2.1.1.2 Device ID (ID type) ..................................................................................... 287
5.2.1.1.3 Device Attribute (ATR type) ...................................................................... 287
5.2.1.1.4 Device Descriptor (ID type) ........................................................................ 288
5.2.1.1.5 Request ID (ID type) ................................................................................... 288
5.2.1.1.6 Data Number (W type, D type) ................................................................. 288
5.2.1.2 Attribute Data .................................................................................................. 289

5.2.2 Device Input/Output Operations .......................................................................... 291
5.2.2.1 tk_open_dev - Open Device ............................................................................. 292
5.2.2.2 tk_cls_dev - Close Device ................................................................................ 294
5.2.2.3 tk_rea_dev - Start Read Device ....................................................................... 295
5.2.2.4  tk_rea_dev_du - Read Device (64-bit, Microseconds) ........................................ 297
5.2.2.5  tk_srea_dev - Synchronous Read ................................................................. 299
5.2.2.6  tk_srea_dev_d - Synchronous Read (64-bit) .................................................. 301
5.2.2.7  tk_wri_dev - Start Write Device ................................................................. 303
5.2.2.8  tk_wri_dev_du - Write Device (64-bit, Microseconds) .................................... 305
5.2.2.9  tk_swri_dev - Synchronous Write ............................................................... 307
5.2.2.10 tk_swri_dev_d - Synchronous Write (64-bit) ................................................. 309
5.2.2.11 tk_wai_dev - Wait for Request Completion for Device .................................... 311
5.2.2.12 tk_wai_dev_u - Wait Device (Microseconds) ................................................ 313
5.2.2.13 tk_sus_dev - Suspends Device ........................................................................ 315
5.2.2.14 tk_get_dev - Get Device Name ....................................................................... 317
5.2.2.15 tk_ref_dev - Get Device Information ............................................................. 318
5.2.2.16 tk_oeref_dev - Get Device Information ......................................................... 319
5.2.2.17 tk_lst_dev - Get Registered Device Information ............................................ 320
5.2.2.18 tk_evt_dev - Send Driver Request Event to Device ....................................... 322
5.2.3  Registration of Device Driver ............................................................................. 323
  5.2.3.1  Registration Method of Device Driver ............................................................ 323
    5.2.3.1.1  tk_def_dev - Register Device ................................................................. 324
    5.2.3.1.2  tk_ref_idv - Reference Device Initialization Information ......................... 327
  5.2.3.2  Device Driver Interface ................................................................................. 328
    5.2.3.2.1  openfn - Open function .......................................................................... 331
    5.2.3.2.2  closefn - Close function ......................................................................... 332
    5.2.3.2.3  execfn - Execute function ....................................................................... 333
    5.2.3.2.4  waitfn - Wait-for-completion function ..................................................... 335
    5.2.3.2.5  abortfn - Abort function ........................................................................ 337
    5.2.3.2.6  eventfn - Event function ....................................................................... 339
  5.2.3.3  Device Event Notification ............................................................................. 341
  5.2.3.4  Device Suspend/Resume Processing ............................................................... 343
    5.2.3.4.1  Device suspend processing ...................................................................... 343
    5.2.3.4.2  Device resume processing ...................................................................... 343

5.3  Interrupt Management Functions ........................................................................... 344
  5.3.1  CPU Interrupt Control ....................................................................................... 345
    5.3.1.1  DI - Disable External Interrupts ................................................................... 346
    5.3.1.2  EI - Enable External Interrupt ..................................................................... 347
    5.3.1.3  isDI - Get Interrupt Disable Status ............................................................. 348
    5.3.1.4  SetCpuIntLevel - Set Interrupt Mask Level in CPU .................................... 349
    5.3.1.5  GetCpuIntLevel - Get Interrupt Mask Level in CPU .................................... 351
  5.3.2  Control of Interrupt Controller .......................................................................... 352
    5.3.2.1  EnableInt - Enable Interrupts .................................................................... 353
5.3.2.2 DisableInt - Disable Interrupts ........................................ 354
5.3.2.3 ClearInt - Clear Interrupt ............................................. 355
5.3.2.4 EndOfInt - Issue EOI to Interrupt Controller ......................... 356
5.3.2.5 CheckInt - Check Interrupt ........................................... 357
5.3.2.6 SetIntMode - Set Interrupt Mode ....................................... 358
5.3.2.7 SetCtrlIntLevel - Set Interrupt Mask Level in Interrupt Controller .................. 360
5.3.2.8 GetCtrlIntLevel - Get Interrupt Mask Level in Interrupt Controller .................. 362

5.4 I/O Port Access Support Functions ........................................... 363
5.4.1 I/O Port Access ............................................................... 363
5.4.1.1 out_b - Write to I/O Port (In Unit of Byte) ........................................ 364
5.4.1.2 out_h - Write to I/O Port (In Unit of Half-word) ......................... 365
5.4.1.3 out_w - Write to I/O Port (In Unit of Word) ............................... 366
5.4.1.4 out_d - Write to I/O Port (In Unit of Double-word) ..................... 367
5.4.1.5 in_b - Read from I/O Port (In Unit of Byte) .............................. 369
5.4.1.6 in_h - Read from I/O Port (In Unit of Half-word) ....................... 370
5.4.1.7 in_w - Read from I/O Port (In Unit of Word) ............................. 371
5.4.1.8 in_d - Read from I/O Port (In Unit of Double-word) ................... 372

5.4.2 Micro Wait ................................................................. 373
5.4.2.1 WaitUsec - Micro Wait (Microseconds) .................................... 373
5.4.2.2 WaitNsec - Micro Wait (Nanoseconds) .................................... 374

5.5 Power Management Functions ............................................... 375
5.5.1 low_pow - Move System to Low-power Mode .............................. 376
5.5.2 off_pow - Move System to Suspend State .................................. 378

5.6 System Configuration Information Management Functions ............... 380
5.6.1 System Configuration Information Acquisition ............................ 381
5.6.1.1 tk_get_cfn - Get Numbers ................................................. 382
5.6.1.2 tk_get_cfs - Get Character String ....................................... 383

5.6.2 Standard System Configuration Information ............................... 384

5.7 Memory Cache Control Functions ............................................ 386
5.7.1 SetCacheMode - Set Cache Mode .......................................... 387
5.7.2 ControlCache - Control Cache ............................................. 389

5.8 Physical Timer Functions ..................................................... 391
5.8.1 Use Case of Physical Timer .................................................. 392
5.8.2 StartPhysicalTimer - Start Physical Timer .................................. 394
5.8.3 StopPhysicalTimer - Stop Physical Timer ................................... 396
5.8.4 GetPhysicalTimerCount - Get Physical Timer Count ....................... 398
5.8.5 DefinePhysicalTimerHandler - Define Physical Timer Handler .................. 399
5.8.6 GetPhysicalTimerConfig - Get Physical Timer Configuration Information .................. 401

5.9 Utility Functions ................................................................. 403
6.1.24 td_ref_flg - Reference Event Flag Status ............................................. 443
6.1.25 td_ref_mbx - Reference Mailbox Status .................................................. 444
6.1.26 td_ref_mtx - Refer Mutex Status ............................................................. 445
6.1.27 td_ref_mbf - Reference Message Buffer Status ....................................... 446
6.1.28 td_ref_mpf - Reference Fixed-size Memory Pool Status .......................... 447
6.1.29 td_ref_mpl - Reference Variable-size Memory Pool Status ..................... 448
6.1.30 td_ref_cyc - Reference Cyclic Handler Status ......................................... 449
6.1.31 td_ref_cyc_u - Reference Cyclic Handler Status (Microseconds) ............... 450
6.1.32 td_ref_alm - Reference Alarm Handler Status ......................................... 451
6.1.33 td_ref_alm_u - Reference Alarm Handler Status (Microseconds) .............. 452
6.1.34 td_ref_sys - Reference System Status ..................................................... 453
6.1.35 td_ref_ssy - Reference Subsystem Status .................................................. 454
6.1.36 td_get_reg - Get Task Register ............................................................... 455
6.1.37 td_set_reg - Set Task Registers .............................................................. 457
6.1.38 td_get_utc - Get System Time ................................................................. 458
6.1.39 td_get_utc_u - Get System Time (Microseconds) ..................................... 460
6.1.40 td_get_tim - Get System Time (TRON) ..................................................... 461
6.1.41 td_get_tim_u - Get System Time (TRON, Microseconds) .......................... 463
6.1.42 td_get_otm - Get Operating Time ............................................................ 464
6.1.43 td_get_otm_u - Get Operating Time (Microseconds) ............................... 466
6.1.44 td_ref_dsname - Refer to DS Object Name ................................................ 467
6.1.45 td_set_dsname - Set DS Object Name ....................................................... 469

6.2 Trace Functions .............................................................................................. 471
6.2.1 td_hok_svc - Define System Call/Extended SVC Hook Routine .................... 472
6.2.2 td_hok_dsp - Define Task Dispatch Hook Routine .................................... 474
6.2.3 td_hok_int - Define Interrupt Handler Hook Routine ................................ 476

7 Appendix ............................................................................................................ 478
7.1 System Configuration ....................................................................................... 479
7.2 Keywords ......................................................................................................... 480

8 Reference .......................................................................................................... 481
8.1 List of C Language Interface ......................................................................... 482
8.1.1 μT-Kernel/OS ............................................................................................. 482
8.1.1.1 Task Management Functions ................................................................. 482
8.1.1.2 Task Synchronization Functions ............................................................. 482
8.1.1.3 Task Exception Handling Functions ..................................................... 483
8.1.1.4 Synchronization and Communication Functions ................................... 483
8.1.1.5 Extended Synchronization and Communication Functions .................. 484
8.1.1.6 Memory Pool Management Functions ........................................ 484
8.1.1.7 Time Management Functions ..................................................... 485
8.1.1.8 Interrupt Management Functions ................................................. 485
8.1.1.9 System Management Functions ................................................... 485
8.1.1.10 Subsystem Management Functions .......................................... 486

8.1.2 \( \mu \)T-Kernel/SM ................................................................. 486
   8.1.2.1 System Memory Management Functions ................................ 486
   8.1.2.2 Device Management Functions .............................................. 486
   8.1.2.3 Interrupt Management Functions ........................................... 487
   8.1.2.4 I/O Port Access Support Functions ....................................... 488
   8.1.2.5 Power Management Functions ............................................... 488
   8.1.2.6 System Configuration Information Management Functions ........ 488
   8.1.2.7 Memory Cache Control Functions ........................................... 488
   8.1.2.8 Physical Timer Functions ..................................................... 488
   8.1.2.9 Utility Functions ................................................................. 489

8.1.3 \( \mu \)T-Kernel/DS ............................................................... 489
   8.1.3.1 Kernel Internal State Acquisition Functions ......................... 489
   8.1.3.2 Trace Functions ................................................................. 490

8.2 List of Error Codes ................................................................. 491
   8.2.1 Normal Completion Error Class (0) ....................................... 491
   8.2.2 Normal completion Internal Error Class (5 to 8) ....................... 491
   8.2.3 Unsupported Error Class (9 to 16) ........................................ 491
   8.2.4 Parameter Error Class (17 to 24) .......................................... 491
   8.2.5 Call Context Error Class (25 to 32) .................................. 492
   8.2.6 Resource Constraint Error Class (33 to 40) ............................ 492
   8.2.7 Object State Error Class (41 to 48) ..................................... 493
   8.2.8 Wait Error Class (49 to 56) ............................................... 493
   8.2.9 Device Error Class (57 to 64) (\( \mu \)T-Kernel/SM) .................. 493
   8.2.10 Status Error Class (65 to 72) (\( \mu \)T-Kernel/SM) ................. 493

8.3 List of APIs and Service Profile Items .......................................... 494
   8.3.1 \( \mu \)T-Kernel/OS ................................................................. 494
      8.3.1.1 Task Management Functions ............................................ 494
      8.3.1.2 Task Synchronization Functions ....................................... 494
      8.3.1.3 Task Exception Handling Functions ................................ 495
      8.3.1.4 Synchronization and Communication Functions .................. 495
      8.3.1.5 Extended Synchronization and Communication Functions .... 495
      8.3.1.6 Memory Pool Management Functions ................................ 496
      8.3.1.7 Time Management Functions ........................................... 496
      8.3.1.8 Interrupt Management Functions ..................................... 497
8.3.1.9 System Management Functions ........................................... 497
8.3.1.10 Subsystem Management Functions ................................. 497
8.3.2 μT-Kernel/SM ................................................................. 498
  8.3.2.1 System Memory Management Functions ....................... 498
  8.3.2.2 Device Management Functions .................................. 498
  8.3.2.3 Interrupt Management Functions ............................... 499
  8.3.2.4 I/O Port Access Support Functions ............................. 500
  8.3.2.5 Power Management Functions .................................. 500
  8.3.2.6 System Configuration Information Management Functions .... 500
  8.3.2.7 Memory Cache Control Functions ............................... 500
  8.3.2.8 Physical Timer Functions ....................................... 500
  8.3.2.9 Utility Functions .................................................. 501
8.3.3 μT-Kernel/DS ................................................................. 501
  8.3.3.1 Kernel Internal State Acquisition Functions ................. 501
  8.3.3.2 Trace Functions .................................................. 502
List of Figures

1.1 Position and Structure of $\mu$T-Kernel 3.0 ............................................. 15
2.1 Task State Transition Diagram ................................................................. 27
2.2 Precedence in Initial State ............................................................... 29
2.3 Precedence After Task B Goes To RUNNING State .................................... 30
2.4 Precedence After Task B Goes To WAITING State ..................................... 30
2.5 Precedence After Task B WAITING State Is Released ............................... 30
2.6 Classification of System States ............................................................. 34
2.7 Interrupt Nesting and Delayed Dispatching ............................................. 35
3.1 Behavior of High-Level Language Support Routine .................................... 50
4.1 Multiple Tasks Waiting for One Event Flag ............................................ 141
4.2 Format of Messages Using a Mailbox ...................................................... 145
4.3 Synchronous Communication by Message Buffer ....................................... 170
4.4 Synchronous Communication Using Message Buffer of $bufsz = 0$ ............... 173
4.5 Precedence Before Issuing $tk\_rot\_rdq$ .................................................... 254
4.6 Precedence After Issuing $tk\_rot\_rdq (tskpri = 2)$ ..................................... 254
4.7 $maker$ Format ....................................................................................... 264
4.8 $prid$ Format ......................................................................................... 264
4.9 $spver$ Format ....................................................................................... 264
4.10 $\mu$T-Kernel Subsystems ........................................................................ 266
5.1 Device Management Functions ................................................................... 285
List of Tables

2.1 State Transitions Distinguishing Invoking Task and Other Tasks ........................................ 28
4.1 Target Task State and Execution Result (tk_ter_tsk) .......................................................... 70
4.2 Values of tskwait and wid ..................................................................................................... 82
4.3 Target Task State and Execution Result (tk_rel_wai) .......................................................... 92
5.1 Whether Concurrent Open of Same Device is Allowed or NOT ............................................ 293
API Notation

In the parts of this specification that describe APIs, the specification of each API (Application Programming Interface) is explained in the format illustrated below. In addition to system calls that directly call kernel functions, APIs include functions implemented as extended SVCs (extended system calls), macros, and libraries.

API Name - Description

This is an API name and its description.

C Language Interface

This is an API’s C language interface and header file(s) to include.

Parameter

Describes an API’s parameter(s), i.e. information passed to the \( \mu \) T-Kernel when the API is issued.

Return Parameter

Describes an API’s return parameter(s), i.e. information returned by the \( \mu \) T-Kernel when the execution of the API ends.

A return parameter that is returned as an API’s function value may be called ”return code.” A return parameter can include, besides return code, a value stored at a pointer that points at memory location where some information can be stored.

Error Code

Describes errors that can occur in an API.

The following error codes are common to all APIs and are not included in the error code listings for each API: E_SYS, E_NOSPT, E_RSFN, E_MACV, E_OACV.

The detection of the error conditions that may result in the following error codes is implementation-dependent; such conditions may not always be detected as errors:

E_PAR, E_MACV, E_CTX.

Error code E_CTX is included in the error code section of individual API only when API can encounter an error due to a semantically wrong caller context: e.g., the case of task-independent portion’s calling an API that can block. If an API’s constraints in the caller’s context are implementation-dependent, and such semantic errors
are not universal across all implementations, the explanation of E_CTX is not included in the error section of the API under discussion. Implementations may generate errors that are not explained in the explanation section of error codes.

Valid Context

Indicates the context (task portion, quasi-task portion, and task-independent portion) that can issue the API under consideration. Note that items marked with "x" are sometimes clearly impossible to use in the context discussed, but the usability of some items in the context discussed may be implementation-dependent, and some may be usable in some implementations.

Related Service Profile Items

The relation of the service profile item(s) associated with API is shown.

Description

Describes the API functions.

When the values to be passed in a parameter are selected from various choices, the following notation is used in the parameter descriptions:

\[( x \| y \| z )\]

Set one of x, y, or z.

\[x | y\]

Both x and y can be set at the same time (in which case the logical sum of x and y is taken).

\[[ x ]\]

x is optional.

Example of Using Parameters Notation

```
wfmode := (TWF_ANDW || TWF_ORW) | [TWF_CLR]
```

The above description means that wfmode can be specified in any of the following four ways:

- TWF_ANDW
- TWF_ORW
- \((TWF_ANDW \mid TWF_CLR)\)
- \((TWF_ORW \mid TWF_CLR)\)

Additional Notes

Supplements the description by noting matters that need special attention or caution, etc.

Rationale for the Specification

Explains the reason for adopting a particular approach and specification.
Index of μT-Kernel/OS APIs

The μT-Kernel/OS system APIs described in this specification are listed below in alphabetical order.

- `tk_can_wup` - Cancel Wakeup Task
- `tk_chg_pri` - Change Task Priority
- `tk_clr_flg` - Clear Event Flag
- `tk_cre_alm` - Create Alarm Handler
- `tk_cre_cyc` - Create Cyclic Handler
- `tk_cre_cyc_u` - Create Cyclic Handler (Microseconds)
- `tk_cre_flg` - Create Event Flag
- `tk_cre_mbf` - Create Message Buffer
- `tk_cre_mbx` - Create Mailbox
- `tk_cre_mpf` - Create Fixed-size Memory Pool
- `tk_cre_mpl` - Create Variable-size Memory Pool
- `tk_cre_mtx` - Create Mutex
- `tk_cre_sem` - Create Semaphore
- `tk_cre_tsk` - Create Task
- `tk_def_int` - Define Interrupt Handler
- `tk_def_ssy` - Define Subsystem
- `tk_def_tex` - Define Task Exception Handler
- `tk_del_alm` - Delete Alarm Handler
- `tk_del_cyc` - Delete Cyclic Handler
- `tk_del_flg` - Delete Event Flag
- `tk_del_mbf` - Delete Message Buffer
- `tk_del_mbx` - Delete Mailbox
- `tk_del_mpf` - Delete Fixed-size Memory Pool
- `tk_del_mpl` - Delete Variable-size Memory Pool
- `tk_del_mtx` - Delete Mutex
- `tk_del_sem` - Delete Semaphore
tk_del_tsk - Delete Task

tk_dis_dsp - Disable Dispatch

tk_dis_tex - Disable Task Exception

tk_dis_wai - Disable Task Wait

tk_dly_tsk - Delay Task

tk_dly_tsk_u - Delay Task (Microseconds)

tk_ena_dsp - Enable Dispatch

tk_ena_tex - Enable Task Exception

tk_ena_wai - Enable Task Wait

tk_end_tex - End Task Exception Handler

tk_evt_ssy - Call Event Function

tk_exd_tsk - Exit and Delete Task

tk_ext_tsk - Exit Task

tk_frsm_tsk - Force Resume Task

tk_get_cpr - Get Task Coprocessor Registers

tk_get_mpf - Get Fixed-size Memory Block

tk_get_mpf_u - Get Fixed-size Memory Block (Microseconds)

tk_get_mpl - Get Variable-size Memory Block

tk_get_mpl_u - Get Variable-size Memory Block (Microseconds)

tk_get_otm - Get Operating Time

tk_get_otm_u - Get Operating Time (Microseconds)

tk_get_reg - Get Task Registers

tk_get_tid - Get Task Identifier

tk_get_tim - Get System Time (TRON)

tk_get_tim_u - Get System Time (TRON, Microseconds)

tk_get_utc - Get System Time

tk_get_utc_u - Get System Time (Microseconds)

tk_loc_mtx - Lock Mutex

tk_loc_mtx_u - Lock Mutex (Microseconds)

tk_ras_tex - Raise Task Exception

tk_rcv_mbf - Receive Message from Message Buffer

tk_rcv_mbf_u - Receive Message from Message Buffer (Microseconds)

tk_rcv_mbx - Receive Message from Mailbox

tk_rcv_mbx_u - Receive Message from Mailbox (Microseconds)

tk_ref_alm - Reference Alarm Handler Status
• tk_ref alm u - Reference Alarm Handler Status (Microseconds)
• tk_ref cyc - Reference Cyclic Handler Status
• tk_ref cyc u - Reference Cyclic Handler Status (Microseconds)
• tk_ref flg - Reference Event Flag Status
• tk_ref mbf - Reference Message Buffer Status
• tk_ref mbx - Reference Mailbox Status
• tk_ref mpf - Reference Fixed-size Memory Pool Status
• tk_ref mpl - Reference Variable-size Memory Pool Status
• tk_ref mtx - Refer Mutex Status
• tk_ref sem - Reference Semaphore Status
• tk_ref ssy - Reference Subsystem Status
• tk_ref sys - Reference System Status
• tk_ref tex - Reference Task Exception Status
• tk_ref tsk - Reference Task Status
• tk_ref ver - Reference Version Information
• tk_rel mpf - Release Fixed-size Memory Block
• tk_rel mpl - Release Variable-size Memory Block
• tk_rel wai - Release Wait
• tk_ret int - Return from Interrupt Handler
• tk_rot rdq - Rotate Ready Queue
• tk_rsm tsk - Resume Task
• tk_set cpr - Set Task Coprocessor Registers
• tk_set flg - Set Event Flag
• tk_set pow - Set Power Mode
• tk_set reg - Set Task Registers
• tk_set tim - Set System Time (TRON)
• tk_set tim u - Set System Time (TRON, Microseconds)
• tk_set utc - Set System Time
• tk_set utc u - Set System Time (Microseconds)
• tk_sig sem - Signal Semaphore
• tk_sig tev - Signal Task Event
• tk_slp tsk - Sleep Task
• tk_slp tsk u - Sleep Task (Microseconds)
• tk_snd mbf - Send Message to Message Buffer
• tk_snd mbf u - Send Message to Message Buffer (Microseconds)
• **tk_snd_mbx** - Send Message to Mailbox
• **tk_sta_alm** - Start Alarm Handler
• **tk_sta_alm_u** - Start Alarm Handler (Microseconds)
• **tk_sta_cyc** - Start Cyclic Handler
• **tk_sta_tsk** - Start Task
• **tk_stp_alm** - Stop Alarm Handler
• **tk_stp_cyc** - Stop Cyclic Handler
• **tk_sus_tsk** - Suspend Task
• **tk_ter_tsk** - Terminate Task
• **tk_unl_mtx** - Unlock Mutex
• **tk_wai_flg** - Wait Event Flag
• **tk_wai_flg_u** - Wait Event Flag (Microseconds)
• **tk_wai_sem** - Wait on Semaphore
• **tk_wai_sem_u** - Wait on Semaphore (Microseconds)
• **tk_wai_tev** - Wait Task Event
• **tk_wai_tev_u** - Wait Task Event (Microseconds)
• **tk_wup_tsk** - Wakeup Task
Index of μT-Kernel/SM APIs

The μT-Kernel/SM system APIs described in this specification are listed below in alphabetical order.

- abortfn - Abort function
- CheckInt - Check Interrupt
- ClearInt - Clear Interrupt
- closefn - Close function
- ControlCache - Control Cache
- CreateLock - Create Fast Lock
- CreateMLock - Create Fast Multi-lock
- DefinePhysicalTimerHandler - Define Physical Timer Handler
- DeleteLock - Delete Fast Lock
- DeleteMLock - Delete Fast Multi-lock
- DI - Disable External Interrupts
- DisableInt - Disable Interrupts
- EI - Enable External Interrupts
- EnableInt - Enable Interrupts
- EndOfInt - Issue EOI to Interrupt Controller
- eventfn - Event function
- execfn - Execute function
- GetCpuIntLevel - Get CPU Interrupt Mask Level
- GetCtrlIntLevel - Get Interrupt Controller Interrupt Mask Level
- GetPhysicalTimerConfig - Get Physical Timer Configuration Information
- GetPhysicalTimerCount - Get Physical Timer Count
- in_b - Read from I/O Port (in Bytes)
- in_d - Read from I/O Port (in Double-words)
- in_h - Read from I/O Port (in Half-words)
- in_w - Read from I/O Port (in Words)
- isDI - Get Interrupt Disable Status
- **Kcalloc** - Allocate Memory and Clear
- **Kfree** - Release Memory
- **Kmalloc** - Allocate Memory
- **Krealloc** - Reallocate Memory
- **Lock** - Lock Fast Lock
- **low_pow** - Move System to Low-power Mode
- **MLock** - Lock Fast Multi-lock
- **MLockTmo** - Lock Fast Multi-lock (with Timeout)
- **MLockTmo_u** - Lock Fast Multi-lock (with Timeout, Microseconds)
- **MUnlock** - Unlock Fast Multi-lock
- **off_pow** - Move System to Suspend State
- **openfn** - Open function
- **out_b** - Write to I/O Port (in Bytes)
- **out_d** - Write to I/O Port (in Double-words)
- **out_h** - Write to I/O Port (in Half-words)
- **out_w** - Write to I/O Port (in Words)
- **SetCacheMode** - Set Cache Mode
- **SetCpuIntLevel** - Set CPU Interrupt Mask Level
- **SetCtrlIntLevel** - Set Interrupt Controller Interrupt Mask Level
- **SetIntMode** - Set Interrupt Mode
- **SetOBJNAME** - Set Object Name
- **StartPhysicalTimer** - Start Physical Timer
- **StopPhysicalTimer** - Stop Physical Timer
- **tk_cls_dev** - Close Device
- **tk_def_dev** - Register Device
- **tk_evt_dev** - Send Driver Request Event to Device
- **tk_get_cfn** - Get Numbers
- **tk_get_cfs** - Get Character String
- **tk_get_dev** - Get Device Name
- **tk_lst_dev** - Get Registered Device Information
- **tk_opn_dev** - Open Device
- **tk_oref_dev** - Get Device Information
- **tk_rea_dev** - Start Read Device
- **tk_rea_dev_du** - Read Device (64-bit, Microseconds)
- **tk_ref_dev** - Get Device Information
• **tk_ref_idv** - Reference Device Initialization Information
• **tk_srea_dev** - Synchronous Read
• **tk_srea_dev_d** - Synchronous Read (64-bit)
• **tk_sus_dev** - Suspends Device
• **tk_swri_dev** - Synchronous Write
• **tk_swri_dev_d** - Synchronous Write (64-bit)
• **tk_wai_dev** - Wait for Request Completion for Device
• **tk_wai_dev_u** - Wait Device (Microseconds)
• **tk_wri_dev** - Start Write Device
• **tk_wri_dev_du** - Write Device (64-bit, Microseconds)
• **Unlock** - Unlock Fast Lock
• **waitfn** - Wait function
• **WaitNsec** - Micro Wait (Nanoseconds)
• **WaitUsec** - Micro Wait (Microseconds)
Index of \( \mu T \)-Kernel/DS APIs

The \( \mu T \)-Kernel/DS APIs described in this specification are listed below in alphabetical order.

- `td_flg_que` - Reference Event Flag Queue
- `td_get_otm` - Get Operating Time
- `td_get_otm_u` - Get Operating Time (Microseconds)
- `td_get_reg` - Get Task Register
- `td_get_tim` - Get System Time (TRON)
- `td_get_tim_u` - Get System Time (TRON, Microseconds)
- `td_get_utc` - Get System Time
- `td_get_utc_u` - Get System Time (Microseconds)
- `td_hok_dsp` - Define Task Dispatch Hook Routine
- `td_hok_int` - Define Interrupt Handler Hook Routine
- `td_hok_svc` - Define System Call/Extended SVC Hook Routine
- `td_lst_alm` - Reference Alarm Handler ID List
- `td_lst_cyc` - Reference Cyclic Handler ID List
- `td_lst_flg` - Reference Event Flag ID List
- `td_lst_mbf` - Reference Message Buffer ID List
- `td_lst_mbx` - Reference Mailbox ID List
- `td_lst_mpf` - Reference Fixed-size Memory Pool ID List
- `td_lst_mpl` - Reference Variable-size Memory Pool ID List
- `td_lst_mtx` - Reference Mutex ID List
- `td_lst_sem` - Reference Semaphore ID List
- `td_lst_ssy` - Reference Subsystem ID List
- `td_lst_tsk` - Reference Task ID List
- `td_mbx_que` - Reference Mailbox Queue
- `td_mpf_que` - Reference Fixed-size Memory Pool Queue
- `td_mpl_que` - Reference Variable-size Memory Pool Queue
- `td_mtx_que` - Reference Mutex Queue
- `td_rdy_que` - Reference Task Precedence
- `td_ref_alm` - Reference Alarm Handler Status
- `td_ref_alm_u` - Reference Alarm Handler Status (Microseconds)
- `td_ref_cyc` - Reference Cyclic Handler Status
- `td_ref_cyc_u` - Reference Cyclic Handler Status (Microseconds)
- `td_ref_dsname` - Refer to DS Object Name
- `td_ref_flg` - Reference Event Flag Status
- `td_ref_mbf` - Reference Message Buffer Status
- `td_ref_mbx` - Reference Mailbox Status
- `td_ref_mpf` - Reference Fixed-size Memory Pool Status
- `td_ref_mpl` - Reference Variable-size Memory Pool Status
- `td_ref_mtx` - Refer Mutex Status
- `td_ref_sem` - Reference Semaphore Status
- `td_ref_ssy` - Reference Subsystem Status
- `td_ref_sys` - Reference System Status
- `td_ref_tex` - Reference Task Exception Status
- `td_ref_tsk` - Get Task Status
- `td_rmbf_que` - Reference Message Buffer Receive Queue
- `td_sem_que` - Reference Semaphore Queue
- `td_set_dsname` - Set DS Object Name
- `td_set_reg` - Set Task Registers
- `td_smbf_que` - Reference Message Buffer Send Queue
Chapter 1

Overview of $\mu$T-Kernel 3.0
1.1 TRON Project and μT-Kernel 3.0

This standard defines the specification of a real-time operating system (RTOS) called "μT-Kernel 3.0." μT-Kernel 3.0 is the latest result from the TRON project (http://www.tron.org/), which was started by Dr. Ken Sakamura, then at the University of Tokyo in 1984.

The TRON Project envisioned that environments optimized to humans would be created by embedding small microprocessors, invented in the prior decade, in many objects in our surroundings and having them talk to each other. In the TRON Project, the computing paradigm to achieve this goal was called a "Highly Functionally Distributed System (HFDS)" and an RTOS called ITRON was created to control such microprocessors efficiently. The specification of the first version of ITRON, namely ITRON1, was published in 1987. The project promoted the industry-academic cooperation and published the technical specification and other information so that anyone can make use of the technology for free under the philosophy of "Open Approach." As a result, ITRON specification OS was born and it ran on many types of processors. It became the de facto standard RTOS for embedded computer systems. Additionally, the development of "μ ITRON," which is an improved version of ITRON and has better adaptability, proceeded concurrently. OSs based on ITRON and μ ITRON specifications have been used widely in many embedded computer systems: they are used in consumer products, such as home electronic appliances and AV equipment, and industrial applications, such as machine control on factory floors, engine control of automobiles, etc.

The concept of "HFDS," which the TRON Project proposed, started to be called "ubiquitous computing" before the turn of the century and is now widely recognized as the Internet of Things (IoT). Since its inception in 1984, the TRON Project has targeted the IoT in today's parlance as the main application field of microprocessors and carried out research and development of OS and computer architecture. The latest result of such research and development of OS is μ T-Kernel, a resource-efficient RTOS suitable for IoT edge nodes. It is an improvement of μ ITRON, and has features for IoT.

Based on the adoptions so far, it has been reported that 60 percent or more of embedded devices use the results of the TRON Project in the 2010s, 30 years after the inception of the project (https://www.tron.org/blog/2017/07/press20170406/).

In 2018, IEEE (Institute of Electrical and Electronics Engineers), a global standard creating organization, published the IEEE 2050-2018 standard specification for RTOS for IoT edge nodes based on the specification of the updated version of μ T-Kernel, μ T-Kernel 2.0.

This document defines the standard of μ T-Kernel 3.0, an updated version of μ T-Kernel 2.0 by streamlining some features to adapt them specifically for controlling IoT edge devices. μ T-Kernel 3.0 maintains high compatibility with IEEE 2050-2018 by adopting the API added in IEEE 2050-2018. As a result, μ T-Kernel 3.0 specification is completely upper compatible with IEEE 2050-2018.
1.2 Design Policy of $\mu$T-Kernel 3.0

$\mu$T-Kernel 3.0 specification defines an embedded real-time OS with small resource footprint meant for controlling IoT edge devices. $\mu$T-Kernel improves development efficiency and interoperability of the software by standardizing basic OS functions and API specification. It is designed to deliver high performance even on a lower-end single-chip microcontroller unit (MCU), including 16-bit MCU, MCUs without a memory management unit (MMU), and small-scale embedded systems with a small amount ROM/RAM. Furthermore, $\mu$T-Kernel has functions such as device driver control and power saving, so it can build low-power systems in which various types of devices and communication methods are embedded for building an IoT network.

The $\%$utk 3.0 specification defines a standard for an RTOS, and it can be implemented on many types of CPUs irrespective of CPU architectures. At the same time, the designers are aware that it makes sense to adapt the OS implementation or limit the OS functions in the case of very resource-poor systems in order to cope with a particular choice of CPU and hardware configuration as in the case of tiny IoT edge nodes. To cope with such situations, a concept and description of ”service profile” has been introduced in $\%$utk to leave room for the flexible implementation of the OS, at the same time retaining the compatibility of software and portability. Service profile makes it possible to formally describe the omission and/or difference of functions available in a particular implementation of the OS. This makes it easy for middleware and applications that run under the OS to learn and cope with the implementation-dependent differences.

The $\%$utk 3.0 specification includes common definitions such as data types, the state transition of tasks, which are the basic units of parallel execution inside programs, non-task behavior such as that of interrupt handler, and API specifications provided by the OS. File system management, network communication, process management, etc., are not included in the $\%$utk 3.0 specification. However, by adding appropriate middleware packages, we can build relatively large systems using $\%$utk 3.0. That is, $\%$utk 3.0 can be used as a microkernel for a large system that has functions such as file system management, network communication, and process management. It can be extended to support multi-core processors as well.
1.3 Structure of μT-Kernel 3.0

%utk 3.0 consists of "%utk/OS" that handles the intrinsic functions of RTOS such as task scheduling, synchronization, and communication between tasks, "%utk/SM" that offers additional system management functions, and "%utk/DS" that offers functions for software debugger. The position and structure of μT-Kernel 3.0 is shown in Figure 1.1, “Position and Structure of μT-Kernel 3.0”.

μT-Kernel/OS provides the following functions:

- Task Management Functions
- Task Synchronization Functions
- Task Exception Handling Functions
- Synchronization and Communication Functions
- Extended Synchronization and Communication Functions
- Memory Pool Management Functions
- Time Management Functions
- Interrupt Management Functions
- System Management Functions
- Subsystem Management Functions

μT-Kernel/SM provides the following kinds of functions:
- System Memory Management Functions
- Device Management Functions
- Interrupt Management Functions
- I/O Port Access Support Functions
- Power Management Functions
- System Configuration Information Management Functions
- Memory Cache Control Functions
- Physical Timer Functions
- Utility Functions

μT-Kernel/DS provides the following kinds of functions exclusively for debugging use:

- Kernel Internal State Acquisition Functions
- Trace Functions
1.4 Reference Code

Since the optimization and customization is very important on small scale embedded systems, μT-Kernel 3.0 does not aim at the uniqueness of source code unlike T-Kernel. Instead it offers reference code, a source code that can be referenced as a sample implementation.

Reference code is an example of an implementation of μT-Kernel 3.0, and is distributed by TRON Forum. A major difference with T-Kernel is that this reference code is not the only implementation of μT-Kernel 3.0, and it is free for any OS implementer to modify this reference code, or implement it from scratch. However, only those that behave exactly as the reference code is officially recognized as μT-Kernel 3.0 specification OS.

The reference code has been provided to specify behaviors which are difficult to describe in the specification, and the introduction of the reference code has made it possible to assure and check the uniform behavior across different implementations that are optimized and customized to target systems.
1.5 Adaptability and Service Profile

\( \mu \text{T-Kernel 3.0} \) has been designed by paying attention to the compatibility with T-Kernel. Namely, porting between \( \mu \text{T-Kernel 3.0} \) and T-Kernel ought to be simple. If only common features are used, a simple re-compilation will do. Even if changes are necessary, the amount of change is small. That is the design goal.

For features that have strong dependency on hardware such as MMU and FPU, unnecessary features for the intended narrow target, and features that have potential implications for run-time efficiency such as hooks for debug support, the specification allows subsetting. To accommodate the subsetting in this manner, and the desire to keep the distribution and portability of middleware and application high, such software needs to obtain information about the implementation details of \( \mu \text{T-Kernel 3.0} \). \( \mu \text{T-Kernel 3.0} \) has introduced a mechanism, "service profile", to let each implementation of \( \mu \text{T-Kernel 3.0} \) describe implementation details clearly. All \( \mu \text{T-Kernel 3.0} \) implementations shall provide a service profile and provide information on the functions that were omitted to create a subset.
1.6 Implementation Specification Document

With the introduction of service profile, not all the implementations of μT-Kernel 3.0 provide all the features in the specification completely.

Hence, to let the user verify the implementation-dependent details of a particular implementation of μT-Kernel 3.0, and avoid spending time to understand unexpected behavior, all μT-Kernel 3.0 implementations shall produce an implementation specification document. The implementation specification document shall describe the following at least.

Version number of μT-Kernel 3.0 specification
   Clearly specify the major and minor version numbers of μT-Kernel 3.0.

Information about the Service Profile
   Explicitly specify the values of all the service profile items.
1.7 Relation with Existing RTOS Specifications

This section lists the major differences between the \( \mu \) T-Kernel 3.0 specification and other legacy RTOS specifications that have close relationship with \( \mu \) T-Kernel 3.0.

1.7.1 Relation with \( \mu \) T-Kernel 2.0

1. Removal of features that assumes process management and virtual memory
\( \mu \) T-Kernel 3.0 defines a real-time OS to control small embedded systems and IoT edge nodes equipped with 16-bit or 32-bit CPU. It is not designed to be used as the OS kernel with process management or virtual memory for generic information processing systems. Because of this design decision, Address Space Management Functions (Address Space Configuration, Address Space Checking, Logical Address Space Management) and System Memory Allocation function which \( \mu \) T-Kernel 2.0 has are not included in \( \mu \) T-Kernel 3.0. Also, among Subsystem Management Functions, startup/cleanup processing and functions for resource group are not included in \( \mu \) T-Kernel 3.0.

2. Addition of handling system time
API that uses 0:00:00 of January 1st 1970 (UTC) as epoch to set system time have been added to \( \mu \) T-Kernel 3.0: these are \texttt{tk_set_utc}, \texttt{tk_set_utc_u}, \texttt{tk_get_utc}, \texttt{tk_get_utc_u}, \texttt{td_get_utc}, \texttt{td_get_utc_u}.

3. Removal of Rendezvous Function
\( \mu \) T-Kernel 3.0 does not have Rendezvous function, one of Extended Synchronization and Communication Functions of T-Kernel.

1.7.2 Relation with T-Kernel 2.0

1. Removal of features that assumes process management and virtual memory
\( \mu \) T-Kernel 3.0 defines a real-time OS to control small embedded systems and IoT edge nodes equipped with 16-bit or 32-bit CPU. It is not designed to be used as the OS kernel with process management or virtual memory for generic information processing systems. Because of this design decision, Address Space Management Functions (Address Space Configuration, Address Space Checking, Logical Address Space Management) and System Memory Allocation function which T-Kernel 2.0 has are not included in \( \mu \) T-Kernel 3.0. Also, among Subsystem Management Functions, startup/cleanup processing and functions for resource group are not included in \( \mu \) T-Kernel 3.0.

2. Introduction of service profile
For \( \mu \) T-Kernel that addresses the needs of small-scale embedded systems, the specification aims at the ease of optimization and customization. However, at the same time, to improve the ease of distribution of middleware and applications by increasing portability, a formal mechanism to describe the issues for implementation-dependency of \( \mu \) T-Kernel is now introduced. For details, see Section 2.8, "Service Profile".

3. Specification of user buffer
APIs that need to use internal memory on the stack or in the memory pools can use a user-specified buffer area instead of using the automatically allocated area by the kernel. Specification by \texttt{TA_USERBUF} is enough to use a user-specified buffer in general.

4. Type changes for supporting 16-bit CPU
\( \mu \) T-Kernel needs to support 16-bit CPU, and the integer that can be represented by INT or UINT type may be restricted to 16-bit integer values. For this reason, some arguments of APIs and members of structures now have wide enough scalar types, instead of INT or UINT types, so that they can present the values adequately.
5. Customization for small-scale embedded systems

μT-Kernel is meant for small-scale embedded systems, and so the specification has been tuned to such usage. For example, an implementation with a smaller value, than in T-Kernel, for the largest value of task priority is allowed.

6. Re-organization and extension of interrupt management function

μT-Kernel 3.0 offers interrupt management functions that are based on those of T-Kernel 2.0 after re-organizing and extending these one way or the other. There are differences as follows.

(a) Addition of functions to obtain and set interrupt mask level

Add APIs for obtaining and setting the interrupt mask level of CPU and/or interrupt controller namely `SetCpuIntLevel`, `GetCpuIntLevel`, `SetCtrlIntLevel`, and `GetCtrlIntLevel`.

(b) Abolishing interrupt vector number (INTVEC)

In order to simplify number systems used for interrupts and make it simple to understand, we abolished with the specification using interrupt vector number (INTVEC) For APIs that take INTVEC as argument in T-Kernel 2.0, we use the common interrupt number used in `tk_def_int()` as the argument instead of INTVEC.

1.7.3 Relation with IEEE 2050-2018

Specification of μT-Kernel 3.0 is upward compatible with the specification of IEEE 2050-2018 standard which IEEE published for the standard RTOS for IoT edge nodes. Because of this, an OS that satisfies the μT-Kernel 3.0 specification automatically satisfies IEEE 2050-2018 specification.

On the other hand, the specification of IEEE 2050-2018 is a subset of μT-Kernel 3.0 specification. The following functions of μT-Kernel 3.0 are not included in IEEE 2050-2018: Subsystem Management Functions, and Kernel Internal State Acquisition Functions and Trace Functions for debugging purposes provided by μT-Kernel/DS. Also the following functions are not in IEEE 2050-2018: Functions that handle DS Object Names(`dsnname`) and APIs to handle system time using 00:00:00, January 1, 1985 (GMT) as epoch (`tk_get_tim, tk_get_tim_u, tk_set_tim, tk_set_tim_u`).
Chapter 2

μT-Kernel Concepts
2.1 Meaning of Basic Terminology

Real-time system and real-time operating system (RTOS)
A system whose response time and delay time are deterministic without uncertainty and non-reproducibility and has an internal configuration that makes the worst value predictable or makes it easy to produce an educated guess value is called a real-time system.

μT-Kernel is the real-time operating system (RTOS) that is used for building real-time systems with the preceding characteristics.

Task, invoking task
The basic logical unit of concurrent program execution is called a "task." Whereas the code in one task is executed in sequence, codes in different tasks can be executed in parallel. This concurrent processing is a conceptual phenomenon, from the standpoint of applications; in actual implementation it is accomplished by time-sharing among tasks as controlled by the kernel.
A task that invokes a system call is called the "invoking task."

Dispatch, dispatcher
The switching of tasks executed by the processor is called "dispatching" (or task dispatching). The kernel mechanism by which dispatching is realized is called a "dispatcher" (or task dispatcher).

Scheduling, scheduler
The processing to determine which task to execute next is called "scheduling" (or task scheduling). The kernel mechanism by which scheduling is realized is called a "scheduler" (or task scheduler). Generally a scheduler is implemented inside system call processing or in the dispatcher.

Context
The environment in which a program runs is generally called "context.” For a context to be called identical, at the very least the processor operation mode must be the same and the stack space must be the same (part of the same contiguous area). Note that context is a conceptual entity from the standpoint of applications; even when processing must be executed in independent contexts, in actual implementation both contexts may sometimes use the same processor operation mode and the same stack space.

Precedence
The execution order of tasks, i.e., the order relation, is called precedence. This refers to the order of tasks when an execution right is given to a task among a group of tasks in the executable state to be in the execution state. If task Y has a higher precedence than task X, task Y will be executed first. If task Y, which has higher precedence than task X, becomes ready for execution while task X is executed, the execution right will be transferred to task Y, and task Y will be in execution state, i.e., RUNNING state. In this case, task X will be in the executable state, i.e., READY state, instead of execution state.

Additional Notes
Precedence has a similar meaning to "priority", and they both affect the execution order of tasks. However, "priority" is an attribute of tasks specified by API parameter, etc., explicitly from applications, whereas "precedence" is a concept that is employed to define the execution order among a group of tasks. The precedence among a group of tasks is determined based on the priority of the tasks. A task with higher priority has higher precedence. On the other hand, tasks with the same priority do not have the same precedence. Among tasks having the same priority, the one that entered an executable state (i.e., RUNNING state or READY state) first has the highest precedence. It is possible, however, to use an API such as tk_rot_rdq to change the precedence among tasks having the same priority.

API and system call
The standard interfaces for calling functions provided by μT-Kernel from applications or middleware are collectively called API (Application Programming Interface). In addition to system calls that directly call kernel functions, APIs include functions implemented as extended SVCs, macros, and libraries.
Extended SVC

System calls that are added at the time of the initial startup of OS or added later are called extended SVC. μT-Kernel 3.0 specification stipulates _Subsystem Management Functions_ can be used to define/implement extended SVC. Implementation of μT-Kernel/SM API can use extended SVC(s).

A program that executes the function of an extended SVC is extended SVC handler.

Kernel

Kernel refers to the portion of μT-Kernel that is not implemented by extended SVCs, compile-time macros, or library functions. μT-Kernel/SM API can be implemented using extended SVC, compile-time macros, and/or library functions. Such APIs are part of μT-Kernel specification. However, it is not deemed to be part of the kernel. On the other hand, all the functions of μT-Kernel/OS and μT-Kernel/DS are included in the kernel.

When we refer to system state while a non-task portion is executing, we need to be aware whether the execution is within the kernel or not.

Implementation-defined

That something is implementation-defined means that something is not standardized in the T-Kernel specification and should be defined for each implementation. The specifics of the implementation should be described clearly in the implementation specifications. In application programs, the portability for the portion dependent on implementation-defined items is not assured.

Implementation-dependent

That something is implementation-dependent means that in the T-Kernel specification, the behavior of something varies according to the target systems or system operating conditions. The behavior should be defined for each implementation. The specifics of the implementation should be described clearly in the implementation specifications. In application programs, the portion dependent on implementation-dependent items needs to be modified when porting in principle.
2.2 Task States and Scheduling Rules

2.2.1 Task States

Task states are classified primarily into the five below. Of these, Waiting state in the broad sense is further classified into three states. Saying that a task is in a RUN state means it is in either RUNNING state or READY state.

RUNNING state
The task is currently being executed. When a task-independent portion is executing, except when otherwise specified, the task that was executing prior to the start of task-independent portion execution is said to be in RUNNING state.

READY state
The task has completed preparations for running, but cannot run because a task with higher precedence is running. In this state, the task is able to run whenever it becomes the task with the highest precedence among the tasks in READY state.

Waiting states
The task cannot run because the conditions for running are not in place. In other words, the task is waiting for the conditions for its execution to be met. While a task is in one of the Waiting states, the program counter and register values, and the other information representing the program execution state, are saved. When the task resumes running from this state, the program counter, registers and other values revert to their values immediately prior to going to the Waiting state. This state is subdivided into the following three states.

WAITING state
Execution is stopped because a system call was invoked that interrupts execution of the invoking task until some condition is met.

SUSPENDED state
Execution was forcibly interrupted by another task.

WAITING-SUSPENDED state
The task is in both WAITING state and SUSPENDED state at the same time. WAITING-SUSPENDED state results when another task requests suspension of a task already in WAITING state.

μ T-Kernel makes a clear distinction between WAITING state and SUSPENDED state. A task cannot go to SUSPENDED state on its own.

DORMANT state
The task has not yet been started or has completed execution. While a task is in DORMANT state, information presenting its execution state is not saved. When a task is started from DORMANT state, execution starts from the task start address. Except when otherwise specified, the register values are not saved.

NON-EXISTENT state
A virtual state before a task is created, or after it is deleted, and is not registered in the system.

Depending on the implementation, there may also be transient states that do not fall into any of the above categories (see Section 2.5, “System States”).

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 RUNNING state. In such a case the task that was in RUNNING state up to that time is said to have been preempted by the task that goes to RUNNING state anew. Note also that in explanations of system call functions, even when a task is said to go to READY state, depending on the task precedence it may go immediately to RUNNING state.

Task starting means transferring a state from DORMANT state to READY state. A task is therefore said to be in "started" state if it is in any state other than DORMANT or NON-EXISTENT. Task exit means that a task in started state goes to DORMANT state.
Task wait release means that a task in WAITING state goes to READY state, or a task in WAITING-SUSPENDED state goes to SUSPENDED state. The resumption of a suspended task means that a task in SUSPENDED state goes to READY state, or a task in WAITING-SUSPENDED state goes to WAITING state.

Task state transitions in a typical implementation are shown in Figure 2.1, “Task State Transition Diagram”. Depending on the implementation, there may be other states besides those shown here.
Figure 2.1: Task State Transition Diagram
A feature of μT-Kernel is the clear distinction made between system calls that perform operations affecting the invoking task and those whose operations affect other tasks (see Table 2.1, “State Transitions Distinguishing Invoking Task and Other Tasks”). The reason for this is to clarify task state transitions and facilitate understanding of system calls. This distinction between system call operations in the invoking task and operations affecting other tasks can also be seen as a distinction between state transitions from RUNNING state and those from other states.

<table>
<thead>
<tr>
<th>Task transition to a waiting state (including SUSPENDED)</th>
<th>Operations in invoking tasks (Transition from RUNNING state)</th>
<th>Operations on other tasks (Transitions from other states)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task exit</td>
<td>tk_slp_tsk&lt;br&gt;READY state → WAITING state</td>
<td>tk_sus_tsk&lt;br&gt;READY state, WAITING state → SUSPENDED state, WAITING-SUSPENDED state</td>
</tr>
<tr>
<td>Task deletion</td>
<td>tk_ext_tsk&lt;br&gt;READY state → DORMANT state</td>
<td>tk_ter_tsk&lt;br&gt;READY state, WAITING state → DORMANT state</td>
</tr>
<tr>
<td></td>
<td>tk_exd_tsk&lt;br&gt;READY state → NON-EXISTENT state</td>
<td>tk_del_tsk&lt;br&gt;DORMANT state → NON-EXISTENT state</td>
</tr>
</tbody>
</table>

Table 2.1: State Transitions Distinguishing Invoking Task and Other Tasks

Additional Notes
WAITING state and SUSPENDED state are orthogonally related, in that a request for transition to SUSPENDED state cannot have any effect on the conditions for task wait release. That is, the task wait release conditions are the same whether the task is in WAITING state or WAITING-SUSPENDED state. Thus even if transition to SUSPENDED state is requested for a task that is in a state of waiting to acquire some resource (semaphore resource, memory block, etc.), and the task goes to WAITING-SUSPENDED state, the conditions for allocation of the resource do not change but remain the same as before the request to go to SUSPENDED state.

Rationale for the Specification
The reason the μT-Kernel makes a distinction between WAITING state (wait caused by the invoking task) and SUSPENDED state (wait caused by another task) is that these states sometimes overlap. By recognising these overlapped states as WAITING-SUSPENDED states, the task state transitions become clearer and system calls are easier to understand. On the other hand, since a task in WAITING state cannot invoke a system call, different types of WAITING state (e.g., waiting for wakeup, or waiting to acquire a semaphore resource) will never overlap. Since there is only one kind of waiting state caused by another task (SUSPENDED state), the μT-Kernel treats repeated entries to SUSPENDED state as nesting, thereby achieving clarity of task state transitions.

2.2.2 Task Scheduling Rules

The μT-Kernel adopts a preemptive priority-based scheduling method based on priority levels assigned to each task. Tasks having the same priority are scheduled on a FCFS (First Come First Served) basis. Specifically, task precedence is used as the task scheduling rule, and precedence among tasks is determined as follows based on the priority of each task. If there are multiple tasks that can be run, the one with the highest precedence goes to RUNNING state and the others go to READY state. In determining precedence among tasks, of those tasks having different priority levels, that with the highest priority has the highest precedence. Among tasks having the same priority, the one that entered a run state (RUNNING state or READY state) first
has the highest precedence. It is possible, however, to use a system call to change the precedence among tasks having the same priority.

When the task with the highest precedence changes from one task to another, a dispatch occurs immediately and the task in RUNNING state is switched. If no dispatch occurs (during execution of a handler, during dispatch disabled state, etc.), however, the switching of the task in RUNNING state is held off until the next dispatch occurs.

Additional Notes
According to the scheduling rules adopted in the μT-Kernel, so long as there is a higher precedence task in a run state, a task with lower precedence will simply not run. That is, unless the highest-precedence task goes to WAITING state or for other reason cannot run, other tasks are not run. This is a fundamental difference from TSS (Time Sharing System) scheduling in which multiple tasks are treated equally. It is possible, however, to issue a system call changing the precedence among tasks having the same priority. An application can use such a system call to realize round-robin scheduling, which is a typical kind of TSS scheduling.

Examples in figures below illustrate how the task that first goes to a run state (RUNNING state or READY state) gains precedence among tasks having the same priority. Figure 2.2, “Precedence in Initial State” shows the precedence among tasks after Task A of priority 1, Task E of priority 3, and Tasks B, C and D of priority 2 are started in that order. The task with the highest precedence, Task A, goes to RUNNING state. When Task A exits, Task B with the next-highest precedence goes to RUNNING state (Figure 2.3, “Precedence After Task B Goes To RUNNING State”). When Task A is again started, Task B is preempted and reverts to READY state; but since Task B went to a run state earlier than Task C and Task D, it still has the highest precedence among tasks with the same priority. In other words, the task precedence reverts to that in Figure 2.2, “Precedence in Initial State”.

Next, consider what happens when Task B goes to WAITING state in the conditions in Figure 2.3, “Precedence After Task B Goes To RUNNING State”. Since task precedence is defined among tasks that can be run, the precedence among tasks becomes as shown in Figure 2.4, “Precedence After Task B Goes To WAITING State”. Thereafter when the Task B waiting state is released, Task B goes to run state after Task C and Task D, and thus assumes the lowest precedence among tasks of the same priority (Figure 2.5, “Precedence After Task B WAITING State Is Released”).

Summarizing the above, immediately after a task that goes from READY state to RUNNING state reverts to READY state, it has the highest precedence among tasks of the same priority; but after a task goes from RUNNING state to WAITING state and then the wait is released, its precedence is the lowest among tasks of the same priority.

Note that after a task goes from SUSPENDED state to a run state, it has the lowest precedence among tasks of the same priority.

![Figure 2.2: Precedence in Initial State](image-url)
Figure 2.3: Precedence After Task B Goes To RUNNING State

Figure 2.4: Precedence After Task B Goes To WAITING State

Figure 2.5: Precedence After Task B WAITING State Is Released
2.3 Interrupt Handling

Interrupts in the μT-Kernel include both external interrupts from devices and interrupts due to CPU exceptions. One interrupt handler may be defined for each interrupt handler number. Interrupt handlers can be started in two ways: one is to start it without the kernel intervention, the other is to start it via a high-level language support routine.

For more details, see Section 4.8, “Interrupt Management Functions”.
2.4 Task Exception Handling

The μT-Kernel defines task exception handling functions for dealing with exceptions. Note that CPU exceptions are treated as interrupts.

A task exception handling function invokes a system call requesting task exception handling by a designated task, interrupts execution by the specified task, and runs a task exception handler. Execution of the task exception handler takes place in the same context as the interrupted task. Upon return from the task exception handler, the interrupted processing continues.

One task exception handler per task can be registered from an application.

For more details, see Section 4.3, “Task Exception Handling Functions”.
2.5 System States

2.5.1 System States While Non-task Portion Is Executing

When programming tasks to run on μT-Kernel, one can keep track of the changes in task states by using a task state transition diagram. In the case of routines such as interrupt handlers or extended SVC handlers, however, the user must perform programming at a level closer to the kernel than tasks. In this case consideration must be made also of system states while a non-task portion is being executed, for application programs to work properly. An explanation of μT-Kernel system states is therefore given here.

System states are classified as in Figure 2.6, “Classification of System States”.

Of these shown in Figure 2.6, “Classification of System States”, a “transient state” is equivalent to the kernel running state (system call execution). From the standpoint of the user, it is important that each of the system calls issued by the user application program be executed indivisibly, and that the internal states while a system call is executing cannot be seen by the user. For this reason the state while the kernel running is considered a “transient state” and internally it is treated as a black box.

However, in the following case, for instance, a transient state may become visible to users.

- When memory is being allocated or freed in the case of a system call that gets or releases memory (while a μT-Kernel/SM system memory management function is called).

When a task is in a transient state such as these, the behavior of a task termination (tk_ter_tsk) system call is not guaranteed. Moreover, task suspension (tk_sus_tsk) may cause a deadlock or other problem by stopping without clearing the transient state.

Accordingly, as a rule tk_ter_tsk and tk_sus_tsk cannot be used in programs. These system calls should be used only in specific middleware or debugger, which is closely related to OS itself.

While being a “non-task portion,” the portion that is considered to be running a processing requested from a specific task (called a “requesting task”) is called “quasi-task portion.” For example, an extended SVC handler is executed as a “quasi-task portion.” The invoking task can be identified in a “quasi-task portion” and the requesting task becomes the invoking task. Similar to the task portion, in the quasi-task portion, the task state transitions can be defined and system calls can be issued to enter into WAITING state from the quasi-task portion. In this way, the quasi-task portion behaves similarly to a subroutine called from a requesting task. “Quasi-task portion” is, however, positioned as an extended part of OS and its processor operation mode and stack space are different from those of the task portion. It means that when a state enters into a quasi-task portion from a task portion, its processor operation mode and stack space are switched. This behavior is different from when a function or subroutine is called in a task portion.

Among the “non-task portion,” a “task-independent portion” is activated due to a factor that completely ignore the progress of the task portion or quasi-task portion processing. Specifically, an interrupt handler that is triggered by an external interrupt or a time event handler (cyclic handler and alarm handler) that is triggered due to the specified elapsed time is executed as a “task-independent portion.” Note that both the external interrupt and the specified elapsed time are the factors that is independent from a task that is incidentally running at that moment.

Finally, ”non-task portion” is separated into three classes: ”transient state,” ”quasi-task portion,” and ”task-independent portion.” The states other than these represent a state where a program for the task is running, this is, the state where ”task portion is running.”
2.5.2 Task-Independent Portion and Quasi-Task Portion

A feature of a task-independent portion (interrupt handlers, time event handlers, etc.) is that it is meaningless to identify the task that was running immediately prior to entering a task-independent portion, and the concept of "invoking task" does not exist. Accordingly, a system call that enters WAITING state, or one that is issued implicitly specifying the invoking task, cannot be called from a task-independent portion. Moreover, since the currently running task cannot be identified in a task-independent portion, there is no task switching (dispatching). If dispatching is necessary, it is delayed until processing leaves the task-independent portion. This is called delayed dispatching.

If dispatching were to take place in the interrupt handler, which is a task-independent portion, the rest of the interrupt handler routine would be delayed for execution after the task started by the dispatching, causing problems in case of interrupt nesting. This is illustrated in Figure 2.7, “Interrupt Nesting and Delayed Dispatching”.

In Figure 2.7, “Interrupt Nesting and Delayed Dispatching”, Interrupt X is raised during Task A execution, and while its interrupt handler is running, a higher-priority interrupt Y is raised. In this case, if dispatching were to occur immediately on return from interrupt Y at (1),\(^1\) starting Task B, the processing of parts (2) to (3) of Interrupt X would be put off until after Task B relinquishes CPU, with parts (2) to (3) executed only after Task A goes to RUNNING state. The danger is that the low-priority Interrupt X handler would be preempted not only by a higher-priority interrupt but even by Task B started by that interrupt. There would no longer be any guarantee of the interrupt handler execution maintaining priority over task execution, making it impossible to write an interrupt handler. This is the reason for introducing the principle of delayed dispatching.

A feature of a quasi-task portion, on the other hand, is that the task executing prior to entering the quasi-task portion (the requesting task) can be identified, making it possible to define task states just as in the task portion; moreover, it is possible to enter WAITING state while in a quasi-task portion. Accordingly, dispatching occurs in a quasi-task portion in the same way as in ordinary task execution. As a result, even though the OS extended part and other quasi-task portion is a non-task portion, its execution does not necessarily have priority at all times over the task portion. This is in contrast to interrupt handlers, which must always be given execution precedence over tasks.

The following two examples illustrate the difference between a task-independent portion and quasi-task portion.

---
\(^1\) If dispatching takes place at (1), the remainder of the handler routine for Interrupt X ((2) to (3)) ends up being put off until later.
• An interrupt is raised while Task A (priority 8 = low) is running, and in its interrupt handler (task-independent portion) \texttt{tk\_wup\_tsk} is issued for Task B (priority 2 = high). In accordance with the principle of delayed dispatching, however, dispatching does not yet occur at this point. Instead, after \texttt{tk\_wup\_tsk} execution, first the remaining part of the interrupt handler are executed. Only when \texttt{tk\_ret\_int} is executed at the end of the interrupt handler does dispatching occur, causing Task B to run.

• An extended SVC is executed in Task A (priority 8 = low), and in its extended SVC handler (quasi-task portion), \texttt{tk\_wup\_tsk} is issued for Task B (priority 2 = high). In this case the principle of delayed dispatching is not applied, so dispatching occurs in \texttt{tk\_wup\_tsk} processing. Task A goes to READY state in a quasi-task portion, and Task B goes to RUNNING state. Task B is therefore executed before the rest of the extended SVC handler is completed. The rest of the extended SVC handler is executed after dispatching occurs again and Task A goes to RUNNING state.

![Figure 2.7: Interrupt Nesting and Delayed Dispatching](image-url)
"Object" is the general term for resources handled by μT-Kernel. Besides tasks, objects include memory pools, semaphores, event flags, mailboxes and other synchronization and communication mechanisms, as well as time event handlers (cyclic handlers and alarm handlers).

Attributes can generally be specified when an object is created. Attributes determine detailed differences in object behavior or the object initial state. When TA_XXXXX is specified for an object, that object is called a "TA_XXXXX attribute object." If there is no particular attribute to be defined, TA_NULL (= 0) is specified. Generally there is no interface provided for reading attributes after an object is registered.

In an object attribute value, the lower bits indicate system attributes and the upper bits indicate implementation-dependent attributes. This specification does not define the bit position at which the upper and lower distinction is to be made. Basically, bits that are not defined in the standard specification can be used as implementation-dependent attributes. In principle, however, the system attribute portion is assigned from the least significant bit (LSB) toward the most significant bit (MSB), and implementation-dependent attributes from the MSB toward the LSB. Bits not defining any attribute must be cleared to 0.

In some cases an object may contain extended information. Extended information is specified when the object is registered. Information passed in parameters when an object starts execution has no effect on μT-Kernel behavior. Extended information can be read by calling an object status reference system call.

An object is identified by an ID number. In μT-Kernel, an ID number is automatically assigned when an object is created. Users cannot specify ID numbers. This makes identifying an object during debugging difficult. We can specify an object name for debugging upon creating each object. This name is used temporarily for debugging and can be referred to only from μT-Kernel/DS functions. No check is performed on the naming by μT-Kernel.
2.7 Protection Levels

In μT-Kernel, four levels from 0 to 3 (meaning privileged mode, user mode, etc.) are defined as the protection level at runtime, and also four levels from 0 to 3 are defined as the protection level of memory to be accessed. The currently running execution task can access only to the memory with the same or lower protection level. This function is useful for protecting a system such as the OS from being illegally accessed by programs.

The uses of each protection level are as follows.

<table>
<thead>
<tr>
<th>Protection Levels</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Kernel, subsystems, device drivers, etc.</td>
</tr>
<tr>
<td>1</td>
<td>System application tasks</td>
</tr>
<tr>
<td>2</td>
<td>(reserved)</td>
</tr>
<tr>
<td>3</td>
<td>User application tasks</td>
</tr>
</tbody>
</table>

Some CPUs support only two protection levels privileged (supervisor mode) and user levels. In such a case protection level 0 is assigned to the privileged level and protection level 3 to the user level. In such a case if protection levels from 0 to 2 are specified in an API the behavior of the system is the same as in the case of privileged level 0 being specified. For example if TA_RNG2 is specified in tskatr when tk_cre_tsk is invoked it is assumed that TA_RNG0 has been specified and the task executes at the privileged level (protection level 0). Another example is specifying TA_RNG2 in mplatr when tk_cre_mpl is invoked. This is assumed to specify TA_RNG0 and the access protection level of the created memory pool is 0. In this case the service profile defines the following macros to be 0: TK_MEM_RNG0, TK_MEM_RNG1, TK_MEM_RNG2.

In the case of CPUs without any distinction for privileged and user modes only protection level 0 is used. In such a case if protection levels 1 to 3 are specified in an API the behavior of the system is the same as in the case of privileged level 0 being specified. In this case the service profile defines the following macros to be 0: TK_MEM_RNG0, TK_MEM_RNG1, TK_MEM_RNG2, TK_MEM_RNG3.

When a protection privilege level of the currently running context is lower than that of the memory being accessed the violation of memory access privilege shall be detected and a CPU exception shall be generated.

Changing from one protection level to another is accomplished by invoking a system call or extended SVC or by interrupt or CPU exception.

A non-task portion (task-independent portion, quasi-task portion, etc.) runs at protection level 0. Only a task portion can run at protection levels 1 to 3. A task portion can also run at protection level 0.
2.8 Service Profile

μT-Kernel 3.0 is an OS specification for small-scale embedded computer systems, and it allows many implementations, and permits customization and optimization suitable for each target platform. For features that have strong dependency on hardware such as floating-point unit (FPU), and features that have potential implications for run-time efficiency such as hooks for debug support, the specification allows subsetting as exceptional case, and this allows efficient implementation of μT-Kernel 3.0 specification OS on target hardware. To accommodate the subsetting in this manner, and the desire to keep the distribution and portability of middleware and application high, μT-Kernel 3.0 has introduced a mechanism to let each implementation of μT-Kernel 3.0 describe the differences in the implementation from other implementation of μT-Kernel 3.0. This description as a whole is called service profile.

Service profile in μT-Kernel 3.0 is realized by enumerating the information about a particular implementation of μT-Kernel 3.0 as a list of C language macros that have constant value. For example, an implementation that allows the specification of TA_USERBUF, the corresponding service profile items must be defined as below to announce the support of TA_USERBUF.

```
#define TK_SUPPORT_USERBUF TRUE /* Support of user-specified buffer (TA_USERBUF) */
```

Applications and middleware can use the service profile information and write code according to the existence of the support of TA_USERBUF. For example, the following is a typical use case.

```
T_CTSK ctsk = {
    .exinf = NULL,
#if TK_SUPPORT_USERBUF
    .tskatr = TA_HLNG|TA_RNG0|TA_USERBUF,
    .bufptr = taskA_stack,
#else
    .tskatr = TA_HLNG|TA_RNG0,
#endif
    .task  = task,
    .itskpri = 10,
    .stksz = 2048
};

tskid = tk_cre_tsk(&ctsk);
```

The code sample above changes, depending on the availability of TA_USERBUF, changes the content of parameter packet ctsk which is passed to tk_cre_tsk. In this manner, it is possible to develop applications and middleware that can be used on both the implementations, those that support TA_USERBUF and those that do not. Middleware developers are requested to improve the portability and thus facilitate distribution of middleware software packages by using the service profile mechanism appropriately.

For the details of service profile items defined in μT-Kernel 3.0, see Section 3.4, “Service Profile”. 

Chapter 3

Common Rules of $\mu$T-Kernel
## 3.1 Data Types

### 3.1.1 General Data Types

```c
typedef signed char B; /* signed 8-bit integer */
typedef signed short H; /* signed 16-bit integer */
typedef signed long W; /* signed 32-bit integer */
typedef signed long long D; /* signed 64-bit integer */
typedef unsigned char UB; /* unsigned 8-bit integer */
typedef unsigned short UH; /* unsigned 16-bit integer */
typedef unsigned long UW; /* unsigned 32-bit integer */
typedef unsigned long long UD; /* unsigned 64-bit integer */

typedef char VB; /* 8-bit data without an intended type */
typedef short VH; /* 16-bit data without an intended type */
typedef long VW; /* 32-bit data without an intended type */
typedef long long VD; /* 64-bit data without an intended type */

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

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

typedef INT SZ; /* Generic Size */
typedef INT ID; /* general ID */
typedef W MSEC; /* general time (in milliseconds) */

typedef void (*FP)(); /* general function address */
typedef INT (*FUNCP)(); /* general function address */

#define LOCAL static /* local symbol definition */
#define EXPORT /* global symbol definition */
#define IMPORT extern /* global symbol reference */
```

/*
 * Boolean values
 * TRUE = 1 is defined, but any value other than 0 is logically TRUE.
 * Do NOT use as in if ( bool == TRUE )
 * use as in if ( bool )
 */
typedef UINT BOOL;
#define TRUE 1 /* true */
#define FALSE 0 /* false */
Note

- VB, VH, VW, and VD differ from B, H, W, and D in that the former mean only the bit width is known, not the contents of the data type, whereas the latter clearly indicate integer type.

- SZ type is an integer data type with implementation-defined bit width, and it shall be properly defined based on the CPU bit width and memory space size for each implementation.

- BOOL defines TRUE as 1, but any value other than 0 is also true. For this reason, TRUE must not be used as left-hand or right-hand value of comparison operators (== and !=) for deciding whether the value is true or false. That is, conditional operations like "if (boolean value == TRUE)" should be avoided, and instead use boolean value directly as condition, like "if (boolean value)".

Related Service Profile Items

The 64-bit data types, D, UD, and VD, are guaranteed to be usable when the following service profile item is set to be effective.

TK_HAS_DOUBLEWORD Support of 64-bit data types (D, UD, VD)

Additional Notes

Parameters such as stkza, wpcnt, and message size that clearly do not take negative values are also in principle signed integer (INT or W) data type. This is in keeping with the overall TRON project rule that integers should be treated as signed numbers as much as possible. As for the timeout (TMO tmout) parameter, its being a signed integer enables 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.

```c
typedef INT FN; /* Function Codes */
typedef UW ATR; /* Object/handler attributes */
typedef INT ER; /* Error Code */
typedef INT PRI; /* Priority */
typedef W TMO; /* Timeout specification in milliseconds */
typedef D TMO_U; /* Timeout specification in microseconds with 64-bit */
    bit integer */
typedef UW RELTIM; /* Relative time in milliseconds */
typedef UD RELTIM_U; /* Relative time in microseconds with 64-bit */
typedef struct systim {
    W hi; /* High 32 bits */
    UW lo; /* Low 32 bits */
} SYSTIM;
typedef D SYSTIM_U; /* System time in microseconds with 64-bit integer */

/*
 * Common constants
*/
```
Note

- A data type that combines two or more data types is represented by its main data type. For example, the value returned by `tk_cre_tsk` can be a task ID or error code, but since it is mainly a task ID, the data type is ID.

---

Related Service Profile Items

TMO_U, RELTIM_U, and SYSTEM_U dealing with date and relative time in microsecond resolution are guaranteed to be usable only when the following service profile items are set to be effective.

- **TK_SUPPORT_USEC**: Support of microsecond

---

Additional Notes

The policy is to append "._u" (u means μ) or "._U" at the end for parameters and data types representing microsecond (μsec), or append "._d" (d means double integer) or "._D" at the end for other parameters and data types representing 64-bit integer. TMO_U, RELTIM_U, and SYSTIM_U are data type names complying to this policy.
3.2 System Calls

3.2.1 System Call Format

μT-Kernel adopts C as the standard high-level language, and standardizes interfaces for system call execution from C language routines.

The method for interfacing with the assembly language shall be implementation-dependent. Calling by means of a C language interface is recommended even when an assembly language is used. In this way, portability is assured for programs written in assembly language even if the OS changes, so long as the CPU is the same.

The following common rules are established for system call interfaces.

- All system calls are defined as C language functions.
- A function return code of 0 or a positive value indicates normal completion, while negative values are used for error codes.

The implementation of the system call interface is not standardized, and is implementation-dependent. For example, we can use C language macros, inline functions, inline assembly language description, etc.

Among C language interfaces for system calls, those which pass parameters using a packet or pointer have CONST modifier attached to explicitly indicate that μT-Kernel does not overwrite a parameter referred to by the pointer.

CONST is intended to be the C language const modifier equivalent. This alias for const is used so that the compiler check can be disabled by using #define macro function when any program that does not support const modifier mixes in.

Specific usage of CONST is as follows: Details, however, depend on the development environment.

1. Include the following descriptions in the common include file:

```c
#ifndef TKERNEL_CHECK_CONST
#define CONST const
#else
#define CONST
#endif
```

2. Describe a function definition or system call definition in the program by using CONST.

<table>
<thead>
<tr>
<th>Description Example of CONST</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_cre_tsk( CONST T_CTSK *pk_ctsk );</td>
</tr>
<tr>
<td>foo_bar( CONST void *buf );</td>
</tr>
</tbody>
</table>

In μT-Kernel 3.0 or later, it is strongly recommended that CONST is used explicitly in a program and the check for const is enabled in the configuration.

3.2.2 APIs Possible from Task-Independent Portion

The following system calls of μT-Kernel/OS can be issued from a task-independent portion and in dispatch disabled state:

<table>
<thead>
<tr>
<th>System call name</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_sta_tsk</td>
<td>Start Task</td>
</tr>
<tr>
<td>tk_ref_tsk</td>
<td>Reference Task Status</td>
</tr>
<tr>
<td>System call name</td>
<td>Summary description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>tk_wup_tsk</td>
<td>Wakeup Task</td>
</tr>
<tr>
<td>tk_rel_wai</td>
<td>Release Wait</td>
</tr>
<tr>
<td>tk_sus_tsk</td>
<td>Suspend Task</td>
</tr>
<tr>
<td>tk_sig_tev</td>
<td>Signal Task Event</td>
</tr>
<tr>
<td>tk_sig_sem</td>
<td>Signal Semaphore</td>
</tr>
<tr>
<td>tk_set_flg</td>
<td>Set Event Flag</td>
</tr>
<tr>
<td>tk_sta_cyc</td>
<td>Start Cyclic Handler</td>
</tr>
<tr>
<td>tk_stp_cyc</td>
<td>Stop Cyclic Handler</td>
</tr>
<tr>
<td>tk_ref_cyc</td>
<td>Reference Cyclic Handler Status</td>
</tr>
<tr>
<td>tk_ref_cyc_u</td>
<td>Reference Cyclic Handler Status (Microseconds)</td>
</tr>
<tr>
<td>tk_sta_alm</td>
<td>Start Alarm Handler</td>
</tr>
<tr>
<td>tk_sta_alm_u</td>
<td>Start Alarm Handler (Microseconds)</td>
</tr>
<tr>
<td>tk_stp_alm</td>
<td>Stop Alarm Handler</td>
</tr>
<tr>
<td>tk_ref_alm</td>
<td>Reference Alarm Handler Status</td>
</tr>
<tr>
<td>tk_ref_alm_u</td>
<td>Reference Alarm Handler Status (Microseconds)</td>
</tr>
<tr>
<td>tk_ret_int</td>
<td>Return from Interrupt Handler (can be issued only from an interrupt handler written in an assembly language)</td>
</tr>
<tr>
<td>tk_rot_rdq</td>
<td>Rotate Ready Queue</td>
</tr>
<tr>
<td>tk_get_tid</td>
<td>Get Task Identifier</td>
</tr>
<tr>
<td>tk_ref_sys</td>
<td>Reference System Status</td>
</tr>
</tbody>
</table>

The following APIs of μT-Kernel/SM can be issued from a task-independent portion and in dispatch disabled state:

<table>
<thead>
<tr>
<th>API name</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>Disable External Interrupts</td>
</tr>
<tr>
<td>EI</td>
<td>Enable External Interrupts</td>
</tr>
<tr>
<td>isDI</td>
<td>Get Interrupt Disable Status</td>
</tr>
<tr>
<td>SetCpuIntLevel</td>
<td>Set CPU Interrupt Mask Level</td>
</tr>
<tr>
<td>GetCpuIntLevel</td>
<td>Get CPU Interrupt Mask Level</td>
</tr>
<tr>
<td>EnableInt</td>
<td>Enable Interrupts</td>
</tr>
<tr>
<td>DisableInt</td>
<td>Disable Interrupts</td>
</tr>
<tr>
<td>ClearInt</td>
<td>Clear Interrupt</td>
</tr>
<tr>
<td>EndOfInt</td>
<td>Issue EOI to Interrupt Controller</td>
</tr>
<tr>
<td>CheckInt</td>
<td>Check Interrupt</td>
</tr>
<tr>
<td>SetIntMode</td>
<td>Set Interrupt Mode</td>
</tr>
<tr>
<td>SetCtrlIntLevel</td>
<td>Set Interrupt Controller Interrupt Mask Level</td>
</tr>
<tr>
<td>GetCtrlIntLevel</td>
<td>Get Interrupt Controller Interrupt Mask Level</td>
</tr>
<tr>
<td>out_b</td>
<td>Write to I/O Port (in Bytes)</td>
</tr>
<tr>
<td>out_h</td>
<td>Write to I/O Port (in Half-words)</td>
</tr>
<tr>
<td>out_w</td>
<td>Write to I/O Port (in Words)</td>
</tr>
<tr>
<td>out_d</td>
<td>Write to I/O Port (in Double-words)</td>
</tr>
<tr>
<td>in_b</td>
<td>Read from I/O Port (in Bytes)</td>
</tr>
<tr>
<td>in_h</td>
<td>Read from I/O Port (in Double-words)</td>
</tr>
<tr>
<td>in_w</td>
<td>Read from I/O Port (in Words)</td>
</tr>
<tr>
<td>in_d</td>
<td>Read from I/O Port (in Double-words)</td>
</tr>
<tr>
<td>WaitUsec</td>
<td>Micro Wait (Microseconds)</td>
</tr>
<tr>
<td>WaitNsec</td>
<td>Micro Wait (Nanoseconds)</td>
</tr>
<tr>
<td>SetOBJNAME</td>
<td>Set Object Name</td>
</tr>
</tbody>
</table>

All system calls of μT-Kernel/DS can be issued from a task-independent portion and in dispatch disabled state.

Whether system calls or APIs other than those above can be issued from a task-independent portion or in dispatch disabled state is implementation-dependent.
3.2.3 Restricting System Call Invocation

The protection levels at which a system call is invokable can be restricted. In this case, if a system call is issued from a task (task portion) running at lower privilege than the specified protection level, the error code E_OACV is returned.

Extended SVC calling cannot be restricted.

If, for example, issuing a system call from a level with lower privilege than level 1 is prohibited, system calls cannot be made from tasks running at protection levels 2 and 3. Tasks running at those levels will only be able to make extended SVC calls, and are programmed using subsystem functions only.

This kind of restriction is used when μT-Kernel is combined with middleware that offers process management function and other functions, to prevent tasks (as part of user process, etc.) that use the functions of such middleware (process management, etc.) from directly accessing μT-Kernel functions. It allows μT-Kernel to be used as a micro-kernel. The idea is that the user process cannot control the micro-kernel directly via available process API, and only the middleware can control the micro-kernel directly.

The protection level restriction on system call invocation is set using the system configuration information management functions. (see Section 5.6, “System Configuration Information Management Functions”).

3.2.4 Modifying a Parameter Packet Format

Some parameters passed to system calls use packet format. The packet format parameters are of two kinds, either input parameters passing information to a system call (e.g., T_CTSK) or output parameters returning information from a system call (e.g., T_RTSK).

Additional information that is implementation-dependent can be added to a parameter packet. When implementation-dependent information is added, it must be positioned after the standard defined information. It is permitted to delete only parameters that are declared ineffective by the service profile, and other parameters shall not be deleted. It is not allowed, however, to change the data types and order of information defined in the standard specification.

When implementation-dependent information is added to a packet of input information passed to a system call (T_CTSK, etc.), if the system call is invoked while this additional information is not yet initialized (memory content is indeterminate), the system call must still function normally.

Ordinarily a flag indicating that valid values are set in the additional information is defined in the implementation-dependent area of attribute flag included in the standard specification. When that flag is set (1), the additional information is to be used; and when the flag is not set (0), the additional information is not initialized (memory content is indeterminate) and the default values are to be used instead.

The reason for this specification is to make sure we can run the same application program merely by recompiling, irrespective of whether implementation dependent function extension is added to an implementation of the specification.
Porting Guideline
A care must be taken now for parameter packet initialization since the parameter may be deleted by declaring it to be ineffective by service profile. For example, it is not recommended to initialize T_CTSK structure in the following manner from the viewpoint of portability.

```c
T_CTSK ctsk = {
    NULL,
    TA_HLNG|TA_RNG0|TA_USERBUF,
    task,
    10,
    2048,
    "",
    buf
};
```

Instead, it is recommended to perform initialization using the syntax specified in ISO/IEC 9899:1999 as follows.

```c
T_CTSK ctsk = {
    .exinf = NULL,
    .tskatr = TA_HLNG|TA_RNG0|TA_USERBUF,
    .task = task,
    .itskpri = 10,
    .stksz = 2048,
    .bufptr = buf
};
```

### 3.2.5 Function Codes

Function codes are numbers assigned to each system call and used to identify the system call. The system call function codes are not specified here but are to be defined in implementation. See `tk_def_ssy` on extended SVC function codes.

### 3.2.6 Error Codes

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 returned values in the case of normal completion is specified individually for each system call. An exception to this principle is that there are some system calls that do not return when called. A system call that does not return is declared in the C language interface as having no return code (i.e., 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, the circumstances in which they occur and other factors.

```c
#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))
```

Note that, in an environment where ER is 16-bit data type, sub error code can be omitted and main error code can be returned as the error code. In this case, SERCD macro shall not be defined.

```c
#define MERCD(er) ((ER)(er))  /* main error code */
#define ERCD(mer, ser) ((ER)(mer))
```
Related Service Profile Items
Only when the service profile items below are set to be effective, the error code contains sub error code, and SERCD macro is supported.

TK_SUPPORT_SERCD Support of sub error code

3.2.7 Timeout

A system call that may enter WAITING state has a timeout function. If processing is not completed by the time the specified timeout interval has elapsed, the processing is canceled and the system call returns error code E_TMOUT.

In accordance with the principle that there should be no side-effects from calling a system call if that system call returns an error code, the calling of a system call that times out should in principle result in no change in system state. An exception to this is when the functioning of the system call is such that it cannot return to its original state if processing is canceled. This is indicated in the system call description.

If the timeout interval is set to 0, a system call does not enter even when a situation arises in which it would ordinarily go to WAITING state. In other words, a system call with timeout set to 0 when it is invoked has no possibility of entering WAITING state. Invoking a system call with timeout set to 0 is called polling; i.e., a system call that performs polling has no chance of entering WAITING state.

The descriptions of individual system calls as a rule describe the behavior when there is no timeout (in other words, when an eternal wait occurs). Even if the system call description states that the system call "enters WAITING state" or "is put in WAITING state," if a timeout is set and that time interval elapses before processing is completed, the WAITING state is released and the system call returns error code E_TMOUT. In the case of polling, the system call returns E_TMOUT without entering WAITING state.

Timeout (TMO and TMO_U types) is given as a positive integer, or as TMO_POL=0 for polling, or as TMO_FEVR (= -1) for eternal wait. If a timeout interval is set, the timeout processing must be guaranteed to take place after the specified interval from the system call issuing has elapsed.

Additional Notes
Since a system call that performs polling does not enter WAITING state, there is no change in the precedence of the task calling it.

In a general implementation, when the timeout is set to 1, timeout processing takes place on the second timer interrupt (sometimes called "time tick") after a system call is invoked. Since a timeout of 0 cannot be specified (0 being allocated to TMO_POL), in this kind of implementation timeout does not occur on the initial timer interrupt after the system call is invoked.

3.2.8 Relative Time and System Time

When the time of an event occurrence is specified relative to another time, such as the time when a system call was invoked, relative time (RELTIM or RELTIM_U type) is used. If relative time is used to specify event occurrence time, it is necessary to guarantee that the event processing will take place after the specified time has elapsed from the time base. Relative time (RELTIM or RELTIM_U type) is also used for e.g. event occurrence. In such cases the method of interpreting the specified relative time is determined for each case.

When time is specified as an absolute value, system time (SYSTIM or SYSTIM_U type) is used. The μT-Kernel provides a function for setting system time, but even if the system time is changed using this function, there is no change in the real world time (actual time) at which an event occurs that was specified using relative time. What changes is the system time at which an event occurs that was specified as relative time.

SYSTIM: System time
Time base 1 millisecond, 64-bit signed integer
typedef struct systim {
    W hi; /* High 32 bits */
    UW lo; /* Low 32 bits */
} SYSTIM;

SYSTIM_U: System time
Time base 1 microsecond, 64-bit signed integer

typedef D SYSTIM_U; /* 64-bit */

RELTIM: Relative time
Time base 1 millisecond, 32-bit unsigned integer (UW)

typedef UW RELTIM;

RELTIM_U: Relative time
Time base 1 microsecond, 64-bit unsigned (UD) integer

typedef UD RELTIM_U; /* Relative time in microseconds with 64-bit integer */

TMO: Timeout time
Time base 1 millisecond, 32-bit signed integer (W)

typedef W TMO;

Eternal wait can be specified as TMO_FEVR (= -1).

TMO_U timeout period
Time base 1 microsecond, 64-bit signed (D) integer

typedef D TMO_U; /* Timeout in microseconds with 64-bit integer */

Eternal wait can be specified as TMO_FEVR (= -1).

Related Service Profile Items
TMO_U, RELTIM_U, and SYSTEM_U dealing with date and relative time in microsecond resolution are guaranteed to be usable only when the following service profile items are set to be effective.

TK_SUPPORT_USEC Support of microsecond

Additional Notes
Timeout or other such processing must be guaranteed to occur after the time specified as RELTIM, RELTIM_U, TMO, or TMO_U has elapsed. For example, if the timer interrupt interval is 1 ms and a timeout of 1 ms is specified, timeout occurs on the second timer interrupt after system call invocation. (The first timer interrupt does not exceed 1 ms.)
When a system time (SYSTIM_U) value that may overflow internally in kernel is specified as an argument, the system call behavior is undefined.
3.3 High-Level Language Support Routines

High-level language support routine capability is provided so that even if a task or handler is written in high-level language, the kernel-related processing can be kept separate from the language environment-related processing. Whether or not a high-level language support routine is used is specified in TA_HLNG, one of the object attributes and handler attributes.

When TA_HLNG is not specified, a task or handler is started directly from the start address passed in a parameter to tk_cre_tsk or tk_def_int, etc.; whereas when TA_HLNG is specified, first the high-level language startup processing routine (high-level language support routine) is started, then from this routine an indirect jump is made to the task start address or handler address passed in a parameter to tk_cre_tsk or tk_def_int. Viewed from the kernel, the task start address or handler address is a parameter given to the high-level language support routine. Separating the kernel processing from the language environment processing in this way facilitates support for different language environments.

Use of high-level language support routines has the further advantage that when a handler is written as a C language function, a system call for return from a handler can be executed automatically, simply by performing a function return (explicit return or "}").

In a system that utilizes CPU’s operating modes, however, whereas it is relatively easy to realize a high-level language support routine in the case of an interrupt handler or the like that runs at the same protection level as the kernel, it is more difficult in the case of a task or task exception handler running at a different protection level from the kernel’s. For this reason, when a high-level language support routine is used for a task, there is no guarantee that the task will exit by a return from the function. Returning a task function using return or "}” leads to an undefined behavior. At the end of a task, Exit Task (tk_ext_tsk) or Exit and Delete Task (tk_exd_tsk) must always be issued.

In the case of a task exception handler, the high-level language support routine is supplied as source code and is to be embedded in the user program.

The internal working of a high-level language support routine is as illustrated in Figure 3.1, “Behavior of High-Level Language Support Routine”.

Figure 3.1: Behavior of High-Level Language Support Routine
3.4 Service Profile

μT-Kernel 3.0 service profile items are shown below. Defining these service profile items is a requirement. The implementor of OS may add original service profile definitions.

3.4.1 Service Profile Items that Represent Function Availability

The service profile item that shows whether a particular function is effective (or enabled) or ineffective (or disabled) is described by using a macro shown below, which is defined to be either TRUE, or FALSE. (The following definitions are given as example only, and each implementation shall define these appropriately.)

3.4.1.1 Device Driver Functions

```
#define TK_SUPPORT_TASKEVENT TRUE /* Support of task event */
#define TK_SUPPORT_DISWAI TRUE /* Support of disabling wait */
#define TK_SUPPORT_IOPORT TRUE /* Support of I/O port access */
#define TK_SUPPORT_MICROWAIT TRUE /* Support of micro wait */
```

Setting TK_SUPPORT_TASKEVENT and TK_SUPPORT_DISWAI to TRUE is recommended on systems that use advanced general-purpose device drivers.

Setting TK_SUPPORT_IOPORT and TK_SUPPORT_MICROWAIT to TRUE is generally recommended.

3.4.1.2 Power Management Functions

```
#define TK_SUPPORT_LOWPOWER TRUE /* Support of power management functions */
```

Setting TK_SUPPORT_LOWPOWER to TRUE is recommended. However, this may as well be set to FALSE on systems with little need for power-saving or restrictions due to used hardware.

3.4.1.3 Static/dynamic Memory Management Functions

```
#define TK_SUPPORT_USERBUF FALSE /* Support of user-specified buffer (TA_USERBUF) */
#define TK_SUPPORT_AUTOBUF TRUE /* Support of automatic buffer allocation (No TA_USERBUF specification) */
#define TK_SUPPORT_MEMLIB TRUE /* Support of memory allocation library */
```

Setting TK_SUPPORT_USERBUF to FALSE is generally recommended.

Setting TK_SUPPORT_AUTOBUF to TRUE is generally recommended.

However, it is acceptable in a system where memory management is staticfally done to set TK_SUPPORT_USERBUF to TRUE, and TK_SUPPORT_AUTOBUF to FALSE.

You cannot set both TK_SUPPORT_USERBUF and TK_SUPPORT_AUTOBUF to FALSE.

Setting TK_SUPPORT_MEMLIB to TRUE is generally recommended.

3.4.1.4 Task Exception Handling Functions

```
#define TK_SUPPORT_TASKEXCEPTION TRUE /* Support of task exception handling functions */
```

Setting TK_SUPPORT_TASKEXCEPTION to TRUE is recommended on a relatively large system that consist of many software modules and that requires flexible handling of abnormal conditions.
3.4.1.5 Subsystem Management Functions

```c
#define TK_SUPPORT_SUBSYSTEM TRUE /* Support of subsystem management functions */
#define TK_SUPPORT_SSYEVENT TRUE /* Support of event processing of subsystems */
```

Setting `TK_SUPPORT_SUBSYSTEM` and `TK_SUPPORT_SSYEVENT` to `TRUE` is recommended on a relatively large system which use middleware.

3.4.1.6 System Configuration Information Acquisition Functions

```c
#define TK_SUPPORT_SYSCONF FALSE /* Support of system configuration information management functions */
```

`TK_SUPPORT_SYSCONF` need to be set to `FALSE` on a system where system configuration information such as the maximum counts of objects (e.g. tasks), is fixed statically at OS build time by hard-coding. On the other hand, if the system configuration information is specified flexibly (e.g. at runtime), `TK_SUPPORT_SYSCONF` need to be set to `TRUE`.

3.4.1.7 Supporting 64-bit and 16-bit CPUs

```c
#define TK_HAS_DOUBLEWORD FALSE /* Support of 64-bit data types (D, UD, VD) */
#define TK_SUPPORT_USEC FALSE /* Support of microsecond */
#define TK_SUPPORT_LARGEDEV FALSE /* Support of large mass-storage device (64-bit) */
#define TK_SUPPORT_SERCD TRUE /* Support of sub error code */
```

`TK_HAS_DOUBLEWORD`, `TK_SUPPORT_USEC`, and `TK_SUPPORT_LARGEDEV` need to be set to either `TRUE` or `FALSE`, according to the target hardware characteristics, and the usage or purpose of the target system.

`TK_SUPPORT_USEC` and `TK_SUPPORT_LARGEDEV` depend on `TK_HAS_DOUBLEWORD`. That is, when `TK_HAS_DOUBLEWORD` is set to `FALSE`, these two profile items are also set to `FALSE`.

Setting `TK_SUPPORT_SERCD` to `TRUE` is recommended on a system where INT and ER are 32 bit entities. Setting `TK_SUPPORT_SERCD` to `FALSE` is recommended on a system where INT and ER are 16 bit entities.

3.4.1.8 Functions that Depend on CPU, Hardware, System, and Compiler

Each of the following profiles needs to be set to `TRUE` or `FALSE` according to the target hardware and the implementation of the OS.

3.4.1.8.1 Interrupt Management Functions

```c
#define TK_SUPPORT_INTCTRL TRUE /* Support of interrupt controller management */
#define TK_HAS_ENAINTLEVEL TRUE /* Can specify interrupt priority level */
#define TK_SUPPORT_CPUINTLEVEL FALSE /* Support of CPU interrupt mask level */
#define TK_SUPPORT_CTRLINTLEVEL TRUE /* Support of interrupt controller mask level */
#define TK_SUPPORT_INTMODE TRUE /* Support of setting interrupt mode */
```
3.4.1.8.2 Memory Cache Control Functions

```
#define TK_SUPPORT_CACHECTRL TRUE /* Support of memory cache control functions */
#define TK_SUPPORT_SETCACHEMODE TRUE /* Support of set cache mode function */
#define TK_SUPPORT_WBCACHE FALSE /* Support of write-back cache */
#define TK_SUPPORT_WTCACHE TRUE /* Support of write-through cache */
```

3.4.1.8.3 FPU(COP) Support Functions

```
#define TK_SUPPORT_FPU TRUE /* Support of FPU */
#define TK_SUPPORT_COP0 TRUE /* Support of co-processor number 0 */
#define TK_SUPPORT_COP1 FALSE /* Support of co-processor number 1 */
#define TK_SUPPORT_COP2 FALSE /* Support of co-processor number 2 */
#define TK_SUPPORT_COP3 FALSE /* Support of co-processor number 3 */
```

3.4.1.8.4 Miscellaneous Functions

```
#define TK_SUPPORT_ASM FALSE /* Support of assembly language function entry/exit */
#define TK_SUPPORT_REGOPS FALSE /* Support for task-register manipulation functions */
#define TK_ALLOW_MISALIGN FALSE /* Memory misalign access is permitted */
#define TK_BIGENDIAN FALSE /* Is big endian (Must be defined) */
#define TK_TRAP_SVC TRUE /* Use CPU Trap instruction for system call entry */
#define TK_HAS_SYSSTACK TRUE /* Task has a separate system stack */
#define TK_SUPPORT_PTIMER TRUE /* Support of physical timer function */
#define TK_SUPPORT_UTC TRUE /* Support of UNIX time */
#define TK_SUPPORT_TRONTIME FALSE /* Support of TRON time */
```

At least one of `TK_SUPPORT_UTC` and `TK_SUPPORT_TRONTIME` must be set to `TRUE`.

3.4.1.9 Debugger Support Functions

```
#define TK_SUPPORT_DSNAME FALSE /* Support of DS object name */
#define TK_SUPPORT_DBGSPRT FALSE /* Support of μT-Kernel/DS */
```

Depending on the user’s need, `TK_SUPPORT_DSNAME` and `TK_SUPPORT_DBGSPRT` may be set to either `TRUE` or `FALSE`. `TK_SUPPORT_DBGSPRT` specifies whether the APIs of μT-Kernel/DS, other than `td_ref_dsname` and `td_set_dsname`, can be used. Even if `TK_SUPPORT_DBGSPRT` is set to `FALSE`, `td_ref_dsname` and `td_set_dsname` can be used if `TK_SUPPORT_DSNAME` is set to `TRUE`.

3.4.1.10 Check Method of Service Profile

Although the implementations of μT-Kernel 3.0 must define the profile items mentioned previously, the use of profile where some definitions are missing should be practiced since other OSs does not provide profile at all, and there bound to be implementation’s failures to define all the profile items. For example, if you want to distinguish the effective/ineffective/undefined status, you can perform the following check:

```
#if defined(TK_SUPPORT_xxx)
  #if TK_SUPPORT_xxx
    /* when a profile item is set to be effective. */
  #else
```

Note that if profile item is directly used for the parameter of "#if" macro as follows, you cannot distinguish whether the profile item is ineffective or undefined.

```c
#if TK_SUPPORT_xxx
  /* when a profile item is set to be effective. */
#else
  /* when a profile item is set to be ineffective or undefined. */
#endif
```

### 3.4.2 Service Profile Items that Represent Values

A service profile item that represents a limit value or version number will be specified as a MACRO that holds the value. (The following definitions are given as example only. The real values of profile items are implementation-dependent.)

```c
#define TK_SPECVER_MAGIC 6  /* Magic number of \uT-Kernel */
#define TK_SPECVER_MAJOR 3   /* Major Version number of \uT-Kernel */
#define TK_SPECVER_MINOR 0   /* Minor Version number of \uT-Kernel */
#define TK_SPECVER (((TK_SPECVER_MAJOR << 8) | TK_SPECVER_MINOR))
#define TK_MAX_TSKPRI 32     /* Version number of \uT-Kernel */
#define TK_WAKEUP_MAXCNT 65535 /* Maximum queuing count of the task wakeup requests (>= 1) */
#define TK_SEMAPHORE_MAXCNT 65535 /* Upper limit of maximum semaphore resource count (maxsem) (>= 32767) */
#define TK_SUSPEND_MAXCNT 65535 /* Maximum nest count of the forced wait of tasks (>= 1) */
#define TK_MEM_RNG0 0        /* Real memory protection level of TA_RNG0 (0~3) */
#define TK_MEM_RNG1 0        /* Real memory protection level of TA_RNG1 (0~3) */
#define TK_MEM_RNG2 0        /* Real memory protection level of TA_RNG2 (0~3) */
#define TK_MEM_RNG3 3        /* Real memory protection level of TA_RNG3 (0~3) */
#define TK_MAX_PTIMER 2      /* Maximum number of physical timers (>= 0) (Values from 1 to TK_MAX_PTIMER can be used as physical timer number) */
```

`TK_MEM_RNGn` defines the real memory protection level of memory specified by TA_RNGn, and if TK_MEM_RNGn == TK_MEM_RNGm, then as far as memory access protection level goes, TA_RNGn and TA_RNGm are equivalent. In other words, it is guaranteed that a task with protection level m can access memory with protection level n without generating access privilege violation exception.

It is recommended that the developer is prepared for the case of missing definitions for service profile items that are supposed to have a value by means of coding such as `defined(...)`.  

### 3.4.3 Examples of Service Profile Items

Following are concrete examples of service profile items.
3.4.3.1 Service Profile Items for a Very Small-scale System using 16-bit CPU

```
#define TK_SUPPORT_TASKEVENT FALSE
#define TK_SUPPORT_DISWAI FALSE
#define TK_SUPPORT_IOPORT TRUE
#define TK_SUPPORT_MICROWAIT TRUE
#define TK_SUPPORT_LOWPOWER TRUE
#define TK_SUPPORT_USERBUF TRUE
#define TK_SUPPORT_AUTOBUF FALSE
#define TK_SUPPORT_MEMLIB FALSE
#define TK_SUPPORT_TASKEXCEPTION FALSE
#define TK_SUPPORT_SUBSYSTEM FALSE
#define TK_SUPPORT_SSYEVENT FALSE
#define TK_SUPPORT_SYSCONF FALSE
#define TK_HAS_DOUBLEWORD FALSE
#define TK_SUPPORT_USEC FALSE
#define TK_SUPPORT_LARGEDEV FALSE
#define TK_SUPPORT_SERCD FALSE
#define TK_SUPPORT_INTCTRL FALSE
#define TK_HAS_ENAINTLEVEL FALSE
#define TK_SUPPORT_CPUINTLEVEL FALSE
#define TK_SUPPORT_CTRLINTLEVEL FALSE
#define TK_SUPPORT_INTMODE TRUE
#define TK_SUPPORT_CACHECTRL FALSE
#define TK_SUPPORT_SETCACHEMODE FALSE
#define TK_SUPPORT_WBCACHE FALSE
#define TK_SUPPORT_WTCACHE FALSE
#define TK_SUPPORT_FPU FALSE
#define TK_SUPPORT_COP0 FALSE
#define TK_SUPPORT_COP1 FALSE
#define TK_SUPPORT_COP2 FALSE
#define TK_SUPPORT_COP3 FALSE
#define TK_SUPPORT_ASM TRUE
#define TK_SUPPORT_REGOPS FALSE
#define TK_ALLOW_MISALIGN FALSE
#define TK_BIGENDIAN FALSE
#define TK_TRAP_SVC FALSE
#define TK_HAS_SYSSTACK FALSE
#define TK_SUPPORT_PTIMER FALSE
#define TK_SUPPORT_UTC TRUE
#define TK_SUPPORT_TRONTIME FALSE
#define TK_SUPPORT_DSNAME FALSE
#define TK_SUPPORT_DBGSPPT FALSE
#define TK_SPECVER_MAGIC 6
#define TK_SPECVER_MAJOR 3
#define TK_SPECVER_MINOR 0
#define TK_SPECVER (((TK_SPECVER_MAJOR << 8) | TK_SPECVER_MINOR))
#define TK_MAX_TSKPRI 16
#define TK_WAKEUP_MAXCNT 4095
```
3.4.3.2 Service Profile Items for a Relatively Large-scale System

```
#define TK_SUPPORT_TASKEVENT TRUE
#define TK_SUPPORT_DISWAI TRUE
#define TK_SUPPORT_IOPORT TRUE
#define TK_SUPPORT_MICROWAIT TRUE
#define TK_SUPPORT_LOWPOWER TRUE
#define TK_SUPPORT_USERBUF FALSE
#define TK_SUPPORT_AUTOBUF TRUE
#define TK_SUPPORT_MEMLIB TRUE
#define TK_SUPPORT_TASKEXCEPTION TRUE
#define TK_SUPPORT_SUBSYSTEM TRUE
#define TK_SUPPORT_SSYEVENT TRUE
#define TK_SUPPORT_SYSCONF TRUE
#define TK_HAS_DOUBLEWORD TRUE
#define TK_SUPPORT_USEC TRUE
#define TK_SUPPORT_LARGEDEV TRUE
#define TK_SUPPORT_SERCD TRUE
#define TK_SUPPORT_INTCTRL TRUE
#define TK_HAS_ENAINTLEVEL TRUE
#define TK_SUPPORT_CPUINTLEVEL FALSE
#define TK_SUPPORT_CTRLINTLEVEL TRUE
#define TK_SUPPORT_INTMODE TRUE
#define TK_SUPPORT_CACHECTRL TRUE
#define TK_SUPPORT_SETCACHEMODE TRUE
#define TK_SUPPORT_WBCACHE TRUE
#define TK_SUPPORT_WTCACHE TRUE
#define TK_SUPPORT_FPU TRUE
#define TK_SUPPORT_COP0 TRUE
#define TK_SUPPORT_COP1 FALSE
#define TK_SUPPORT_COP2 FALSE
#define TK_SUPPORT_COP3 FALSE
#define TK_SUPPORT_ASM TRUE
#define TK_SUPPORT_REGOPS TRUE
#define TK_ALLOW_MISALIGN FALSE
#define TK_BIGENDIAN FALSE
#define TK_TRAP_SVC TRUE
#define TK_HAS_SYSSTACK TRUE
#define TK_SUPPORT_PTIMER TRUE
#define TK_SUPPORT_UTC TRUE
#define TK_SUPPORT_TRONTIME FALSE
```
#define TK_SUPPORT_DSNAME TRUE
#define TK_SUPPORT_DBGSPC TRUE

#define TK_SPECVER_MAGIC 6
#define TK_SPECVER_MAJOR 3
#define TK_SPECVER_MINOR 0
#define TK_SPECVER ((TK_SPECVER_MAJOR << 8) | TK_SPECVER_MINOR)

#define TK_MAX_TSKPRI 140
#define TK_WAKEUP_MAXCNT 65535
#define TK_SEMAPHORE_MAXCNT 65535
#define TK_SUSPEND_MAXCNT 65535
#define TK_MEM_RNG0 0
#define TK_MEM_RNG1 0
#define TK_MEM_RNG2 0
#define TK_MEM_RNG3 3
#define TK_MAX_PTIMER 10
Chapter 4

μT-Kernel/OS Functions

This chapter describes details of the system calls provided by μT-Kernel/OS (Operating System).
4.1 Task Management Functions

Task management functions are functions that directly manipulate or reference task states. Functions are provided for creating and deleting a task, for task starting and exit, changing task priority, and referencing task state. A task is an object identified by an ID number called a task ID. Task states and scheduling rules are explained in Section 2.2, “Task States and Scheduling Rules”.

For control of execution order, a task has a base priority and current priority. When simply "task priority" is mentioned, this means the current priority. The base priority of a task is initialized to the startup priority when a task is started. If the mutex function is not used, the task current priority is always identical to its base priority. For this reason, the current priority immediately after a task is started is the task startup priority. When the mutex function is used, the current priority is set as discussed in Section 4.5.1, “Mutex”.

The kernel does not perform processing for freeing of resources acquired by a task (semaphore resources, memory blocks, etc.) upon task exit, other than mutex unlocking. Freeing of task resources is the responsibility of the application.
4.1.1  tk_cre_tsk - Create Task

C Language Interface

#include <tk/tkernel.h>

ID tskid = tk_cre_tsk(CONST T_CTSK *pk_ctsk);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_CTSK*</td>
<td>pk_ctsk</td>
<td>Packet to Create Task</td>
</tr>
</tbody>
</table>

pk_ctsk Detail:

- void* exinf: Extended Information
- ATR tskatr: Task Attribute
- FP task: Task Start Address
- PRI itskpri: Initial Task Priority
- SZ stksz: Stack Size
- SZ sstksz: System Stack Size
- void* stkptr: User Stack Pointer
- UB dsname[8]: DS Object name
- void* bufptr: Buffer Pointer

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID or Error Code</th>
</tr>
</thead>
</table>

Error Code

- E_NOMEM: Insufficient memory (memory for control block or user stack cannot be allocated)
- E_LIMIT: Number of tasks exceeds the system limit
- E_RSATR: Reserved attribute (tskatr is invalid or cannot be used), or the specified coprocessor does not exist
- E_NOSPT: Unsupported functions(when the specification of TA_ASM, TA_USERSTACK, TA_TASKSPACE, or TA_USERBUF is not supported.)
- E_PAR: Parameter error
- E_NOCOP: The specified coprocessor cannot be used (not installed, or abnormal operation detected)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items
TK_SUPPORT_ASM
Support for specifying TA_ASM for task attribute

TK_SUPPORT_USERBUF
Support for specifying TA_USERBUF for task attribute

TK_SUPPORT_AUTOBUF
Automatic buffer allocation is supported (by not specifying TA_USERBUF to task attribute)

TK_SUPPORT_FPU
Support for specifying TA_FPU for task attribute

TK_SUPPORT_COPn
Support for specifying TA_COPn for task attribute

TK_HAS_SYSSTACK
Task can have a system stack independent of user-stack, and each can be specified separately using (TA_USERSTACK, TA_SSTKSZ)

TK_SUPPORT_DSNAME
Support for specifying TA_DSNAME for task attribute

TK_MAX_TSKPRI
Maximum task priority that can be specified (must be 16 or higher)

Description

Creates a task, assigning to it a task ID number. This system call allocates a TCB (Task Control Block) to the created task and initializes it based on itskpri, task, stksz and other parameters.

After the task is created, it is initially in DORMANT state.

itskpri is used to specify the startup priority when a task is started. Task priority level can be specified by a positive integer, and the smaller the value, higher priority the task has. The largest task priority level is defined by TK_MAX_TSKPRI.

exinf can be used freely by the user to insert miscellaneous information about the task. The information set here is passed to the task as startup parameter information and can be referred to by calling tk_ref_tsk. If a larger area is needed for indicating user information, or if the information may need to be changed after the task is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in exinf. The kernel pays no attention to the contents of exinf.

tskatr indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of tskatr is as follows.

tskatr := (TA_ASM || TA_HLNG)
| [TA_SSTKSZ] | [TA_USERSTACK] | [TA_USERBUF] | [TA_DSNAME]
| [TA_RNG0] || [TA_RNG1] || [TA_RNG2] || [TA_RNG3]
| [TA_COP0] | [TA_COP1] | [TA_COP2] | [TA_COP3] | [TA_FPU]

TA_ASM Indicates that the task is written in assembly language
TA_HLNG Indicates that the task is written in high-level language
TA_SSTKSZ Specifies the system stack size
TA_USERSTACK Points to the user stack
TA_USERBUF Use of user-specified memory area as stack
TA_DSNAME Specifies DS object name
TA_RNGn Indicates that the task runs at protection level n
TA_COPn Specifies use of the nth coprocessor (including floating point coprocessor or DSP)
TA_FPU Specifies use of a floating point coprocessor (when a coprocessor specified in TA_COPn is a general-purpose FPU particularly for floating point processing and not dependent on the CPU)

The function for specifying implementation-dependent attributes can be used, for example, to specify that a task is subject to debugging. One use of the remaining system attribute fields is for indicating multiprocessor attributes in the future.

#define TA_ASM 0x00000000 /* Task in Assembly Language */
define TA_HLNG 0x00000001 /* Task in High-level language */
define TA_SSTKSZ 0x00000002 /* System stack size */
define TA_USERSTACK 0x00000004 /* User stack pointer */
When TA_HLNG is specified, starting the task jumps to the task address not directly but by going through a high-level language environment configuration program (high-level language support routine). The task takes the following form in this case.

```c
void task( INT stacd, void *exinf )
{
    /* (processing) */
    tk_ext_tsk(); or tk_exd_tsk(); /* Exit task */
}
```

The startup parameters passed to the task include the task startup code stacd specified in tk_sta_tsk, and the extended information exinf specified in tk_cre_tsk.

The task cannot (must not) be terminated by a simple return from the function, otherwise the operation will be indeterminate (implementation-dependent).

The form of the task when the TA_ASM attribute is specified in implementation-dependent, but stacd and exinf must be passed as startup parameters.

The task runs at the protection level specified in the TA_RNGn attribute. When a system call or extended SVC is called, the protection level goes to 0, then goes back to its original level upon return from the system call or extended SVC.

Each task has two stack areas, a system stack and user stack. The user stack is used at the protection level specified in TA_RNGn while the system stack is used at protection level 0. When the calling of a system call or extended SVC causes the protection level to change, the stack is also switched.

Note that a task running at TA_RNG0 does not switch protection levels, so there is no stack switching either. When TA_RNG0 is specified, the combined total of the user stack size and system stack size is the size of one stack, employed as both a user stack and system stack.

When TA_SSTKSZ is specified, sstksz is valid. If TA_SSTKSZ is not specified, sstksz is ignored and the default size applies.

When TA_USERSTACK is specified, stkptr is valid. In this case a user stack is not provided by the OS, but must be allocated by the caller. stkksz must be set to 0. If TA_USERSTACK is not specified, stkptr is ignored. Note that if TA_RNG0 is set, TA_USERSTACK cannot be specified. E_PAR occurs if TA_RNG0 and TA_USERSTACK are specified at the same time.

TA_USERBUF can be specified for implementation where there is no distinction of user stack and system stack and there is only one unified stack for a task. When this attribute is specified, bufptr becomes effective, and the memory area starting at bufptr containing stkksz octets is used as the unified user and system stack area. In this case, the kernel does not provide the stack area.

When TA_DSNAME is specified, dsname is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, td_ref_dsname and td_set_dsname. For more details, see the description of td_ref_dsname and td_set_dsname. If TA_DSNAME is not specified, dsname is ignored. Then td_ref_dsname and td_set_dsname return E_OBJ error.
Additional Notes

A task runs either at the protection level set in TA_RNGn or at protection level 0. For example, a task for which TA_RNG3 is specified in no case runs at protection level 1 or 2.

In a system with separate interrupt stack, interrupt handlers also use the system stack. An interrupt handler runs at protection level 0.

The system stack default size is decided taking into account the amount taken up by system call execution and, in a system with separate interrupt stack, the amount used by interrupt handlers.

The definition of TA_COPn is dependent on the CPU and other hardware and is not portable.

TA_FPU is provided as a portable notation method only for the definition in TA_COPn of a floating point coprocessor. If, for example, the floating point coprocessor is TA_COP0, then TA_FPU = TA_COP0. If there is no particular need to specify the use of a coprocessor for floating point operations, TA_FPU = 0 is set.

Even in a system with a single CPU’s operating mode, for the sake of portability all attributes including TA_RNGn must be accepted. It is possible, for example, to handle all TA_RNGn as equivalent to TA_RNG0, but error must not be returned.

Porting Guideline

The T-Kernel 2.0 specification does not define TA_USERBUF and its associated notion of bufptr. So if this feature is used, a modification is necessary to port the software to T-Kernel 2.0. However, if stksz is properly set already, simply removing TA_USERBUF and bufptr will complete the modification for porting.

The largest task priority is defined by TK_MAX_TSKPRI. Although TK_MAX_TSKPRI is variable, but is guaranteed to be equal to or larger than 16, and so by restricting the used task priorities only to the range from 1 to 16, there shall be no need for modifying the task priorities during porting.
4.1.2 tk_del_tsk - Delete Task

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_del_tsk(ID tskid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Task ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_ID: Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the task specified in tskid does not exist)
- E_OBJ: Invalid object state (the task is not in DORMANT state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes the task specified in tskid.

This system call changes the state of the task specified in tskid from DORMANT state to NONEXISTENT state (no longer exists in the system), releasing the TCB and stack area that were assigned to the task. The task ID number is also released. When this system call is issued for a task not in DORMANT state, error code E_OBJ is returned.

This system call cannot specify the invoking task. If the invoking task is specified, error code E_OBJ is returned since the invoking task is not in DORMANT state. The invoking task is deleted not by this system call but by the tk_exd_tsk system call.
4.1.3  tk_sta_tsk - Start Task

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_sta_tsk(ID tskid, INT stacd);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>stacd</td>
<td>Task Start Code</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_ID: Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the task specified in tskid does not exist)
- E_OBJ: Invalid object state (the task is not in DORMANT state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Starts the task specified in tskid. This system call changes the state of the specified task from DORMANT state to READY state.

Parameters to be passed to the task when it starts can be set in stacd. These parameters can be referred to from the started task, enabling use of this feature for simple message passing.

The task priority when it starts is the task startup priority (itskpri) specified when the started task was created.

Start requests by this system call are not queued. If this system call is issued while the target task is in a state other than DORMANT state, the system call is ignored and error code E_OBJ is returned to the calling task.

Porting Guideline

Note that stacd is INT type, and its value range is implementation-dependent, so care must be taken.
4.1.4  tk_ext_tsk - Exit Task

C Language Interface

#include <tk/tkernel.h>

void tk_ext_tsk(void);

Parameter

None.

Return Parameter

Does not return to the context issuing the system call.

Error Codes

The following kind of error may be detected, but no return is made to the context issuing the system call even if the error is detected. For this reason the error code cannot be passed directly as a system call return parameter. The behavior in case an error occurs is implementation-dependent.

  E_CTX  Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

+----------------+-----------------+------------------+
| Task portion   | Quasi-task portion | Task-independent portion |
| YES           | YES              | NO                |
+

Related Service Profile Items

None.

Description

Exits the invoking task normally and changes its state to DORMANT state.

Additional Notes

When a task terminates by tk_ext_tsk, the resources acquired by the task up to that time (memory blocks, semaphores, etc.) are not automatically freed. The user is responsible for releasing such resources before the task exits.

tk_ext_tsk is a system call that does not return to the context from which it was called. Even if an error code is returned when an error of some kind is detected, normally no error checking is performed in the context from which the system call was invoked, leaving the possibility that the program will behave in an unexpected manner. For this reason these system calls do not return even if error is detected.

As a rule, the task priority and other information included in the TCB is reset when the task returns to DORMANT state. If, for example, the task priority is changed by tk_chg_pri and later terminated by tk_ext_tsk, the
task priority reverts to the startup priority (itskpri) specified by tk_cre_tsk at startup. It does not keep the task priority in effect at the time tk_ext_tsk was executed.

System calls that do not return to the calling context are those named tk_ret_?? or tk_ext_?? (tk_exd_??).
4.1.5  tk_exd_tsk - Exit and Delete Task

C Language Interface

#include <tk/tkernel.h>

void tk_exd_tsk(void);

Parameter

None.

Return Parameter

Does not return to the context issuing the system call.

Error Codes

The following kind of error may be detected, but no return is made to the context issuing the system call even if the error is detected. For this reason the error code cannot be passed directly as a system call return parameter. The behavior in case an error occurs is implementation-dependent.

E_CTX Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Terminates the invoking task normally and also deletes it. This system call changes the state of the invoking task to NON-EXISTENT state (no longer exists in the system).

Additional Notes

When a task terminates by tk_exd_tsk, the resources acquired by the task up to that time (memory blocks, semaphores, etc.) are not automatically freed. The user is responsible for releasing such resources before the task exits.

tk_exd_tsk is a system call that does not return to the context from which it was called. Even if an error code is returned when an error of some kind is detected, normally no error checking is performed in the context from which the system call was invoked, leaving the possibility that the program will behave in an unexpected manner. For this reason these system calls do not return even if error is detected.
4.1.6 tk_ter_tsk - Terminate Task

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ter_tsk(ID tskid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

| E_OK | Normal completion |
| E_ID | Invalid ID number (tskid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the task specified in tskid does not exist) |
| E_OBJ | Invalid object state (the target task is in DORMANT state or is the invoking task) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Forcibly terminates the task specified in tskid. This system call changes the state of the target task specified in tskid to DORMANT state.

Even if the target task was in the waiting state (including SUSPENDED state), the waiting state is released and the task is terminated. If the target task was in some kind of queue (semaphore wait, etc.), executing tk_ter_tsk results in its removal from the queue.

This system call cannot specify the invoking task. If the invoking task is specified, error code E_OBJ is returned.

The relationships between target task states and the results of executing tk_ter_tsk are summarized in Table 4.1, “Target Task State and Execution Result (tk_ter_tsk)”.

Additional Notes

When a task is terminated by tk_ter_tsk, the resources acquired by the task up to that time (memory blocks, semaphores, etc.) are not automatically freed. The user is responsible for releasing such resources before the task is terminated.
<table>
<thead>
<tr>
<th>Target Task State</th>
<th>tk_ter_tsk Return Value</th>
<th>(processing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run state (RUNNING or READY) (not for invoking task)</td>
<td>E_OK</td>
<td>Forced termination</td>
</tr>
<tr>
<td>Running state (RUNNING) (invoking task)</td>
<td>E_OBJ</td>
<td>No operation</td>
</tr>
<tr>
<td>Waiting state (WAITING)</td>
<td>E_OK</td>
<td>Forced termination</td>
</tr>
<tr>
<td>Suspended state (SUSPENDED)</td>
<td>E_OK</td>
<td>Forced termination</td>
</tr>
<tr>
<td>Waiting-suspended state (WAITING-SUSPENDED)</td>
<td>E_OK</td>
<td>Forced termination</td>
</tr>
<tr>
<td>Dormant state (DORMANT)</td>
<td>E_OBJ</td>
<td>No operation</td>
</tr>
<tr>
<td>Non-existent state (NON-EXISTENT)</td>
<td>E_NOEXS</td>
<td>No operation</td>
</tr>
</tbody>
</table>

Table 4.1: Target Task State and Execution Result (tk_ter_tsk)

As a rule, the task priority and other information included in the TCB is reset when the task returns to DORMANT state. If, for example, the task priority is changed by tk_chg_pri and later terminated by tk_ter_tsk, the task priority reverts to the startup priority (itskpri) that is specified by tk_cre_tsk at startup. The task priority at task termination by tk_ter_tsk is not used after the task is restarted by tk_sta_tsk.

Forcible termination of another task is intended for use only by a debugger or a few other tasks closely related to the OS. As a rule, this system call is not to be used by ordinary applications or middleware, for the following reason.

Forced termination occurs regardless of the running state of the target task. If, for example, a task were forcibly terminated while the task was calling a middleware function, the task would terminate right while the middleware was executing. If such a situation were allowed, normal operation of the middleware could not be guaranteed.

This is an example of how task termination should not be allowed when the task status (what it is executing) is unknown. Ordinary applications therefore must not use the forcible termination function.
### 4.1.7 tk_chg_pri - Change Task Priority

#### C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_chg_pri(ID tskid, PRI tskpri);
```

#### Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>tskpri</td>
<td>Task Priority</td>
</tr>
</tbody>
</table>

#### Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

#### Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (tskid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the task specified in tskid does not exist)
- **E_PAR**: Parameter error (tskpri is invalid or cannot be used)
- **E_ILUSE**: Illegal use (upper priority limit exceeded)

#### Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

#### Related Service Profile Items

- **TK_MAX_TSKPRI**: Maximum task priority that can be specified (must be 16 or higher)

#### Description

Changes the base priority of the task specified in tskid to the value specified in tskpri. The current priority of the task also changes as a result.

Task priority values are specified from 1 to **TK_MAX_TSKPRI**, with the smaller numbers indicating higher priority.

When **TSK_SELF** (= 0) is specified in tskid, the invoking task is the target task. Note, however, that when tskid=TSK_SELF is specified in a system call issued from a task-independent portion, error code E_ID is returned. When **TPRI_INI** (= 0) is specified as tskpri, the target task base priority is changed to the initial priority when the task was started (itskpri).

A priority changed by this system call remains valid until the task is terminated. When the task reverts to DORMANT state, the task priority before its exit is discarded, with the task again assigned to the initial priority when the task was started (itskpri). However, the priority changed in DORMANT state is valid. The next time the task is started, it has the new initial priority.
If as a result of this system call execution the target task current priority matches the base priority (this condition is always met when the mutex function is not used), processing is as follows.

If the target task is in a run state, the task precedence changes according to its priority. The target task has the lowest precedence among tasks of the same priority after the change.

If the target task is in some kind of priority-based queue, the order in that queue changes in accordance with the new task priority. Among tasks of the same priority after the change, the target task is queued at the end.

If the target task has locked a TA_CEILING attribute mutex or is waiting for a lock, and the base priority specified in tskpri is higher than any of the ceiling priorities, error code E_ILUSE is returned.

Additional Notes

In some cases when this system call results in a change in the queued order of the target task in a task priority-based queue, it may be necessary to release the wait state of another task waiting in that queue (in a message buffer send queue, or in a queue waiting to acquire a variable-size memory pool).

In some cases when this system call results in a base priority change while the target task is waiting for a mutex lock with TA_INHERIT dynamic priority inheritance processing may be necessary.

When a mutex function is not used and the system call is issued specifying the invoking task as the target task, setting the new priority to the base priority of the invoking task, the order of execution of the invoking task becomes the lowest among tasks of the same priority. This system call can therefore be used to relinquish execution privilege.

Porting Guideline

The largest task priority is defined by TK_MAX_TSKPRI. Although TK_MAX_TSKPRI is variable, but is guaranteed to be equal to or larger than 16, and so by restricting the used task priorities only to the range from 1 to 16, there shall be no need for modifying the task priorities during porting.
4.1.8 tk_get_reg - Get Task Registers

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_get_reg(ID tskid, T_REGS *pk_regs, T_EIT *pk_eit, T_CREGS *pk_cregs);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_REGS*</td>
<td>pk_regs</td>
<td>Packet of Registers</td>
<td>Pointer to the area to return the general register values</td>
</tr>
<tr>
<td>T_EIT*</td>
<td>pk_eit</td>
<td>Packet of EIT Registers</td>
<td>Pointer to the area to return the values of registers saved when an exception occurs</td>
</tr>
<tr>
<td>T_CREGS*</td>
<td>pk_cregs</td>
<td>Packet of Control Registers</td>
<td>Pointer to the area to return the control register values</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

The contents of T_REGS, T_EIT, and T_CREGS are defined for each CPU and implementation.

Error Code

- E_OK Normal completion
- E_ID Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS Object does not exist (the task specified in tskid does not exist)
- E_OBJ Invalid object state (called for the invoking task)
- E_CTX Context error (called from task-independent portion)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- TK_SUPPORT_REGOPS Support for task-register manipulation functions

Description

Gets the current register contents of the task specified in tskid.

If NULL is set in pk_regs, pk_eit, or pk_cregs, the corresponding registers are not referenced.
The referenced register values are not necessarily the values at the time the task portion was executing. If this system call is issued for the invoking task, error code E_OBJ is returned.

Additional Notes

In principle, all registers in the task context can be referenced. This includes not only physical CPU registers but also those treated by the kernel as virtual registers.
4.1.9  tk_set_reg - Set Task Registers

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_set_reg(ID tskid, CONST T_REGS *pk_regs, CONST T_EIT *pk_eit, CONST T_CREGS *pk_cregs);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_REGS*</td>
<td>pk_regs</td>
<td>Packet of Registers</td>
<td>General registers</td>
</tr>
<tr>
<td>CONST T_EIT*</td>
<td>pk_eit</td>
<td>Packet of EIT Registers</td>
<td>Registers saved when EIT occurs</td>
</tr>
<tr>
<td>CONST T_CREGS*</td>
<td>pk_cregs</td>
<td>Packet of Control Registers</td>
<td>Control registers</td>
</tr>
</tbody>
</table>

The contents of T_REGS, T_EIT, and T_CREGS are defined for each CPU and implementation.

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

| E_OK | Normal completion |
| E_ID | Invalid ID number (tskid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the task specified in tskid does not exist) |
| E_OBJ | Invalid object state (called for the invoking task) |
| E_CTX | Context error (called from task-independent portion) |
| E_PAR | Invalid register value (implementation-dependent) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_REGOPS  Support for task-register manipulation functions

Description

Sets the current register contents of the task specified in tskid.

If NULL is set in pk_regs, pk_eit, or pk_cregs, the corresponding registers are not set.

The set register values are not necessarily the values while the task portion is executing. The kernel is not responsible for handling the side-effects of register value changes.
It is possible, however, that some registers or register bits cannot be changed if the kernel does not allow such changes. (Implementation-dependent)

If this system call is issued for the invoking task, error code E_OBJ is returned.
4.1.10 tk_get_cpr - Get Task Coprocessor Registers

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_get_cpr(ID tskid, INT copno, T_COPREGS *pk_copregs);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>copno</td>
<td>Coprocessor Number</td>
<td>Coprocessor number (0 to 3)</td>
</tr>
<tr>
<td>T_COPREGS*</td>
<td>pk_copregs</td>
<td>Packet of Coprocessor Registers</td>
<td>Pointer to the area to return coprocessor register values</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>pk_copregs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Detail:

- T_COP0REG cop0 Coprocessor Number 0 Register
- T_COP1REG cop1 Coprocessor Number 1 Register
- T_COP2REG cop2 Coprocessor Number 2 Register
- T_COP3REG cop3 Coprocessor Number 3 Register

The contents of T_COPnREG are defined for each CPU and implementation.

Error Code

- E_OK Normal completion
- E_ID Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS Object does not exist (the task specified in tskid does not exist)
- E_OBJ Invalid object state (called for the invoking task)
- E_CTX Context error (called from task-independent portion)
- E_PAR Parameter error (copno is invalid or the specified coprocessor does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

TK_SUPPORT_COPn Support of co-processor number n
If `TK_SUPPORT_COPn` is ineffective for all \( n \), this API is unsupported.

**Description**

Gets the current contents of the register specified in `copno` of the task specified in `tskid`.
The referenced register values are not necessarily the values at the time the task portion was executing.
If this system call is issued for the invoking task, error code `E_OBJ` is returned.

**Additional Notes**

In principle, all registers in the task context can be referenced. This includes not only physical CPU registers but also those treated by the kernel as virtual registers.
4.1.11 tk_set_cpr - Set Task Coprocessor Registers

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_set_cpr(ID tskid, INT copno, CONST T_COPREGS *pk_copregs);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tskid</td>
<td>Task ID</td>
<td>Task ID</td>
</tr>
<tr>
<td>copno</td>
<td>Coprocessor Number</td>
<td>Coprocessor number (0 to 3)</td>
</tr>
<tr>
<td>pk_copregs</td>
<td>Packet of Coprocessor Registers</td>
<td>Coprocessor register</td>
</tr>
</tbody>
</table>

pk_copregs Detail:

| T_COP0REG | cop0 | Coprocessor Number 0 Register | Coprocessor number 0 register |
| T_COP1REG | cop1 | Coprocessor Number 1 Register | Coprocessor number 1 register |
| T_COP2REG | cop2 | Coprocessor Number 2 Register | Coprocessor number 2 register |
| T_COP3REG | cop3 | Coprocessor Number 3 Register | Coprocessor number 3 register |

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ercd</td>
<td>Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Invalid ID number (tskid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>Invalid object state (called for the invoking task)</td>
</tr>
<tr>
<td>E_CTX</td>
<td>Context error (called from task-independent portion)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (copno is invalid or the specified coprocessor does not exist), or the set register value is invalid (implementation-dependent)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

Related Service Profile Items

TK_SUPPORT_COPn Support of co-processor number n
If `TK_SUPPORT_COPn` is ineffective for all $n$, this API is unsupported.

**Description**

Sets the contents of the register specified in `copno` of the task specified in `tskid`.

The set register values are not necessarily the values while the task portion is executing. The kernel is not responsible for handling the side-effects of register value changes.

It is possible, however, that some registers or register bits cannot be changed if the kernel does not allow such changes. (Implementation-dependent)

If this system call is issued for the invoking task, error code E_OBJ is returned.
4.1.12 tk_ref_tsk - Reference Task Status

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RTSK*</td>
<td>pk_rtsk</td>
<td>Packet to Return Task Status</td>
</tr>
<tr>
<td>Task ID</td>
<td>Pointer to the area to return the task status</td>
<td></td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Code</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

pk_rtsk Detail:

- void* exinf Extended Information
- PRI tskpri Task Priority
- PRI tskbpri Task Base Priority
- UINT tskstat Task State
- UW tskwait Task Wait Factor
- ID wid Waiting Object ID
- INT wupcnt Wakeup Count
- INT suscnt Suspend Count
- UW waitmask Wait Mask
- UINT texmask Task Exception Mask
- UINT tskevent Task Event

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

- E_OK Normal completion
- E_ID Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS Object does not exist (the task specified in tskid does not exist)
- E_PAR Parameter error (invalid pk_rtsk)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

- TK_SUPPORT_DISWAI Information about disabled wait factors (waitmask) is obtainable
- TK_SUPPORT_TASKEXCEPTION Task exception information (texmask) can be acquired
- TK_SUPPORT_TASKEVENT Generated task event (tskevent) can be acquired
Description

Gets the state of the task specified in \texttt{tskid}.

\texttt{tskstat} takes the following values.

\begin{verbatim}
TTS_RUN 0x0001 RUNNING state
TTS_RDY 0x0002 READY state
TTS_WAI 0x0004 WAITING state
TTS_SUS 0x0008 SUSPENDED state
TTS_WAS 0x000c WAITING-SUSPENDED state
TTS_DMT 0x0010 DORMANT state
TTS_NODISWAI 0x0080 Disabling of wait by \texttt{tk_dis_wai} is prohibited
\end{verbatim}

Task states such as \texttt{TTS_RUN} and \texttt{TTS_WAI} are expressed by corresponding bits, which is useful when making a complex state decision (e.g., deciding that the state is one of either RUNNING or READY state). Note that of the above states, \texttt{TTS_WAS} is a combination of \texttt{TTS_SUS} and \texttt{TTS_WAI} but \texttt{TTS_SUS} is never combined with other states (\texttt{TTS_RUN}, \texttt{TTS_RDY}, \texttt{TTS_DMT}).

In the case of \texttt{TTS_WAI} (including \texttt{TTS_WAS}), disabling of wait by the \texttt{tk_dis_wai} is prohibited, \texttt{TTS_NODISWAI} is set. \texttt{TTS_NODISWAI} is never combined with states other than \texttt{TTS_WAI}.

When \texttt{tk_ref_tsk} is executed for an interrupted task from an interrupt handler, \texttt{RUNNING} (\texttt{TTS_RUN}) is returned as \texttt{tskstat}.

When \texttt{tskstat} is \texttt{TTS_WAI} (including \texttt{TTS_WAS}), the values of \texttt{tskwait} and \texttt{wid} are as shown in Table 4.2, “Values of \texttt{tskwait} and \texttt{wid}”.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\texttt{tskwait} & \texttt{Value} & \texttt{Description} & \texttt{wid} \\
\hline
\texttt{TTW_SLP} & 0x00000001 & Wait caused by \texttt{tk_slp_tsk} & 0 \\
\texttt{TTW_DLY} & 0x00000002 & Wait caused by \texttt{tk_dly_tsk} & 0 \\
\texttt{TTW_SEM} & 0x00000004 & Wait caused by \texttt{tk_wai_sem} & \texttt{semid} \\
\texttt{TTW_FLG} & 0x00000008 & Wait caused by \texttt{tk_wai_flg} & \texttt{flgid} \\
\texttt{TTW_MBX} & 0x00000040 & Wait caused by \texttt{tk_rcv_mbx} & \texttt{mbxid} \\
\texttt{TTW_MTX} & 0x00000080 & Wait caused by \texttt{tk_loc_mtx} & \texttt{mtxid} \\
\texttt{TTW_SMBF} & 0x00000100 & Wait caused by \texttt{tk_snd_mbf} & \texttt{mbfid} \\
\texttt{TTW_RMBF} & 0x00000200 & Wait caused by \texttt{tk_rcv_mbf} & \texttt{mbfid} \\
\texttt{TTW_CAL} & 0x00000400 & (reserved) & (reserved) \\
\texttt{TTW_ACP} & 0x00000800 & (reserved) & (reserved) \\
\texttt{TTW_RDV} & 0x00001000 & (reserved) & (reserved) \\
\texttt{TTW_CAL} | \texttt{TTW_RDV} & 0x00001400 & (reserved) & (reserved) \\
\texttt{TTW_MPF} & 0x00002000 & Wait caused by \texttt{tk_get_mpf} & \texttt{mpfid} \\
\texttt{TTW_MPL} & 0x00004000 & Wait caused by \texttt{tk_get_mpl} & \texttt{mplid} \\
\texttt{TTW_EV1} & 0x00010000 & Wait for task event \#1 & 0 \\
\texttt{TTW_EV2} & 0x00020000 & Wait for task event \#2 & 0 \\
\texttt{TTW_EV3} & 0x00040000 & Wait for task event \#3 & 0 \\
\texttt{TTW_EV4} & 0x00080000 & Wait for task event \#4 & 0 \\
\texttt{TTW_EV5} & 0x00100000 & Wait for task event \#5 & 0 \\
\texttt{TTW_EV6} & 0x00200000 & Wait for task event \#6 & 0 \\
\texttt{TTW_EV7} & 0x00400000 & Wait for task event \#7 & 0 \\
\texttt{TTW_EV8} & 0x00800000 & Wait for task event \#8 & 0 \\
\hline
\end{tabular}
\caption{Values of \texttt{tskwait} and \texttt{wid}}
\end{table}

When \texttt{tskstat} is not \texttt{TTS_WAI} (including \texttt{TTS_WAS}), both \texttt{tskwait} and \texttt{wid} are 0.

\texttt{waitmask} is the same bit array as \texttt{tskwait}.
**texmask** is a logical OR bit array representing permitted task exception codes in the form $1 \ll$ task exception code for each code.

**tskevent** shows the list of generated and pending task events by representing each event as $1 \ll$ (task event number - 1) and calculating the logical OR of the bit values.

For a task in DORMANT state, $\text{wupcnt} = 0$, $\text{suscnt} = 0$, and $\text{tskevent} = 0$.

The invoking task can be specified by setting $\text{tskid} = \text{TSK_SELF} = 0$. Note, however, that when $\text{tskid} = \text{TSK_SELF} = 0$ is specified in a system call issued from a task-independent portion, error code E_ID is returned.

When the task specified with $\text{tk_ref_tsk}$ does not exist, error code E_NOEXS is returned.

**Additional Notes**

Even when $\text{tskid} = \text{TSK_SELF}$ is specified with this system call, the ID of the invoking task is not known. Use $\text{tk_get_tid}$ to find out the ID of the invoking task.
4.2 Task Synchronization Functions

Task synchronization functions achieve synchronization among tasks by direct manipulation of task states. They include functions for task sleep and wakeup, for canceling wakeup requests, for forcibly releasing task WAITING state, for changing a task state to SUSPENDED state, for delaying execution of the invoking task, and for disabling task WAITING state.

Wakeup requests for a task are queued. That is, when it is attempted to wake up a task that is not sleeping, the wakeup request is remembered, and the next time the task is to go to a sleep state (waiting for wakeup), it does not enter that state. The queuing of task wakeup requests is realized by having the task keep a task wakeup request queuing count. When the task is started, this count is cleared to 0.

Suspend requests for a task are nested. That is, if it is attempted to suspend a task already in SUSPENDED state (including WAITING-SUSPENDED state), the request is remembered, and later when it is attempted to resume the task in SUSPENDED state (including WAITING-SUSPENDED state), it is not resumed. The nesting of suspend requests is realized by having the task keep a suspend request nesting count. When the task is started, this count is cleared to 0.
4.2.1 tk_slp_tsk - Sleep Task

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>TMO</th>
<th>tmout</th>
<th>Timeout</th>
<th>Timeout (ms)</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

- **E_OK** Normal completion
- **E_PAR** Parameter error (tmout ≦ (-2))
- **E_RLWAI** Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI** Wait released due to disabling of wait
- **E_TMOUT** Polling failed or timeout
- **E_CTX** Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Changes the state of the invoking task from RUNNING state to sleep state (WAITING state for tk_wup_tsk). Note if the wakeup requests for the invoking task are queued, i.e., the wakeup request queuing count of the invoking task is 1 or more, the count is decremented by 1, and the execution is continued without moving the invoking task to the waiting state.

If tk_wup_tsk is issued for the invoking task before the time specified in tmout has elapsed, this system call completes normally. If timeout occurs before tk_wup_tsk is issued, the timeout error code E_TMOUT is returned. Specifying tmout = TMO_FEVR (= -1) means eternal wait. In this case, the task stays in waiting state until tk_wup_tsk is issued.

Additional Notes

Since tk_slp_tsk is a system call that puts the invoking task into the waiting state, tk_slp_tsk can never be nested. It is possible, however, for another task to issue tk_sus_tsk for a task that was put in the waiting state.
by `tk_slp_tsk`. In this case the task goes to WAITING-SUSPENDED state.

For simply delaying a task, `tk_dly_tsk` should be used rather than `tk_slp_tsk`.

The task sleep function is intended for use by applications and as a rule should not be used by middleware.

The reason is as follows.

Attempting to achieve synchronization by putting a task to sleep in two or more places would cause confusion, leading to mis-operation. For example, if sleep were used by both an application and middleware for synchronization, a wakeup request might arise in the application while middleware has a task sleeping. In such a situation, normal operation would not be possible in either the application or middleware.

In this manner, proper task synchronization is not possible if it is not clear where the wait for wakeup originated. Task sleep is often used as a simple means of task synchronization. Applications should be able to use it freely, which means as a rule it should not be used by middleware.
4.2.2  tk_slp_tsk_u - Sleep Task (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_slp_tsk_u(TMO_U tmout_u);

Parameter

<table>
<thead>
<tr>
<th>TMO_U</th>
<th>tmout_u</th>
<th>Timeout</th>
<th>Timeout (in microseconds)</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_PAR: Parameter error (tmout_u ≤ (-2))
- E_RLWAI: Waiting state released (tk_rel_wai received in waiting state)
- E_DISWAI: Wait released due to disabling of wait
- E_TMOUT: Polling failed or timeout
- E_CTX: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_USEC: Support of microsecond

Description

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_slp_tsk.
The specification of this system call is same as that of tk_slp_tsk, except that the parameter is replaced with tmout_u. For more details, see the description of tk_slp_tsk.
4.2.3  tk_wup_tsk - Wakeup Task

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_wup_tsk(ID tskid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>Invalid ID number (tskid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>Invalid object state (called for the invoking task or for a task in DORMANT state)</td>
</tr>
<tr>
<td>E_QOVR</td>
<td>Queuing or nesting overflow (too many queued wakeup requests in wupcnt )</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

TK_WAKEUP_MAXCNT Maximum queuing count of the task wakeup requests (>= 1)

Description

If the task specified in tskid has been put in WAITING state by tk_slp_tsk, this system call releases the WAITING state.

This system call cannot be called for the invoking task. If the invoking task is specified, error code E_OBJ is returned.

If the target task has not called tk_slp_tsk and is not in WAITING state, the wakeup request by tk_wup_tsk is queued. That is, the calling of tk_wup_tsk for the target task is recorded, then when tk_slp_tsk is called after that, the task does not go to WAITING state. This is what is meant by queuing of wakeup requests.

The queuing of wakeup requests works as follows. Each task keeps a wakeup request queuing count (wupcnt) in its TCB. Its initial value (when tk_sta_tsk is executed) is 0. When tk_wup_tsk is issued for a task not sleeping (not in WAITING state), the count is incremented by 1; but each time tk_slp_tsk is executed, the count is decremented by 1. When tk_slp_tsk is executed for a task whose wakeup queuing count is 0, the queuing count is not made negative but rather the task goes to WAITING state.
It is always possible to queue `tk_wup_tsk` at least one time (`wupcnt = 1`), but the maximum queuing count (\(\text{wupcnt}\)) is implementation-dependent and its maximum value is defined by serviced profile item, `TK_WAKEUP_MAXCNT`. In other words, issuing `tk_wup_tsk` once for a task not in WAITING state does not return an error, but whether an error is returned for the second or subsequent call of `tk_wup_tsk` is implementation-dependent.

When calling `tk_wup_tsk` causes `wupcnt` to exceed the allowed maximum value, error code `E_QOVR` is returned.
4.2.4 tk_can_wup - Cancel Wakeup Task

C Language Interface

```c
#include <tk/tkernel.h>

INT wupcnt = tk_can_wup(ID tskid);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>wupcnt</th>
<th>Wakeup Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Number of queued wakeup requests</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or Error Code</td>
</tr>
</tbody>
</table>

Error Code

- E_ID: Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the task specified in tskid does not exist)
- E_OBJ: Invalid object state (called for a task in DORMANT state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Passes in the return value the wakeup request queuing count (wupcnt) for the task specified in tskid, at the same time canceling all wakeup requests. That is, this system call clears the wakeup request queuing count (wupcnt) to 0 for the specified task.

The invoking task can be specified by setting tskid = TSK_SELF = 0. Note, however, that when tskid = TSK_SELF = 0 is specified in a system call issued from a task-independent portion, error code E_ID is returned.

Additional Notes

This system call can be used to determine whether the processing was completed within the allotted time when processing is performed that involves cyclic wakeup of a task. Before processing of a prior wakeup request is completed and tk_slp_tsk is called by the waken up task, the task monitoring this task calls tk_can_wup. If wupcnt in the return parameter is 1 or above, this means the previous wakeup request was not processed within the allotted time. Measure can then be taken accordingly to compensate for the delay.
4.2.5  tk_rel_wai - Release Wait

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_rel_wai(ID tskid);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID tskid</td>
<td>Task ID</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER ercd</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (*tskid* is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the task specified in *tskid* does not exist)
- **E_OBJ**: Invalid object state (called for a task not in WAITING state (including when called for the invoking task, or for a task in DORMANT state))

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

If the task specified in *tskid* is in some kind of waiting state (not including SUSPENDED state), forcibly releases that state.

To the task whose WAITING state was released by *tk_rel_wai*, the error code E_RLWAI is returned. At this time, the target task is guaranteed to be released from its wait state without the allocation of the waited resource (without the wait release conditions being met).

Wait release requests are not queued by *tk_rel_wai*. That is, if the task specified in *tskid* is already in WAITING state, the WAITING state is cleared; but if it is not in WAITING state when this system call is issued, error code E_OBJ is returned to the caller. Likewise, error code E_OBJ is returned when this system call is issued specifying the invoking task.

The *tk_rel_wai* system call does not release a SUSPENDED state. If *tk_rel_wai* is issued for a task in WAITING-SUSPENDED state, the task goes to SUSPENDED state. If it is necessary to release SUSPENDED state, the separate system call *tk_rsm_tsk* or *tk_frsm_tsk* is used.

The states of the target task when *tk_rel_wai* is called and the results of its execution in each state are shown in Table 4.3, “Target Task State and Execution Result (*tk_rel_wai)*”. 
### Target Task State and Execution Result (tk_rel_wai)

<table>
<thead>
<tr>
<th>Target Task State</th>
<th><code>tk_rel_wai</code> Return Value</th>
<th>(processing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run state (RUNNING or READY) (not for invoking task)</td>
<td>E_OBJ</td>
<td>No operation</td>
</tr>
<tr>
<td>Running state (RUNNING) (invoking task)</td>
<td>E_OBJ</td>
<td>No operation</td>
</tr>
<tr>
<td>Waiting state (WAITING)</td>
<td>E_OK</td>
<td>Wait released/release wait</td>
</tr>
<tr>
<td>Suspended state (SUSPENDED)</td>
<td>E_OBJ</td>
<td>No operation</td>
</tr>
<tr>
<td>Waiting-suspended state (WAITING-SUSPENDED)</td>
<td>E_OK</td>
<td>Goes to SUSPENDED state</td>
</tr>
<tr>
<td>Dormant state (DORMANT)</td>
<td>E_OBJ</td>
<td>No operation</td>
</tr>
<tr>
<td>Non-existent state (NON-EXISTENT)</td>
<td>E_NOEXS</td>
<td>No operation</td>
</tr>
</tbody>
</table>

### Additional Notes

A function similar to timeout can be realized by using an alarm handler or the like to issue this system call after a given task has been in WAITING state for a set time.

The main differences between `tk_rel_wai` and `tk_wup_tsk` are the following.

- Whereas `tk_wup_tsk` releases only WAITING state effected by `tk_slp_tsk`, `tk_rel_wai` releases also WAITING state caused by other factors (`tk_wai_flg`, `tk_wai_sem`, `tk_rcv_mbx`, `tk_get_mpl`, `tk_dly_tsk`, etc.).

- Seen from the task in WAITING state, release of the WAITING state by `tk_wup_tsk` returns a Normal completion (E_OK), whereas release by `tk_rel_wai` returns an error code (E_RLWAI).

- Wakeup requests by `tk_wup_tsk` are queued if `tk_slp_tsk` has not yet been executed. If `tk_rel_wai` is issued for a task not in WAITING state, error code E_OBJ is returned.
4.2.6  tk_sus_tsk - Suspend Task

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

| ID   | tskid | Task ID |

Return Parameter

| ER   | ercd  | Error Code |

Error Code

- E_OK: Normal completion
- E_ID: Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the task specified in tskid does not exist)
- E_OBJ: Invalid object state (called for the invoking task or for a task in DORMANT state)
- E_CTX: A task in RUNNING state was specified in dispatch disabled state
- E_QOVR: Queuing or nesting overflow (too many nested requests in suscnt)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

**TK_SUSPEND_MAXCNT**

Maximum nest count of the forced wait of tasks (>= 1)

Description

Puts the task specified in tskid in SUSPENDED state and interrupts execution by the task. SUSPENDED state is released by issuing system call tk_rsm_tsk or tk_frsm_tsk.

If tk_sus_tsk is called for a task already in WAITING state, the state goes to a combination of WAITING state and SUSPENDED state (WAITING-SUSPENDED state) after the execution of tk_sus_tsk. Thereafter when the task wait release conditions are met, the task goes to SUSPENDED state. If tk_rsm_tsk is issued for the task in WAITING-SUSPENDED state, the task state reverts to WAITING state. (See Figure 2.1, “Task State Transition Diagram”).

Since SUSPENDED state means task interruption by a system call issued by another task, this system call cannot be issued for the invoking task. If the invoking task is specified, error code E_OBJ is returned.

When this system call is issued from a task-independent portion, if a task in RUNNING state is specified while dispatching is disabled, error code E_CTX is returned.
If \texttt{tk\_sus\_tsk} is issued more than once for the same task, the task is put in nested SUSPENDED state. This is called nesting of suspend requests. In this case, the task reverts to its original state only when \texttt{tk\_rsm\_tsk} has been issued for the same number of times as \texttt{tk\_sus\_tsk} (\texttt{suscnt}). Accordingly, nesting of the pair of system calls \texttt{tk\_sus\_tsk} and \texttt{tk\_rsm\_tsk} is possible.

The nesting feature of suspend requests (issuing \texttt{tk\_sus\_tsk} two or more times for the same task) and upper limits on nesting count are implementation-dependent.

If \texttt{tk\_sus\_tsk} is issued multiple times in a system that does not allow suspend request nesting, or if the nesting count exceeds the allowed limit, error code E\_QOVR is returned.

**Additional Notes**

When a task is in WAITING state for resource acquisition (semaphore wait, etc.) and is also in SUSPENDED state, the resource allocation (semaphore allocation, etc.) takes place under the same conditions as when the task is not in SUSPENDED state. Resource allocation is not delayed by the SUSPENDED state, and there is no change whatsoever in the priority of resource allocation or release from WAITING state. In this way SUSPENDED state is in an orthogonal relation with other processing and task states.

In order to delay resource allocation to a task in SUSPENDED state (temporarily lowering its priority), the user can employ \texttt{tk\_sus\_tsk} and \texttt{tk\_rsm\_tsk} in combination with \texttt{tk\_chg\_pri}.

Task suspension is intended only for very limited uses closely related to the OS, such as breakpoint processing in a debugger. As a rule it should not be used in ordinary applications or in middleware. The reason is as follows.

Task suspension takes place regardless of the target task running state. If, for example, a task is put in SUSPENDED state while it is calling a middleware function, the task will be stopped in the course of middleware internal processing. In some cases middleware performs resource management or other mutual exclusion control. If a task stops inside middleware while it has resources allocated, other tasks may not be able to use that middleware. This situation can cause chain reactions, with other tasks stopping and leading to system-wide deadlock.

For this reason a task must not be stopped without knowing its status (what it is doing at the time), and ordinary tasks should not use the task suspension function.
4.2.7 tk_rsm_tsk - Resumes a task in a SUSPENDED state

C Language Interface

```
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Invalid ID number (tskid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>Invalid object state (the specified task is not in SUSPENDED state (including when this system call specifies the invoking task or a task in DORMANT state))</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Releases the SUSPENDED state of the task specified in tskid. If the target task was earlier put in SUSPENDED state by the tk_sus_tsk system call, this system call releases that SUSPENDED state and resumes the task execution.

When the target task is in a combined WAITING state and SUSPENDED state (WAITING-SUSPENDED state), executing tk_rsm_tsk releases only the SUSPENDED state, putting the task in WAITING state (see Figure 2.1, "Task State Transition Diagram").

This system call cannot be called for the invoking task. If the invoking task is specified, error code E_OBJ is returned.

Executing tk_rsm_tsk once clears only one nested suspend request (suscnt). If tk_sus_tsk was issued more than once for the target task (suscnt \( \geq 2 \)), the target task remains in SUSPENDED state even after tk_rsm_tsk is executed.
Additional Notes

After a task in RUNNING state or READY state is put in SUSPENDED state by \texttt{tk\_sus\_tsk} and then resumed by \texttt{tk\_rsm\_tsk} or \texttt{tk\_frsm\_tsk}, the task has the lowest precedence among tasks of the same priority. When, for example, the following system calls are executed for tasks A and B of the same priority, the result is as indicated below.

\begin{verbatim}
tk\_sta\_tsk (tskid=task\_A, stacd\_A);
tk\_sta\_tsk (tskid=task\_B, stacd\_B);
    /* By the rule of FCFS, precedence becomes task\_A → task\_B. */
tk\_sus\_tsk (tskid=task\_A);
tk\_rsm\_tsk (tskid=task\_A);
    /* In this case precedence becomes task\_B → task\_A. */
\end{verbatim}
4.2.8  tk_frsm_tsk - Force Resume Task

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_frsm_tsk(ID tskid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

- E_OK  Normal completion
- E_ID  Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS  Object does not exist (the task specified in tskid does not exist)
- E_OBJ  Invalid object state (the specified task is not in SUSPENDED state (including when this system call specifies the invoking task or a task in DORMANT state))

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Releases the SUSPENDED state of the task specified in tskid. If the target task was earlier put in SUSPENDED state by the tk_sus_tsk system call, this system call releases that SUSPENDED state and resumes the task execution.

When the target task is in a combined WAITING state and SUSPENDED state (WAITING-SUSPENDED state), executing tk_frsm_tsk releases only the SUSPENDED state, putting the task in WAITING state (see Figure 2.1, “Task State Transition Diagram”).

This system call cannot be called for the invoking task. If the invoking task is specified, error code E_OBJ is returned.

Executing tk_frsm_tsk once clears all the nested suspend requests (suscnt) (suscnt = 0). Therefore, all suspend requests are released (suscnt is cleared to 0) even if tk_sus_tsk was issued more than once (suscnt \( \geq 2 \)). The SUSPENDED state is always cleared, and unless the task was in the WAITING-SUSPENDED state, its execution resumes.
Additional Notes

After a task in RUNNING state or READY state is put in SUSPENDED state by `tk_sus_tsk` and then resumed by `tk_rsm_tsk` or `tk_frsm_tsk`, the task has the lowest precedence among tasks of the same priority.

When, for example, the following system calls are executed for tasks A and B of the same priority, the result is as indicated below.

```c
tk_sta_tsk (tskid=task_A, stacd_A);
tk_sta_tsk (tskid=task_B, stacd_B);
/* By the rule of FCFS, precedence becomes task_A → task_B. */
tk_sus_tsk (tskid=task_A);
tk_frsm_tsk (tskid=task_A);
/* In this case precedence becomes task_B → task_A. */
```
4.2.9  tk_dly_tsk - Delay Task

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_dly_tsk(RELTIM dlytim);

Parameter

RELTIM  dlytim         Delay Time         Delay time (ms)

Return Parameter

ER      ercd           Error Code          Error code

Error Code

E_OK     Normal completion
E_PAR    Parameter error (dlytim is invalid)
E_CTX    Context error (issued from task-independent portion, or in dispatch disabled state)
E_RLWAI  Waiting state released (tk_rel_wai received in waiting state)
E_DISWAI Wait released due to disabling of wait

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Temporarily stops execution of the invoking task and waits for time dlytim to elapse.

The state while the task waits for the delay time to elapse is a WAITING state and is subject to release by tk_rel_wai.

If the task issuing this system call goes to SUSPENDED state or WAITING-SUSPENDED state while it is waiting for the delay time to elapse, the elapsed time continues to be counted in the SUSPENDED state.

The time unit for dlytim (time unit) is the same as that for system time (= 1 ms).

Additional Notes

This system call differs from tk_slp_tsk in that normal completion, not an error code, is returned when the specified delay time elapses. Moreover, the wait is not released even if tk_wup_tsk is executed during the delay time. The only way to terminate tk_dly_tsk before the delay time elapses is by calling tk_ter_tsk or tk_rel_wai.
4.2.10  tk_dly_tsk_u - Delay Task (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_dly_tsk_u(RELTIM_U dlytim_u);

Parameter

| RELTIM_U | dlytim_u | Delay Time | Delay time (microseconds) |

Return Parameter

| ER | ercd | Error Code | Error code |

Error Code

- E_OK   Normal completion
- E_PAR  Parameter error (dlytim_u is invalid)
- E_CTX  Context error (issued from task-independent portion, or in dispatch disabled state)
- E_RLWAI Waiting state released (tk_rel_wai received in waiting state)
- E_DISWAI Wait released due to disabling of wait

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_USEC  Support of microsecond

Description

This system call takes 64-bit dlytim_u in microseconds instead of the parameter dlytim of tk_dly_tsk.

The specification of this system call is same as that of tk_dly_tsk, except that the parameter is replaced with dlytim_u. For more details, see the description of tk_dly_tsk.
4.2.11  tk_sig_tev - Signal Task Event

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_sig_tev(ID tskid, INT tskevt);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>tskevt</td>
<td>Task Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Task ID number (1 to 8)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th></th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td></td>
</tr>
<tr>
<td>E_ID</td>
<td>Invalid ID number (tskid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>Invalid object state (called for a task in DORMANT state)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (tskevt is invalid)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_TASKEVENT  Support of task event

Description

Sends the task event specified in tskevt to the task specified in tskid.

There are eight task event types stored for each task, specified by numbers 1 to 8.

The task event send count is not saved, only whether the event occurs or not.

The invoking task can be specified by setting tskid = TSK_SELF = 0. Note, however, that when tskid = TSK_SELF = 0 is specified in a system call issued from a task-independent portion, error code E_ID is returned.

Additional Notes

The task event function is used for task synchronization much like tk_slp_tsk and tk_wup_tsk, but differs from the use of these system calls in the following ways.
• The wakeup request (task event) count is not kept.
• Wakeup requests can be classified by the eight event types.

Using the same event type for synchronization in two or more places in the same task would cause confusion. Event type allocation should be clearly defined.

The task event function is intended for use in middleware, and as a rule should not be used in ordinary applications. Use of `tk_slp_tsk` and `tk_wup_tsk` is recommended for applications.
4.2.12  tk_wai_tev - Wait Task Event

C Language Interface

#include <tk/tkernel.h>

INT tevptn = tk_wai_tev(UINT waiptn, TMO tmout);

Parameter

<table>
<thead>
<tr>
<th>UINT</th>
<th>waiptn</th>
<th>Wait Event Pattern</th>
<th>Task event pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
<td>Timeout (ms)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>tevptn</th>
<th>Task Event Pattern</th>
<th>Task event status when wait released</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>or Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

- **E_PAR** Parameter error (waiptn or tmout is invalid)
- **E_RLWAI** Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI** Wait released due to disabling of wait
- **E_TMOUT** Polling failed or timeout
- **E_CTX** Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_TASKEVENT** Support of task event

Description

Waits for the occurrence of one of the task events specified in waiptn. When the wait is released by a task event, the task events specified in waiptn are cleared (raised task event &= `waiptn`). The task event status when the wait was released (the state before clearing) is passed in the return code (tevptn).

The parameters waiptn and tevptn consist of logical OR values of the bits for each task event in the form 1 << (task event number -1).

A maximum wait time (timeout) can be set in tmout. The time unit for tmout is the same as that for system time (= 1 ms). If the tmout time elapses before the wait release condition is met (tk_sig_tev is not executed), the system call terminates, returning timeout error code E_TMOUT.

When TMO_POL=0 is set in tmout, this means 0 was specified as the timeout value, and E_TMOUT is returned.
without entering WAITING state even if no task event occurs. When \texttt{TMO\_FEVR\text{=}-1} is set in \texttt{tmout}, this means infinity was specified as the timeout value, and the task continues to wait for a task event without timing out.
4.2.13  tk_wai_tev_u - Wait Task Event (Microseconds)

C Language Interface

#include <tk/tkernel.h>

INT tevptn = tk_wai_tev_u(UINT waiptn, TMO_U tmout_u);

Parameter

<table>
<thead>
<tr>
<th>UINT</th>
<th>waiptn</th>
<th>Wait Event Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Task event pattern
Timeout (in microseconds)

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>tevptn</th>
<th>Task Event Pattern</th>
</tr>
</thead>
</table>

or  Error Code

Task event status when wait released
Error code

Error Code

<table>
<thead>
<tr>
<th>E_PAR</th>
<th>Parameter error (waiptn or tmout_u is invalid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_RLWAI</td>
<td>Waiting state released (tk_rel_wai received in waiting state)</td>
</tr>
<tr>
<td>E_DISWAI</td>
<td>Wait released due to disabling of wait</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>Polling failed or timeout</td>
</tr>
<tr>
<td>E_CTX</td>
<td>Context error (issued from task-independent portion, or in dispatch disabled state)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

<table>
<thead>
<tr>
<th>TK_SUPPORT_TASKEVENT</th>
<th>Support of task event</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_USEC</td>
<td>Support of microsecond</td>
</tr>
</tbody>
</table>

Description

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_wai_tev.
The specification of this system call is same as that of tk_wai_tev, except that the parameter is replaced with tmout_u. For more details, see the description of tk_wai_tev.
4.2.14  tk_dis_wai - Disable Task Wait

C Language Interface

#include <tk/tkernel.h>

INT tskwait = tk_dis_wai(ID tskid, UW waitmask);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW</td>
<td>waitmask</td>
<td>Wait Mask</td>
</tr>
</tbody>
</table>

Task ID

Task ID disabled setting

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>tskwait</th>
<th>Task Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_ID</th>
<th>Invalid ID number (tskid is invalid or cannot be used)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (waitmask is invalid)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DISWAI  Support of disabling of wait of a task

Description

Disables waits for the wait factors set in waitmask by the task specified in tskid. If the task is already waiting for a factor specified in waitmask, that wait is released.

waitmask is specified as the logical OR of any combination of the following wait factors.

```
#define TTW_SLP 0x00000001 /* Wait caused by sleep */
#define TTW_DLY 0x00000002 /* Wait for task delay */
#define TTW_SEM 0x00000004 /* Wait for semaphore */
#define TTW_FLG 0x00000008 /* Wait for event flag */
#define TTW_MBX 0x00000040 /* Wait for mailbox */
#define TTW_MTX 0x00000080 /* Wait for mutex */
#define TTW_SMBF 0x00000100 /* Wait for message buffer send */
#define TTW_RMBF 0x00000200 /* Wait for message buffer receive */
#define TTW_CAL 0x00000400 /* (reserved) */
```
TTX_SVC is a special value disabling not the task wait but the calling of an extended SVC. If TTX_SVC has been set when a task attempts to call an extended SVC, E_DISWAI is returned without calling the extended SVC. This value does not have the effect of terminating an already called extended SVC.

The return value (tskwait) includes the waiting state of a task as a pattern of concatenated bits (bit width of INT data type - 1) after the waiting states are disabled by tk_dis_wai. If bit width of INT data type is 32, then this value is the same as the value tskwait returned by tk_ref_tsk. Information concerning TTX_SVC is not returned in tskwait. A tskwait value of 0 means the task has not entered WAITING state (or the wait was released). If tskwait is not 0, this means the task is in WAITING state for a cause other than those disabled in waitmask. If the bit width of INT data type is less than 32, the information represented by upper bits that will not fit into INT data are not returned. Hence, in this case, even if tskwait is zero, there is a possibility that the task is waiting for a cause that is not specified in waitmask.

When a task wait is cleared by tk_dis_wai or the task is prevented from entering WAITING state after this system call has taken effect, E_DISWAI is returned.

When a system call for which there is the possibility of entering the WAITING state is invoked during wait-disabled state, E_DISWAI is returned even if the processing could be performed without waiting. For example, when message buffer space is available and it is possible to send message without entering the WAITING state, and if a message is sent to message buffer (tk_snd_mbf is called), the message is not sent and E_DISWAI is returned.

Disabling of wait that is set during an extended SVC will be cleared automatically upon return from the extended SVC to its caller. It is automatically cleared also when an extended SVC is called, reverting to the original setting upon return from the extended SVC.

Disabling of wait that is set is cleared also when the task reverts to DORMANT state. The setting made while a task is in DORMANT state, however, is valid and the disabling of wait is applied the next time the task is started.

In the case of semaphores and most other objects, TA_NODISWAI can be specified when the object is created. An object created with TA_NODISWAI specified cannot have wait disabled, and rejects any attempt to disable wait by tk_dis_wai.

The invoking task can be specified by setting tskid = TSK_SELF = 0. Note, however, that when tskid = TSK_SELF = 0 is specified in a system call issued from a task-independent portion, error code E_ID is returned.

Additional Notes

The function to disable wait is provided for aborting the execution of an extended SVC handler in midway, but it is not restricted to that purpose only.

Porting Guideline

Note that the data type of return value of tk_dis_wai (tskwait), is of type INT, and its value range is implementation-dependent, so care must be taken. For example, you can not receive information concerning waiting task...
events under an implementation on 16-bits CPU. If it is desired to obtain task wait status without regard to the CPU bit width under \( \mu \text{T-Kernel} \), it is necessary to reference \texttt{tskwait} by invoking \texttt{tk_ref_tsk}. On the other hand, under T-Kernel, INT is defined to be 32 bits or wider, the return value of \texttt{tk_dis_wai} can show all the details of the wait status of a task.
4.2.15  tk_ena_wai - Enable Task Wait

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ena_wai(ID tskid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_ID: Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the task specified in tskid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_DISWAI**: Support of disabling of wait of a task

Description

Releases all disabling of waits set by tk_dis_wai for the task specified in tskid.

The invoking task can be specified by setting tskid = TSK_SELF = 0. Note, however, that when tskid = TSK_SELF = 0 is specified in a system call issued from a task-independent portion, error code E_ID is returned.
4.3 Task Exception Handling Functions

Task exception handling functions handle exception events that are raised for a task in the context of that task.

The task exception handler is started when all the following processing has taken place:

1. Register task exception handler by tk_def
2. Enable task exception by tk_ena
3. Raise task exception by tk_ras

A task exception handler is executed as a part of the task where the task exception occurred, in the context of that task and at the protection level specified when the task was created. The task states in a task exception handler, except for those states concerning task exceptions, are the same as the states when running an ordinary task portion; and the same set of system calls are available.

A task exception handler can be started only when the target task is running in a task portion. If the task is running in any other portion when a task exception is raised, the task exception handler is started only after the control returns to the task portion. If a quasi-task portion (extended SVC) is being executed when a task exception is raised, the processing of the extended SVC handler is aborted and the control returns to the task portion. If it is needed to abort the processing of the extended SVC handler (called "break processing" for the extended SVC handler), it is performed before the control returns to the task portion where the extended SVC handler is called. "Break processing" is executed by the break function of Subsystem Management Functions.

Requested task exceptions are cleared when the task exception handler is called (when the task exception handler starts running).

Task exception is identified by a task exception code: from 1 to (bit width of UINT data type - 1). For example, if UINT is 16 bits, a number from 0 to 15 can be used as task exception code. 0 corresponds to the highest priority, and (bit width of UINT data type - 1) corresponds to the lowest priority. Task exception code 0 is handled differently from the others, as explained below.

Processing of task exception code from 1 to (bit width of UINT data type) - 1:

- These task exception handlers cannot be executed by nesting them. A task exception (other than task exception code 0) raised while a task exception handler is running will be made pending.
- On return from a task exception handler, the task resumes from the point where processing was interrupted by the exception.
- It is also possible to use longjmp() or the like to jump to any point in the task without returning from the task exception handler.

Task exception code 0:

- This exception can be executed by nesting even while a task exception handler is executing for an exception of task exception code from 1 to (bit width of UINT data type - 1). Nesting does not take place when the task exception handler of task exception code 0 is executed.
- A task exception handler runs after setting the user stack pointer to the initial setting when the task was started. In a system without a separate user stack and system stack, however, the stack pointer is not reset to its initial setting.
- A task exception code 0 handler does not return to task processing. The task must be terminated by calling tk_ext_tsk or tk_exd_tsk.

Porting Guideline

Be warned that the available number of task exception codes is now dependent on the bit width of UINT data type. For example, task exception code can take the value from 0 to 15 in 16-bit environment.
4.3.1 tk_def_tex - Define Task Exception Handler

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_def_tex(ID tskid, CONST T_DTEX *pk_dtex);
```

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_DTEX*</td>
<td>pk_dtex</td>
<td>Packet to Define Task Exception</td>
</tr>
</tbody>
</table>

**pk_dtex** Detail:

- ATR texatr Task Exception Attribute Task exception handler attributes
- FP texhdr Task Exception Handler Task exception handler address
  (Other implementation-dependent parameters may be added beyond this point.)

**Return Parameter**

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

**Error Code**

- E_OK Normal completion
- E_NOMEM Insufficient memory (memory for control block cannot be allocated)
- E_ID Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS Object does not exist (the task specified in tskid does not exist)
- E_OBJ Invalid object state (the task specified in tskid runs at protection level 0 (TA_RNG0))
- E_RSATR Reserved attribute (texatr is invalid or cannot be used)
- E_PAR Parameter error (pk_dtex is invalid or cannot be used)

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_TASKEXCEPTION** Support of task exception handling functions

**Description**

Defines a task exception handler for the task specified in tskid. Only one task exception handler can be defined per task; if one is already defined, the last-defined handler is valid. Setting pk_dtex = NULL cancels a
definition.

Defining or canceling a task exception handler clears pending task exception requests and disables all task exceptions.

texatr indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The texatr system attributes are not assigned in the present version of T-Kernel specification, and system attributes are not used.

A task exception handler takes the following form.

```c
void texhdr( INT texcd )
{}
```  /* Task exception handling */
/* Task exception handler termination */
if ( texcd == 0 ) {
tk_ext_tsk() or tk_exd_tsk();
} else {
tk_end_tex();
return or longjmp();
}
}

A task exception handler behaves like a TA_ASM attribute object and cannot be called via a high-level language support routine. The entry part of the task exception handler must be written in assembly language. The kernel vendor must provide the assembly language source code of the entry routine for calling the above C language task exception handler. That is, source code equivalent to a high-level language support routine must be provided.

A task set to protection level TA_RNG0 when it is created cannot use task exceptions.

**Additional Notes**

At the time a task is created, no task exception handler is defined and task exceptions are disabled.

When a task reverts to DORMANT state, the task exception handler definition is canceled and task exceptions are disabled. Pending task exceptions are cleared. It is possible, however, to define a task exception handler for a task in DORMANT state.

Task exceptions are software interrupts raised by tk_ras_tex, with no direct relation to CPU exceptions.
4.3.2 tk_ena_tex - Enable Task Exception

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ena_tex(ID tskid, UINT texptn);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>texptn</td>
<td>Task Exception Pattern</td>
<td>Task exception pattern</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>Invalid ID number (tskid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist or no task exception handler is defined)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (texptn is invalid or cannot be used)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_TASKEXCEPTION Support of task exception handling functions

Description

Enables task exceptions for the task specified in tskid.

The parameter texptn is a logical OR bit array representing task exception codes in the form 1 << task exception code.

tk_ena_tex enables the task exceptions specified in texptn. If the current exception enabled status is texmask, it changes as follows.

enable: texmask |= texptn

If all the bits of texptn are cleared to 0, no operation is made to texmask. No error will result in this case.

Task exceptions cannot be enabled for a task with no task exception handler defined.

This system call can be called to tasks in DORMANT state.
Porting Guideline

Be warned that the available number of task exception codes is now dependent on the bit width of UINT data type. For example, task exception code can take the value from 0 to 15 in 16-bit environment.
4.3.3  tk_dis_tex - Disable Task Exception

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_dis_tex(ID tskid, UINT texptn);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>texptn</td>
<td>Task Exception Pattern</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Invalid ID number (tskid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist or no task exception handler is defined)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (texptn is invalid or cannot be used)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_TASKEXCEPTION Support of task exception handling functions

Description

Disables task exceptions for the task specified in tskid.

The parameter texptn is a logical OR bit array representing task exception codes in the form 1 << task exception code.

tk_dis_tex disables the task exceptions specified in texptn. If the current exception enabled status is texmask, it changes as follows.

disable: texmask &= ~texptn

If all the bits of texptn are cleared to 0, no operation is made to texmask. No error will result in either case.

A disabled task exception is ignored, and is not made pending. If exceptions are disabled for a task while there are pending task exceptions, the pending task exception requests are discarded (their pending status is cleared).
This system call can be called to tasks in DORMANT state.

**Porting Guideline**

Be warned that the available number of task exception codes is now dependent on the bit width of UINT data type. For example, task exception code can take the value from 0 to 15 in 16-bit environment.
4.3.4  tk_ras_tex - Raise Task Exception

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ras_tex(ID tskid, INT texcd);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID tskid</td>
<td>Task ID</td>
</tr>
<tr>
<td>INT texcd</td>
<td>Task Exception Code</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Return Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER ercd</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Invalid ID number (tskid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist or no task exception handler is defined)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>Invalid object state (the task specified in tskid is in DORMANT state)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (texcd is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_CTX</td>
<td>Context error (issued from task-independent portion, or in dispatch disabled state)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Valid Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_TASKEXCEPTION  Support of task exception handling functions

Description

Raises the task exception specified in texcd for the task specified in tskid. If the task specified in tskid disables the task exception specified in texcd, the raised task exception is ignored, and is not made pending. In this case, E_OK is returned to this system call.

If a task exception handler is already running in the task specified in tskid, the newly raised task exception is made pending. If an exception is pending, the break processing (break function) for the extended SVC handler is not performed even if the target task is executing an extended SVC.

In the case of texcd = 0, however, exceptions are not made pending even if the target task is executing an exception handler. If the target task is running a task exception handler for an exception of task exception codes from 1 to (bit width of UINT data type - 1), the task exception is accepted; and if an extended SVC is
executing, the break processing (break function) for the extended SVC handler is performed. If the target task is running a task exception handler for an exception of task exception code 0, task exceptions are ignored. The invoking task can be specified by setting \texttt{tskid = TSK_SELF = 0}.

If this system call is issued from a task-independent portion, error code E_CTX is returned.

**Additional Notes**

If the target task is executing an extended SVC, the break processing (break function) corresponding to the extended SVC runs as a quasi-task portion of the task that issued \texttt{tk_ras_tex}. That is, it is executed in the context of the quasi-task portion whose requesting task is the task that issued \texttt{tk_ras_tex}.

In such a case \texttt{tk_ras_tex} does not return control until the break processing ends. For this reason, the specification does not allow \texttt{tk_ras_tex} to be issued from a task-independent portion.

Task exceptions raised in the task that called \texttt{tk_ras_tex} while the break processing is running are held until the break processing (break function) ends.

**Porting Guideline**

Be warned that the available number of task exception codes is now dependent on the bit width of UINT data type. For example, task exception code can take the value from 0 to 15 in 16-bit environment.
4.3.5   tk_end_tex - end task exception handler

C Language Interface

#include <tk/tkernel.h>

INT texcd = tk_end_tex(BOOL enatex);

Parameter

BOOL enatex Enable Task Exception Task exception handler calling enabled flag

Return Parameter

INT texcd Task Exception Code Raised exception code or Error Code Error code

Error Code

E_CTX Context error (called for other than a task exception handler or task exception code 0 (detection is implementation-dependent))

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_TASKEXCEPTION Support of task exception handling functions

Description

Ends a task exception handler and enables the new task exception handler. If there are pending task exceptions, the highest-priority task exception code among them is passed in the return code. If there are no pending task exceptions, 0 is returned.

If enatex = FALSE and there are pending task exception, calling the new task exception handler is not allowed. In this case, the exception handler specified in return code texcd is in running state upon return from tk_end_tex. If there are no pending task exceptions, calling the new task exception handler is allowed.

If enatex = TRUE, calling the new task exception handler is allowed regardless of whether there are pending task exceptions. Even if there are pending task exceptions, the task exception handler is in terminated status.

There is no way of ending a task exception handler other than by calling tk_end_tex. A task exception handler continues executing from the time it is started until tk_end_tex is called. Even if return is made from a task exception handler without calling tk_end_tex, the task exception handler will still be running at the point of
return. Similarly, even if `longjmp` is used to get out of a task exception handler without calling `tk_end_tex`, the task exception handler will still be running at the jump destination.

Calling `tk_end_tex` while task exceptions are pending results in a new task exception being accepted. At this time even when `tk_end_tex` is called from an extended SVC handler, a break processing (break function) cannot be called for that extended SVC handler. If extended SVC calls are nested, then when the extended SVC nesting goes down one level, the break processing (break function) corresponding to the extended SVC return destination can be called. Calling of a task exception handler takes place upon return to the task portion.

The `tk_end_tex` system call cannot be issued in the case of task exception code 0 since the task exception handler cannot be ended in this case. The task must be terminated by calling `tk_ext_tsk` or `tk_exd_tsk`. If `tk_end_tex` is called while processing the task exception code 0, the behavior is undefined (implementation-dependent).

This system call cannot be issued from other than a task exception handler. The behavior when it is called from other than a task exception handler is undefined (implementation-dependent).

**Additional Notes**

When `tk_end_tex` (TRUE) is called and there are pending task exceptions, another task exception handler call is made immediately following `tk_end_tex`. In this case, a task exception handler is called without restoring the stack, giving rise to possible stack overflow.

Ordinarily `tk_end_tex` (FALSE) can be used, and processing looped as illustrated below while there are task exceptions pending.

```c
void texhdr( INT texcd )
{
    if ( texcd == 0 ){
        /*
         *  Processing for task exception 0
         */
        tk_exd_tsk();
    }
    do {
        /*
         *  Processing of task exception: from 1 to (bit width of UINT data type) - 1
         */
    } while ( (texcd = tk_end_tex(FALSE)) > 0 );
}
```

Strictly speaking, if a task exception were to occur during the interval after 0 is returned by `tk_end_tex` ending the loop and before exit from `texhdr`, the possibility exists of reentering `texhdr` without restoring the stack. Since task exceptions are software driven, however, ordinarily they do not occur independently of executing tasks; so in practice this is not a problem.

**Porting Guideline**

Be warned that the available number of task exception codes is now dependent on the bit width of UINT data type. For example, task exception code can take the value from 0 to 15 in 16-bit environment.
4.3.6 tk_ref_tex - Reference Task Exception Status

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_ref_tex(ID tskid, T_RTEX *pk_rtex);
```

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RTEX*</td>
<td>pk_rtex</td>
<td>Packet to Return Task Exception Status</td>
</tr>
</tbody>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

**pk_rtex Detail:**

<table>
<thead>
<tr>
<th>UINT</th>
<th>pendtex</th>
<th>Pending Task Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>texmask</td>
<td>Task Exception Mask</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

**Error Code**

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (tskid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the task specified in tskid does not exist)
- **E_PAR**: Parameter error (invalid pk_rtex)

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_TASKEXCEPTION**: Support of task exception handling functions

**Description**

Gets the status of task exceptions for the task specified in tskid.

`pendtex` indicates the currently pending task exceptions. A raised task exception is indicated in `pendtex` from the time the task exception is raised until its task exception handler is called.

`texmask` indicates allowed task exceptions.
Both \texttt{pendtx} and \texttt{texmask} are bit arrays of the form $1 << \text{task exception code}$.

The invoking task can be specified by setting $\texttt{tsk id} = \texttt{TSK\_SELF} = 0$. Note, however, that when $\texttt{tsk id} = \texttt{TSK\_SELF} = 0$ is specified in a system call issued from a task-independent portion, error code \texttt{E\_ID} is returned.
4.4 Synchronization and Communication Functions

Synchronization and communication functions use objects independent of tasks used to synchronize tasks and achieve communication between tasks. The objects available for these purposes include semaphores, event flags, and mailboxes.
4.4.1 Semaphore

A semaphore is an object indicating the availability of a resource and its quantity as a numerical value. A semaphore is used to realize mutual exclusion control and synchronization when using a resource. Functions are provided for creating and deleting a semaphore, acquiring and returning resources corresponding to semaphores, and referencing semaphore status. A semaphore is an object identified by an ID number. The ID number for the semaphore is called a semaphore ID.

A semaphore contains a resource count (semaphore resource count) indicating whether the corresponding resource exists and in what quantity, and a queue of tasks waiting to acquire the resource. When a task (the task making event notification) returns m resources, it increments the semaphore resource count by m. When a task (the task waiting for an event) acquires n resources, it decreases the semaphore resource count by n. If the number of semaphore resources is insufficient (i.e., further reducing the semaphore resource count would cause it to be negative), a task attempting to acquire resources goes into WAITING state until the next time resources are returned. A task waiting for semaphore resources is put in the semaphore queue.

To prevent too many resources from being returned to a semaphore, a maximum value of semaphore resource count can be set for each semaphore. Error is reported if it is attempted to return resources to a semaphore that would cause this maximum count to be exceeded.
4.4.1.1  tk_cre_sem - Create Semaphore

C Language Interface

```c
#include <tk/tkernel.h>

ID semid = tk_cre_sem(CONST T_CSEM *pk_csem);
```

Parameter

<table>
<thead>
<tr>
<th>CONST T_CSEM* pk_csem</th>
<th>Packet to Create Semaphore</th>
</tr>
</thead>
</table>

pk_csem Detail:

<table>
<thead>
<tr>
<th>void* exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR sematr</td>
<td>Semaphore Attribute</td>
<td>Semaphore attribute</td>
</tr>
<tr>
<td>INT isemcnt</td>
<td>Initial Semaphore Count</td>
<td>Initial semaphore resource count</td>
</tr>
<tr>
<td>INT maxsem</td>
<td>Maximum Semaphore Count</td>
<td>Maximum semaphore resource count</td>
</tr>
<tr>
<td>UB dsname[8]</td>
<td>DS Object name</td>
<td>DS object name</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>ID semid</th>
<th>Semaphore ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>or Error Code</td>
<td>Semaphore ID Error code</td>
</tr>
</tbody>
</table>

Error Code

- **E_NOMEM** Insufficient memory (memory for control block cannot be allocated)
- **E_LIMIT** Semaphore count exceeds the system limit
- **E_RSATR** Reserved attribute (`sematr` is invalid or cannot be used)
- **E_PAR** Parameter error (`pk_csem` is invalid, or `isemcnt` or `maxsem` is negative or invalid)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

- **TK_SUPPORT_DISWAI** Support for specifying `TA_NODISWAI` (reject request to disable wait) to semaphore attribute
- **TK_SUPPORT_DSNAME** Support for specifying `TA_DSNAME` for semaphore attribute
- **TK_SEMAPHORE_MAXCNT** Upper limit of maximum number of semaphore resource count (`>= 32767`)
Description

Creates a semaphore, assigning a semaphore ID to it. This system call allocates a control block to the created semaphore and sets its initial value of semaphore resource count to $isemcnt$, and its maximum (upper limit) to $maxsem$. Note that the lowest number that can be specified to $maxsem$ shall be 32767. Whether a number larger than 32767 can be set is implementation-dependent.

$exinf$ can be used freely by the user to set miscellaneous information about the created semaphore. The information set in this parameter can be referenced by $tk\_ref\_sem$. If a larger area is needed for indicating user information, or if the information may need to be changed after the message buffer is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in $exinf$. The kernel pays no attention to the contents of $exinf$.

$sematr$ indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of $sematr$ is as follows.

$$sematr := (TA\_TFIFO || TA\_TPRI) | (TA\_FIRST || TA\_CNT) | [TA\_DSNAME] | [TA\_NODISWAI]$$

- **TA\_TFIFO**: Tasks are queued in FIFO order
- **TA\_TPRI**: Tasks are queued in priority order
- **TA\_FIRST**: The first task in the queue has precedence
- **TA\_CNT**: Tasks with fewer requests have precedence
- **TA\_DSNAME**: Specifies DS object name
- **TA\_NODISWAI**: Disabling of wait by $tk\_dis\_wai$ is prohibited

The queuing order of tasks waiting for a semaphore can be specified in **TA\_TFIFO** or **TA\_TPRI**. If the attribute is **TA\_TFIFO**, tasks are ordered by FIFO, whereas **TA\_TPRI** specifies queuing of tasks in order of their priority setting.

**TA\_FIRST** and **TA\_CNT** specify precedence of resource acquisition. **TA\_FIRST** and **TA\_CNT** do not change the order of the queue, which is determined by **TA\_TFIFO** and **TA\_TPRI**.

When **TA\_FIRST** is specified, resources are allocated starting from the first task in the queue regardless of requested semaphore resource count. As long as the first task in the queue cannot obtain the requested number of resources, tasks behind it in the queue are prevented from obtaining resources.

**TA\_CNT** means resources are assigned based on the order in which tasks are able to obtain the requested semaphore resource count. The requested semaphore resource counts are checked starting from the first task in the queue, and tasks to which their requested counts can be allocated receive resources. This is not the same as allocating in order of fewest requests.

When **TA\_DSNAME** is specified, $dsname$ is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, $td\_ref\_dsname$ and $td\_set\_dsname$. For more details, see the description of $td\_ref\_dsname$ and $td\_set\_dsname$. If **TA\_DSNAME** is not specified, $dsname$ is ignored. Then $td\_ref\_dsname$ and $td\_set\_dsname$ return E_OBJ error.

```c
#define TA\_TFIFO 0x00000000 /* manage queue by FIFO */
#define TA\_TPRI 0x00000001 /* manage queue by priority */
#define TA\_FIRST 0x00000000 /* first task in queue has precedence */
#define TA\_CNT 0x00000002 /* tasks with fewer requests have precedence */
#define TA\_DSNAME 0x00000040 /* DS object name */
#define TA\_NODISWAI 0x00000080 /* reject request to disable wait */
```
4.4.1.2  tk_del_sem - Delete Semaphore

C Language Interface
#include <tk/tkernel.h>

ER ercd = tk_del_sem(ID semid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>semid</th>
<th>Semaphore ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes the semaphore specified in semid.
The semaphore ID and control block area are released as a result of this system call.
This system call completes normally even if there is a task waiting for condition fulfillment on the semaphore, but error code E_DLT is returned to the task in WAITING state.
4.4.1.3  

**tk_sig_sem - Signal Semaphore**

**C Language Interface**

```c
#include <tk/tkernel.h>

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

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>semid</th>
<th>Semaphore ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>cnt</td>
<td>Count</td>
</tr>
</tbody>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

**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 (semaphore resource count semcnt over limit)
- **E_PAR**: Parameter error (cnt ≤ 0)

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

None.

**Description**

Returns to the semaphore specified in semid the number of resources indicated in cnt. If there is a task waiting for the semaphore, the requested semaphore resource count is checked and resources allocated if possible. A task allocated resources goes to READY state. In some conditions more than one task may be allocated resources and put in READY state.

If the semaphore resource count increases to the point where the maximum semaphore resource count (maxsem) would be exceeded by the return of more resources, error code E_QOVR is returned. In this case no resources are returned and the semaphore resource count (semcnt) does not change.

**Additional Notes**

Error is not returned even if semcnt goes over the initial semaphore resource count (isemcnt). When semaphores are used not for mutual exclusion control but for synchronization (like tk_wup_tsk and tk_slp_tsk), the semaphore resource count (semcnt) will sometimes go over the initial setting (isemcnt). The semaphore function can be used for mutual exclusion control by setting isemcnt and the maximum semaphore resource count (maxsem) to the same value and checking for the error that is returned when the count increases.
4.4.1.4 tk_wai_sem - Wait on Semaphore

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>semid</th>
<th>Semaphore ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>cnt</td>
<td>Count</td>
</tr>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

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 \( \leq -2 \), cnt \( \leq 0 \))
- **E_DLT**: The object being waited for was deleted (the specified semaphore was deleted while waiting)
- **E_RLWAI**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Obtains from the semaphore specified in semid the number of resources indicated in cnt. If the requested resources can be allocated, the task issuing this system call does not enter WAITING state but continues executing. In this case the semaphore resource count (semcnt) is decreased by the size of cnt. If the resources are not available, the task issuing this system call enters WAITING state, and is put in the queue of tasks waiting for the semaphore. The semaphore resource count (semcnt) for this semaphore does not change in this case.

A maximum wait time (timeout) can be set in tmout. The time unit for tmout is the same as that for system time (= 1 ms). If the tmout time elapses before the wait release condition is met (tk_sig_sem is not executed), the system call terminates, returning timeout error code E_TMOUT.
When $\text{TMO\_POL}=0$ is set in $\text{tmout}$, this means 0 was specified as the timeout value, and E\_TMOUT is returned without entering WAITING state even if no resources are acquired. When $\text{TMO\_FEVR}=(-1)$ is set in $\text{tmout}$, this means infinity was specified as the timeout value, and the task continues to wait for resource acquisition without timing out.
4.4.1.5  tk_wai_sem_u - Wait on Semaphore (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_wai_sem_u(ID semid, INT cnt, TMO_U tmout_u);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>semid</th>
<th>Semaphore ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>cnt</td>
<td>Count</td>
</tr>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Semaphore ID Semaphore ID

Count Resource request count

Timeout Timeout (in microseconds)

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

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_u ≦ (-2), cnt ≦ 0)
E_DLT The object being waited for was deleted (the specified semaphore was deleted while waiting)
E_RLWAI Waiting state released (tk_rel_wai received in waiting state)
E_DISWAI Wait released due to disabling of wait
E_TMOUT Polling failed or timeout
E_CTX Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_USEC Support of microsecond

Description

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_wai_sem.

The specification of this system call is same as that of tk_wai_sem, except that the parameter is replaced with tmout_u. For more details, see the description of tk_wai_sem.
4.4.1.6 tk_ref_sem - Reference Semaphore Status

C Language Interface

#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>semid</th>
<th>Semaphore ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RSEM*</td>
<td>pk_rsem</td>
<td>Packet to Return Semaphore Status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

pk_rsem Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Waiting Task ID</td>
</tr>
<tr>
<td>INT</td>
<td>semcnt</td>
<td>Semaphore Count</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

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 (invalid pk_rsem)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

References the status of the semaphore specified in semid, passing in the return parameters the current semaphore resource count (semcnt), the waiting task ID (wtsk), and extended information (exinf).

wtsk indicates the ID of a task waiting for the semaphore. If there are two or more such tasks, the ID of the task at the head of the queue is returned. If there are no waiting tasks, wtsk = 0 is returned.

If the specified semaphore does not exist, error code E_NOEXS is returned.
4.4.2 Event Flag

An event flag is an object used for synchronization, consisting of a pattern of bits used as flags to indicate the existence of the corresponding events. Functions are provided for creating and deleting an event flag, for event flag setting and clearing, event flag waiting, and event flag status reference. An event flag is an object identified by an ID number. The ID number for the event flag is called an event flag ID.

In addition to the bit pattern indicating the existence of corresponding events, an event flag has a queue of tasks waiting for the event flag. The event flag bit pattern is sometimes called simply event flag. The event notifier sets or clears the specified bits of the event flag. A task can be made to wait for all or some of the event flag bits to be set. A task waiting for an event flag is put in the queue of that event flag.
4.4.2.1  tk_cre_flg - Create Event Flag

C Language Interface

```c
#include <tk/tkernel.h>

ID flgid = tk_cre_flg(CONST T_CFLG *pk_cflg);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CONST T_CFLG* pk_cflg</code></td>
<td>Packet to Create EventFlag Event flag creation information</td>
</tr>
</tbody>
</table>

**pk_cflg** Detail:

- `void* exinf` Extended Information Extended information
- `ATR flgatr` EventFlag Attribute Event flag attribute
- `UINT iflgptn` Initial EventFlag Pattern Event flag initial value
- `UB dsname[8]` DS Object name DS object name

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ID flgid</code></td>
<td>EventFlag ID Event flag ID</td>
</tr>
<tr>
<td>or</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

- **E_NOMEM** Insufficient memory (memory for control block cannot be allocated)
- **E_LIMIT** Number of event flags exceeds the system limit
- **E_RSATR** Reserved attribute (`flgatr` is invalid or cannot be used)
- **E_PAR** Parameter error (`pk_cflg` is invalid)

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

- **TK_SUPPORT_DISWAI** Support for specifying prohibition of "disabling wait" (`TA_NODISWAI`) to event flag attribute
- **TK_SUPPORT_DSNAME** Support for specifying `TA_DSNAME` for event flag attribute

Description

Creates an event flag, assigning to it an event flag ID. This system call allocates a control block to the created event flag and sets its initial value to `iflgptn`. An event flag handles one word’s worth of bits as a group. All operations are performed in single word units.

`exinf` can be used freely by the user to set miscellaneous information about the created event flag. The information set in this parameter can be referenced by `tk_ref_flg`. If a larger area is needed for indicating user
information, or if the information may need to be changed after the message buffer is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in `exinf`. The kernel pays no attention to the contents of `exinf`.

`flgatr` indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of `flgatr` is as follows.

\[
flgatr := (TA_TFIFO || TA_TPRI) | (TA_WMUL || TA_WSGL) | [TA_DSNAME] | [TA_NODISWAI]
\]

- **TA_TFIFO**: Tasks are queued in FIFO order
- **TA_TPRI**: Tasks are queued in priority order
- **TA_WSGL**: Waiting by multiple tasks is not allowed (Wait Single Task)
- **TA_WMUL**: Waiting by multiple tasks is allowed (Wait Multiple Tasks)
- **TA_DSNAME**: Specifies DS object name
- **TA_NODISWAI**: Disabling of wait by `tk_dis_wai` is prohibited

When `TA_WSGL` is specified, multiple tasks cannot be in the WAITING state at the same time. Specifying `TA_WMUL` allows waiting by multiple tasks at the same time.

The queuing order of tasks waiting for an event flag can be specified in `TA_TFIFO` or `TA_TPRI`. If the attribute is `TA_TFIFO`, tasks are ordered by FIFO, whereas `TA_TPRI` specifies queuing of tasks in order of their priority setting. When `TA_WSGL` is specified, however, since tasks cannot be queued, `TA_TFIFO` or `TA_TPRI` makes no difference.

When multiple tasks are waiting for an event flag, tasks are checked in order from the head of the queue, and the wait is released for tasks meeting the conditions. The first task to have its WAITING state released is therefore not necessarily the first in the queue. If multiple tasks meet the conditions, wait state is released for each of them.

When `TA_DSNAME` is specified, `dsname` is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, `td_ref_dsname` and `td_set_dsname`. For more details, see the description of `td_ref_dsname` and `td_set_dsname`. If `TA_DSNAME` is not specified, `dsname` is ignored. Then `td_ref_dsname` and `td_set_dsname` return E_OBJ error.

```c
#define TA_TFIFO 0x00000000 /* manage queue by FIFO */
#define TA_TPRI 0x00000001 /* manage queue by priority */
#define TA_WSGL 0x00000000 /* prohibit multiple task waiting */
#define TA_WMUL 0x00000080 /* permit multiple task waiting */
#define TA_DSNAME 0x00000040 /* DS object name */
#define TA_NODISWAI 0x00000080 /* reject request to disable wait */
```

Porting Guideline

Note that member, `iflgptn`, of `T_CFLG` is UINT type, and its value range is implementation-dependent, so care must be taken.
4.4.2.2  tk_del_flg - Delete Event Flag

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_del_flg(ID flgid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>flgid</th>
<th>EventFlag ID</th>
<th>Event flag ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes the event flag specified in flgid.

Issuing this system call releases the corresponding event flag ID and control block memory space.

This system call is completed normally even if there are tasks waiting for the event flag, but error code E_DLT is returned to each task in WAITING state.
4.4.2.3  

**tk_set_flg - Set Event Flag**

**C Language Interface**

```
#include <tk/tkernel.h>
```

ER ercd = tk_set_flg(ID flgid, UINT setptn);

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>flgid</th>
<th>EventFlag ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>setptn</td>
<td>Set Bit Pattern</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
</tr>
</tbody>
</table>

**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)

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

None.

**Description**

*tk_set_flg* sets the bits indicated in *setptn* in a one-word event flag specified in *flgid*. That is, a logical sum is taken of the values of the event flag specified in *flgid* and the values indicated in *setptn*. (the processing *flgptn |= setptn* is executed for the event flag value *flgptn*)

After event flag values are changed by *tk_set_flg*, if the condition for releasing the wait state of a task that called *tk_wai_flg* is met, the WAITING state of that task is cleared, putting it in RUNNING state or READY state (or SUSPENDED state if the waiting task was in WAITING-SUSPENDED state).

If all the bits of *setptn* are cleared to 0 in *tk_set_flg*, no operation is made to the target event flag. No error will result in either case.

Multiple tasks can wait for a single event flag if that event flag has the TA_WMUL attribute. The event flag in that case has a queue for the waiting tasks. A single *tk_set_flg* call for such an event flag may result in the release of multiple waiting tasks.

**Porting Guideline**

Note that *setptn* is UINT type, and its value range is implementation-dependent, so care must be taken.
4.4.2.4 tk_clr_flg - Clear Event Flag

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_clr_flg(ID flgid, UINT clrptn);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>flgid</th>
<th>EventFlag ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>clrptn</td>
<td>Clear Bit Pattern</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

The `tk_clr_flg` function clears the bits of the one-word event flag specified in `flgid`, based on the corresponding zero bits of `clrptn`. That is, a logical product is taken of the values of the event flag specified in `flgid` and the values indicated in `clrptn` (the processing `flgptn & clrptn` is executed for the event flag value `flgptn`).

Issuing `tk_clr_flg` never results in wait conditions being released for a task waiting for the specified event flag; that is, dispatching never occurs with `tk_clr_flg`.

If all the bits of `clrptn` are set to 1 in `tk_clr_flg`, no operation is made to the target event flag. No error will be returned in either case.

Porting Guideline

Note that `clrptn` is UINT type, and its value range is implementation-dependent, so care must be taken.
4.4.2.5  tk_wai_flg - Wait Event Flag

C Language Interface

#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>flgid</th>
<th>EventFlag ID</th>
<th>Event flag ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>waiptn</td>
<td>Wait Bit Pattern</td>
<td>Wait bit pattern</td>
</tr>
<tr>
<td>UINT</td>
<td>wfmode</td>
<td>Wait EventFlag Mode</td>
<td>Wait release condition</td>
</tr>
<tr>
<td>UINT</td>
<td>p_flgptn</td>
<td>Pointer to EventFlag Bit Pattern</td>
<td>Pointer to the area to return the return parameter flgptn</td>
</tr>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
<td>Timeout (ms)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>flgptn</td>
<td>EventFlag Bit Pattern</td>
<td>Event flag bit pattern</td>
</tr>
</tbody>
</table>

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_OBJ**: 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**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Waits for the event flag specified in flgid to be set, fulfilling the wait release condition specified in wfmode.

If the event flag specified in flgid already meets the wait release condition set in wfmode, the waiting task continues executing without going to WAITING state.
**wfmode** is specified as follows.

\[
\text{wfmode} = (\text{TWF\_ANDW} || \text{TWF\_ORW}) | \text{[TWF\_CLR || TWF\_BITCLR]} \\
\text{TWF\_ANDW} & 0x00 & \text{AND wait condition} \\
\text{TWF\_ORW} & 0x01 & \text{OR wait condition} \\
\text{TWF\_CLR} & 0x10 & \text{Clear all} \\
\text{TWF\_BITCLR} & 0x20 & \text{Clear condition bit only}
\]

If **TWF\_ORW** is specified, the issuing task waits for any of the bits specified in **waiptn** to be set for the event flag specified in **flgid** (OR wait). If **TWF\_ANDW** is specified, the issuing task will wait for all of the bits specified in **waiptn** to be set for the event flag specified in **flgid** (AND wait).

If **TWF\_CLR** specification is not specified, the event flag values will remain unchanged even after the conditions have been satisfied and the task has been released from WAITING state. If **TWF\_CLR** is specified, all bits of the event flag will be cleared to 0 once wait conditions of the waiting task have been met. If **TWF\_BITCLR** is specified, then when the conditions are met and the task is released from WAITING state, only the bits matching the event flag wait release conditions are cleared to 0 (event flag values &= ~wait release conditions).

The return parameter **flgptn** returns the value of the event flag after the WAITING state of a task has been released due to this system call. If **TWF\_CLR** or **TWF\_BITCLR** was specified, the value before event flag bits were cleared is returned. The value returned by **flgptn** meets the wait release conditions of this system call. The contents of **flgptn** are indeterminate if the wait is released due to timeout or the like.

A maximum wait time (timeout) can be set in **tmout**. The time unit for **tmout** is the same as that for system time (= 1 ms). If the **tmout** time elapses before the wait release condition is met, the system call terminates, returning timeout error code **E\_TMOUT**.

When **TMO\_POL**=0 is set in **tmout**, this means 0 was specified as the timeout value, and **E\_TMOUT** is returned without entering WAITING state even if the condition is not met. When **TMO\_FEVR**=(-1) is set in **tmout**, this means infinity was specified as the timeout value, and the task continues to wait for the condition to be met without timing out.

In the case of a timeout, the event flag bits are not cleared even if **TWF\_CLR** or **TWF\_BITCLR** was specified.

Setting **waiptn** to 0 results in Parameter error **E\_PAR**.

A task cannot execute **tk\_wai\_flg** for an event flag having the **TA\_WSGL** attribute while another task is waiting for it. Error code **E\_OBJ** will be returned for the task issuing the subsequent **tk\_wai\_flg**, regardless of whether that task would have gone to WAITING state; i.e., regardless of whether the wait release conditions would be met.

If an event flag has the **TA\_WMUL** attribute, multiple tasks can wait for it at the same time. The event flag in that case has a queue for the waiting tasks. A single **tk\_set\_flg** call for such an event flag may result in the release of multiple waiting tasks.

If multiple tasks are queued for an event flag with **TA\_WMUL** attribute, the behavior is as follows.

- Tasks are queued in either FIFO or priority order. (Release of wait state does not always start from the head of the queue, however, depending on factors such as **waiptn** and **wfmode** settings.)
- If **TWF\_CLR** or **TWF\_BITCLR** was specified by a task in the queue, the event flag is cleared when that task is released from WAITING state.
- Tasks later in the queue than a task specifying **TWF\_CLR** or **TWF\_BITCLR** will see the event flag after it has already been cleared.

If multiple tasks having the same priority are released from waiting simultaneously as a result of **tk\_set\_flg**, the order of tasks in the ready queue (precedence) after release will continue to be the same as their original order in the event flag queue.
Additional Notes

If a logical sum of all bits is specified as the wait release condition when `tk_wai_flg` is called (`waiptn = 0xfff...ff, wfmode = TWF ORW`), it is possible to transfer messages using one-word bit patterns in combination with `tk_set_flg`. However, it is not possible to send a message containing only 0s for all bits. Moreover, if the next message is sent by `tk_set_flg` before a previous message has been read by `tk_wai_flg`, the previous message will be lost; that is, message queuing is not possible.

Since setting `waiptn = 0` will result in an `E_PAR` error, it is guaranteed that the `waiptn` of tasks waiting for an event flag will not be 0. The result is that if `tk_set_flg` sets all bits of an event flag to 1, the task at the head of the queue will always be released from waiting no matter what its wait condition is.

The ability to have multiple tasks wait for the same event flag is useful in situations like the following. Suppose, for example, that Task B and Task C are waiting for `tk_wai_flg` calls (2) and (3) until Task A issues (1) `tk_set_flg`. If multiple tasks are allowed to wait for the event flag, the result will be the same regardless of the order in which system calls (1)(2)(3) are executed (see Figure 4.1, "Multiple Tasks Waiting for One Event Flag"). On the other hand, if multiple task waiting is not allowed and system calls are executed in the order (2), (3), (1), an `E_OBJ` error will result from the execution of (3) `tk_wai_flg`.

![Figure 4.1: Multiple Tasks Waiting for One Event Flag](image)

Rationale for the Specification

The reason for returning `E_PAR` error for specifying `waiptn = 0` is that if `waiptn = 0` were allowed, it would not be possible to get out of WAITING state regardless of the subsequent event flag values.

Porting Guideline

Note that the data pointed at `waiptn` and `p_flgptn` are UINT type, and their value range is implementation-dependent, so care must be taken.
4.4.2.6 tk_wai_flg_u - Wait Event Flag (Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_wai_flg_u(ID flgid, UINT waiptn, UINT wfmode, UINT *p_flgptn, TMO_U tmout_u);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>flgid</th>
<th>EventFlag ID</th>
<th>Event flag ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>waiptn</td>
<td>Wait Bit Pattern</td>
<td>Wait bit pattern</td>
</tr>
<tr>
<td>UINT</td>
<td>wfmode</td>
<td>Wait EventFlag Mode</td>
<td>Wait mode</td>
</tr>
<tr>
<td>UINT*</td>
<td>p_flgptn</td>
<td>Pointer to EventFlag Bit Pattern</td>
<td>Pointer to the area to return the return parameter flgptn</td>
</tr>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout</td>
<td>Timeout (in microseconds)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>flgptn</td>
<td>EventFlag Bit Pattern</td>
<td>Bit pattern of wait releasing</td>
</tr>
</tbody>
</table>

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_u \(\leq (-2))\)
- **E_OBJ**: 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**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_USEC**: Support of microsecond

Description

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_wai_flg.
The specification of this system call is same as that of tk_wai_flg, except that the parameter is replaced with tmout_u. For more details, see the description of tk_wai_flg.

Porting Guideline

Note that the data pointed at waiptn and p_flagptn are UINT type, and their value range is implementation-dependent, so care must be taken.
4.4.2.7  tk_ref_flg - Reference Event Flag Status

C Language Interface

#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>flgid</th>
<th>EventFlag ID</th>
<th>Event flag ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RFLG*</td>
<td>pk_rflg</td>
<td>Packet to Return EventFlag Status</td>
<td>Pointer to the area to return the event flag status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

pk_rflg Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Waiting Task ID</td>
<td>Waiting task ID</td>
</tr>
<tr>
<td>UINT</td>
<td>flgptn</td>
<td>EventFlag Bit Pattern</td>
<td>The current event flag bit pattern</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>Invalid ID number (flgid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the event flag specified in flgid does not exist)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (invalid pk_rflg)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

References the status of the event flag specified in flgid, passing in the return parameters the current flag pattern (flgptn), waiting task ID (wtsk), and extended information (exinf).

wtsk returns the ID of a task waiting for this event flag. If more than one task is waiting (only when the TA_WMUL was specified), the ID of the first task in the queue is returned. If there are no waiting tasks, wtsk = 0 is returned.

If the specified event flag does not exist, error code E_NOEXS is returned.
4.4.3 Mailbox

A mailbox is an object used to achieve synchronization and communication by passing messages in system (shared) memory space. Functions are provided for creating and deleting a mailbox, sending and receiving messages in a mailbox, and referencing the mailbox status. A mailbox is an object identified by an ID number. The ID number for the mailbox is called a mailbox ID.

A mailbox has a message queue for sent messages, and a task queue for tasks waiting to receive messages. At the message sending end (posting event notification), messages to be sent go in the message queue. On the message receiving end (waiting for event notification), a task fetches one message from the message queue. If there are no queued messages, the task goes to WAITING state for receipt from the mailbox until the next message is sent. Tasks waiting for message receipt from a mailbox are put in the task queue of that mailbox.

Since the contents of messages using this function are in memory space shared both by the sending and receiving sides, only the start address of a message located in this shared space is actually sent and received. The contents of the messages themselves are not copied. T-Kernel manages messages in the message queue by means of a linked listed. An application program must allocate space at the beginning of a message to be sent, for linked list processing by T-Kernel. This area is called the message header. The message header and the message body together are called a message packet. When a system call sends a message to a mailbox, the start address of the message packet \((\text{pk\_msg})\) is passed in a parameter.

When a system call receives a message from a mailbox, the start address of the message packet is passed in a return parameter.

If messages are assigned a priority in the message queue, the message priority \((\text{msgpri})\) of each message must be specified in the message header. [Figure 4.2, “Format of Messages Using a Mailbox”]

The user puts the message contents not at the beginning of the packet but after the header part (the message contents part in the figure).

![Figure 4.2: Format of Messages Using a Mailbox](image)

T-Kernel overwrites the contents of the header when a message is put in the message queue (except for the message priority area). An application, on the other hand, must not overwrite the header of a message in
the queue (including the message priority area). The behavior when an application overwrites the message header is not defined. This specification applies not only to the direct writing of a message header by an application program, but also to the multiple passing of a header address to T-Kernel and having T-Kernel overwrite the message header. Accordingly, the behavior when a message already in the message queue is again sent to a mailbox is undefined.

Additional Notes
Since the application program allocates the message header space for this mailbox function, there is no limit on the number of messages that can be queued. A system call sending a message does not enter WAITING state. Memory blocks allocated dynamically from a fixed-size memory pool or variable-size memory pool, or else a statically allocated area can be used for message packets. Generally, a sending task allocates a memory block from a memory pool, sending it as a message packet. After a task on the receiving end fetches the message, it returns the memory block directly to its memory pool.

The following sample programs show the above usage:

```c
/* Message type definition */
typedef struct {
    T_MSG msgque; /* Message header with T_MFIFO attribute */
    UB msgcont[MSG_SIZE]; /* Message content */
} T_MSG_PACKET;

/* Task operation that acquires a memory block and sends a message */
T_MSG_PACKET *pk_msg;
...
/* Acquire a memory block from the fixed-size memory pool. */
/* Fixed-memory block size must be sizeof(T_MSG_PACKET) or more */
tk_get_mpf( mpfid, (void**)&pk_msg, TMO_FEVR );
/* Create a message at pk_msg -> msgcont[] */
...
/* Send a message */
tk_snd_mbx( mbxid, (T_MSG*)pk_msg );

/* Task operation that receives a message and releases a memory block */
T_MSG_PACKET *pk_msg;
...
/* Receive a message */
tk_rcv_mbx( mbxid, (T_MSG**)&pk_msg, TMO_FEVR );
/* Check message content at pk_msg -> msgcont[] and process them accordingly */
...
/* Return the memory block to the fixed-size memory pool. */
tk_rel_mpf( mpfid, (void*)pk_msg );
```
4.4.3.1  tk_cre_mbx - Create Mailbox

C Language Interface
#include <tk/tkernel.h>

ID mbxid = tk_cre_mbx(CONST T_CMBX *pk_cmbx);

Parameter

CONST T_CMBX* pk_cmbx  Packet to Create Mailbox  Mailbox creation information

pk_cmbx Detail:

<table>
<thead>
<tr>
<th>void* exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR mbxatr</td>
<td>Mailbox Attribute</td>
<td>Mailbox attribute</td>
</tr>
<tr>
<td>UB dsname[8]</td>
<td>DS Object name</td>
<td>DS object name</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

ID mbxid  Mailbox ID  Mailbox ID or Error Code

Error Code

E_NOMEM Insufficient memory (memory for control block cannot be allocated)
E_LIMIT Number of mailboxes exceeds the system limit
E_RSATR Reserved attribute (mbxatr is invalid or cannot be used)
E_PAR Parameter error (pk_cmbx is invalid)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

TK_SUPPORT_DISWAI  Support for specifying TA_NODISWAI (reject request to disable wait) to mailbox attribute
TK_SUPPORT_DSNAME  Support for specifying TA_DSNAME for mailbox attribute

Description

Creates a mailbox, assigning to it a mailbox ID. This system call allocates a control block, etc. for the created mailbox.

exinf can be used freely by the user to set miscellaneous information about the created mailbox. The information set in this parameter can be referenced by tk_ref_mbx. If a larger area is needed for indicating user information, or if the information may need to be changed after the message buffer is created, this can be
done by allocating separate memory for this purpose and putting the memory packet address in `exinf`. The kernel pays no attention to the contents of `exinf`.

`mbxatr` indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of `mbxatr` is as follows.

```
mbxatr:= (TA_TFIFO || TA_TPRI) | (TA_MFIFO || TA_MPRI) | [TA_DSNAME] | [TA_NODISWAI]
```

- **TA_TFIFO**: Tasks are queued in FIFO order
- **TA_TPRI**: Tasks are queued in priority order
- **TA_MFIFO**: Messages are queued in FIFO order
- **TA_MPRI**: Messages are queued in priority order
- **TA_DSNAME**: Specifies DS object name
- **TA_NODISWAI**: Disabling of wait by `tk_dis_wai` is prohibited

The queuing order of tasks waiting for a mailbox can be specified in `TA_TFIFO` or `TA_TPRI`. If the attribute is `TA_TFIFO`, tasks are ordered by FIFO, whereas `TA_TPRI` specifies queuing of tasks in order of their priority setting.

`TA_MFIFO` and `TA_MPRI` are used to specify the order of messages in the message queue (messages waiting to be received). If the attribute is `TA_MFIFO`, messages are ordered by FIFO; `TA_MPRI` specifies queuing of messages in priority order. Message priority is set in a special field in the message packet. Message priority is specified by positive values, with 1 indicating the highest priority and higher numbers indicating successively lower priority. The largest value that can be expressed in the PRI type is the lowest priority. Messages having the same priority are ordered as FIFO.

When `TA_DSNAME` is specified, `dsname` is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, `td_ref_dsname` and `td_set_dsname`. For more details, see the description of `td_ref_dsname` and `td_set_dsname`. If `TA_DSNAME` is not specified, `dsname` is ignored. Then `td_ref_dsname` and `td_set_dsname` return E_OBJ error.

<table>
<thead>
<tr>
<th>#define</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TA_TFIFO</code></td>
<td>0x00000000</td>
<td>manage queue by FIFO</td>
</tr>
<tr>
<td><code>TA_TPRI</code></td>
<td>0x00000001</td>
<td>manage queue by priority</td>
</tr>
<tr>
<td><code>TA_MFIFO</code></td>
<td>0x00000000</td>
<td>manage message queue by FIFO</td>
</tr>
<tr>
<td><code>TA_MPRI</code></td>
<td>0x00000002</td>
<td>manage message queue by priority</td>
</tr>
<tr>
<td><code>TA_DSNAME</code></td>
<td>0x00000040</td>
<td>DS object name</td>
</tr>
<tr>
<td><code>TA_NODISWAI</code></td>
<td>0x00000080</td>
<td>reject request to disable wait</td>
</tr>
</tbody>
</table>

### Additional Notes

The body of a message passed by the mailbox function is located in memory; only its start address is actually sent and received.
4.4.3.2  tk_del_mbx - Delete Mailbox

C Language Interface
#include <tk/tkernel.h>

ER ercd = tk_del_mbx(ID mbxid);

Parameter

| ID mbxid | Mailbox ID |
Return Parameter

| ER ercd | 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) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes the mailbox specified in mbxid.
Issuing this system call releases the mailbox ID and control block memory space, etc., associated with the mailbox.
This system call completes normally even if there are tasks waiting for messages in the deleted mailbox, but error code E_DLT is returned to each of the tasks in WAITING state. Even if there are messages still in the deleted mailbox, the mailbox is deleted without returning an error code.
4.4.3.3  tk_snd_mbx - Send Message to Mailbox

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbxid</th>
<th>Mailbox ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T.MSG</td>
<td>pk_msg</td>
<td>Packet of Message</td>
</tr>
</tbody>
</table>

Start address of message packet

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

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 (invalid pk_msg, or msgpri ≤ 0)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Sends the message packet having pk_msg as its start address to the mailbox specified in mbxid.

The message packet contents are not copied; only the start address (pk_msg) is passed at the time of message receipt. Therefore, the content of the message packet must not be overwritten until it is fetched by the task that receives this message.

If tasks are already waiting for messages in the same mailbox, the WAITING state of the task at the head of the queue is released, and the pk_msg specified in tk_snd_mbx is sent to that task, becoming a parameter returned by tk_rcv_mbx. If there are no tasks waiting for messages in the specified mailbox, the sent message goes in the message queue of that mailbox. In neither case does the task issuing tk_snd_mbx enter WAITING state.

pk_msg is the start address of the packet containing the message, including header. The message header has the following format.

```c
typedef struct t_msg {
    ? ?                  /* Implementation-dependent content (fixed-size) */
} T_MSG;

typedef struct t_msg_pri {
```
The message header is `T_MSG` (if `TA_MFIFO` attribute is specified) or `T_MSG_PRI` (if `TA_MPRI`). In either case the message header has a fixed-size, which can be obtained by `sizeof(T_MSG)` or `sizeof(T_MSG_PRI)`.

The actual message must be put in the area after the header. There is no limit on message size, which may be variable.

Additional Notes

Messages are sent by `tk_snd_mbx` regardless of the status of the receiving tasks. In other words, message sending is asynchronous. What waits in the queue is not the sending task itself, but the sent message. So while there are queues of waiting messages and receiving tasks, the sending task does not go to WAITING state.
4.4.3.4  tk_rcv_mbx - Receive Message from Mailbox

C Language Interface

#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbxid</th>
<th>Mailbox ID</th>
<th>Mailbox ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_MSG*</td>
<td>ppk_msg</td>
<td>Pointer to Packet of Message</td>
<td>Pointer to the area to return the return parameter pk_msg</td>
</tr>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
<td>Timeout (ms)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_MSG*</td>
<td>pk_msg</td>
<td>Packet of Message</td>
<td>Start address of message packet</td>
</tr>
</tbody>
</table>

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: Waiting state released (tk_rel_wai received in waiting state)
- E_DISWAI: Wait released due to disabling of wait
- E_TMOUT: Polling failed or timeout
- E_CTX: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

tk_rcv_mbx receives a message from the mailbox specified in mbxid.

If no messages have been sent to the mailbox (the message queue is empty), the task issuing this system call enters WAITING state and is queued for message arrival. If there are messages in the mailbox, the task issuing this system call fetches the first message in the message queue, passing this in the return parameter pk_msg.

A maximum wait time (timeout) can be set in tmout. The time unit for tmout is the same as that for system time (= 1 ms). If the tmout time elapses before the wait release condition is met (before a message arrives), the system call terminates, returning timeout error code E_TMOUT.
When \texttt{TMO\_POL}=0 is set in \texttt{tmout}, this means 0 was specified as the timeout value, and \texttt{E\_TMOUT} is returned without entering WAITING state even if no message arrives. When \texttt{TMO\_FEVR}=(-1) is set in \texttt{tmout}, this means infinity was specified as the timeout value, and the task continues to wait for message arrival without timing out.

Additional Notes

\texttt{pk\_msg} is the start address of the packet containing the message, including header. The message header is \texttt{T\_MSG} (if \texttt{TA\_MFIFO} attribute is specified) or \texttt{T\_MSG\_PRI} (if \texttt{TA\_MPRI}).
4.4.3.5  tk_rcv_mbx_u - Receive Message from Mailbox (Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_rcv_mbx_u(ID mbxid, T_MSG **ppk_msg, TMO_U tmout_u);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbxid</th>
<th>Mailbox ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_MSG**</td>
<td>ppk_msg</td>
<td>Pointer to Packet of Message</td>
</tr>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_MSG*</td>
<td>pk_msg</td>
<td>Packet of Message</td>
</tr>
</tbody>
</table>

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_u ≦ (-2))
- **E_DLT**: The object being waited for was deleted (the mailbox was deleted while waiting)
- **E_RLWAI**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_USEC**: Support of microsecond

Description

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_rcv_mbx.

The specification of this system call is same as that of tk_rcv_mbx, except that the parameter is replaced with tmout_u. For more details, see the description of tk_rcv_mbx.
4.4.3.6 tk_ref_mbx - Reference Mailbox Status

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbxid</th>
<th>Mailbox ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RMBX*</td>
<td>pk_rmbx</td>
<td>Packet to Refer Mailbox Status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

pk_rmbx Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Waiting Task ID</td>
</tr>
<tr>
<td>T_MSG*</td>
<td>pk_msg</td>
<td>Packet of Message</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

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 (invalid pk_rmbx)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

References the status of the mailbox specified in mbxid, passing in the return parameters the next message to be received (the first message in the message queue), waiting task ID (wtsk), and extended information (exinf).

wtsk indicates the ID of a task waiting for the mailbox. If there are multiple waiting tasks, the ID of the first task in the queue is returned. If there are no waiting tasks, wtsk = 0 is returned.

If the specified mailbox does not exist, error code E_NOEXS is returned.
pk_msg indicates the message that will be received the next time tk_rcv_mbx is issued. If there are no messages in the message queue, pk_msg = NULL is returned. At least one of pk_msg = NULL and wtsk = 0 is always true for this system call.
4.5 Extended Synchronization and Communication Functions

Extended synchronization and communication functions use objects independent of tasks to realize more sophisticated synchronization and communication between tasks. The functions specified here include mutex and message buffer functions.
4.5.1 Mutex

A mutex is an object for mutual exclusion control among tasks that use shared resources. Priority inheritance mutexes and priority ceiling mutexes are supported, as a mechanism to prevent the problem of unbounded priority inversion that can occur in mutual exclusion control.

Functions are provided for creating and deleting a mutex, locking and unlocking a mutex, and referencing mutex status. A mutex is identified by an ID number. The ID number for the mutex is called a mutex ID.

A mutex has a status (locked or unlocked) and a queue for tasks waiting to lock the mutex. For each mutex, T-Kernel keeps track of the tasks locking it; and for each task, it keeps track of the mutexes it has locked. Before a task uses a resource, it locks a mutex associated with that resource. If the mutex is already locked by another task, the task waits for the mutex to become unlocked. Tasks in mutex lock waiting state are put in the mutex queue. When a task finishes with a resource, it unlocks the mutex.

A mutex with TA_INHERIT (= 0x02) specified as mutex attribute supports priority inheritance protocol while one with TA_CEILING (= 0x03) specified supports priority ceiling protocol. When a mutex with TA_CEILING attribute is created, a ceiling priority is assigned to it, indicating the base priority of the task having the highest base priority among the tasks that will lock that mutex. If a task having a higher base priority than the ceiling priority of the mutex with TA_CEILING attribute tries to lock it, error code E_ILUSE is returned. If tk_chg_pri is issued in an attempt to set the base priority of a task having locked a mutex with TA_CEILING attribute to a value higher than the ceiling priority of that mutex, E_ILUSE is returned by the tk_chg_pri system call.

When these protocols are used, unbounded priority inversion is prevented by automatically changing the current priority of a task in a mutex operation. Strict adherence to the priority inheritance protocol and priority ceiling protocol requires that the task current priority must always be changed to match the peak value of the following priorities. This is called strict priority control.

- Task base priority
- When tasks lock mutexes with TA_INHERIT attribute, the current priority of the task having the highest current priority of the tasks waiting for those mutexes.
- When tasks lock mutexes with TA_CEILING attribute, the highest ceiling priority of the mutex among those mutexes.

Note that when the current priority of a task waiting for a mutex with TA_INHERIT attribute changes as the result of a base priority change brought about by mutex operation or tk_chg_pri, it may become necessary to change the current priority of the task locking that mutex. This is called dynamic priority inheritance. Further, if this task is waiting for another mutex with TA_INHERIT attribute, dynamic priority inheritance processing may become necessary also for the task locking that mutex.

The T-Kernel defines, in addition to the above strict priority control, a simplified priority control limiting the situations in which the current priority is changed. The choice between the two is implementation-dependent. In the simplified priority control, whereas all changes in the direction of raising the task current priority are carried out, changes in the direction of lowering that priority are made only when a task is no longer locking any mutexes. (In this case the task current priority reverts to the base priority.) More specifically, processing to change the current priority is needed only in the following circumstances.

- When a task with a higher current priority than that of the task locking a mutex with TA_INHERIT attribute starts waiting for that mutex.
- When task B is waiting for a mutex with TA_INHERIT attribute being locked by another task called A, and if the current priority of B is changed to a higher one than that of task A.
- When a task locks a mutex with TA_CEILING attribute having a higher ceiling priority than the task’s current priority.
- When a task is no longer locking any mutexes.
When the current priority of a task is changed in connection with a mutex operation, the following processing is performed.

If the task whose priority changed is in a run state, the task precedence is changed in accordance with the new priority. Its precedence among other tasks having the same priority is implementation-dependent. Likewise, if the task whose priority changes is waiting in a queue of some kind, its order in that queue is changed based on its new priority. Its order among other tasks having the same priority is implementation-dependent. When a task terminates and there are mutexes still locked by that task, all the mutexes are unlocked. The order in which multiple locked mutexes are unlocked is implementation-dependent. See the description of `tk_unl_mtx` for the specific processing involved.

**Additional Notes**

TA_TFIFO attribute mutex or TA_TPRI attribute mutex has functionality equivalent to that of a semaphore with a maximum of one resource (binary semaphore). The main differences are that a mutex can be unlocked only by the task that locked it, and a mutex is automatically unlocked when the task locking it terminates.

The term "priority ceiling protocol" is used here in a broad sense. The protocol described here is not the same as the algorithm originally proposed. Strictly speaking, it is what is otherwise referred to as a highest locker protocol or by other names.

When the change in current priority of a task due to a mutex operation results in that task’s order being changed in a priority-based queue, it may be necessary to release the waiting state of other tasks waiting for that task or for that queue.

**Rationale for the Specification**

The precedence of tasks having the same priority as the result of a change in task current priority in a mutex operation is left as implementation-dependent, for the following reason. Depending on the application, the mutex function may lead to frequent changes in current priority. It would not be desirable for this to result in constant task switching, which is what would happen if the precedence were made the lowest each time among tasks of the same priority. Ideally task precedence rather than priority should be inherited, but that results in large overhead in implementation. This aspect of the specification is therefore made an implementation-dependent matter.
4.5.1.1  tk_cre_mtx - Create Mutex

C Language Interface

```c
#include <tk/tkernel.h>

ID mtxid = tk_cre_mtx(CONST T_CMTX *pk_cmtx);
```

Parameter

- **CONST T_CMTX** `pk_cmtx` Packet to Create Mutex Information about the mutex to be created

`pk_cmtx` Detail:

- `void* exinf` Extended Information
- `ATR mtxatr` Mutex Attribute
- `PRI ceilpri` Ceiling Priority of Mutex
- `UB dsname[8]` DS Object name

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

- **ID mtxid** Mutex ID
- **or Error Code** Error code

Error Code

- `E_NOMEM` Insufficient memory (memory for control block cannot be allocated)
- `E_LIMIT` Number of mutexes exceeds the system limit
- `E_RSATR` Reserved attribute (`mtxatr` is invalid or cannot be used)
- `E_PAR` Parameter error (`pk_cmtx` or `ceilpri` is invalid)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

- `TK_SUPPORT_DISWAI` Support for specifying `TA_NODISWAI` (reject request to disable wait) to mutex attribute
- `TK_SUPPORT_DSNAME` Support for specifying `TA_DSNAME` for mutex attribute

Description

Creates a mutex, assigning to it a mutex ID. This system call allocates a control block, etc. for the created mutex.

`exinf` can be used freely by the user to set miscellaneous information about the created mutex. The information set in this parameter can be referenced by `tk_ref_mtx`. If a larger area is needed for indicating user information,
or if the information may need to be changed after the message buffer is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in \texttt{exinf}. The kernel pays no attention to the contents of \texttt{exinf}.

\texttt{mtxatr} indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of \texttt{mtxatr} is specified as follows.

\begin{verbatim}
mtxatr:= (TA_TFIFO || TA_TPRI || TA_INHERIT || TA_CEILING) | [TA_DSNAME] | [TA_NODISWAI]
\end{verbatim}

\begin{itemize}
\item \texttt{TA_TFIFO} Tasks are queued in FIFO order
\item \texttt{TA_TPRI} Tasks are queued in priority order
\item \texttt{TA_INHERIT} Priority inheritance protocol
\item \texttt{TA_CEILING} Priority ceiling protocol
\item \texttt{TA_DSNAME} Specifies DS object name
\item \texttt{TA_NODISWAI} Disabling of wait by \texttt{tk_dis_wai} is prohibited
\end{itemize}

When the \texttt{TA_TFIFO} attribute is specified, the order of the mutex task queue is FIFO. If \texttt{TA_TPRI}, \texttt{TA_INHERIT}, or \texttt{TA_CEILING} is specified, tasks are ordered by their priority. \texttt{TA_INHERIT} indicates that priority inheritance protocol is used, and \texttt{TA_CEILING} specifies priority ceiling protocol.

Only when \texttt{TA_CEILING} is specified, \texttt{ceilpri} is valid and specifies the mutex ceiling priority.

When \texttt{TA_DSNAME} is specified, \texttt{dsname} is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, \texttt{td_ref_dsname} and \texttt{td_set_dsname}. For more details, see the description of \texttt{td_ref_dsname} and \texttt{td_set_dsname}. If \texttt{TA_DSNAME} is not specified, \texttt{dsname} is ignored. Then \texttt{td_ref_dsname} and \texttt{td_set_dsname} return \texttt{E_OBJ} error.

\begin{verbatim}
#define TA_TFIFO 0x00000000 /* manage queue by FIFO */
#define TA_TPRI 0x00000001 /* manage queue by priority */
#define TA_INHERIT 0x00000002 /* priority inheritance protocol */
#define TA_CEILING 0x00000003 /* priority ceiling protocol */
#define TA_DSNAME 0x00000004 /* DS object name */
#define TA_NODISWAI 0x00000080 /* reject request to disable wait */
\end{verbatim}
4.5.1.2  tk_del_mtx - Delete Mutex

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_del_mtx(ID mtxid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mtxid</th>
<th>Mutex ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes the mutex specified in mtxid.

Issuing this system call releases the mutex ID and control block memory space allocated to the mutex.

This system call completes normally even if there are tasks waiting to lock the deleted mutex, but error code E_DLT is returned to each of the tasks in WAITING state.

When a mutex is deleted, a task locking the mutex will have one fewer locked mutexes. If the mutex to be deleted was a priority inheritance mutex (TA_INHERIT) or priority ceiling mutex (TA_CEILING), then deleting the mutex might change the priority of the task that has locked it.
4.5.1.3  tk_loc_mtx - Lock Mutex

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mtxid</th>
<th>Mutex ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

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**  Waiting state released (`tk_rel_wai` received in waiting state)
- **E_DISWAI**  Wait released due to disabling of wait
- **E_TMOUT**  Polling failed or timeout
- **E_CTX**  Context error (issued from task-independent portion, or in dispatch disabled state)
- **E_ILUSE**  Illegal use (multiple lock, or upper priority limit exceeded)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Locks the mutex specified in `mtxid`. If the mutex can be locked immediately, the task issuing this system call continue executing without entering WAITING state, and the mutex goes to locked status. If the mutex cannot be locked, the task issuing this system call enters WAITING state. That is, the task is put in the queue of this mutex.

A maximum wait time (timeout) can be set in `tmout`. The time unit for `tmout` is the same as that for system time (= 1 ms). If the `tmout` time elapses before the wait release condition is met, the system call terminates, returning timeout error code `E_TMOUT`.

When `TMO POL=0` is set in `tmout`, this means 0 was specified as the timeout value, and `E TMOUT` is returned without entering WAITING state even if the resource cannot be locked. When `TMO FEVR=(-1)` is set in `tmout`,
this means infinity was specified as the timeout value, and the task continues wait to until the resource is
locked.
If the invoking task has already locked the specified mutex, error code E_ILUSE (multiple lock) is returned.
If the specified mutex is a priority ceiling mutex (TA_CEILING) and the base priority\(^1\) of the invoking task is
higher than the ceiling priority of the mutex, error code E_ILUSE (upper priority limit exceeded) is returned.

**Additional Notes**

- **Priority inheritance mutex (TA_INHERIT attribute)**
  
  If the invoking task is waiting to lock a mutex and the current priority of the task currently locking that
  mutex is lower than that of the invoking task, the priority of the locking task is raised to the same level
  as the invoking task. If the wait ends before the waiting task can obtain a lock (timeout or other reason),
  the priority of the task locking that mutex can be lowered to the highest of the following three priorities.
  Whether this lowering takes place is implementation-dependent.
    a. The highest priority among the current priorities of tasks waiting to lock the mutex.
    b. The highest priority among all the other mutexes locked by the task currently locking this mutex.
    c. The base priority of the locking task.

- **Priority ceiling mutex (TA_CEILING attribute)**
  
  If the invoking task obtains a lock and its current priority is lower than the mutex ceiling priority, the priority
  of the invoking task is raised to the mutex ceiling priority.

---

\(^1\) Base priority: The task priority before it is automatically raised by the mutex. This is the priority last set by `tk_chg_pri` (including
while the mutex is locked), or if `tk_chg_pri` has never been issued, the priority that was set when the task was created.
4.5.1.4  tk_loc_mtx_u - Lock Mutex (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_loc_mtx_u(ID mtxid, TMO_U tmout_u);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mtxid</th>
<th>Mutex ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout (in microseconds)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

- E_OK  Normal completion
- E_ID  Invalid ID number (mxid 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_u ≤ (-2))
- E_DLT  The object being waited for was deleted (the mutex was deleted while waiting for a lock)
- E_RLWAI Waiting state released (tk_rel_wai received in waiting state)
- E_DISWAI Wait released due to disabling of wait
- E_TMOUT Polling failed or timeout
- E_CTX  Context error (issued from task-independent portion, or in dispatch disabled state)
- E_ILUSE Illegal use (multiple lock, or upper priority limit exceeded)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

| TK_SUPPORT_USEC | Support of microsecond |

Description

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_loc_mtx.

The specification of this system call is same as that of tk_loc_mtx, except that the parameter is replaced with tmout_u. For more details, see the description of tk_loc_mtx.
4.5.1.5  tk_unl_mtx - Unlock Mutex

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

| ID      | mtxid | Mutex ID | Mutex ID |

Return Parameter

| ER   | ercd  | Error Code | 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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Unlocks the mutex specified in \(mtxid\).

If there are tasks waiting to lock the mutex, the WAITING state of the task at the head of the queue for that
mutex is released and that task locks the mutex.

If a mutex that was not locked by the invoking task is specified, error code E_ILUSE is returned.

Additional Notes

If the unlocked mutex is a priority inheritance mutex (TA_INHERIT) or priority ceiling mutex (TA_CEILING), task
priority must be lowered as follows.

If as a result of this operation the invoking task no longer has any locked mutexes, the invoking task priority
is lowered to its base priority.

If the invoking task continues to have locked mutexes after the operation above, the invoking task priority is
lowered to whichever of the following priority is highest.
a. The highest priority among the current priority of the tasks in the queue of the mutex with the \texttt{TA\_INHERIT} attribute locked by the invoking task

b. The highest priority among the ceiling priority of the mutexes with the \texttt{TA\_CEILING} attribute locked by the invoking task

c. Base priority of the invoking task

Note that the lowering of priority when locked mutexes remain is implementation-dependent.

If a task terminates (goes to DORMANT state or NON-EXISTENT state) without explicitly unlocking mutexes, all its locked mutexes are automatically unlocked by $\mu$T-Kernel.
4.5.1.6  tk_ref_mtx - Refer Mutex Status

C Language Interface

#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mtxid</th>
<th>Mutex ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RMTX*</td>
<td>pk_rmtx</td>
<td>Packet to Return Mutex Status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

pk_rmtx Detail:

<table>
<thead>
<tr>
<th>ID</th>
<th>exinf</th>
<th>Extended Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>htsk</td>
<td>Locking Task ID</td>
</tr>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Lock Waiting Task ID</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

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 (invalid pk_rmtx) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

References the status of the mutex specified in mtxid, passing in the return parameters the task currently locking the mutex (htsk), tasks waiting to lock the mutex (wtsk), and extended information (exinf).

htsk indicates the ID of the task locking the mutex. If no task is locking it, htsk = 0 is returned.

wtsk indicates the ID of a task waiting to lock the mutex. If there are two or more such tasks, the ID of the task at the head of the queue is returned. If there are no waiting tasks, wtsk = 0 is returned.

If the specified mutex does not exist, error code E_NOEXS is returned.
4.5.2 Message Buffer

A message buffer is an object for achieving synchronization and communication by the passing of variable-size messages. Functions are provided for creating and deleting a message buffer, sending and receiving messages using a message buffer, and referencing message buffer status. A message buffer is an object identified by an ID number. The ID number for the message buffer is called a message buffer ID.

A message buffer keeps a queue of tasks waiting to send a message (send queue) and a queue of tasks waiting for receive a message (receive queue). It also has a message buffer space for holding sent messages. The message sender (the side posting event notification) copies a message it wants to send to the message buffer. If there is insufficient space in the message buffer area, the task trying to send the message is queued for sending until enough space is available.

A task waiting to send a message to the message buffer is put in the send queue. On the message receive side (waiting for event notification), one message is fetched from the message buffer. If the message buffer has no messages, the task enters WAITING state until the next message is sent. A task waiting for receiving a message from a message buffer is put in the receive queue of that message buffer.

A synchronous message function can be realized by setting the message buffer space size to 0. In that case both the sending task and receiving task wait for a system call to be invoked by each other, and the message is passed when both sides issue system calls.

Additional Notes

The message buffer behavior when the size of the message buffer space is set to 0 is explained here using the example in Figure 4.3, “Synchronous Communication by Message Buffer”. In this example Task A and Task B run asynchronously.

- If Task A calls `tk_snd_mbf` first, it goes to WAITING state until Task B calls `tk_rcv_mbf`. In this case Task A is put in the message buffer send queue [Figure 4.3, “Synchronous Communication by Message Buffer” (a)].
- If Task B calls `tk_rcv_mbf` first, on the other hand, Task B goes to WAITING state until Task A calls `tk_snd_mbf`. Task B is put in the message buffer receive queue [Figure 4.3, “Synchronous Communication by Message Buffer” (b)].
- At the point where both Task A has called `tk_snd_mbf` and Task B has called `tk_rcv_mbf`, a message is passed from Task A to Task B; Thereafter both tasks enter a run state.

Tasks waiting to send to a message buffer send messages in their queued order. Suppose Task A wanting to send a 40-byte message to a message buffer, and Task B wanting to send a 10-byte message, are queued in that order. If another task receives a message opening 20 bytes of space in the message buffer, Task B is still required to wait until Task A sends its message.

A message buffer is used to pass variable-size messages by copying them. It is the copying of messages that makes this function different from the mailbox function. It is assumed that the message buffer will be implemented as a ring buffer.
Figure 4.3: Synchronous Communication by Message Buffer
4.5.2.1 tk_cre_mbf - Create Message Buffer

C Language Interface

#include <tk/tkernel.h>

ID mbfid = tk_cre_mbf(CONST T_CMBF *pk_cmbf);

Parameter

<table>
<thead>
<tr>
<th>CONST T_CMBF</th>
<th>pk_cmbf</th>
<th>Packet to Create Message Buffer</th>
<th>Message buffer creation information</th>
</tr>
</thead>
</table>

pk_cmbf Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR</td>
<td>mbfatr</td>
<td>Message Buffer Attribute</td>
<td>Message buffer attribute</td>
</tr>
<tr>
<td>SZ</td>
<td>bufsz</td>
<td>Buffer Size</td>
<td>Message buffer size (in bytes)</td>
</tr>
<tr>
<td>INT</td>
<td>maxmsz</td>
<td>Max Message Size</td>
<td>Maximum message size (in bytes)</td>
</tr>
<tr>
<td>UB</td>
<td>dname[8]</td>
<td>DS Object name</td>
<td>DS object name</td>
</tr>
<tr>
<td>void*</td>
<td>bufptr</td>
<td>Buffer Pointer</td>
<td>User buffer pointer</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbfid</th>
<th>Message Buffer ID</th>
<th>Message buffer ID or Error Code</th>
</tr>
</thead>
</table>

Error Code

| E_NOMEM | Insufficient memory (memory for control block or ring buffer area cannot be allocated) |
| E_LIMIT | Number of message buffers exceeds the system limit |
| E_RSATR | Reserved attribute (mbfatr is invalid or cannot be used) |
| E_PAR   | Parameter error (pk_cmbf is illegal, bufsz, maxmsz is negative or invalid, bufptr is illegal) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

| TK_SUPPORT_USERBUF | Support for specifying TA_USERBUF for message buffer attribute |
| TK_SUPPORT_AUTOBUF| Automatic buffer allocation is supported (by not specifying TA_USERBUF to message buffer attribute) |
| TK_SUPPORT_DISWAI | Support for specifying TA_NODISWAI (reject request to disable wait) to message buffer attribute |
| TK_SUPPORT_DSNAME | Support for specifying TA_DSNAME for message buffer attribute |
Description

Creates a message buffer, assigning to it a message buffer ID. This system call allocates a control block to the created message buffer. Based on the information specified in `bufsz`, it allocates a ring buffer area for message queue use (for messages waiting to be received).

A message buffer is an object for managing the sending and receiving of variable-size messages. If differs from a mailbox (mbx) in that the contents of the variable-size messages are copied when the message is sent and received. It also has a function for putting the sending task in WAITING state when the buffer is full.

`exinf` can be used freely by the user to set miscellaneous information about the created message buffer. The information set in this parameter can be referenced by `tk_ref_mbf`. If a larger area is needed for indicating user information, or if the information may need to be changed after the message buffer is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in `exinf`. The kernel pays no attention to the contents of `exinf`.

`mbfatr` indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of `mbfatr` is specified as follows.

```plaintext
mbfatr := (TA_TFIFO || TA_TPRI) | [TA_DSNAME] | [TA_USERBUF] | [TA_NODISWAI]
```

- **TA_TFIFO**: Tasks waiting on call are queued in FIFO order
- **TA_TPRI**: Tasks waiting on call are queued in priority order
- **TA_DSNAME**: Specifies DS object name
- **TA_USERBUF**: Support of user-specified memory area as message buffer area
- **TA_NODISWAI**: Disabling of wait by tk_dis_wai is prohibited

The queuing order of tasks waiting for sending a message when the buffer is full can be specified in `TA_TFIFO` or `TA_TPRI`. If the attribute is `TA_TFIFO`, tasks are ordered by FIFO, whereas `TA_TPRI` specifies queuing of tasks in order of their priority setting. Messages themselves are queued in FIFO order only.

Tasks waiting for receiving a message from a message buffer are queued in FIFO order only.

When `TA_USERBUF` is specified, `bufptr` becomes effective, and the memory area starting at `bufptr` and containing `bufsz` octets is used as message buffer area. In this case, the message buffer area is not provided by he OS, but must be allocated by the caller. When `TA_USERBUF` is not specified, `bufptr` is ineffective, and the message buffer area is provided by the kernel.

When `TA_DSNAME` is specified, `dsname` is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, `td_ref_dsname` and `td_set_dsname`. For more details, see the description of `td_ref_dsname` and `td_set_dsname`. If `TA_DSNAME` is not specified, `dsname` is ignored. Then `td_ref_dsname` and `td_set_dsname` return E_OBJ error.

```plaintext
#define TA_TFIFO 0x00000000 /* manage task queue by FIFO */
#define TA_TPRI 0x00000001 /* manage task queue by priority */
#define TA_USERBUF 0x00000020 /* Use user-specified buffer */
#define TA_DSNAME 0x00000040 /* DS object name */
#define TA_NODISWAI 0x00000080 /* reject request to disable wait */
```

Additional Notes

When there are multiple tasks waiting to send messages, the order in which their messages are sent when buffer space becomes available is always in their queued order.

If, for example, a Task A wanting to send a 30-byte message is queued with a Task B wanting to send a 10-byte message, in the order A-B, even if 20 bytes of message buffer space becomes available, Task B never sends its message before Task A.

The ring buffer in which messages are queued also contains information for managing each message. For this reason the total size of queued messages will ordinarily not be identical to the ring buffer size specified
in \texttt{bufsz}. Normally the total message size will be smaller than \texttt{bufsz}. In this sense \texttt{bufsz} does not strictly represent the total message capacity.

It is possible to create a message buffer with \texttt{bufsz} = 0. In this case communication using the message buffer is completely synchronous between the sending and receiving tasks. That is, if either \texttt{tk_snd_mbf} or \texttt{tk_rcv_mbf} is executed ahead of the other, the task executing the first system call goes to WAITING state. When the other system call is executed, the message is passed (copied), then both tasks resume running.

In the case of a \texttt{bufsz} = 0 message buffer, the specific functioning is as follows.

1. In Figure 4.4, "Synchronous Communication Using Message Buffer of \texttt{bufsz} = 0", Task A and Task B operate asynchronously. If Task A arrives at point (1) first and executes \texttt{tk_snd_mbf}(mbfid), Task A goes to send waiting state until Task B arrives at point (2). If \texttt{tk_ref_tsk} is issued for Task A in this state, \texttt{tskwait}=TTW\_SMBF is returned. If, on the other hand, Task B gets to point (2) first and calls \texttt{tk_rcv_mbf}(mbfid), Task B goes to receive waiting state until Task A gets to point (1). If \texttt{tk_ref_tsk} is issued for Task B in this state, \texttt{tskwait}=TTW\_RMBF is returned.

2. At the point where both Task A has executed \texttt{tk_snd_mbf}(mbfid) and Task B has executed \texttt{tk_rcv_mbf}(mbfid), a message is passed from Task A to Task B, their wait states are released and both tasks resume running.

![Figure 4.4: Synchronous Communication Using Message Buffer of \texttt{bufsz} = 0](image)

**Porting Guideline**

Note that member, \texttt{maxmsz}, of \texttt{T\_CMBF} is INT type, and its value range is implementation-dependent, so care must be taken.

The T-Kernel 2.0 specification does not define \texttt{TA\_USERBUF} and its associated notion of \texttt{bufptr}. So if this feature is used, a modification is necessary to port the software to T-Kernel 2.0. However, if \texttt{bufsz} is properly set already, simply removing \texttt{TA\_USERBUF} and \texttt{bufptr} will complete the modification for porting.
4.5.2.2  tk_del_mbf - Delete Message Buffer

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbfid</th>
<th>Message Buffer ID</th>
<th>Message buffer ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes the message buffer specified in mbfid.

Issuing this system call releases the corresponding message buffer and control block memory space, as well as the message buffer space.

This system call completes normally even if there were tasks queued in the message buffer for message receipt or message sending, but error code E_DLT is returned to the tasks in WAITING state. If there are messages left in the message buffer when it is deleted, the message buffer is deleted anyway. No error code is returned and the messages are discarded.
4.5.2.3  tk_snd_mbf - Send Message to Message Buffer

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_snd_mbf(ID mbfid, CONST void *msg, INT msgsz, TMO tmout);

Parameter

| ID  | mbfid  | Message Buffer ID
|-----|--------|-------------------|
| CONST void* | msg | Send Message
| INT  | msgsz  | Send Message Size
| TMO  | tmout  | Timeout

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 \( \leq 0, \text{msgsz} > \text{maxmsz}, \text{invalid msg}, \text{or tmout} \leq (-2)) |
| E_DLT   | The object being waited for was deleted (message buffer was deleted while waiting) |
| E_RLWAI | Waiting state released (tk_rel_wai received in waiting state) |
| E_DISWAI| Wait released due to disabling of wait |
| E_TMOUT | Polling failed or timeout |
| E_CTX   | Context error (issued from task-independent portion, or in dispatch disabled state) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO(*) Available in some circumstances</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

\texttt{tk_snd_mbf} sends the message at the address specified in \texttt{msg} to the message buffer indicated in \texttt{mbfid}. The message size is specified in \texttt{msgsz}. This system call copies \texttt{msgsz} bytes starting from \texttt{msg} to the message queue of message buffer \texttt{mbfid}. The message queue is assumed to be implemented as a ring buffer.

If \texttt{msgsz} is larger than the \texttt{maxmsz} specified in \texttt{tk_cre_mbf}, error code \texttt{E_PAR} is returned.

If there is not enough available buffer space to accommodate message \texttt{msg} in the message queue, the task issuing this system call goes to send waiting state and is put in the send queue of the message buffer waiting for buffer space to become available. Waiting tasks are queued in either FIFO or priority order, depending on the attribute specified in \texttt{tk_cre_mbf}. 


A maximum wait time (timeout) can be set in \texttt{tmout}. The time unit for \texttt{tmout} is the same as that for system time (= 1 ms). If the \texttt{tmout} time elapses before the wait release condition is met (before there is sufficient buffer space), the system call terminates, returning timeout error code E\_TMOUT.

When \texttt{TMO\_POL=0} is specified in \texttt{tmout}, it means 0 is specified as the timeout value, and if there is not enough buffer space, then E\_TMOUT is returned without entering WAITING state. When \texttt{TMO\_FEVR=(-1)} is specified in \texttt{tmout}, this means infinity was specified as the timeout value, and the task continues to wait for buffer space to become available, without timing out.

A message of size 0 cannot be sent. When \texttt{msgsz \leq 0}, error code E\_PAR is returned.

When this system call is invoked from a task-independent portion or in dispatch disabled state, error code E\_CTX is returned; but in the case of \texttt{tmout = TMO\_POL}, there may be implementations where execution from a task-independent portion or in dispatch disabled state is possible.

\textbf{Porting Guideline}

Note that \texttt{msgsz} is INT type, and its value range is implementation-dependent, so care must be taken. For example, there is a chance that the message size that can sent at once might be limited to 32767 octets on 16-bit CPU.
4.5.2.4  tk_snd_mbf_u - Send Message to Message Buffer (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_snd_mbf_u(ID mbfid, CONST void *msg, INT msgsz, TMO_U tmout_u);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbfid</th>
<th>Message Buffer ID</th>
<th>Message buffer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST void*</td>
<td>msg</td>
<td>Send Message</td>
<td>Start address of send message</td>
</tr>
<tr>
<td>INT</td>
<td>msgsz</td>
<td>Send Message Size</td>
<td>Send message size (in bytes)</td>
</tr>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout</td>
<td>Timeout (in microseconds)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

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 > maxmsg, invalid msg, or tmout_u ≤ (-2))
- E_DLT  The object being waited for was deleted (message buffer was deleted while waiting)
- E_RLWAI  Waiting state released (tk_rel_wai received in waiting state)
- E_DISWAI  Wait released due to disabling of wait
- E_TMOUT  Polling failed or timeout
- E_CTX  Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO(* Available in certain circumstance)</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_USEC  Support of microsecond

Description

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_snd_mbf.

The specification of this system call is same as that of tk_snd_mbf, except that the parameter is replaced with tmout_u. For more details, see the description of tk_snd_mbf.
Porting Guideline

Note that \texttt{msgsz} is INT type, and its value range is implementation-dependent, so care must be taken. For example, there is a chance that the message size that can sent at once might be limited to 32767 octets on 16-bit CPU.
4.5.2.5  tk_rcv_mbf - Receive Message from Message Buffer

C Language Interface

```c
#include <tk/tkernel.h>

INT msgsz = tk_rcv_mbf(ID mbfid, void *msg, TMO tmout);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbfid</th>
<th>Message Buffer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>msg</td>
<td>Receive Message</td>
</tr>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>msgsz</th>
<th>Receive Message Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>Error Code</td>
<td>Received message size (in bytes)</td>
</tr>
</tbody>
</table>

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 (invalid msg, or tmout ≤ (-2))
- **E_DLT**: The object being waited for was deleted (message buffer was deleted while waiting)
- **E_RLWAI**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

`tk_rcv_mbf` receives a message from the message buffer specified in mbfid, copying it in the location specified in msg. This system call copies the contents of the first queued message in the message buffer specified in mbfid, and copies it to an area of msgsz bytes starting at address msg.

If no message has been sent to the message buffer specified in mbfid (the message queue is empty), the task issuing this system call goes to WAITING state and is put in the receive queue of the message buffer to wait for message arrival. Tasks in the receive queue are ordered by FIFO only.

A maximum wait time (timeout) can be set in tmout. The time unit for tmout is the same as that for system time (= 1 ms). If the tmout time elapses before the wait release condition is met (before a message arrives), the system call terminates, returning timeout error code E_TMOUT.
When `TMO_POL=0` is set in `tmout`, this means 0 was specified as the timeout value, and `E_TMOUT` is returned without entering WAITING state even if there is no message. When `TMO_FEV=-1` is set in `tmout`, this means infinity was specified as the timeout value, and the task continues to wait for message arrival without timing out.
4.5.2.6 \texttt{tk_rcv_mbf_u} - Receive Message from Message Buffer (Microseconds)

\textbf{C Language Interface}

\texttt{#include <tk/tkernel.h>}

\texttt{INT msgsz = tk_rcv_mbf_u(ID mbfid, void *msg, TMO_U tmout_u);}  

\textbf{Parameter}

<table>
<thead>
<tr>
<th>ID</th>
<th>\texttt{mbfid}</th>
<th>Message Buffer ID</th>
<th>Message buffer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>\texttt{msg}</td>
<td>Receive Message</td>
<td>Address of the receive message</td>
</tr>
<tr>
<td>TMO_U</td>
<td>\texttt{tmout_u}</td>
<td>Timeout</td>
<td>Timeout (in microseconds)</td>
</tr>
</tbody>
</table>

\textbf{Return Parameter}

<table>
<thead>
<tr>
<th>INT</th>
<th>\texttt{msgsz}</th>
<th>Receive Message Size</th>
<th>Received message size (in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>or Error Code</td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

\textbf{Error Code}

- \texttt{E\_ID}  
  Invalid ID number (\texttt{mbfid} is invalid or cannot be used)

- \texttt{E\_NOEXS}  
  Object does not exist (the message buffer specified in \texttt{mbfid} does not exist)

- \texttt{E\_PAR}  
  Parameter error (invalid \texttt{msg}, or \texttt{tmout\_u} \(\leq (-2)\))

- \texttt{E\_DLT}  
  The object being waited for was deleted (message buffer was deleted while waiting)

- \texttt{E\_RLWAI}  
  Waiting state released (\texttt{tk_rel_wai} received in waiting state)

- \texttt{E\_DISWAI}  
  Wait released due to disabling of wait

- \texttt{E\_TMOUT}  
  Polling failed or timeout

- \texttt{E\_CTX}  
  Context error (issued from task-independent portion, or in dispatch disabled state)

\textbf{Valid Context}

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

\textbf{Related Service Profile Items}

Only when all the service profile items below are set to be effective, this system call can be used.

- \texttt{TK\_SUPPORT\_USEC}  
  Support of microsecond

\textbf{Description}

This system call takes 64-bit \texttt{tmout\_u} in microseconds instead of the parameter \texttt{tmout} of \texttt{tk_rcv_mbf}.

The specification of this system call is same as that of \texttt{tk_rcv_mbf}, except that the parameter is replaced with \texttt{tmout\_u}. For more details, see the description of \texttt{tk_rcv_mbf}. 
4.5.2.7 tk_ref_mbf - Reference Message Buffer Status

C Language Interface

```c
#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbfid</th>
<th>Message Buffer ID</th>
<th>Message buffer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RMBF*</td>
<td>pk_rmbf</td>
<td>Packet to Return Message Buffer Status</td>
<td>Pointer to the area to return the message buffer status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>pk_rmbf</td>
<td>Detail:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
<td>Extended information</td>
</tr>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Waiting Task ID</td>
<td>Receive waiting task ID</td>
</tr>
<tr>
<td>ID</td>
<td>stsk</td>
<td>Send Waiting Task ID</td>
<td>Send waiting task ID</td>
</tr>
<tr>
<td>INT</td>
<td>msgsz</td>
<td>Message Size</td>
<td>Size of the next message to be received (in bytes)</td>
</tr>
<tr>
<td>SZ</td>
<td>frbufsz</td>
<td>Free Buffer Size</td>
<td>Free buffer size (in bytes)</td>
</tr>
<tr>
<td>INT</td>
<td>maxmsz</td>
<td>Maximum Message Size</td>
<td>Maximum message size (in bytes)</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

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 (invalid pk_rmbf)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

References the status of the message buffer specified in mbfid, passing in the return parameters the send waiting task ID (stsk), the size of the next message to be received (msgsz), free buffer size (frbufsz), maximum message size (maxmsz), receive waiting task ID (wtsk), and extended information (exinf).
\textbf{wtsk} indicates the ID of a task waiting to receive a message from the message buffer. \textbf{stsk} indicates the ID of a task waiting to send a message to the message buffer. If multiple tasks are waiting in the message buffer queues, the ID of the task at the head of the queue is returned. If no tasks are waiting, 0 is returned.

If the specified message buffer does not exist, error code E_NOEXXS is returned.

The size of the message at the head of the queue (the next message to be received) is returned in \textbf{msgsz}. If there are no queued messages, \textbf{msgsz} = 0 is returned. A message of size 0 cannot be sent.

At least one of \textbf{msgsz} = 0 and \textbf{wtsk} = 0 is always true for this system call.

\textbf{frbufsz} indicates the free space in the ring buffer of which the message queue consists. This value indicates the approximate size of messages that can be sent.

The maximum message size as specified in \textbf{tk_cre_mbf} is returned to \textbf{maxmsz}. 
4.6 Memory Pool Management Functions

Memory pool management functions are for managing memory pools and allocating memory blocks by using software.

There are fixed-size memory pools and variable-size memory pools, which are considered separate objects and require separate sets of system calls for their operation. Memory blocks allocated from a fixed-size memory pool are all of one fixed size, whereas memory blocks from a variable-size memory pool can be of various sizes.
4.6.1 Fixed-size Memory Pool

A fixed-size memory pool is an object used for dynamic management of fixed-size memory blocks. Functions are provided for creating and deleting a fixed-size memory pool, getting and returning memory blocks in a fixed-size memory pool, and referencing the status of a fixed-size memory pool. A fixed-size memory pool is an object identified by an ID number. The ID number for the fixed-size memory pool is called a fixed-size memory pool ID.

A fixed-size memory pool has a memory space used as the fixed-size memory pool (called a fixed-size memory pool area or simply memory pool area), and a queue for tasks waiting for memory block allocation. A task wanting to allocate a memory block from a fixed-size memory pool that lacks sufficient available memory space goes to WAITING state for fixed-size memory block until memory blocks are returned to the pool. A task in this state is put in the task queue of the fixed-size memory pool.

Additional Notes
When memory blocks of various sizes are needed from fixed-size memory pools, it is necessary to provide multiple memory pools of different sizes.
4.6.1.1 tk_cre_mpf - Create Fixed-size Memory Pool

C Language Interface

```c
#include <tk/tkernel.h>

ID mpfid = tk_cre_mpf(CONST T_CMPF *pk_cmpf);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_CMBX* pk_cmpf</td>
<td>Packet to Create Memory Pool</td>
</tr>
<tr>
<td></td>
<td>Information about the fixed-size memory pool to be created</td>
</tr>
</tbody>
</table>

`pk_cmpf` Detail:

- `void* exinf` Extended Information
- `ATR mpfatr` Memory Pool Attribute
- `SZ mpfcnt` Memory Pool Block Count
- `SZ blfsz` Memory Block Size
- `UB dsname[8]` DS Object name
- `void* bufptr` Buffer Pointer

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID mpfid</td>
<td>Memory Pool ID</td>
</tr>
<tr>
<td></td>
<td>or Error Code</td>
</tr>
</tbody>
</table>

Error Code

- **E_NOMEM** Insufficient memory (memory for control block or memory pool area cannot be allocated)
- **E_LIMIT** Number of fixed-size memory pools exceeds the system limit
- **E_RSATR** Reserved attribute (mpfatr is invalid or cannot be used)
- **E_PAR** Parameter error (pk_cmpf is illegal, mpfcnt, blfsz is negative or invalid, or bufptr is illegal)

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

- **TK_SUPPORT_USERBUF** Support for specifying `TA_USERBUF` for fixed-size memory pool attribute
- **TK_SUPPORT_AUTOBUF** Automatic buffer allocation is supported (by not specifying `TA_USERBUF` to fixed-size memory pool attribute)
- **TK_SUPPORT_DISWAI** Support for specifying `TA_NODISWAI` (reject request to disable wait) to fixed-size memory pool attribute
Support for specifying TA_DSNAME for fixed-size memory pool attribute

Description

Creates a fixed-size memory pool, assigning to it a fixed-size memory pool ID. This system call allocates a memory space for use as a memory pool based on the information specified in parameters mpfcnt and blfsz, and assigns a control block to the memory pool. A memory block of size blfsz can be allocated from the created memory pool by calling the `tk_get_mpf` system call.

`exinf` can be used freely by the user to set miscellaneous information about the created memory pool. The information set in this parameter can be referenced by `tk_ref_mpf`. If a larger area is needed for indicating user information, or if the information may need to be changed after the message buffer is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in `exinf`. The kernel pays no attention to the contents of `exinf`.

`mpfatr` indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of `mpfatr` is as follows.

```
TA_TFIFO Tasks waiting for memory allocation are queued in FIFO order
TA_TPRI Tasks waiting for memory allocation are queued in priority order
TA_RNGn Specifies DS object name
TA_USERBUF Support of user-specified memory area as memory pool area
TA_NODISWAI Disabling of wait by `tk_dis_wai` is prohibited
```

The queuing order of tasks waiting for memory block allocation from a memory pool can be specified in TA_TFIFO or TA_TPRI. If the attribute is TA_TFIFO, tasks are ordered by FIFO, whereas TA_TPRI specifies queuing of tasks in order of their priority setting.

TA_RNGn is specified to limit the protection levels from which memory can be accessed. Only tasks running at the same or higher protection level than the one specified can access the allocated memory. If a task running at a lower protection level attempts an access, a CPU protection fault exception is raised. For example, memory allocated from a memory pool specified as TA_RNG1 can be accessed by tasks running at levels TA_RNG0 or TA_RNG2, but not by tasks running at levels TA_RNG2 or TA_RNG3.

When TA_DSNAME is specified, dsname is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, `td_ref_dsname` and `td_set_dsname`. For more details, see the description of `td_ref_dsname` and `td_set_dsname`. If TA_DSNAME is not specified, dsname is ignored. Then `td_ref_dsname` and `td_set_dsname` return E_OBJ error.

Additional Notes

In the case of a fixed-size memory pool, separate memory pools must be provided for different block sizes. That is, if various memory block sizes are required, memory pools must be created for each block size.
For the sake of portability, the $\text{TA}_{\text{RNG}}n$ attribute must be accepted even by a system with a single CPU’s operating mode. It is possible, for example, to handle all $\text{TA}_{\text{RNG}}n$ as equivalent to $\text{TA}_{\text{RNG}}0$, but error must not be returned.

**Porting Guideline**

The T-Kernel 2.0 specification does not define $\text{TA}_{\text{USERBUF}}$ and its associated notion of $\text{bufptr}$. So if this feature is used, a modification is necessary to port the software to T-Kernel 2.0. However, if $\text{mpcnt}$ and $\text{blfsz}$ is properly set already, simply removing $\text{TA}_{\text{USERBUF}}$ and $\text{bufptr}$ will complete the modification for porting.
4.6.1.2  tk_del_mpf - Delete Fixed-size Memory Pool

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_del_mpf(ID mpfid);

Parameter

| ID   | mpfid | Memory Pool ID | Fixed-size memory pool ID |

Return Parameter

| ER   | ercd  | Error Code | Error code |

Error Code

- E_OK: Normal completion
- E_ID: Invalid ID number (mpfid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the fixed-size memory pool specified in mpfid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes the fixed-size memory pool specified in mpfid.

No check or notification is made as to whether there are tasks using memory allocated from this memory pool. The system call completes normally even if not all blocks have been returned to the pool.

Issuing this system call releases the memory pool ID number, the control block memory space and the memory pool space itself.

This system call completes normally even if there are tasks waiting for memory block allocation from the deleted memory pool, but error code E_DLT is returned to the tasks in WAITING state.
4.6.1.3 tk_get_mpf - Get Fixed-size Memory Block

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_get_mpf(ID mpfid, void **p_blf, TMO tmout);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mpfid</th>
<th>Memory Pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>p_blf</td>
<td>Pointer to Block Start Address</td>
</tr>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>blf</td>
<td>Memory block start address</td>
</tr>
</tbody>
</table>

Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (mpfid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the fixed-size memory pool specified in mpfid does not exist)
- **E_PAR**: Parameter error (tmout \(\leq (-2)\))
- **E_DLT**: The object being waited for was deleted (the memory pool was deleted while waiting)
- **E_RLWAI**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Gets a memory block from the fixed-size memory pool specified in mpfid. The start address of the allocated memory block is returned in blf. The size of the allocated memory block is the value specified in the blfsz parameter when the fixed-size memory pool was created.

The allocated memory is not cleared to zero, and the memory block contents are indeterminate.

If a block cannot be allocated from the specified memory pool, the task that issued tk_get_mpf is put in the queue of tasks waiting for memory allocation from that memory pool, and waits until memory can be allocated.
A maximum wait time (timeout) can be set in `tmout`. The time unit for `tmout` is the same as that for system time (= 1 ms). If the `tmout` time elapses before the wait release condition is met (memory space does not become available), the system call terminates, returning timeout error code E_TMOUT.

When `TMO_POL=0` is set in `tmout`, this means 0 was specified as the timeout value, and E_TMOOUT is returned without entering WAITING state even if memory cannot be allocated.

When `TMO_FEVR=-1` is set in `tmout`, this means infinity was specified as the timeout value, and the task continues to wait for memory allocation without timing out.

The queuing order of tasks waiting for memory block allocation is either FIFO or task priority order, depending on the memory pool attribute.
4.6.1.4  tk_get_mpf_u - Get Fixed-size Memory Block (Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_get_mpf_u(ID mpfid, void **p_blf, TMO_U tmout_u);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID mpfid</td>
<td>Memory Pool ID</td>
</tr>
<tr>
<td>void** p_blf</td>
<td>Pointer to Block Start Address</td>
</tr>
<tr>
<td>TMO_U tmout_u</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER ercd</td>
<td>Error Code</td>
</tr>
<tr>
<td>void** blf</td>
<td>Block Start Address</td>
</tr>
</tbody>
</table>

Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (mpfid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the fixed-size memory pool specified in mpfid does not exist)
- **E_PAR**: Parameter error (tmout_u ≤ (-2))
- **E_DLT**: The object being waited for was deleted (the memory pool was deleted while waiting)
- **E_RLWAI**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_USEC**: Support of microsecond

Description

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_get_mpf.

The specification of this system call is same as that of tk_get_mpf, except that the parameter is replaced with tmout_u. For more details, see the description of tk_get_mpf.
4.6.1.5  tk_rel_mpf - Release Fixed-size Memory Block

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_rel_mpf(ID mpfid, void *blf);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mpfid</th>
<th>Memory Pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>blf</td>
<td>Block Start Address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fixed-size memory pool ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory block start address</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>Invalid ID number (mpfid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the fixed-size memory pool specified in mpfid does not exist)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (blf is invalid, or block returned to wrong memory pool)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Returns the memory block specified in blf to the fixed-size memory pool specified in mpfid.

Executing tk_rel_mpf may enable memory block acquisition by another task waiting to allocate memory from the memory pool specified in mpfid, releasing the WAITING state of that task.

When a memory block is returned to a fixed-size memory pool, it must be the same fixed-size memory pool from which the block was allocated. If an attempt to return a memory block to a different memory pool is detected, error code E_PAR is returned. Whether this error detection is performed or not is implementation-dependent.
4.6.1.6  tk_ref_mpf - Reference Fixed-size Memory Pool Status

C Language Interface

#include <tk/tkernel.h>

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

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mpfid</th>
<th>Memory Pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RMPF*</td>
<td>pk_rmpf</td>
<td>Packet to Return Memory Pool Status</td>
</tr>
</tbody>
</table>

Fixed-size memory pool ID

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>Invalid ID number (mpfid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the fixed-size memory pool specified in mpfid does not exist)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (invalid pk_rmpf)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

References the status of the fixed-size memory pool specified in mpfid, passing in return parameters the current free block count (frbcnt), waiting task ID (wtsk), and extended information (exinf).

wtsk indicates the ID of a task waiting for memory block allocation from this fixed-size memory pool. If multiple tasks are waiting for the fixed-size memory pool, the ID of the task at the head of the queue is returned. If there are no waiting tasks, wtsk = 0 is returned.

If the fixed-size memory pool specified with tk_ref_mpf does not exist, error code E_NOEXS is returned.

At least one of frbcnt = 0 and wtsk = 0 is always true for this system call.
Additional Notes

Whereas `frsz` returned by `tk_ref_mpl` gives the total free memory size in bytes, `frbcnt` returned by `tk_ref_mpf` gives the number of unused memory blocks.
4.6.2 Variable-size Memory Pool

A variable-size memory pool is an object for dynamically managing memory blocks of any size. Functions are provided for creating and deleting a variable-size memory pool, allocating and returning memory blocks in a variable-size memory pool, and referencing the status of a variable-size memory pool. A variable-size memory pool is an object identified by an ID number. The ID number for the variable-size memory pool is called a variable-size memory pool ID.

A variable-size memory pool has a memory space used as the variable-size memory pool (called a variable-size memory pool area or simply memory pool area), and a queue for tasks waiting for memory block allocation. A task wanting to allocate a memory block from a variable-size memory pool that lacks sufficient available memory space goes to WAITING state for variable-size memory block until memory blocks are returned to the pool. A task in this state is put in the task queue of the variable-size memory pool.

Additional Notes
When tasks are waiting for memory block allocation from a variable-size memory pool, they are served in queued order. If, for example, Task A requesting a 400-byte memory block from a variable-size memory pool is queued along with Task B requesting a 100-byte block, in A-B order, then even if 200-byte block of space are free, Task B is made to wait until Task A has acquired the requested memory block.
4.6.2.1 tk_cre_mpl - Create Variable-size Memory Pool

C Language Interface

```
#include <tk/tkernel.h>
```

```
ID mplid = tk_cre_mpl(CONST T_CMPL *pk_cmpl);
```

Parameter

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_CMPL *</td>
<td>pk_cmpl</td>
</tr>
</tbody>
</table>

pk_cmpl Detail:

- `void* exinf` Extended Information
- `ATR mplatr` Memory Pool Attribute
- `SZ mpsz` Memory Pool Size
- `UB dsnname[8]` DS Object name
- `void* bufptr` Buffer Pointer

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>mplid</td>
</tr>
<tr>
<td>or</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

- `E_NOMEM` Insufficient memory (memory for control block or memory pool area cannot be allocated)
- `E_LIMIT` Number of variable-size memory pools exceeds the system limit
- `E_RSATR` Reserved attribute (mplatr is invalid or cannot be used)
- `E_PAR` Parameter error: (pk_cmpl is invalid, mpsz is negative or invalid, or bufptr is illegal)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

- `TK_SUPPORT_USERBUF` Support for specifying TA_USERBUF for variable-size memory pool attribute
- `TK_SUPPORT_AUTOBUF` Automatic buffer allocation is supported (by not specifying TA_USERBUF to variable-size memory pool attribute)
- `TK_SUPPORT_DISWAI` Support for specifying TA_NODISWAI (reject request to disable wait) to variable-size memory pool attribute
- `TK_SUPPORT_DSNAME` Support for specifying TA_DSNAME for variable-size memory pool attribute
Description

Creates a variable-size memory pool, assigning to it a variable-size memory pool ID. This system call allocates a memory space for use as a memory pool, based on the information in parameter mplsz, and assigns a control block to the memory pool.

exinf can be used freely by the user to set miscellaneous information about the created memory pool. The information set in this parameter can be referenced by tk_ref_mpl. If a larger area is needed for indicating user information, or if the information may need to be changed after the message buffer is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in exinf. The kernel pays no attention to the contents of exinf.

mplatr indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of mplatr is as follows.

mplatr := (TA_TFIFO || TA_TPRI) | [TA_DSNAME] | [TA_USERBUF] | [TA_NODISWAI] | (TA_RNG0 || TA_RNG1 || TA_RNG2 || TA_RNG3)

TA_TFIFO Tasks waiting for memory allocation are queued in FIFO order
TA_TPRI Tasks waiting for memory allocation are queued in priority order
TA_RNGn Memory access privilege is set to protection level n
TA_DSNAME Specifies DS object name
TA_USERBUF Support of user-specified memory area as memory pool area
TA_NODISWAI Disabling of wait by tk_dis_wai is prohibited

#define TA_TFIFO 0x00000000 /* manage task queue by FIFO */
#define TA_TPRI 0x00000001 /* manage task queue by priority */
#define TA_USERBUF 0x00000020 /* Use user-specified buffer */
#define TA_DSNAME 0x00000040 /* DS object name */
#define TA_RNG0 0x00000000 /* protection level 0 */
#define TA_RNG1 0x00000010 /* protection level 1 */
#define TA_RNG2 0x00000020 /* protection level 2 */
#define TA_RNG3 0x00000030 /* protection level 3 */

The queuing order of tasks waiting for memory block allocation from a memory pool can be specified in TA_TFIFO or TA_TPRI. If the attribute is TA_TFIFO, tasks are ordered by FIFO, whereas TA_TPRI specifies queuing of tasks in order of their priority setting.

When tasks are queued waiting for memory allocation, memory is allocated in the order of queuing. Even if other tasks in the queue are requesting smaller amounts of memory than the task at the head of the queue, they do not acquire memory blocks before the first task. If, for example, Task A requesting a 400-byte memory block from a variable-size memory pool is queued along with Task B requesting a 100-byte block, in A-B order, then even if 200-byte block of space are freed by tk_rel_mpl of another task, Task B is made to wait until Task A has acquired the requested memory block.

TA_RNGn is specified to limit the protection levels from which memory can be accessed. Only tasks running at the same or higher protection level than the one specified can access the allocated memory. If a task running at a lower protection level attempts an access, a CPU protection fault exception is raised. For example, memory allocated from a memory pool specified as TA_RNG1 can be accessed by tasks running at levels TA_RNG0 or TA_RNG1, but not by tasks running at levels TA_RNG2 or TA_RNG3.

When TA_DSNAME is specified, dsname is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, td_ref_dsname and td_set_dsname. For more details, see the description of td_ref_dsname and td_set_dsname. If TA_DSNAME is not specified, dsname is ignored. Then td_ref_dsname and td_set_dsname return E_OBJ error.
Additional Notes

If the task at the head of the queue waiting for memory allocation has its WAITING state forcibly released, or if a different task becomes the first in the queue as a result of a change in task priority, memory allocation is attempted to that task. If memory can be allocated, the WAITING state of that task is released. In this way it is possible under some circumstances for memory allocation to take place and task WAITING state to be released even when memory is not released by tk_rel_mpl.

For the sake of portability, the TA_RNGn attribute must be accepted even by a system with a single CPU’s operating mode. It is possible, for example, to handle all TA_RNGn as equivalent to TA_RNG0, but error must not be returned.

Rationale for the Specification

The capability of creating multiple variable-size memory pools can be used for memory allocation as needed for error handling or in emergent situations in programming, etc.

Porting Guideline

The T-Kernel 2.0 specification does not define TA_USERBUF and its associated notion of bufptr. So if this feature is used, a modification is necessary to port the software to T-Kernel 2.0. However, if mplsz is properly set already, simply removing TA_USERBUF and bufptr will complete the modification for porting.
4.6.2.2 tk_del_mpl - Delete Variable-size Memory Pool

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_del_mpl(ID mplid);

Parameter

| ID mplid | Memory Pool ID        | Variable-size memory pool ID |

Return Parameter

| ER ercd | Error Code | Error code |

Error Code

| E_OK  | Normal completion |
| E_ID  | Invalid ID number (mplid is invalid or cannot be used) |
| E_NOEXS | Object does not exist (the variable-size memory pool specified in mplid does not exist) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes the variable-size memory pool specified in mplid.

No check or notification is made as to whether there are tasks using memory allocated from this memory pool. The system call completes normally even if not all blocks have been returned to the pool.

Issuing this system call releases the memory pool ID number, the control block memory space and the memory pool space itself.

This system call completes normally even if there are tasks waiting for memory block allocation from the deleted memory pool, but error code E_DLT is returned to the tasks in WAITING state.
4.6.2.3  tk_get_mpl - Get Variable-size Memory Block

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_get_mpl(ID mplid, SZ blksz, void **p_blk, TMO tmout);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mplid</th>
<th>Memory Pool ID</th>
<th>Variable-size memory pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>blksz</td>
<td>Memory Block Size</td>
<td>Memory block size (in bytes)</td>
</tr>
<tr>
<td>void*</td>
<td>p_blk</td>
<td>Pointer to Block Start Address</td>
<td>Pointer to the area to return the block start address</td>
</tr>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
<td>Timeout (ms)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>blk</td>
<td>Block Start Address</td>
<td>Memory block start address</td>
</tr>
</tbody>
</table>

Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (mplid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the variable-size memory pool specified in mplid does not exist)
- **E_PAR**: Parameter error (tmout ≤ (-2))
- **E_DLT**: The object being waited for was deleted (the memory pool was deleted while waiting)
- **E_RLWAI**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

 Gets a memory block of size blksz (bytes) from the variable-size memory pool specified in mplid. The start address of the allocated memory block is returned in blk.

The allocated memory is not cleared to zero, and the memory block contents are indeterminate.

If memory cannot be allocated, the task issuing this system call enters WAITING state.
A maximum wait time (timeout) can be set in `tmout`. The time unit for `tmout` is the same as that for system time (= 1 ms). If the `tmout` time elapses before the wait release condition is met (memory space does not become available), the system call terminates, returning timeout error code E_TMOUT.

When `TMO_POL=0` is set in `tmout`, this means 0 was specified as the timeout value, and E_TMOUT is returned without entering WAITING state even if memory cannot be allocated.

When `TMO_FEVR=(-1)` is set in `tmout`, this means infinity was specified as the timeout value, and the task continues to wait for memory allocation without timing out.

The queuing order of tasks waiting for memory block allocation is either FIFO or task priority order, depending on the memory pool attribute.
4.6.2.4  tk_get_mpl_u - Get Variable-size Memory Block (Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_get_mpl_u(ID mplid, SZ blksz, void **p_blk, TMO_U tmout_u);
```

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>mplid</th>
<th>Parameters: Memory Pool ID</th>
<th>Variable-size memory pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>blksz</td>
<td>Parameters: Memory Block Size</td>
<td>Memory block size (in bytes)</td>
</tr>
<tr>
<td>void*</td>
<td>p_blk</td>
<td>Parameters: Pointer to Block Start Address</td>
<td>Pointer to the area to return the block start address blk</td>
</tr>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Parameters: Timeout</td>
<td>Timeout (in microseconds)</td>
</tr>
</tbody>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>blk</td>
<td>Block Start Address</td>
<td>Memory block start address</td>
</tr>
</tbody>
</table>

**Error Code**

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (mplid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the variable-size memory pool specified in mplid does not exist)
- **E_PAR**: Parameter error (tmout_u ≤ (-2))
- **E_DLT**: The object being waited for was deleted (the memory pool was deleted while waiting)
- **E_RLWAI**: Waiting state released (tk_rel_wai received in waiting state)
- **E_DISWAI**: Wait released due to disabling of wait
- **E_TMOUT**: Polling failed or timeout
- **E_CTX**: Context error (issued from task-independent portion, or in dispatch disabled state)

**Valid Context**

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_USEC**: Support of microsecond

**Description**

This system call takes 64-bit tmout_u in microseconds instead of the parameter tmout of tk_get_mpl.

The specification of this system call is same as that of tk_get_mpl, except that the parameter is replaced with tmout_u. For more details, see the description of tk_get_mpl.
4.6.2.5  tk_rel_mpl - Release Variable-size Memory Block

C Language Interface

```c
#include <tk/tkernel.h>
ER ercd = tk_rel_mpl(ID mplid, void *blk);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mplid</th>
<th>Memory Pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>blk</td>
<td>Memory block start address</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

- **E_OK** Normal completion
- **E_ID** Invalid ID number (mplid is invalid or cannot be used)
- **E_NOEXS** Object does not exist (the variable-size memory pool specified in mplid does not exist)
- **E_PAR** Parameter error (blk is invalid, or block returned to wrong memory pool)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Returns the memory block specified in blk to the variable-size memory pool specified in mplid.

Executing `tk_rel_mpl` may enable memory block acquisition by another task waiting to allocate memory from the memory pool specified in mplid, releasing the WAITING state of that task.

When a memory block is returned to a variable-size memory pool, it must be the same variable-size memory pool from which the block was allocated. If an attempt to return a memory block to a different memory pool is detected, error code E_PAR is returned. Whether this error detection is performed or not is implementation-dependent.

Additional Notes

When memory is returned to a variable-size memory pool in which multiple tasks are queued, multiple tasks may be released at the same time depending on the amount of memory returned and their requested memory size. The task precedence among tasks of the same priority after their WAITING state is released in such a case is the order in which they were queued.
4.6.2.6  tk_ref_mpl - Reference Variable-size Memory Pool Status

C Language Interface

```c
#include <tk/tkernel.h>

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

### Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mplid</th>
<th>Memory Pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T_RMPL*</td>
<td>pk_rmpl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pointer to the area to return the memory pool status</td>
</tr>
</tbody>
</table>

### Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

pk_rmpl Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Waiting Task ID</td>
</tr>
<tr>
<td>SZ</td>
<td>frsz</td>
<td>Free Memory Size</td>
</tr>
<tr>
<td>SZ</td>
<td>maxsz</td>
<td>Max Memory Size</td>
</tr>
</tbody>
</table>

(Maximum memory space size (in bytes))

(Other implementation-dependent parameters may be added beyond this point.)

### Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (mplid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the variable-size memory pool specified in mplid does not exist)
- **E_PAR**: Parameter error (invalid pk_rmpl)

### Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

### Related Service Profile Items

None.

### Description

References the status of the variable-size memory pool specified in mplid, passing in return parameters the total size of free space (frsz), the maximum size of memory immediately available (maxsz), the waiting task ID (wtsk), and extended information (exinf).

wtsk indicates the ID of a task waiting for memory block allocation from this variable-size memory pool. If multiple tasks are waiting for the variable-size memory pool, the ID of the task at the head of the queue is returned. If there are no waiting tasks, wtsk = 0 is returned.
If the variable-size memory pool specified with `tk_ref_mpl` does not exist, error code `E_NOEXS` is returned.
4.7 Time Management Functions

Time management functions perform time-dependent processing. They include functions for system time management, cyclic handlers, and alarm handlers.

The generic name used in the following for cyclic handlers and alarm handlers is time event handlers.
4.7.1 System Time Management

System time is the time which a system that runs μT-Kernel uses as timing reference for its operation. Functions are provided for system clock setting and reference, and for referencing system operating time.

System time of μT-Kernel 3.0 starts from the epoch, January 1st 1970, 0:00:00 (UTC). It is represented either in the elapsed milliseconds or in microseconds. System time is set using `tk_set_utc` or `tk_set_utc_u`. It can be referenced by `tk_get_utc` or `tk_get_utc_u`.

Additional Notes
System time epoch in μT-Kernel 3.0 is 0:00:00, January 1, 1970 (UTC). The epoch, 0:00:00, January 1, 1970 (UTC), is the same epoch used by UNIX operating systems that conform to the POSIX standard.
4.7.1.1  tk_set_utc - Set System Time

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_set_utc(CONST SYSTIM *pk_tim);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST SYSTIM*</td>
<td>Packet of Current Time</td>
</tr>
<tr>
<td>pk_tim</td>
<td>Packet indicating current time (ms)</td>
</tr>
</tbody>
</table>

pk_tim Detail:

- W hi         : High 32 bits Higher 32 bits of current time for setting the system time
- UW lo        : Low 32 bits Lower 32 bits of current time for setting the system time

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>Error Code Error code</td>
</tr>
<tr>
<td>ercd</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

- E_OK: Normal completion
- E_PAR: Parameter error (pk_tim is invalid, or time setting is invalid)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- TK_SUPPORT_UTC: Support of UNIX time

Description

Sets the system clock to the value specified in pk_tim.
System time is expressed as cumulative milliseconds from 0:00:00, January 1, 1970 (UTC).

Additional Notes

The relative time specified in RELTIM or TMO does not change even if the system clock is changed by calling tk_set_utc during system operation. For example, if a timeout is set to elapse in 60 seconds and the system...
clock is advanced by 60 seconds by \texttt{tk\_set\_utc} while waiting for the timeout, the timeout occurs not immediately but 60 seconds after it was set. Instead, \texttt{tk\_set\_utc} changes the system time at which the timeout occurs.

The time specified in \texttt{pk\_tim} for \texttt{tk\_set\_utc} is not restricted to the resolution of the timer interrupt cycle. But the time that is read later by \texttt{tk\_get\_utc} changes according to the time resolution of the timer interrupt cycle. For example, in the system where the timer interrupt cycle is 10 milliseconds, if the time of 10005 (ms) is specified in \texttt{tk\_set\_utc}, then the time obtained later by \texttt{tk\_get\_utc} changes as follows: 10005 (ms) → 10015 (ms) → 10025 (ms).
4.7.1.2  tk_set_utc_u - Set Time (Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_set_utc_u(SYSTIM_U tim_u);
```

Parameter

| SYSTIM_U | tim_u | Current Time | Current time (in microseconds) |

Return Parameter

| ER | ercd | Error Code | Error code |

Error Code

| E_OK | Normal completion |
| E_PAR | Parameter error (tim_u is invalid, or time setting is invalid) |

Valid Context

| Task portion | Quasi-task portion | Task-independent portion |
| YES | YES | NO |

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- TK_SUPPORT.utc
- TK_SUPPORT.usec

Description

This system call takes 64-bit tim_u in microseconds instead of the parameter pk_tim of tk_set_utc. In the parameter tim_u of this API, system time is expressed as cumulative microseconds from 0:00:00, January 1, 1970 (UTC).

Whereas the parameter pk_tim of tk_set_utc is passed in packet using the structure SYSTIM, the parameter tim_u of tk_set_utc_u is passed by value (not packet) using the 64-bit signed integer SYSTIM_U.

The specification of this system call is same as that of tk_set_utc, except the above-mentioned point. For more details, see the description of tk_set_utc.
4.7.1.3  tk_set_tim - Set System Time (TRON)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_set_tim(CONST SYSTIM *pk_tim);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CONST SYSTIM* pk_tim</code></td>
<td>Packet of Current Time Packet indicating current time (ms)</td>
</tr>
</tbody>
</table>

**pk_tim Detail:**

| W   | hi     | High 32 bits  | Higher 32 bits of current time for setting the system time |
| UW  | lo     | Low 32 bits   | Lower 32 bits of current time for setting the system time  |

**Return Parameter**

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (pk_tim is invalid, or time setting is invalid)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_TRONTIME** Support of TRON time

**Description**

Sets the system clock to the value specified in `pk_tim`. In the parameter `hi` and `lo` of this API, system time is expressed as cumulative milliseconds from 0:00:00 (GMT), January 1, 1985.

**Additional Notes**

`tk_set_tim` is very similar to `tk_set_utc`. However, it uses the time system with a different epoch. `tk_set_tim` is an API to keep compatibility with legacy μT-Kernel or T-Kernel specifications.
4.7.1.4 tk_set_tim_u - Set Time (TRON, Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_set_tim_u(SYSTIM_U tim_u);

Parameter

SYSTIM_U tim_u Current Time Current time (in microseconds)

Return Parameter

ER ercd Error Code Error code

Error Code

E_OK Normal completion
E_PAR Parameter error (tim_u is invalid, or time setting is invalid)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_TRONTIME Support of TRON time
TK_SUPPORT_USEC Support of microsecond

Description

This system call takes 64-bit tim_u in microseconds instead of the parameter pk_tim of tk_set_tim. In the parameter tim_u of this API, system time is expressed as cumulative microseconds from 0:00:00 (GMT), January 1, 1985.

Whereas the parameter pk_tim of tk_set_tim is passed in packet using the structure SYSTIM, the parameter tim_u of tk_set_tim_u is passed by value (not packet) using the 64-bit signed integer SYSTIM_U.

The specification of this system call is same as that of tk_set_tim, except the above-mentioned point. For more details, see the description of tk_set_tim.

Additional Notes

tk_set_tim_u is very similar to tk_set_utc_u. However, it uses the time system with a different epoch. tk_set_tim_u is an API to keep compatibility with legacy μT-Kernel or T-Kernel specifications.
4.7.1.5  tk_get_utc - Get System Time

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_get_utc(SYSTIM *pk_tim);
```

Parameter

| SYSTIM* pk_tim | Packet of Current Time | Pointer to the area to return the current time (ms) |

Return Parameter

| ER ercd | Error Code | Error code |

pk_tim Details:

| W hi | High 32 bits |
| UW lo | Low 32 bits |

Higher 32 bits of current time of the system time
Lower 32 bits of current time of the system time

Error Code

| E_OK Normal completion |
| E_PAR Parameter error (pk_tim is invalid) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

| TK_SUPPORT_UTC | Support of UNIX time |

Description

Reads the current value of the system clock and returns in it pk_tim.
System time is expressed as cumulative milliseconds from 0:00:00, January 1, 1970 (UTC).

Additional Notes

The resolution of the current system time read by this API varies depending on the time resolution of the timer interrupt interval (cycle). Hence, this API cannot be used to get the elapsed time that is shorter than
the timer interrupt interval (cycle). For more details, see the Additional Notes of \texttt{tk\_set\_utc}. To find out the elapsed time shorter than the timer interrupt interval (cycle), use the return parameter \texttt{ofs} of \texttt{tk\_get\_utc\_u} or \texttt{td\_get\_utc}. 
4.7.1.6 tk_get_utc_u - Get System Time (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_get_utc_u(SYSTIM_U *tim_u, UW *ofs);

Parameter

SYSTIM_U* tim_u Time Pointer to the area to return the current time (in microseconds)
UW* ofs Offset Pointer to the area to return the return parameter ofs

Return Parameter

ER ercd Error Code Error code
SYSTIM_U* tim_u Time Current time (in microseconds)
UW* ofs Offset Relative elapsed time from tim_u (in nanoseconds)

Error Code

E_OK Normal completion
E_PAR Parameter error (invalid tim_u or ofs)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_UTC Support of UNIX time
TK_SUPPORT_USEC Support of microsecond

Description

This system call takes 64-bit tim_u in microseconds instead of the return parameter pk_tim of tk_get_utc. System time is expressed as cumulative microseconds from 0:00:00 (UTC), January 1, 1970. It also includes the return parameter ofs that returns the relative time in nanoseconds.

tim_u has the resolution of time interrupt interval (cycle), but even more precise time information is obtained in ofs as the elapsed time from tim_u in nanoseconds. The resolution of ofs is implementation-dependent, but generally is the resolution of hardware timer.

If ofs = NULL, the information of ofs is not stored.

The specification of this system call is same as that of tk_get_utc, except the above-mentioned point. In addition, the specification of this system call is the same as that of td_get_utc, except that the data type of
tim_u is SYSTIM_U. For more details, see the description of \texttt{tk\_get\_utc} and \texttt{td\_get\_utc}.
4.7.1.7 tk_get_tim - Get System Time (TRON)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_get_tim(SYSTIM *pk_tim);
```

Parameter

- **SYSTIM** `pk_tim`:
  - Packet of Current Time
  - Pointer to the area to return the current time (ms)

Return Parameter

- **ER** `ercd`:
  - Error Code
  - Error code

- **pk_tim**:
  - High 32 bits: `hi`
  - Lower 32 bits: `lo`

  - Higher 32 bits of current time of the system time
  - Lower 32 bits of current time of the system time

Error Code

- **E_OK**:
  - Normal completion

- **E_PAR**:
  - Parameter error (`pk_tim` is invalid)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_TRONTIME**:
  - Support of TRON time

Description

Reads the current value of the system clock and returns it in `pk_tim`. In the return parameter `hi` and `lo` of this API, system time is expressed as cumulative milliseconds from 0:00:00 (GMT), January 1, 1985.

Additional Notes

The resolution of the current system time read by this API varies depending on the time resolution of the timer interrupt interval (cycle). Hence, this API cannot be used to get the elapsed time that is shorter than
the timer interrupt interval (cycle). For more details, see the Additional Notes of `tk_set_utc`. To find out the elapsed time shorter than the timer interrupt interval (cycle), use the return parameter `ofs` of `tk_get_tim_u` or `td_get_tim`.

`tk_get_tim` is very similar to `tk_get_utc`. However, it uses the time system with a different epoch. `tk_get_tim` is an API to keep compatibility with legacy μT-Kernel or T-Kernel specifications.
4.7.1.8  tk_get_tim_u - Get System Time (TRON, Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_get_tim_u(SYSTIM_U *tim_u, UW *ofs);
```

Parameter

<table>
<thead>
<tr>
<th>SYSTIM_U*</th>
<th>tim_u</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW*</td>
<td>ofs</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Parameter

- `SYSTIM_U* tim_u`: Pointer to the area to return the current time (in microseconds)
- `UW* ofs`: Pointer to the area to return the return parameter `ofs`

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTIM_U</td>
<td>tim_u</td>
<td>Time</td>
</tr>
<tr>
<td>UW</td>
<td>ofs</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Error Code

- `E_OK`: Normal completion
- `E_PAR`: Parameter error (invalid `tim_u` or `ofs`)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- `TK_SUPPORT_TRONTIME`: Support of TRON time
- `TK_SUPPORT_USEC`: Support of microsecond

Description

This system call takes 64-bit `tim_u` in microseconds instead of the return parameter `pk_tim` of `tk_get_tim`. In the return parameter `tim_u` of this API, system time is expressed as cumulative microseconds from 0:00:00 (GMT), January 1, 1985. It also includes the return parameter `ofs` that returns the relative time in nanoseconds.

`tim_u` has the resolution of time interrupt interval (cycle), but even more precise time information is obtained in `ofs` as the elapsed time from `tim_u` in nanoseconds. The resolution of `ofs` is implementation-dependent, but generally is the resolution of hardware timer.

If `ofs = NULL`, the information of `ofs` is not stored.

The specification of this system call is same as that of `tk_get_tim`, except the above-mentioned point. In addition, the specification of this system call is the same as that of `td_get_tim`, except that the data type of
tim_u is SYSTIM_U. For more details, see the description of tk_get_tim and td_get_tim.

Additional Notes

tk_get_tim_u is very similar to tk_get_utc_u. However, it uses the time system with a different epoch. tk_get_tim_u is an API to keep compatibility with legacy μT-Kernel or T-Kernel specifications.
4.7.1.9  tk_get_otm - Get Operating Time

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_get_otm(SYSTIM *pk_tim);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTIM* <code>pk_tim</code></td>
<td>Packet of Operating Time</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER <code>ercd</code></td>
<td>Error Code</td>
</tr>
</tbody>
</table>

`pk_tim` Detail:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W hi</td>
<td>High 32 bits</td>
</tr>
<tr>
<td>UW lo</td>
<td>Low 32 bits</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (<code>pk_tim</code> is invalid)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Gets the system operating time (up time).

System operating time, unlike system time, indicates the length of time elapsed linearly since the system was started. It is not affected by clock settings made by `tk_set_utc` or `tk_set_tim`.

System operating time must have the same precision as system time.
4.7.1.10  tk_get_otm_u - Get Operating Time (Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_get_otm_u(SYSTIM_U *tim_u, UW *ofs);
```

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTIM_U*</td>
<td>Pointer to the area to return the operating time (in microseconds)</td>
</tr>
<tr>
<td>UW*</td>
<td>Pointer to the area to return the return parameter ofs</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>Error Code Error Codes</td>
</tr>
<tr>
<td>SYSTIM_U</td>
<td>Time</td>
</tr>
<tr>
<td>UW</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Error Code

- **E_OK**: Normal completion
- **E_PAR**: Parameter error (invalid tim_u or ofs)

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

<table>
<thead>
<tr>
<th>Service Profile Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_USEC</td>
<td>Support of microsecond</td>
</tr>
</tbody>
</table>

Description

This system call takes 64-bit tim_u in microseconds instead of the return parameter pk_tim of tk_get_otm. It also includes the return parameter ofs that returns the relative time in nanoseconds.

tim_u has the resolution of time interrupt interval (cycle), but even more precise time information is obtained in ofs as the elapsed time from tim_u in nanoseconds. The resolution of ofs is implementation-dependent, but generally is the resolution of hardware timer.

If ofs = NULL is set, the information of ofs is not stored.

The specification of this system call is same as that of tk_get_otm, except the above-mentioned point. In addition, the specification of this system call is the same as that of td_get_otm, except that the data type of tim_u is SYSTIM_U. For more details, see the description of tk_get_otm and td_get_otm.
4.7.2 Cyclic Handler

A cyclic handler is a time event handler started at regular intervals. Cyclic handler functions are provided for creating and deleting a cyclic handler, activating and deactivating a cyclic handler operation, and referencing cyclic handler status. A cyclic handler is an object identified by an ID number. The ID number for the cyclic handler is called a cyclic handler ID.

The time interval at which a cyclic handler is started (cycle time) and the cycle phase are specified for each cyclic handler when it is created. When a cyclic handler operation is requested, T-Kernel determines the time at which the cyclic handler should next be started based on the cycle time and cycle phase set for it. When a cyclic handler is created, the time when it is to be started next is the time of its creation plus the cycle phase. When the time comes to start a cyclic handler, `exinf`, containing extended information about the cyclic handler, is passed to it as a starting parameter. The time when the cyclic handler is started plus its cycle time becomes the next start time. Sometimes when a cyclic handler is activated, the next start time will be newly set.

In principle the cycle phase of a cyclic handler is no longer than its cycle time. The behavior is implementation-dependent when the cycle phase is made longer than the cycle time.

A cyclic handler has two activation states, active and inactive. While a cyclic handler is inactive, it is not started even when its start time arrives, although calculation of the next start time does take place. When a system call for activating a cyclic handler is called (`tk_sta_cyc`), the cyclic handler goes to active state, and the next start time is decided if necessary. When a system call for deactivating a cyclic handler is called (`tk_stp_cyc`), the cyclic handler goes to inactive state. Whether a cyclic handler upon creation is active or inactive is decided by a cyclic handler attribute.

The cycle phase of a cyclic handler is a relative time specifying the first time the cyclic handler is to be started, in relation to the time when the system call creating it was invoked. The cycle time of a cyclic handler is likewise a relative time, specifying the next time the cyclic handler is to be started in relation to the time it should have started (not the time it started). For this reason, the intervals between times the cyclic handler is started will individually be shorter than the cycle time in some cases, but their average over a longer time span will match the cycle time.

Additional Notes

Actual time resolution in \(\mu\) T-Kernel time management functions processing uses one that is specified by the "timer interrupt interval" (TTimPeriod) in Section 5.6.2, "Standard System Configuration Information". It also means that a cyclic handler or an alarm handler is actually started at the time according to the time resolution provided by the timer interrupt interval (TTimPeriod). For this reason, the cyclic handler is actually started at the time of timer interrupt occurrence immediately after the time when the cyclic handler should be started. A general \(\mu\) T-Kernel implementation checks if a cyclic handler or an alarm handler that is to be started within the processing of timer interrupt exists, and then starts them as necessary.
4.7.2.1 tk_cre_cyc - Create Cyclic Handler

C Language Interface

```c
#include <tk/tkernel.h>

ID cycid = tk_cre_cyc(CONST T_CCYC *pk_ccyc);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_CCYC*</td>
<td>pk_ccyc</td>
</tr>
<tr>
<td>Parameter</td>
<td>Packet to Create Cyclic</td>
</tr>
<tr>
<td>handler definition</td>
<td></td>
</tr>
<tr>
<td>Information</td>
<td>Cyclic handler definition</td>
</tr>
</tbody>
</table>

`pk_ccyc` Detail:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>exinf</td>
</tr>
<tr>
<td>ATR</td>
<td>cycatr</td>
</tr>
<tr>
<td>FP</td>
<td>cychdr</td>
</tr>
<tr>
<td>RELTIM</td>
<td>cyctim</td>
</tr>
<tr>
<td>RELTIM</td>
<td>cycphs</td>
</tr>
<tr>
<td>UB</td>
<td>dsname[8]</td>
</tr>
</tbody>
</table>

- Extended Information
- Cyclic Handler Attribute
- Cyclic Handler Address
- Cycle Time Interval of cyclic start (ms)
- Cycle Phase (ms)
- DS Object name

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>cycid</td>
</tr>
<tr>
<td>or</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOMEM</td>
<td>Insufficient memory (memory for control block cannot be allocated)</td>
</tr>
<tr>
<td>E_LIMIT</td>
<td>Number of cyclic handlers exceeds the system limit</td>
</tr>
<tr>
<td>E_RSATR</td>
<td>Reserved attribute (cycatr is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (pk_ccyc, cychdr, cyctim, or cycphs is invalid or cannot be used)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td></td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

- TK_SUPPORT_ASM  Support for specifying TA_ASM for cyclic handler attribute
- TK_SUPPORT_DSNAME Support for specifying TA_DSNAME for cyclic handler attribute

Description

Creates a cyclic handler, assigning to it a cyclic handler ID. This is performed by assigning a control block for the generated cyclic handler.

A cyclic handler is a handler running at specified intervals as a task-independent portion.
exinf can be used freely by the user to set miscellaneous information about the created cyclic handler. The information set in this parameter can be referenced by tk_ref_cyc. If a larger area is needed for indicating user information, or if the information may need to be changed after the message buffer is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in exinf. The kernel pays no attention to the contents of exinf.

cycatr indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of cycatr is as follows.

\[
\text{cycatr} := (\text{TA ASM} || \text{TA HLNG}) | \text{TA_STA} | \text{TA_PHS} | \text{TA_DSNAME}
\]

- TA_ASM: The handler is written in assembly language
- TA_HLNG: The handler is written in high-level language
- TA_STA: Activate immediately upon cyclic handler creation
- TA_PHS: Save the cycle phase
- TA_DSNAME: Specifies DS object name

#define TA_ASM 0x00000000 /* assembly language program */
#define TA_HLNG 0x00000001 /* high-level language program */
#define TA_STA 0x00000002 /* activate cyclic handler */
#define TA_PHS 0x00000004 /* save cyclic handler cycle phase */
#define TA_DSNAME 0x00000040 /* DS object name */

cychdr specifies the cyclic handler start address, cyctime the cycle time, and cycphs the cycle phase.

When the TA_HLNG attribute is specified, the cyclic handler is started via a high-level language support routine. The high-level language support routine takes care of saving and restoring register values. The cyclic handler terminates by a simple return from a function. The cyclic handler takes the following format when the TA_HLNG attribute is specified.

```c
void cychdr( void *exinf )
{
    /* (processing) */
    return; /* Exit cyclic handler*/
}
```

The cyclic handler format when the TA_ASM attribute is specified is implementation-dependent, but exinf must be passed in a starting parameter.

cycphs indicates the length of time until the cyclic handler is initially started after being created by tk_cre_cyc. Thereafter it is started periodically at the interval set in cyctime. If zero is specified in cycphs, the cyclic handler starts immediately after it is created. Zero cannot be specified in cyctime.

The starting of the cyclic handler for the nth time occurs after at least cycphs + cyctime*(n - 1) time has elapsed from the cyclic handler creation.

When TA_STA is specified, the cyclic handler goes to active state immediately on creation, and starts at the intervals noted above. If TA_STA is not specified, the cycle time is calculated but the cyclic handler is not actually started.

When TA_PHS is specified, then even if tk_sta_cyc is called activating the cyclic handler, the cycle time is not reset, and the cycle time calculated as above from the time of cyclic handler creation continues to apply. If TA_PHS is not specified, calling tk_sta_cyc resets the cycle time and the cyclic handler is started at cyctime intervals measured from the time tk_sta_cyc was called. Note that the resetting of cycle time by tk_sta_cyc does not affect cycphs. In this case the starting of the cyclic handler for the nth time occurs after at least cyctime * n has elapsed from the calling of tk_sta_cyc.
Even if a system call is invoked from a cyclic handler and this causes the task in RUNNING state up to that time to go to another state, with a different task going to RUNNING state, dispatching (task switching) does not occur while the cyclic handler is running. Completion of execution by the cyclic handler has precedence even if dispatching is necessary; only when the cyclic handler terminates does the dispatch take place. In other words, a dispatch request that is generated while a cyclic handler is running is not processed immediately, but is delayed until the cyclic handler terminates. This is called delayed dispatching.

A cyclic handler runs as a task-independent portion. As such, it is not possible to call in a cyclic handler a system call that can enter WAITING state, or one that is intended for the invoking task.

When TA_DSNAME is specified, dsname is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, td_ref_dsname and td_set_dsname. For more details, see the description of td_ref_dsname and td_set_dsname. If TA_DSNAME is not specified, dsname is ignored. Then td_ref_dsname and td_set_dsname return E_OBJ error.

Additional Notes

Once a cyclic handler is defined, it continues to run at the specified cycles either until tk_stp_cyc is called to deactivate it or until it is deleted. There is no parameter to specify the number of cycles in tk_cre_cyc.

When multiple time event handlers or interrupt handlers operate at the same time, it is implementation-dependent whether to have them run serially (after one handler exits, another starts) or in a nested manner (one handler operation is suspended, another runs, and when that one finishes the previous one resumes). In either case, since time event handlers and interrupt handlers run as task-independent portion, the principle of delayed dispatching applies.

If 0 is specified in cycphs, the first startup of the cyclic handler is executed immediately after this system call execution. However, depending on the implementation, the first startup (execution) of the cyclic handler may be executed while processing this system call, instead of immediately after the completion of this system call execution. In such case, the interrupt disabled or other state in the cyclic handler may differ from the state at the second and subsequent ordinary startups. In addition, when 0 is set to cycphs, the first startup of the cyclic handler is executed without waiting for a timer interrupt, that is, regardless of the timer interrupt interval. This behavior also differs from the second and subsequent startups of the cyclic handler, and from the startup of the cyclic handler with cycphs set to other than 0.
4.7.2.2  tk_cre_cyc_u - Create Cyclic Handler (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ID cycid = tk_cre_cyc_u(CONST T_CCYC_U *pk_ccyc_u);

Parameter

CONST T_CCYC_U*  pk_ccyc_u  Packet to Create Cyclic Handler
Cyclic handler definition information

pk_ccyc_u Detail:

void*  exinf  Extended Information
ATR cycatr  Cyclic Handler Attribute
FP cyhdr  Cyclic Handler Address
RELTIM_U cyctim_u  Cycle Time
Interval of cyclic start (in microseconds)
RELTIM_U cyctim_u  Cycle Phase
Cycle phase (in microseconds)
UB dsname[8]  DS Object name
DS object name
(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

ID cycid  Cyclic Handler ID
or Error Code Cyclic handler ID
Error code

Error Code

E_NOMEM Insufficient memory (memory for control block cannot be allocated)
E_LIMIT Number of cyclic handlers exceeds the system limit
E_RSATR Reserved attribute (cycatr is invalid or cannot be used)
E_PAR Parameter error (pk_ccyc_u, cyhdr, cyctim_u, or cyctim_u is invalid or cannot be used)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_USEC  Support of microsecond

Additionally, the following service profile items are related to this system call.

TK_SUPPORT_ASM  Support for specifying TA_ASM for cyclic handler attribute
**TK_SUPPORT_DSNAME**

Support for specifying TA_DSNAME for cyclic handler attribute

**Description**

This system call takes 64-bit `cyctim_u` and `cycphs_u` in microseconds instead of the parameters `cyctim` and `cycphs` of `tk_cre_cyc`.

The specification of this system call is same as that of `tk_cre_cyc`, except that the parameter is replaced with `cyctim_u` and `cycphs_u`. For more details, see the description of `tk_cre_cyc`.
4.7.2.3 tk_del_cyc - Delete Cyclic Handler

C Language Interface

```c
#include <tk/tkernel.h>
```

```c
ER ercd = tk_del_cyc(ID cycid);
```

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>cycid</th>
<th>Cyclic Handler ID</th>
<th>Cyclic handler ID</th>
</tr>
</thead>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

**Error Code**

- **E_OK** Normal completion
- **E_ID** Invalid ID number (cycid is invalid or cannot be used)
- **E_NOEXS** Object does not exist (the cyclic handler specified in cycid does not exist)

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

None.

**Description**

Deletes a cyclic handler.
4.7.2.4  tk_sta_cyc - Start Cyclic Handler

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_sta_cyc(ID cycid);

Parameter

| ID   | cycid | Cyclic Handler ID | Cyclic handler ID |

Return Parameter

| ER   | ercd | Error Code | Error code |

Error Code

- E_OK      Normal completion
- E_ID      Invalid ID number (cycid is invalid or cannot be used)
- E_NOEXS   Object does not exist (the cyclic handler specified in cycid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Activates a cyclic handler, putting it in active state.

If the TA_PHS attribute was specified, the cycle time of the cyclic handler is not reset when the cyclic handler goes to active state. If it was already in active state when this system call was executed, it continues unchanged in active state.

If the TA_PHS attribute was not specified, the cycle time is reset when the cyclic handler goes to active state. If it was already in active state, it continues in active state but its cycle time is reset. In this case, the next time the cyclic handler starts is after cyctim has elapsed.
### 4.7.2.5 tk_stp_cyc - Stop Cyclic Handler

**C Language Interface**

```c
#include <tk/tkernel.h>

ER ercd = tk_stp_cyc(ID cycid);
```

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>cycid</th>
<th>Cyclic Handler ID</th>
<th>Cyclic handler ID</th>
</tr>
</thead>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

**Error Code**

- **E_OK** Normal completion
- **E_ID** Invalid ID number (`cycid` is invalid or cannot be used)
- **E_NOEXS** Object does not exist (the cyclic handler specified in `cycid` does not exist)

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

None.

**Description**

Deactivates a cyclic handler, putting it in inactive state. If the cyclic handler was already in inactive state, this system call has no effect (no operation).
4.7.2.6 tk_ref_cyc - Reference Cyclic Handler Status

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_ref_cyc(ID cycid, T_RCYC *pk_rcyc);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>cycid</th>
<th>Cyclic Handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RCYC*</td>
<td>pk_rcyc</td>
<td>Packet to Return Cyclic Handler Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cyclic handler ID</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

pk_rcyc Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELTIM</td>
<td>lfttim</td>
<td>Left Time (ms)</td>
</tr>
<tr>
<td>UINT</td>
<td>cycstat</td>
<td>Cyclic Handler Status</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (cycid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the cyclic handler specified in cycid does not exist)
- **E_PAR**: Parameter error (invalid pk_rcyc)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

References the status of the cyclic handler specified in cycid, passing in return parameters the cyclic handler activation state (cycstat), the time remaining until the next start (lfttim), and extended information (exinf).

The following information is returned in cycstat:

```
cycstat := (TCYC_STP | TCYC_STA)
```
#define TCYC_STP 0x00 /* cyclic handler is inactive */
#define TCYC_STA 0x01 /* cyclic handler is active */

lfttim returns the remaining time (milliseconds) until the next time when the cyclic handler is invoked. It does not matter whether the cyclic handler is currently running or stopped.

exinf returns the extended information specified as a parameter when the cyclic handler is generated. exinf is passed to the cyclic handler as a parameter.

If the cyclic handler specified in cycid does not exist for, error code E_NOEXS is returned.

The time remaining lfttim returned in the cyclic handler status information (T_RCYC) is a value rounded to milliseconds. To know the value in microseconds, call tk_ref_cyc_u.
4.7.2.7 tk_ref_cyc_u - Reference Cyclic Handler Status (Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_ref_cyc_u(ID cycid, T_RCYC_U *pk_rcyc_u);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th></th>
<th>Cyclic Handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycid</td>
<td></td>
<td>Cyclic handler ID</td>
</tr>
<tr>
<td>T_RCYC_U*</td>
<td>pk_rcyc_u</td>
<td>Packet to Return Cyclic Handler Status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th></th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ercd</td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

`pk_rcyc_u` Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELTIM_U</td>
<td>lfttim_u</td>
<td>Left Time</td>
</tr>
<tr>
<td>UINT</td>
<td>cycstat</td>
<td>Cyclic Handler Status</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (`cycid` is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the cyclic handler specified in `cycid` does not exist)
- **E_PAR**: Parameter error (invalid `pk_rcyc_u`)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- **TK_SUPPORT_USEC**: Support of microsecond

Description

This system call takes 64-bit `lfttim_u` in microseconds instead of the return parameter `lfttim` of `tk_ref_cyc`. The specification of this system call is same as that of `tk_ref_cyc`, except that the return parameter is replaced with `lfttim_u`. For more details, see the description of `tk_ref_cyc`. 
4.7.3 Alarm Handler

An alarm handler is a time event handler that starts at a specified time. Functions are provided for creating and deleting an alarm handler, activating and deactivating the alarm handler, and referencing the alarm handler status. An alarm handler is an object identified by an ID number. The ID number for an alarm handler is called an alarm handler ID.

The time at which an alarm handler starts (called the alarm time) can be set independently for each alarm handler. When the alarm time arrives, exinf, containing extended information about the alarm handler, is passed to it as a starting parameter.

After an alarm handler is created, initially it has no alarm time set and is in inactive state. The alarm time is set when the alarm handler is activated by calling tk_sta_alm, as relative time from the time that system call is executed. When tk_stp_alm is called deactivating the alarm handler, the alarm time setting is canceled. Likewise, when an alarm time arrives and the alarm handler runs, the alarm time is canceled and the alarm handler becomes inactive.

Additional Notes
An alarm handler is actually started at the time according to the time resolution provided by the timer interrupt interval (TTimPeriod). For more details, see the additional notes for Section 4.7.2, “Cyclic Handler”.
4.7.3.1 tk_cre_alm - Create Alarm Handler

C Language Interface

```c
#include <tk/tkernel.h>

ID almid = tk_cre_alm(CONST T_CALM *pk_calm);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_CALM* pk_calm</td>
<td>Packet to Create Alarm Handler</td>
</tr>
<tr>
<td></td>
<td>Alarm handler definition information</td>
</tr>
</tbody>
</table>

pk_calm Detail:

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extended information</td>
</tr>
<tr>
<td>ATR</td>
<td>almatr</td>
<td>Alarm Handler Attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alarm handler attributes</td>
</tr>
<tr>
<td>FP</td>
<td>almhdr</td>
<td>Alarm Handler Address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alarm handler address</td>
</tr>
<tr>
<td>UB</td>
<td>dsname[8]</td>
<td>DS Object name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DS object name</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>Return Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID almid</td>
<td>Alarm Handler ID</td>
</tr>
<tr>
<td>or Error Code</td>
<td>Alarm handler ID or Error code</td>
</tr>
</tbody>
</table>

Error Code

- E_NOMEM: Insufficient memory (memory for control block cannot be allocated)
- E_LIMIT: Number of alarm handlers exceeds the system limit
- E_RSATR: Reserved attribute (almatr is invalid or cannot be used)
- E_PAR: Parameter error (pk_calm or almhdr is invalid or cannot be used)

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task portion</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

- TK_SUPPORT_ASM: Support for specifying TA_ASM for alarm handler attribute
- TK_SUPPORT_DSNAME: Support for specifying TA_DSNAME for alarm handler attribute

Description

Creates an alarm handler, assigning to it an alarm handler ID. This is performed by assigning a control block for the generated alarm handler.

An alarm handler is a handler running at the specified time as a task-independent portion.

exinf can be used freely by the user to set miscellaneous information about the created alarm handler. The information set in this parameter can be referenced by tk_ref_alm. If a larger area is needed for indicating
user information, or if the information may need to be changed after the message buffer is created, this can be done by allocating separate memory for this purpose and putting the memory packet address in `exinf`. The kernel pays no attention to the contents of `exinf`.

`almatr` indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of `almatr` is as follows.

```
almatr := (TA_ASM || TA_HLNG) || [TA_DSNAME]
```

- **TA_ASM**
  - The handler is written in assembly language
- **TA_HLNG**
  - The handler is written in high-level language
- **TA_DSNAME**
  - Specifies DS object name

```c
#define TA_ASM 0x00000000 /* assembly language program */
#define TA_HLNG 0x00000001 /* high-level language program */
#define TA_DSNAME 0x00000040 /* DS object name */
```

`almhdr` specifies the alarm handler start address.

When the **`TA_HLNG`** attribute is specified, the alarm handler is started via a high-level language support routine. The high-level language support routine takes care of saving and restoring register values. The alarm handler terminates by a simple return from a function. The alarm handler takes the following format when the **`TA_HLNG`** attribute is specified.

```c
void almhdr( void *exinf )
{
    /*
     * (processing)
     */

    return; /* exit alarm handler */
}
```

The alarm handler format when the **`TA_ASM`** attribute is specified is implementation-dependent, but `exinf` must be passed in a starting parameter.

Even if a system call is invoked from an alarm handler and this causes the task in RUNNING state up to that time to go to another state, with a different task going to RUNNING state, dispatching (task switching) does not occur while the alarm handler is running. Completion of execution by the alarm handler has precedence even if dispatching is necessary; only when the alarm handler terminates does the dispatch take place. In other words, a dispatch request that is generated while an alarm handler is running is not processed immediately, but is delayed until the alarm handler terminates. This is called delayed dispatching.

An alarm handler runs as a task-independent portion. As such, it is not possible to call in an alarm handler a system call that can enter WAITING state, or one that is intended for the invoking task.

When **`TA_DSNAME`** is specified, `dsname` is valid and specifies the DS object name. DS object name is used to identify objects by debugger, and it is handled only by T-Kernel/DS API, `td_ref_dsname` and `td_set_dsname`. For more details, see the description of `td_ref_dsname` and `td_set_dsname`. If **`TA_DSNAME`** is not specified, `dsname` is ignored. Then `td_ref_dsname` and `td_set_dsname` return E_OBJ error.

### Additional Notes

When multiple time event handlers or interrupt handlers operate at the same time, it is an implementation-dependent whether to have them run serially (after one handler exits, another starts) or in a nested manner (one handler operation is suspended, another runs, and when that one finishes the previous one resumes). In either case, since time event handlers and interrupt handlers run as task-independent portion, the principle of delayed dispatching applies.
4.7.3.2  tk_del_alm - Delete Alarm Handler

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_del_alm(ID almid);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>almid</th>
<th>Alarm Handler ID</th>
<th>Alarm handler ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_ID: Invalid ID number (almid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the alarm handler specified in almid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes an alarm handler.
4.7.3.3  tk_sta_alm - Start Alarm Handler

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_sta_alm(ID almid, RELTIM almtim);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>almid</th>
<th>Alarm Handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELTIM</td>
<td>almtim</td>
<td>Alarm Time</td>
</tr>
</tbody>
</table>

Alarm handler ID

Alarm handler start relative time (ms)

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error code

Error Code

- **E_OK**: Normal completion
- **E_ID**: Invalid ID number (almid is invalid or cannot be used)
- **E_NOEXS**: Object does not exist (the alarm handler specified in almid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Sets the alarm time of the alarm handler specified in almid to the time given in almtim, putting the alarm handler in active state. almtim is specified as relative time from the time of calling tk_sta_alm. After the time specified in almtim has elapsed, the alarm handler starts. If the alarm handler is already active when this system call is invoked, the existing almtim setting is canceled and the alarm handler is activated anew with the alarm time specified here.

If almtim = 0 is set, the alarm handler starts as soon as it is activated.
4.7.3.4  tk_sta_alm_u - Start Alarm Handler (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_sta_alm_u(ID almid, RELTIM_U almtim_u);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>almid</th>
<th>Alarm Handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELTIM_U</td>
<td>almtim_u</td>
<td>Alarm Time</td>
</tr>
</tbody>
</table>

Alarm handler ID
Alarm handler start relative time (in microseconds)

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_ID: Invalid ID number (almid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the alarm handler specified in almid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_USEC: Support of microsecond

Description

This system call takes 64-bit almtim_u in microseconds instead of the parameter almtim of tk_sta_alm.

The specification of this system call is same as that of tk_sta_alm, except that the parameter is replaced with almtim_u. For more details, see the description of tk_sta_alm.
4.7.3.5 tk_stp_alm - Stop Alarm Handler

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_stp_alm(ID almid);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>almid</th>
<th>Alarm Handler ID</th>
<th>Alarm handler ID</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

- E_OK Normal completion
- E_ID Invalid ID number (almid is invalid or cannot be used)
- E_NOEXS Object does not exist (the alarm handler specified in almid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Cancels the alarm time of the alarm handler specified in almid, putting it in inactive state. It the cyclic handler was already in inactive state, this system call has no effect (no operation).
4.7.3.6  tk_ref_alm - Reference Alarm Handler Status

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ref_alm(ID almid, T_RALM *pk_ralm);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>almid</th>
<th>Alarm Handler ID</th>
<th>Alarm handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RALM*</td>
<td>pk_ralm</td>
<td>Packet to Return Alarm Handler Status</td>
<td>Pointer to the area to return the alarm handler status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

pk_ralm Detail:

void* exinf          Extended Information | Extended information |
RELTIM lfttim        Left Time       | Time remaining until the handler starts (ms) |
UINT almstat         Alarm Handler Status | Alarm handler activation state |

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

E_OK Normal completion
E_ID Invalid ID number (almid is invalid or cannot be used)
E_NOEXS Object does not exist (the alarm handler specified in almid does not exist)
E_PAR Parameter error (invalid pk_ralm)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

References the status of the alarm handler specified in almid, passing in return parameters the time remaining until the handler starts (lfttim), and extended information (exinf).

The following information is returned in almstat.

almstat:= (TALM_STP | TALM_STA)
If the alarm handler is active (TALM_STA), the relative time until the alarm handler is scheduled to be started next time is returned to lfttim. This value is within the range \( \text{almtim} \geq \text{lfttim} \geq 0 \) specified with tk_sta alm. Since lfttim is decremented with each timer interrupt, \( \text{lfttim} = 0 \) means the alarm handler will start at the next timer interrupt.

\( \text{exinf} \) returns the extended information specified as a parameter when the alarm handler is generated. \( \text{exinf} \) is passed to the alarm handler as a parameter.

If the alarm handler is inactive (TALM_STP), lfttim is indeterminate.

If the alarm handler specified with tk_ref alm in almid does not exist, error code E_NOEXS is returned.

The time remaining lfttim returned in the alarm handler status information (T_RALM) is a value rounded to milliseconds. To know the value in microseconds, call tk_ref alm u.
4.7.3.7  tk_ref_alm_u - Reference Alarm Handler Status (Microseconds)

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ref_alm_u(ID almid, T_RALM_U *pk_ralm_u);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>almid</th>
<th>Alarm Handler ID</th>
<th>Alarm handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RALM_U*</td>
<td>pk_ralm_u</td>
<td>Packet to Return Alarm Handler Status</td>
<td>Pointer to the area to return the alarm handler status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

pk_ralm_u Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELTIM_U</td>
<td>lfttim_u</td>
<td>Left Time</td>
<td>Time remaining until the handler starts (in microseconds)</td>
</tr>
<tr>
<td>UINT</td>
<td>almstat</td>
<td>Alarm Handler Status</td>
<td>Alarm handler activation state</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Invalid ID number (almid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the alarm handler specified in almid does not exist)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (invalid pk_ralm_u)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_USEC  Support of microsecond

Description

This system call takes 64-bit lfttim_u in microseconds instead of the return parameter lfttim of tk_ref alm.

The specification of this system call is same as that of tk_ref_alm, except that the return parameter is replaced with lfttim_u. For more details, see the description of tk_ref alm.
4.8 Interrupt Management Functions

Interrupt management functions are for defining and manipulating handlers for external interrupts and CPU exceptions.

An interrupt handler runs as a task-independent portion. System calls can be invoked in a task-independent portion in the same way as in a task portion, but the following restriction applies to system call issuing in a task-independent portion.

- A system call that implicitly specifies the invoking task, or one that may put the invoking task in WAITING state cannot be issued. Error code E_CTX is returned in such cases.

During task-independent portion execution, task switching (dispatching) does not occur. If system call processing results in a dispatch request, the dispatch is delayed until processing leaves the task-independent portion. This is called delayed dispatching.
4.8.1 tk_def_int - Define Interrupt Handler

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_def_int(UINT intno, CONST T_DINT *pk_dint);

Parameter

| UINT intno | Interrupt Number          | Interrupt number         |
| CONST T_DINT* pk_dint | Packet to Define Interrupt Handler | Interrupt handler definition information |

pk_dint Detail:

| ATR intatr | Interrupt Handler Attribute | Interrupt handler attribute |
| FP inthdr   | Interrupt Handler Address   | Interrupt handler address |

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

| ER ercd | Error Code | Error code |

Error Code

| E_OK  | Normal completion |
| E_NOMEM | Insufficient memory (memory for control block cannot be allocated) |
| E_RSATR | Reserved attribute (intatr is invalid or cannot be used) |
| E_PAR | Parameter error (intno, pk_dint, or inthdr is invalid or cannot be used) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

| TK_SUPPORT_ASM | Support for specifying TA_ASM for alarm handler attribute |

Description

"Interrupts" include both external interrupts from devices and interrupts due to CPU exceptions. Defines an interrupt handler for the interrupt number intno to enable use of the interrupt handler. This system call maps the interrupt number specified by intno to the address and attributes of the interrupt handler. intno is the number used to distinguish different interrupts. Its specific meaning is defined for each implementation, but generally the interrupt number used by the interrupt mechanism of the CPU hardware (such as IRQ number) is used as it is, or any number that can be mapped to such number is used.
\texttt{intatr} indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute part of \texttt{intatr} is specified as follows.

\[
\text{intatr} := (\text{TA\_ASM} \, || \, \text{TA\_HLNG})
\]

- \texttt{TA\_ASM}: The handler is written in assembly language
- \texttt{TA\_HLNG}: The handler is written in high-level language

```
#define TA_ASM 0x00000000 /* assembly language program */
#define TA_HLNG 0x00000001 /* high-level language program */
```

As a rule, the kernel is not involved in the starting of a \texttt{TA\_ASM} attribute interrupt handler. When an interrupt is raised, the interrupt handling function in the CPU hardware directly starts the interrupt handler defined by this system call (depending on the implementation, processing by program may be included). Accordingly, processing for saving and restoring registers used by the interrupt handler is necessary at the beginning and end of the interrupt handler. An interrupt handler is terminated by execution of the \texttt{tk\_ret\_int} system call or by the CPU interrupt return instruction (or an equivalent mechanism).

Support of a mechanism for return from an interrupt handler without using \texttt{tk\_ret\_int} and hence without kernel intervention is mandatory. Note that if \texttt{tk\_ret\_int} is not used, delayed dispatching does not need to be performed.

Support for return from an interrupt handler using \texttt{tk\_ret\_int} is also mandatory, and in this case delayed dispatching must be performed.

When the \texttt{TA\_HLNG} attribute is specified, the interrupt handler is started via a high-level language support routine. The high-level language support routine takes care of saving and restoring register values. The interrupt handler terminates by a return from a C language function. The interrupt handler takes the following format when the \texttt{TA\_HLNG} attribute is specified.

```c
void inthdr( UINT intno )
{
    /* Interrupt Handling */
    return; /* Exit interrupt handler */
}
```

The parameter \texttt{intno} passed to an interrupt handler is the interrupt number identifying the interrupt that was raised, and is the same as that specified with \texttt{tk\_def\_int}. Depending on the implementation, other information about the interrupt may be passed in addition to \texttt{intno}. If such information is used, it must be defined for each implementation in the second parameter or subsequent parameters passed to the interrupt handler.

If the \texttt{TA\_HLNG} attribute is specified, it is assumed that the CPU interrupt flag will be set to interrupts disabled state from the time the interrupt is raised until the interrupt handler is called. In other words, as soon as an interrupt is raised, multiple interrupts are disabled, and this state remains when the interrupt handler is called. If multiple interrupts are to be allowed, the interrupt handler must include processing that handles multiple interrupts by manipulating the CPU interrupt flag.

Also in the case of the \texttt{TA\_HLNG} attribute, upon entry into the interrupt handler, issuing system call must be possible. Note, however, that assuming standard provision of the functionality described above, extensions are allowed such as adding a function for entering an interrupt handler with multiple interrupts enabled.

When the \texttt{TA\_ASM} attribute is specified, the state upon entry into the interrupt handler shall be defined for each implementation. Such matters as the stack and register status upon interrupt handler entry, whether system calls can be made, the method of invoking system calls, and the method of returning from the interrupt handler without kernel intervention must all be defined explicitly.
In the case of the `TA_ASM` attribute, depending on the implementation there may be cases where interrupt handler execution is not considered to be a task-independent portion. In such a case the following points need to be noted carefully.

- If interrupts are enabled, there is a possibility that task dispatching will occur.
- When a system call is invoked, it will be processed as having been called from a task portion or quasi-task portion.

If a method is provided for performing some kind of operation in an interrupt handler to detected whether it runs as task-independent portion, that method shall be announced for each implementation.

Even if a system call is invoked from an interrupt handler and this causes the task in RUNNING state up to that time to go to another state, with a different task going to RUNNING state, dispatching (task switching) does not occur while the interrupt handler is running. Completion of execution of the interrupt handler has precedence even if dispatching is necessary; only when the interrupt handler terminates does the dispatch take place. In other words, a dispatch request that is generated while an interrupt handler is running is not processed immediately, but is delayed until the interrupt handler terminates. This is called delayed dispatching.

An interrupt handler runs as a task-independent portion. As such, it is not possible to call in an interrupt handler a system call that can enter WAITING state, or one that is intended for the invoking task.

When `pk_dint = NULL` is set, a previously defined interrupt handler is canceled. When the handler definitions are canceled, the default handler defined by the system is used.

It is possible to redefine an interrupt handler for an interrupt number for which a handler is already defined. It is not necessary first to cancel the definition for that number. Defining a new handler for a `intno` already having an interrupt handler defined does not return error.

**Additional Notes**

The various specifications governing the `TA_ASM` attribute are mainly concerned with realizing an interrupt hook. For example, when an exception is raised due to illegal address access, ordinarily an interrupt handler defined in a higher-level program detects this and performs the error processing; but in the case of debugging, in place of error processing by a higher-level program, the default interrupt handler defined by the system may perform the processing and starts a debugger. In this case, the interrupt handler defined by high-level program hooks the default interrupt handler defined by the system. And, according to the situation, the handler either passes the interrupt handling to a system program such as a debugger, or it just processes it for itself.
4.8.2 tk_ret_int - Return from Interrupt Handler

C Language Interface

#include <tk/tkernel.h>

void tk_ret_int(void);

Although this system call is defined in the form of a C language interface, it will not be called in this format if a high-level language support routine is used.

Parameter

None.

Return Parameter

Does not return to the context issuing the system call.

Error Codes

The following kind of error may be detected, but no return is made to the context issuing the system call even if the error is detected. For this reason the error code cannot be passed directly as a system call return parameter. The behavior in case an error occurs is implementation-dependent.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX</td>
<td>Context error (issued from other than an interrupt handler (implementation-dependent error))</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

<table>
<thead>
<tr>
<th>Service Profile Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_ASM</td>
<td>Support for specifying TA_ASM for interrupt handler attribute</td>
</tr>
</tbody>
</table>

Description

Exits from an interrupt handler.

System calls invoked from an interrupt handler do not result in dispatching while the handler is running; instead, the dispatching is delayed until tk_ret_int is called ending the interrupt handler processing (delayed dispatching). Accordingly, tk_ret_int results in the processing of all dispatch requests made while the interrupt handler was running.

tk_ret_int is invoked only if the interrupt handler was defined specifying the TA_ASM attribute. In the case of a TA_HLNG attribute interrupt handler, the functionality equivalent to tk_ret_int is executed implicitly in the high-level language support routine, so tk_ret_int is not (must not be) called explicitly.
As a rule, the kernel is not involved in the starting of a TA_ASM attribute interrupt handler. When an interrupt is raised, the defined interrupt handler is started directly by the CPU hardware interrupt processing function. The saving and restoring of registers used by the interrupt handler must therefore be taken care of in the interrupt handler.

For the same reason, the stack and register states at the time tk_ret_int is issued must be the same as those at the time of entry into the interrupt handler. Because of this, in some cases function codes cannot be used in tk_ret_int, in which case tk_ret_int can be implemented using a trap instruction of another vector separate from that used for other system calls.

Additional Notes

tk_ret_int is a system call that does not return to the context from which it was called. Even if an error code is returned when an error of some kind is detected, normally no error checking is performed in the context from which the system call was invoked, leaving the possibility that the program will hang. For this reason these system calls do not return even if error is detected.

Using an assembly language return-from-interrupt instruction instead of tk_ret_int to exit the interrupt handler is possible if it is clear no dispatching will take place on return from the handler (the same task is guaranteed to continue executing), or if there is no need for dispatching to take place.

Depending on the CPU architecture and method of implementing the kernel, it may be possible to perform delayed dispatching even when an interrupt handler exits using an assembly language return-from-interrupt instruction. In such cases, it is permissible for the assembly language return-from-interrupt instruction to be interpreted as if it were a tk_ret_int system call.

Performing of E_CTX error checking when tk_ret_int is called from a time event handler is implementation-dependent. Depending on implementation, control may return from a different type of handler immediately.
4.9 System Management Functions

System management functions sets and references system states. Functions are provided for rotating task precedence in a queue, getting the ID of the task in RUNNING state, disabling and enabling task dispatching, referencing context and system states, setting low-power mode, and referencing the T-Kernel version.
4.9.1 tk_rot_rdq - Rotate Ready Queue

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_rot_rdq(PRI tskpri);

Parameter

PRI tskpri Task Priority Task priority

Return Parameter

ER ercd Error Code Error code

Error Code

E_OK Normal completion
E_PAR Parameter error (tskpri is invalid)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Rotates the precedence among tasks having the priority specified in tskpri. This system call changes the precedence of tasks in RUN or READY state having the specified priority, so that the task with the highest precedence among those tasks is given the lowest precedence.

By setting tskpri = TPRI_RUN = 0, this system call rotates the precedence of tasks having the priority level of the task currently in RUNNING state. When tk_rot_rdq is called from an ordinary task, it rotates the precedence of tasks having the same priority as the invoking task. When calling from a cyclic handler or other task-independent portion, it is also possible to call tk_rot_rdq (tskpri = TPRI_RUN).

Additional Notes

If there are no tasks in a run state having the specified priority, or only one such task, the system call completes normally with no operation (no error code is returned).

When this system call is issued in dispatch enabled state, specifying as the priority either TPRI_RUN or the current priority of the invoking task, the precedence of the invoking task will be the lowest among tasks of the same priority. This system call can therefore be used to relinquish execution privilege.
In dispatch disabled state, the task with highest precedence among tasks of the same priority is not always the currently executing task. The precedence of the invoking task will therefore not always become the lowest among tasks having the same priority when the above method is used in dispatch disabled state.

Examples of \texttt{tk\_rot\_rdq} execution are given in Figure 4.5, "Precedence Before Issuing \texttt{tk\_rot\_rdq}" and Figure 4.6, "Precedence After Issuing \texttt{tk\_rot\_rdq} (\texttt{tskpri} = 2)". When this system call is issued in the state shown in Figure 4.5, "Precedence Before Issuing \texttt{tk\_rot\_rdq}" specifying \texttt{tskpri} = 2, the new precedence order becomes that in Figure 4.6, "Precedence After Issuing \texttt{tk\_rot\_rdq} (\texttt{tskpri} = 2)", and Task C becomes the executing task.

![Figure 4.5: Precedence Before Issuing tk\_rot\_rdq](image)

![Figure 4.6: Precedence After Issuing tk\_rot\_rdq (tskpri = 2)](image)
4.9.2  tk_get_tid - Get Task Identifier

C Language Interface
#include <tk/tkernel.h>

ID tskid = tk_get_tid(void);

Parameter
None.

Return Parameter

| ID | tskid | Task ID | ID of the task in RUNNING state |

Error Codes
None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items
None.

Description

Gets the ID number of the task currently in RUNNING state. Unless the task-independent portion is executing, the current RUNNING state task will be the invoking task.

If there is no task currently in RUNNING state, 0 is returned.

Additional Notes

The task ID returned by tk_get_tid is identical to runtskid returned by tk_ref_sys.
4.9.3  tk_dis_dsp - Disable Dispatch

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_dis_dsp(void);

Parameter

None.

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX</td>
<td>Context error (issued from task-independent portion)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Disables task dispatching. Dispatch disabled state remains in effect until tk_ena_dsp is called enabling task dispatching. While dispatching is disabled, the invoking task does not change from RUNNING state to READY state or to WAITING state. External interrupts, however, are still enabled, so even in dispatch disabled state an interrupt handler can be started. In dispatch disabled state, the running task can be preempted by an interrupt handler, but not by another task.

The specific operations during dispatch disabled state are as follows.

- Even if a system call issued from an interrupt handler or by the task that called tk_dis_dsp results in a task going to READY state with a higher priority than the task that called tk_dis_dsp, that task will not be dispatched. Dispatching of the higher-priority task is delayed until dispatch disabled state ends.
- If the task that called tk_dis_dsp issues a system call that may cause the invoking task to be put in WAITING state (e.g., tk_slp_tsk or tk_wai_sem), error code E_CTX is returned.
- When system status is referenced by tk_ref_sys, TSS_DDSP is returned in sysstat.

If tk_dis_dsp is called for a task already in dispatch disabled state, that state continues with no error code returned. No matter how many times tk_dis_dsp is called, calling tk_ena_dsp just one time is enough to enable dispatching again. The sophisticated operation when the pair of system calls tk_dis_dsp and tk_ena_dsp are used in a nested manner must therefore be managed by the user as necessary.
Additional Notes

A task in RUNNING state cannot go to DORMANT state or NON-EXISTENT state while dispatching is disabled. If `tk_ext_tsk` or `tk_exd_tsk` is called for a task in RUNNING state while interrupts or dispatching is disabled, error code E_CTX is detected. Since, however, `tk_ext_tsk` and `tk_exd_tsk` are system calls that do not return to their original context, such errors are not passed in return parameters by these system calls.
4.9.4  tk_ena_dsp - Enable Dispatch

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ena_dsp(void);

Parameter

None.

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

- E_OK    Normal completion
- E_CTX   Context error (issued from task-independent portion)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Enables task dispatching. This system call cancels the disabling of dispatching by the tk_dis_dsp system call. If tk_ena_dsp is called from a task not in dispatch disabled state, the dispatch enabled state continues and no error code is returned.
4.9.5  tk_ref_sys - Reference System Status

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ref_sys(T_RSYS *pk_rsys);

Parameter

T_RSYS* pk_rsys  Packet to Refer System Status  Pointer to the area to return the system status

Return Parameter

ER ercd  Error Code  Error code

pk_rsys Detail:

<table>
<thead>
<tr>
<th>UINT</th>
<th>sysstat</th>
<th>System State</th>
<th>System State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>runtskid</td>
<td>Running Task ID</td>
<td>ID of the task currently in RUNNING state</td>
</tr>
<tr>
<td>ID</td>
<td>schedtskid</td>
<td>Scheduled Task ID</td>
<td>ID of the task scheduled to run next</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

E_OK  Normal completion
E_PAR  Parameter error (invalid pk_rsys)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Gets the current system execution status, passing in return parameters such information as the dispatch disabled state and whether a task-independent portion is executing.

The following values are returned in `sysstat`.

```plaintext
sysstat := ( TSS_TSK | [TSS-DDSP] | [TSS-DINT] )
    || ( TSS_QTSK | [TSS-DDSP] | [TSS-DINT] )
    || ( TSS_INDP )
```
### Additional Notes

Depending on the kernel implementation, the information returned by `tk_ref_sys` is not necessarily guaranteed to be accurate at all times.
4.9.6  tk_set_pow - Set Power Mode

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_set_pow(UINT powmode);

Parameter

UINT  powmode   Power Mode  Low-power mode

Return Parameter

ER  ercd       Error Code  Error code

Error Code

E_OK   Normal completion
E_PAR  Parameter error (value that cannot be used in powmode )
E_QOVR Low-power mode disable count overflow
E_OBJ  TPW_ENALOWPOW was requested with low-power mode disable count at 0

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_LOWPOWER  Support of power management functions

Description

The following two power-saving functions are supported.

- Switching to low-power mode when the system is idle
  When there are no tasks to be executed, the system switches to a low-power mode provided in hardware.
  Low-power mode is a function for reducing power use during very short intervals, such as from one
timer interrupt to the next. This is accomplished, for example, by lowering the CPU clock frequency.
  It does not require complicated mode-switching in software but is implemented mainly using hardware
  functionality.

- Automatic power-off
  When the operator performs no operations for a certain length of time, the system automatically cuts the
  power and goes to suspended state. If there is a start request (interrupt, etc.) from a peripheral device
  or if the operator turns on the power, the system resumes from the state when the power was cut.
In the case of a power supply problem such as low battery, the system likewise cuts the power and goes to suspended state.

In suspended state, the power is cut to peripheral devices and circuits as well as to the CPU, but the main memory contents are retained.

\texttt{tk\_set\_pow} sets the low-power mode.

\begin{verbatim}
powmode := ( TPW\_DOSUSPEND || TPW\_DISLOWPOW || TPW\_ENALOWPOW )
\end{verbatim}

\begin{verbatim}
#define TPW\_DOSUSPEND 1 Suspended state
#define TPW\_DISLOWPOW 2 Switching to low-power mode disabled
#define TPW\_ENALOWPOW 3 Switching to low-power mode enabled (default)
\end{verbatim}

- \texttt{TPW\_DOSUSPEND}
  Execution of all tasks and handlers is stopped, peripheral circuits (timers, interrupt controllers, etc.) are stopped, and the power is cut (suspended). (\texttt{off\_pow} is called.)

  When power is turned back on, peripheral circuits are restarted, execution of all tasks and handlers is resumed, operations resume from the point before power was cut, and the system call returns.

  If for some reason the resume processing fails, normal startup processing (for reset) is performed and the system boots fresh.

- \texttt{TPW\_DISLOWPOW}
  Switching to low-power mode in the dispatcher is disabled. (\texttt{low\_pow} is not called.)

- \texttt{TPW\_ENALOWPOW}
  Switching to low-power mode in the dispatcher is enabled (\texttt{low\_pow} is called).

The default at system startup is low-power mode enabled (\texttt{TPW\_ENALOWPOW}).

Each time \texttt{TPW\_DISLOWPOW} is specified, the request count is incremented. Low-power mode is enabled only when \texttt{TPW\_ENALOWPOW} is requested for as many times as \texttt{TPW\_DISLOWPOW} was requested. The maximum request count is implementation-dependent, but a count of at least 255 times must be possible.

\section*{Additional Notes}

\texttt{off\_pow} and \texttt{low\_pow} are \textmu T-Kernel/SM functions. For more details, see Section 5.5, “Power Management Functions”.

\textmu T-Kernel does not detect power supply problems or other factors for suspending the system. Actual suspension requires suspend processing in each of the peripheral devices (device drivers). The system is suspended not by calling \texttt{tk\_set\_pow} directly but by use of the \textmu T-Kernel/SM suspend function.
4.9.7  tk_ref_ver - Reference Version Information

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ref_ver(T_RVER *pk_rver);

Parameter

<table>
<thead>
<tr>
<th>T_RVER* pk_rver</th>
<th>Packet to Return Version Information</th>
<th>Pointer to the area to return the version information</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

pk_rver Detail:

<table>
<thead>
<tr>
<th>UH maker</th>
<th>Maker Code</th>
<th>T-Kernel maker code</th>
</tr>
</thead>
<tbody>
<tr>
<td>UH prid</td>
<td>Product ID</td>
<td>T-Kernel identification number</td>
</tr>
<tr>
<td>UH spver</td>
<td>Specification Version</td>
<td>Specification version</td>
</tr>
<tr>
<td>UH prver</td>
<td>Product Version</td>
<td>T-Kernel version</td>
</tr>
<tr>
<td>UH prno[4]</td>
<td>Product Number</td>
<td>T-Kernel products management information</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_PAR</td>
<td>Parameter error (invalid pk_rver)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Gets information about the T-Kernel version in use, returning that information in the packet specified in pk_rver. The following information can be obtained.

maker is the maker code assigned to the developer who has implemented the version of μ T-Kernel. maker format is described in [Figure 4.7, “maker Format”].
**μT-Kernel 3.0 Specification**

**Figure 4.7: maker Format**

`prid` is a number indicating the T-Kernel type. The `prid` field has the format shown in Figure 4.8, “`prid Format`”.

Assignment of values to `prid` is left to the vendor who has implemented this version of μT-Kernel. Note, however, that this is the only number distinguishing product types, and that vendors should give careful thought to how they assign these numbers, doing so in a systematic way. In this way, the combination of `maker` and `prid` becomes a unique identifier of the kernel version.

The reference source code of μT-Kernel is provided from TRON Forum, and its `maker` and `prid` are as follows.

```
maker = 0x0000
prid = 0x0000
```

**Figure 4.8: prid Format**

The upper 4 bits of `spver` give the TRON specification series. The lower 12 bits indicate the T-Kernel specification version implemented. The `spver` field has the format shown in Figure 4.9, “`spver Format`”.

If, for example, a product conforms to the μT-Kernel specification Ver 3.01.xx, `spver` is as follows.

```
MAGIC = 0x6 (μT-Kernel)
SpecVer = 0x301 (Ver 3.01)
spver = 0x6301
```

If a product implements the draft version of μT-Kernel specification, that is, Ver 3.B0.xx draft specification, `spver` is as follows.

```
MAGIC = 0x6 (μT-Kernel)
SpecVer = 0x3B0 (Ver 3.B0)
spver = 0x63B0
```

**Figure 4.9: spver Format**

**MAGIC:**
Type of OS specification
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>TRON common (TAD, etc.)</td>
</tr>
<tr>
<td>0x1</td>
<td>reserved</td>
</tr>
<tr>
<td>0x2</td>
<td>reserved</td>
</tr>
<tr>
<td>0x3</td>
<td>reserved</td>
</tr>
<tr>
<td>0x4</td>
<td>reserved</td>
</tr>
<tr>
<td>0x5</td>
<td>reserved</td>
</tr>
<tr>
<td>0x6</td>
<td>μT-Kernel</td>
</tr>
<tr>
<td>0x7</td>
<td>T-Kernel</td>
</tr>
</tbody>
</table>

**SpecVer:**

The version of the specification that the kernel complies with. This is given as a three-digit packed-format BCD code. In the case of a draft version, the letter A, B, or C may appear in the second digit. In this case the corresponding hexadecimal form of A, B, or C is inserted.

**prver** is the version number of the T-Kernel implementation. The specific values assigned to **prver** are left to the T-Kernel implementing vendor to decide.

**prno** is a return parameter for use in indicating T-Kernel product management information, product number or the like. The specific meaning of values set in **prno** is left to the T-Kernel implementing vendor to decide.

**Additional Notes**

The format of the packet and structure members for getting version information is mostly uniform across each version of T-Kernel or μT-Kernel specification.

The value obtained by **tk_ref_ver** in **SpecVer** is the first three digits of the specification version number. The numbers after that indicate minor revisions such as those issued to correct misprints and the like, and are not obtained by **tk_ref_ver**. For the purpose of matching to the specification contents, the first three numbers of the specification version are sufficient.

A kernel implementing a draft version may have A, B, or C as the second number of **SpecVer**. It must be noted that in such cases the specification order of release may not correspond exactly to higher and lower **SpecVer** values. For example, specifications may be released in the following order: Ver 2.A1 → Ver 2.A2 → Ver 2.B1 → Ver 2.C1 → Ver 2.00 → Ver 2.01... In this example, when going from Ver 2.Cx to Ver 2.00, **SpecVer** goes from a higher to a lower value.
4.10 Subsystem Management Functions

Subsystem management functions extends the functions of μT-Kernel itself by adding a user-defined function called “subsystem” to the kernel in order to implement middleware and others running on the μT-Kernel. Some functions provided by μT-Kernel/SM are also implemented by utilizing the subsystem management functions.

A subsystem consists of extended SVC handlers to execute user-defined system calls (called “extended SVCs”), a break function that performs the required processing when any exception occurs, and an event handling function that performs the required processing when any event is raised from devices, etc. (Figure 4.10, “μT-Kernel Subsystems”)

![Diagram of subsystem management functions](image)

The extended SVC handler directly accepts requests from applications and others. A break function and event processing function are so-called callback type functions and accept requests from the kernel.

Additional Notes

Generally speaking, upper layer middleware including the process management functions and the file management functions are also implemented by utilizing the subsystem management functions. Other examples of middleware that are implemented by utilizing the subsystem management functions include TCP/IP manager, USB manager, and PC card manager.

Though subsystem management functions are meant to allow users to add custom system calls (SVC: Supervisor Calls) as the primary purpose, they can be used to build complex and advanced middleware through not only the addition of just user-defined system calls but also through provision of exception processing functions to handle the exceptions, which are required for the added system calls.

In addition to the subsystem management functions, μT-Kernel also provides the device driver functions in order to extend itself. Both subsystems and device drivers are function modules independent from μT-Kernel itself. They can be used by loading their corresponding binary programs and then calling them from a task on μT-Kernel. Both run at the protection level 0. While API is limited to using open/close and read/write type when calling a device driver, API for calling a subsystem can be defined without any restriction.

Subsystems are identified by subsystem IDs (ssid), more than one subsystem can be defined and used at the same time. One subsystem can be called and used from within another subsystem.
4.10.1  tk_def_ssy - Define Subsystem

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_def_ssy(ID ssid, CONST T_DSSY *pk_dssy);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>ssid</th>
<th>Subsystem ID</th>
<th>Subsystem ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_DSSY</td>
<td>pk_dssy</td>
<td>Packet to Define Subsystem definition information</td>
<td></td>
</tr>
</tbody>
</table>

pk_dssy Detail:

<table>
<thead>
<tr>
<th>ATR</th>
<th>ssyatr</th>
<th>Subsystem Attributes</th>
<th>Subsystem attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRI</td>
<td>ssypri</td>
<td>Subsystem Priority</td>
<td>Subsystem priority</td>
</tr>
<tr>
<td>FP</td>
<td>svchdr</td>
<td>Extended SVC Handler Address</td>
<td>Extended SVC handler address</td>
</tr>
<tr>
<td>FP</td>
<td>breakfn</td>
<td>Break Function Address</td>
<td>Break function address</td>
</tr>
<tr>
<td>FP</td>
<td>eventfn</td>
<td>Event Handling Function Address</td>
<td>Event handling function address</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

E_OK  Normal completion
E_ID  Invalid ID number (ssid is invalid or cannot be used)
E_NOMEM Insufficient memory (memory for control block cannot be allocated)
E_RSATR Reserved attribute (ssyatr is invalid or cannot be used)
E_PAR Parameter error (pk_dssy is invalid or cannot be used)
E_OBJ  ssid is already defined (when pk_dssy ≠ NULL)
E_NOEXS ssid is not defined (when pk_dssy = NULL)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_SUBSYSTEM Support of subsystem management functions
When the service profile items below is set to be effective, subsystem priority ($ssypri$) can be specified.

**TK_SUPPORT_SYSEVENT**  
Support of event processing of subsystems

Only when the service profile items below are set to be effective, break function can be specified.

**TK_SUPPORT_TASKEXCEPTION**  
Support of task exception handling functions

**Description**

Defines subsystem specified in $ssid$.

One subsystem ID must be assigned to one subsystem without overlapping with other subsystems. The kernel does not have a function for assigning subsystem IDs automatically.

Subsystem IDs 1 to 9 are reserved for $\mu$T-Kernel use. 10 to 255 are numbers used by middleware, etc. The maximum usable subsystem ID value is implementation-dependent and may be lower than 255 in some implementations.

$syattr$ indicates system attributes in its lower bits and implementation-dependent attributes in its higher bits. The system attribute in $syattr$ are not assigned in this version, and no system attributes are used.

$sypr$ indicates the subsystem priority. The startup function, cleanup function, and event handling function are called in order of priority. The calling order is undefined when these subsystems have the same priority. Subsystem priority 1 is the highest priority, with larger numbers indicating lower priorities. The range of priorities that can be specified is implementation-dependent, but it must be possible to assign at least priorities 1 to 16.

$NULL$ can be specified in $breakfn$ and $eventfn$, in which case the corresponding function will not be called. Specifying $pk_dssy = NULL$ deletes a subsystem definition.

• **Extended SVC handler**

An extended SVC handler accepts requests from applications and other programs as an application programming interface (API) for a subsystem. It can be called in the same way as an ordinary system call, and is normally invoked using a trap instruction or the like.

The format of an extended SVC handler is as follows.

```c
INT svchdr( void *pk_para, FN fncl )
{
    /*
        branching by fncl
    */
    return retcode; /* exit extended SVC handler */
}
```

$fncl$ is a function code. The lower 8 bits of the instruction code are the subsystem ID. The remaining higher bits can be used in any way by the subsystem. Ordinarily they are used as a function code inside the subsystem. A function code must be a positive value, so the most significant bit is always 0.

$pk_para$ points to a packet of parameters passed to this system call. The packet format can be decided by the subsystem. Generally a format like the stack passed to a C language function is used, which in many cases is the same format as a C language structure.

The return code passed from an extended SVC handler is passed to the caller transparently as the function return code. As a rule, negative values are error codes and 0 or positive values are the return code for normal completion. If an extended SVC call fails for some reason, the error code (negative value) set by T-Kernel is returned to the caller without invoking the extended SVC handler, so it is best to avoid confusion with these values.
The format by which an extended SVC is called is dependent on the kernel implementation. As a subsystem API, however, it must be specified in a C language function format independent of the kernel implementation. The subsystem must provide an interface library for converting from the C language function format to the kernel-dependent extended SVC calling format.

An extended SVC handler runs as a quasi-task portion.

It can be called from a task-independent portion, and in this case the extended SVC handler also runs as a task-independent portion.

- **Break function**

  A break function is a function called when a task exception is raised for a task while an extended SVC handler is executing.

  When a break function is called, the processing by the extended SVC handler running at the time the task exception was raised must be stopped promptly and control must be returned from the extended SVC handler to its caller. The role of a break function is to abort the processing of the currently running extended SVC handler.

  The format of a break function is as follows.

  ```c
  void breakfn( ID tskid )
  {
      /*
         stop the running extended SVC handler
      */
  }
  ```

  `tskid` is the ID of the task in which the task exception was raised.

  A break function is called when a task exception is raised by `tk_ras_tex`. If extended SVC handler calls are nested, then when the nesting level of the extended SVC handler is decreased by the return from the latest extended SVC handler, the break function corresponding to the former extended SVC handler to which the control will be returned next, is called.

  A break function is called only once for one extended SVC handler per one task exception.

  If another nested extended SVC call is made while a task exception is raised, no break function is called for the called extended SVC handler.

  A break function runs as a quasi-task portion. Its requesting task is identified as follows: If a break function is called by `tk_ras_tex`, it runs as a quasi-task portion of the task that issued `tk_ras_tex`. On the other hand, when the nesting level of extended SVC handler is decreased, the break function runs as a quasi-task portion of the task that raised the task exception (the task running the extended SVC handler). This means that the task executing the break function may be different from the task executing the extended SVC handler. In such a case, the break function and extended SVC handler run concurrently as controlled by task scheduling.

  It is thus conceivable that the extended SVC handler will return to its caller before the break function finished executing, but in that case the extended SVC handler waits at the point right before returning, until the break function completes. How this waiting state maps to the task state transitions is implementation-dependent, but preferably it should remain in READY state (a READY state that does not go to RUNNING state). The precedence of a task may change while it is waiting for a break function to complete, but how task precedence is treated is implementation-dependent.

  Similarly, an extended SVC handler cannot call an extended SVC until break function execution completes.

  In other words, during the time from the raising of a task interrupt until the break function completes, the affected task must stay in the extended SVC handler that was executing at the time of the task exception.

  In the case where the requesting task of the break function differs from that of the extended SVC handler, that is, where the break function and the extended SVC handler run in different task contexts, the task priority of the break function is raised to the same as that of the extended SVC handler only while the break handler is executing if the former is lower than the latter. On the other hand, if the break function task priority is the same as or higher than that of the extended SVC handler, the priority does not change. The priority that gets changed is the current priority; the base priority stays the same.
The change in priority occurs only immediately before entry into the break function; any changes after
that of the extended SVC handler task priority are not followed by further changes in priority of the break
function task. In no case does a change in the break function priority while a break function is running
results in a priority change in the extended SVC handler task. At the same time, there is no restriction on
priority changes due to a running break function.

When the break function completes, the current priority of its task reverts to base priority. If a mutex was
locked, however, the priority reverts to that as adjusted by the mutex. (In other words, the ability is provided
to adjust the current priority at the entry and exit of the break function only; other than that, the priority is
the same as when an ordinary task is running.)

- Event handling function

An event handling function is called by issuing the `tk_evt_ssy` system call.

It processes various requests made to a subsystem.

Note that it has to process all requests for all subsystems. If processing is not required, it can simply return
E_OK without performing any operation.

The format of an event handling function is as follows.

```c
ER eventfn( INT evtty, INT info )
{
    /*
        event processing
    */
    return ercd;
}
```

`evtty` indicates the request type and `info` is a parameter that can be used freely. These parameters are
specified in `tk_evt_ssy`.

If processing completes normally, E_OK is passed in the return code; otherwise an error code (negative
value) is returned.

The following event types `evtty` are defined. For more details, see Section 5.2, “Device Management Func-
tions”.

```c
#define TSEVT_SUSPEND_BEGIN    1 /* before suspending device */
#define TSEVT_SUSPEND_DONE      2 /* after suspending device */
#define TSEVT_RESUME_BEGIN      3 /* before resuming device */
#define TSEVT_RESUME_DONE       4 /* after resuming device */
#define TSEVT_DEVICE_REGIST     5 /* device registration notice */
#define TSEVT_DEVICE_DELETE     6 /* device deletion notice */
```

An event handling function runs as a quasi-task portion of the task that issued `tk_evt_ssy`.

Additional Notes

Extended SVC handlers as well as break functions and event handling functions all have the equivalent of the
`TA_HLNG` attribute only. There is no means of specifying the `TA_ASM` attribute.

It is possible to issue a system call that enters WAITING state in the extended SVC handler, but in that case
the program must be designed so that it can be stopped by calling a break function. The specific processing
flow is as follows: If `tk_ras_tex` is issued for the caller task while an extended SVC handler is executing, it
is necessary to stop the running extended SVC handler as soon as possible and return a stop error to the
caller task. For this purpose the break function is used. In order to stop the running extended SVC handler
immediately, the break function must forcibly release the WAITING state, even if the system call is in WAITING
state during processing the extended SVC handler. For this purpose, the `tk_dis_wai` system call is generally
used. `tk_dis_wai` can prevent the system call from entering WAITING state until the control returns from the
extended SVC handler to the caller task, but the implementor should also make it possible to stop the program.
of the extended SVC handler by calling a break function. For example, leaving from WAITING state with the 
error code E_DISWAI can mean that the execution is stopped by a break function. So it is best to stop the 
extended SVC handler immediately and return a stop error to the caller task, without continuing to execute 
the subsequent processing.

An extended SVC handler may be called concurrently by multiple tasks. If the tasks share same resources, 
the mutual exclusion control must be performed in the extended SVC handler.

**Porting Guideline**

Note that, in an environment where INT data type is 16 bits, part of function code that can be used for 
subsystem function code is only 7 bits wide (0-127), and care must be taken.
4.10.2  tk_evt_ssy - Call Event Function

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_evt_ssy(ID ssid, INT evttyp, ID resid, INT info);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>ssid</th>
<th>Subsystem ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>evttyp</td>
<td>Event Type</td>
</tr>
<tr>
<td>INT</td>
<td>info</td>
<td>Information</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_ID: Invalid ID number (ssid is invalid or cannot be used)
- E_NOEXS: Object does not exist (the subsystem specified in ssid is not defined)
- E_CTX: Context error (issued from task-independent portion, or in dispatch disabled state)
- Other: Error code returned by the event handling function

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- TK_SUPPORT_SUBSYSTEM: Support of subsystem management functions
- TK_SUPPORT_SSYEVENT: Support of event processing of subsystems

Description

Calls the event handling function of the subsystem specified in ssid.

Specifying ssid = 0 makes the system call applied to all currently defined subsystems. In this case the event handling function of each subsystem is called in sequence.

When evttyp is an odd number:
- Calls subsystems in descending order of priority.

When evttyp is an even number:
- Calls subsystems in ascending order of priority.
The calling order is undefined when these subsystems have the same priority.

If this system call is issued for a subsystem with no event handling function defined, the function is simply not called; no error results.

If the event handling function returns an error, the error code is passed transparently in the system call return code. When ssid = 0 and an event handler returns an error, the event handling functions of all other subsystems continue to be called. In the system call return code, only one error code is returned even if more than one event handling function returned an error. It is not possible to know which subsystem’s event handling function returned the error.

If a task exception is raised for the task that called tk_evt_ssy, during the execution of event handling function, the task exception is held until the event handling function completes its processing.

Additional Notes

An example of using an event handling function is to perform the suspend/resume processing for the power management functions. Specifically, when the system enters the power-off state (device suspended state) due to power failure or other reason, it notifies each subsystem of its transition to suspended state. Then the event handling function of each subsystem is called to perform the appropriate processing for it. In µT-Kernel/SM, tk_evt_ssy is executed for this purpose during the processing of tk_sus_dev. The event handling function of each subsystem performs any necessary operations before going to suspended state, such as saving the data. On the other hand, when the system returns (resumes) from the suspended state due to power on or other reason, it notifies each subsystem of its return from suspended state. Then the event handling function of each subsystem is called again to perform the appropriate processing for it. For more details, see the description of tk_sus_dev.

For another example, when a new device is registered by tk_def_dev, the system notifies each subsystem of the registration, and the event handling function of each subsystem is called to perform the appropriate processing for it. In µT-Kernel/SM, tk_evt_ssy is executed for this purpose during the processing of tk_def_dev.

Porting Guideline

Note that info is INT type, and its value range is implementation-dependent, so care must be taken.
4.10.3  tk_ref_ssy - Reference Subsystem Status

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_ref_ssy(ID ssid, T_RSSY *pk_rssy);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>ssid</th>
<th>Subsystem ID</th>
<th>T_RSSY*</th>
<th>pk_rssy</th>
<th>Subsystem ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Packet to Return Subsystem Status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

pk_rssy Detail:

<table>
<thead>
<tr>
<th>PRI</th>
<th>sspri</th>
<th>Subsystem Priority</th>
<th>Subsystem priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Other implementation-dependent parameters may be added beyond this point.)</td>
<td></td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Invalid ID number ($ssid$ is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the subsystem specified in $ssid$ is not defined)</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (invalid $pk_rssy$)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_SUBSYSTEM

Support of subsystem management functions

When the service profile items below is set to be effective, subsystem priority ($ssypri$) can be acquired.

TK_SUPPORT_SSYEVENT

Support of event processing of subsystems

Description

References information about the subsystem specified in $ssid$. 
ssypr returns the subsystem priority specified in tk_def_ssy.

If the subsystem specified in ssid is not defined, E_NOEXS is returned.
Chapter 5

μT-Kernel/SM Functions

This chapter describes details of the functions provided by μT-Kernel/SM (System Manager).

Overall Note and Supplement

• There are two types of API names that are defined in μT-Kernel/SM specification: ones beginning with ‘tk_’ and others. It is generally assumed that APIs with a name beginning with ‘tk_’ are implemented using extended SVC (a Subsystem Management Function), and other APIs are implemented as library functions (including in-line functions) or macros of the C language. However, μT-Kernel specification does not define the implementation of these APIs. So the developers are free to adopt different implementation methods. API implemented by libraries and macros may call extended SVCs or system calls indirectly.

• Error codes such as E_PAR, E_MACV, and E_NOMEM that can be returned in many situations are not described here always unless there is some special reason for doing so.

• Except where otherwise noted, extended SVC and libraries of μT-Kernel/SM cannot be called from a task-independent portion and while dispatching and interrupts are disabled. There may be some limitations, however, imposed by particular implementations (E_CTX).

• Extended SVC and libraries of μT-Kernel/SM cannot be invoked from a lower protection level than that at which T-Kernel/OS system calls can be invoked (lower than TSVCLimit)(E_OACV).

• Extended SVC and libraries of μT-Kernel/SM are reentrant except when a special explanation is given. Note that some functions perform mutual exclusion internally.
5.1 System Memory Management Functions

The system memory management functions manage all the memory (system memory) allocated dynamically by \( \mu \)T-Kernel. This includes memory used internally by \( \mu \)T-Kernel as well as task stacks, message buffers, and memory pools.

System memory management functions include memory allocation libraries that manage memory through subdividing system memory into smaller blocks.

The system memory management functions are for use not only within \( \mu \)T-Kernel but also used by applications, subsystems, and device drivers.
5.1.1 Memory Allocation Library Functions

Memory allocation library provides functions equivalent to \texttt{malloc/calloc/realloc/free} provided by C standard library.

These memories are all allocated as memory with a protection level specified in \texttt{TSVCLimit}.
5.1.1.1 Kmalloc - Allocate Memory

C Language Interface

#include <tk/tkernel.h>

void* Kmalloc(size_t size);

Parameter

size_t size

Size Memory size to be allocated (in bytes)

Return Parameter

void* addr

Memory Start Address Start address of the allocated memory

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_MEMLIB Support of memory allocation library

Description

Allocates the memory of bytes specified in size and returns the start address of the allocated memory in addr. When the specified size of memory cannot be allocated or 0 is specified in size, NULL is returned in addr. APIs in the memory allocation library, including Kmalloc, cannot be called from a task-independent portion and while dispatch or interrupt is disabled. Such a call may lead to an undefined behavior including possible system failure, and the caller is responsible for guaranteeing the state on the call.

Additional Notes

Any value can be specified in size. Note that a larger memory size than the number of bytes specified in size may be allocated internally due to allocating the management space, aligning the allocated memory address, or other reasons. For example, when the implementation specifies that the least allocatable memory size is 16 bytes and the alignment is an 8-byte unit, 16-byte memory is allocated internally even if a value less than 16 bytes is specified in size. Similarly, 24-byte memory is allocated even if 20 bytes is specified in size.
5.1.1.2 Kcalloc - Allocate Memory and Clear

C Language Interface

#include <tk/tkernel.h>

void* Kcalloc(size_t nmemb, size_t size);

Parameter

<table>
<thead>
<tr>
<th>size_t</th>
<th>nmemb</th>
<th>Number of Memory Blocks</th>
<th>Number of memory blocks to be allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t</td>
<td>size</td>
<td>Size</td>
<td>Memory block size to be allocated (in bytes)</td>
</tr>
</tbody>
</table>

Return Parameter

| void*  | addr  | Memory Start Address       | Start address of the allocated memory |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_MEMLIB Support of memory allocation library

Description

Allocates the specified number (nmemb) of contiguous memory blocks of the specified bytes (size), clears them with 0, then returns the start address of them in addr. This memory allocation operation is identical to allocating one memory block of the number of size * nmemb bytes.

When the specified number of memory blocks cannot be allocated or 0 is specified in nmemb or size, NULL is returned in addr.

APIs in the memory allocation libraries, including Kcalloc, cannot be called from a task-independent portion and while dispatch or interrupt is disabled. Such a call may lead to an undefined behavior including possible system failure, and the caller is responsible for guaranteeing the state on the call.

Additional Notes

A larger memory size than the number of size * nmemb bytes may be allocated internally. For more details, see the additional note for Kmalloc.
5.1.1.3 Krealloc - Reallocate Memory

C Language Interface

#include <tk/tkernel.h>

void* Krealloc(void *ptr, size_t size);

Parameter

<table>
<thead>
<tr>
<th>void*</th>
<th>ptr</th>
<th>Pointer to Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t</td>
<td>size</td>
<td>Size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reallocated memory size (in bytes)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>void*</th>
<th>addr</th>
<th>Memory Start Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Start address of the reallocated memory</td>
</tr>
</tbody>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_MEMLIB Support of memory allocation library

Description

Changes the size of the previously allocated memory specified in `ptr` to the size specified in `size`. At that time, reallocates the memory and returns the start address of the reallocated memory in `addr`.

Generally, `addr` results in different value from `ptr` because the memory start address is moved by reallocating the memory with resizing. The content of the reallocated memory is retained. To do so, the memory content is copied during the `Krealloc` processing. The memory that becomes free by reallocation will be released.

The start address of the memory allocated previously by `Kmalloc`, `Kcalloc`, or `Krealloc` must be specified in `ptr`. The caller must guarantee the validity of `ptr`.

If `NULL` is specified in `ptr`, only the new memory allocation is performed. This operation is identical to `Kmalloc`.

When the specified size of memory cannot be reallocated or 0 is specified in `size`, `NULL` is returned in `addr`.

In this case, the memory specified by `ptr` is only released if a value other than `NULL` is specified in `ptr`. This operation is identical to `Kfree`.

APIs in the memory allocation library, including `Krealloc`, cannot be called from a task-independent portion and while dispatch or interrupt is disabled. Such a call may lead to an undefined behavior including possible system failure, and the caller is responsible for guaranteeing the state on the call.
Additional Notes

The memory address returned in `addr` may be the same as `ptr` in some cases, for example, when the memory size becomes smaller than before by reallocation or when the reallocation is performed without moving the memory start address because an unallocated memory area was around the memory specified in `ptr`.

A larger memory size than the number of bytes specified in `size` may be allocated internally. For more details, see the additional note for `Kmalloc`. 
5.1.1.4  Kfree - Release Memory

C Language Interface
#include <tk/tkernel.h>

void Kfree(void *ptr);

Parameter

void*  ptr  Pointer to Memory  Start address of memory to be released

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_MEMLIB  Support of memory allocation library

Description

Releases the memory specified in ptr.

The start address of the memory allocated previously by Kmalloc, Kcalloc, or Krealloc must be specified in ptr. The caller must guarantee the validity of ptr.

APIs in the memory allocation library, including Kfree, cannot be called from a task-independent portion and while dispatch or interrupt is disabled. Such a call may lead to an undefined behavior including possible system failure, and the caller is responsible for guaranteeing the state on the call.
5.2 Device Management Functions

Device management functions manage device drivers running on \( \mu \text{T-Kernel} \).

A device driver is a program that is implemented independent from \( \mu \text{T-Kernel} \) itself to control a hardware device or perform I/O processing with the hardware device. Since the difference of specifications among individual devices is absorbed by the device driver when an application or middleware operates a device or performs I/O processing with the device via the device driver, the application or middleware can enhance its hardware independence and compatibility.

Device management functions include a function to define a device driver, or to register the device driver to \( \mu \text{T-Kernel} \), and a function to use the registered device driver from an application or middleware.

While this registration of device drivers is mostly performed in the initialization at system startup, it can also be performed dynamically during the normal operation of the system. A device driver is registered in the device registration information (\texttt{ddev}) that is one of parameters for the API, \texttt{tk_def_dev}, by specifying the set of functions (driver processing functions) of a program that actually implements device driver. These functions include the open function (\texttt{openfn}) that is called when a device is opened, the execute function (\texttt{execfn}) that is called when read or write processing starts, wait-for-completion function (\texttt{waitfn}) that waits for completion of read or write processing, etc. The actual operation of a device or I/O processing with the devices are performed in these driver processing functions.

As these driver execute functions are executed at protection level 0 as quasi-task portion, they can also access hardware directly. I/O processing with a device may be performed directly in these driver execute functions or may be performed in another task that runs based on the request from one of these driver execute functions. The specification of parameters, etc. when these driver execute functions are called is defined as part of the device driver interface. The device driver interface is an interface between a device driver and the \( \mu \text{T-Kernel} \) device management functions.

When a device driver program is implemented, it is recommended to separate three layers of interface, logical, and physical layers carefully in order to enhance their maintainability and portability. The interface layer is responsible for implementing an interface between the \( \mu \text{T-Kernel} \) device management functions and a device driver. The logical layer is responsible for performing a common processing according to the type of device. The physical layer is responsible for performing an operation dependent on the actual hardware or control chip. The interface specification, however, among the interface layer, logical layer, and physical layer is not specified in the \( \mu \text{T-Kernel} \), so that the actual layer separation can be implemented appropriately in each device driver. Programs that process the interface layer may be provided as libraries since there are many common processing steps that are independent of individual devices in the physical layer.

APIs are provided such as open (\texttt{tk_opn_dev}), close (\texttt{tk_cls_dev}), read (\texttt{tk_rea_dev}), write (\texttt{tk_wri_dev}), etc. to use the registered device driver from an application or middleware. The specification of these APIs is called an application interface. For example, when an application executes \texttt{tk_opn_dev} to open a device, the \( \mu \text{T-Kernel} \) calls the open function (\texttt{openfn}) for the corresponding device driver to request the device open processing.

The positioning and structure of \( \mu \text{T-Kernel} \) device management functions are shown in Figure 5.1, “Device Management Functions”.

Additional Notes
The device drivers have common features with the subsystems as being implemented independent from μT-Kernel itself and also being a system program to add or extend functions for μT-Kernel. Additionally, both are also same in that they operate at protection level 0, and can access a hardware. Notable differences between the two, is that while API for calling a device driver is limited to using open/close and read/write type, API for calling a subsystem can be defined without any restriction.
Though μT-Kernel device drivers managed by device management functions are assumed to be drivers for physical devices or hardware, they are not necessarily required to handle real physical devices or hardware. Also, system program for operating a device could be implemented as a subsystem rather than a device driver if it is not compatible with open/close or read/write type APIs.
5.2.1 Common Notes Related to Device Drivers

5.2.1.1 Basic Concepts

In addition to a physical device that represents a device as a physical hardware, there is a logical device that represents a perceived unit of a device from the viewpoint of software.

Although both devices match for most devices, when partitions were created on a hard disk or any other storage type device (SD card, USB storage, etc.), entire device represents a physical device and each partition represents a logical device.

The physical devices of same type are identified by “unit” while logical devices in one physical device are identified by “subunit.” For example, the information that distinguishes the first hard disk from the second is called “unit,” and the information that distinguishes the first partition from the second within that first hard disk is called “subunit.”

The data definitions used in device management functions are explained in the subsequent subsections.

In the following description, the references and mentions are made to particular types of devices and their names. These are not meant to be the part of μT-Kernel specification, but rather are offered as a common guideline for defining device driver specifications. Each device driver does not have to implement all the functions described here. However, each driver should be designed so that their behavior is compliant with the description in the following if applicable.

5.2.1.1.1 Device Name (UB* type)

A device name is a string of up to eight characters that is given to each device. US-ASCII is the used character code. It consists 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 of type + unit + subunit. Some devices may not have a unit or subunit, in which case the corresponding field is omitted.

The subunit is usually used to distinguish partitions in a hard disk. In other devices also, it can be used to create multiple logical devices in one physical device.

A name consisting of type + unit is called a physical device name. A name consisting of type + unit + subunit is called a logical 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.
### 5.2.1.1.2 Device ID (ID type)

By registering a device (device driver) with \(\mu\) 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).

- **devid**: The device ID assigned at device registration
- **devid + n+1**: The nth subunit (logical device)

<table>
<thead>
<tr>
<th>Device name</th>
<th>Target device</th>
<th>Example of Device ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>hda</td>
<td>Hard disk (entire disk)</td>
<td>devid</td>
</tr>
<tr>
<td>hda0</td>
<td>Hard disk (1st partition)</td>
<td>devid + 1</td>
</tr>
<tr>
<td>fda</td>
<td>Floppy disk</td>
<td>devid + 2</td>
</tr>
<tr>
<td>rsa</td>
<td>Serial port</td>
<td></td>
</tr>
<tr>
<td>kbdp</td>
<td>Keyboard/pointing device</td>
<td></td>
</tr>
<tr>
<td>fla</td>
<td>Flash memory</td>
<td></td>
</tr>
<tr>
<td>neta</td>
<td>Network</td>
<td></td>
</tr>
</tbody>
</table>

### 5.2.1.1.3 Device Attribute (ATR type)

Device attributes are defined in order to represent a feature for each device and classify a device for each type. Device attributes should be specified when registering a device driver.

The specification method of device attributes is as follows:

```
III IIII IIII IIII PRxx xxxx 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 protected */
#define TD_REMOVABLE 0x4000 /* R: removable media */
#define TD_DEVKIND 0x00ff /* K: device/media kind */
#define TD_DEVTYPE 0x00f0 /* device type */
#define TDK_UNDEF 0x0000 /* undefined/unknown */
#define TDK_DISK 0x0010 /* disk device */
```
Within the realm of \(\mu\)-T-Kernel, the device type other than disk type is not defined. Defining the device type other than disk type does not affect the behavior of \(\mu\)-T-Kernel. Other devices are assigned to undefined type (TDK_UNDEF).

For the disk device, the disk kinds are additionally defined. The typical disk kinds are as follows:

```c
#define TDK_DISK_UNDEF 0
#define TDK_DISK_RAM 0x0011 /* RAM disk (used as main memory) */
#define TDK_DISK_ROM 0x0012 /* ROM disk (used as main memory) */
#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_CDROM 0x0016 /* CD-ROM */
```

The definition of disk kinds does not affect the \(\mu\)-T-Kernel behavior. These definitions are used only when they are required in a device driver or an application. For example, when an application must change its processing according to the kind of devices or media, the disk kind information is used. Devices or media that do not need such distinctions do not have to be assigned a device type.

### 5.2.1.1.4 Device Descriptor (ID type)

A device descriptor is an identifier used to access a device. The device descriptor is assigned a positive value (> 0) by the \(\mu\)-T-Kernel/SM when a device is opened.

### 5.2.1.1.5 Request ID (ID type)

When an I/O request is made to a device, a request ID (> 0) is assigned identifying the request. This ID can be used to wait for I/O completion.

### 5.2.1.1.6 Data Number (W type, D type)

Data input/output from/to device is specified by a data number. Data is roughly classified into device-specific data and attribute data.

**Device-specific data: Data number \(\geq 0\)**

As device-specific data, the data numbers are defined separately for each device.

#### Example of Device-specific Data

<table>
<thead>
<tr>
<th>device</th>
<th>Data number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>Data number = physical block number</td>
</tr>
<tr>
<td>Serial port</td>
<td>Data number = 0 only</td>
</tr>
</tbody>
</table>

**Attribute data: Data number < 0**

Attribute data specifies driver or device state acquisition and setting modes, and special functions, etc. Data numbers common to devices are defined, but device-dependent attribute data can also be defined. For more details, see Section 5.2.1.2, “Attribute Data”.
5.2.1.2 Attribute Data

Attribute data are classified broadly into the following three types of data.

Common attributes
Attributes defined in common for all devices (device drivers).

Device kind attributes
Attributes defined in common for devices (device drivers) of the same kind.

Device-specific attributes
Attributes defined individually for each device (device driver).

Device kind attributes and device-specific attributes are out of scope of this specification and defined in device driver’s specifications. Only the common attributes are defined here.

Common attributes are assigned attribute data numbers in the range from -1 to -99. While common attribute data numbers are the same for all devices, not all devices necessarily support all the common attributes. If an unsupported data number is specified, error code E_PAR is returned.

The definition of common attributes is as follows:

```c
#define TDN_EVENT (-1) /* RW: event notification message buffer ID */
#define TDN_DISKINFO (-2) /* R: disk information */
#define TDN_DISPSPEC (-3) /* reserved */
#define TDN_PCMCIAINFO (-4) /* reserved */
#define TDN_DISKINFO_D (-5) /* R: disk information (64-bit device) */
```

**TDN_EVENT**

Event notification message buffer ID

<table>
<thead>
<tr>
<th>Data type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The ID of the message buffer used for device event notification.

As a device is registered by `tk_def_dev` when a device driver is started and the system default event notification message buffer ID (`evtmbfid`) is returned as this API return parameter, the value is held in the device driver and is used as the initial value of this attribute data.

If 0 is set, device events are not notified. For device event notification, see Section 5.2.3.3, “Device Event Notification”.

**TDN_DISKINFO**

32-bit device and disk information

<table>
<thead>
<tr>
<th>Data type</th>
<th>DiskInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
typedef enum {
    DiskFmt_STD = 0, /* standard (HD, etc.) */
    DiskFmt_CDROM = 4 /* CD-ROM 640MB */
} DiskFormat;

typedef struct {
    DiskFormat format; /* format */
    UW protect:1; /* protected status */
    UW removable:1; /* removable */
    UW rsv:30; /* reserved (always 0) */
} DiskInfo;
```
μT-Kernel 3.0 Specification

Blocksize

```
W blocksize; /* block size in bytes */
W blockcont; /* total block count */
```

For definition of DiskFormat other than the above description, see the specification related to device drivers.

**TDN_DISPSPEC**
Display Device Specification

Data type DEV_SPEC

For the definition of DEV_SPEC, see the specification related to device drivers.

**TDN_DISKINFO_D**
64-bit device and disk information

Data type DiskInfo_D

```c
typedef struct diskinfo_d {
    DiskFormat format; /* format */
    BOOL protect:1; /* protected status */
    BOOL removable:1; /* removable */
    UW rsv:30; /* reserved (0) */
    W blocksize; /* block size in bytes */
    D blockcont_d; /* total number of blocks in 64-bit */
} DiskInfo_D;
```

Difference between DiskInfo_D and DiskInfo is only the part of their names being blockcont or blockcont_d, and the data type.

μT-Kernel/SM does not convert a data between DiskInfo and DiskInfo_D. **TDN_DISKINFO** and **TDN_DISKINFO_D** just pass the request to device driver without any modification.

The disk device driver must support one of **TDN_DISKINFO** and **TDN_DISKINFO_D**, or both. It is recommended that **TDN_DISKINFO** is supported wherever possible.

Even if the total number of blocks of entire disk exceeds W, the number of blocks of individual partition may fit within W. In that case, the preferable implementation is such that a partitions fitting within W correspond to **TDN_DISKINFO** and partitions not fitting within W are determined to be an error (E_PAR) by **TDN_DISKINFO**. It is also preferable that **TDN_DISKINFO_D** is supported even if the number of blocks fit within W.

There is no direct dependency between the support for **TDN_DISKINFO_D** and the device driver attribute **TDA_DEV_D**. A device driver does not always have **TDA_DEV_D** attribute even if **TDN_DISKINFO_D** is supported. Also, **TDN_DISKINFO_D** is not always supported even if the device driver has **TDA_DEV_D** attribute.

As the definition of common attributes described above is a part of the specification of device driver rather than μT-Kernel, it does not directly affect the μT-Kernel behavior. Each device driver does not need to implement all the functions defined in the common attributes. However, as the definition of common attributes is applicable to all the device drivers, the specification of each device driver must be specified in a way that does not conflict with these definitions.
5.2.2 Device Input/Output Operations

The application interface is used to make use of the registered device drivers from an application or middleware. API of µT-Kernel provides the following functions. These functions cannot be called from a task-independent portion or while dispatch or interrupts are disabled (E_CTX).

```c
ID tk_opn_dev( CONST UB *devnm, UINT omode )
ER tk_cls_dev( ID dd, UINT option )
ID tk_rea_dev( ID dd, W start, void *buf, SZ size, TMO tmout )
ID tk_rea_dev_du( ID dd, D start_d, void *buf, SZ size, TMO_U tmout_u )
ER tk_srea_dev( ID dd, W start, void *buf, SZ size, TMO_U tmout_u )
ID tk_srea_dev_d( ID dd, D start_d, void *buf, SZ size, TMO_U tmout_u )
ID tk_wri_dev( ID dd, W start, CONST void *buf, SZ size, TMO tmout )
ER tk_swri_dev( ID dd, W start, CONST void *buf, SZ size, TMO_U tmout_u )
ID tk_wri_dev_du( ID dd, D start_d, CONST void *buf, SZ size, TMO_U tmout_u )
ER tk_swri_dev_d( ID dd, D start_d, CONST void *buf, SZ size, TMO_U tmout_u )
ID tk_wai_dev( ID dd, ID reqid, SZ *asize, ER *ioer, TMO tmout )
ID tk_wai_dev_u( ID dd, ID reqid, SZ *asize, ER *ioer, TMO_U tmout_u )
INT tk_sus_dev( UINT mode )
ID tk_get_dev( ID devid, UB *devnm )
ID tk_ref_dev( CONST UB *devnm, T_RDEV *rdev )
ID tk_oref_dev( ID dd, T_RDEV *rdev )
INT tk_lst_dev( T_LDEV *ldev, INT start, INT ndev )
INT tk_evt_dev( ID devid, INT evttyp, void *evtinf )
```
5.2.2.1  tk_opn_dev - Open Device

C Language Interface

```c
#include <tk/tkernel.h>

ID dd = tk_opn_dev(CONST UB *devnm, UINT omode);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST UB* <code>devnm</code></td>
<td>Device Name</td>
</tr>
<tr>
<td>UINT <code>omode</code></td>
<td>Open Mode</td>
</tr>
</tbody>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID <code>dd</code></td>
<td>Device Descriptor</td>
</tr>
<tr>
<td>or Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

**Error Code**

- E_BUSY: Device BUSY (exclusive open)
- E_NOEXS: Device does not exist
- E_LIMIT: Open count exceeds the limit
- Other: Error code returned by device driver

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

None.

**Description**

Opens the device specified in `devnm` in the mode specified in `omode`, and prepares for device access. The device descriptor is passed in the return code.

`omode := (TD_READ || TD_WRITE || TD_UPDATE) | [TD_EXCL || TD_WEXCL || TD_REXCL]`

```c
#define TD_READ 0x0001 /* read only */
#define TD_WRITE 0x0002 /* write only */
#define TD_UPDATE 0x0003 /* read/write */
#define TD_EXCL 0x0100 /* exclusive */
#define TD_WEXCL 0x0200 /* exclusive write */
#define TD_REXCL 0x0400 /* exclusive read */
```

- **TD_READ**
  - read only
**TD_WRITE**
Write only

**TD_UPDATE**
Read/write
Sets the access mode.
When **TD_READ** is set, **tk_wri_dev** cannot be used.
When **TD_WRITE** is set, **tk_rea_dev** cannot be used.

**TD_EXCL**
Exclusive

**TD_WEXCL**
Exclusive write

**TD_REXCL**
Exclusive read
Sets the exclusive mode.
When **TD_EXCL** is set, all concurrent opening is prohibited.
When **TD_WEXCL** is set, concurrent opening in write mode (**TD_WRITE** or **TD_UPDATE**) is prohibited.
When **TD_REXCL** is set, concurrent opening in read mode (**TD_READ** or **TD_UPDATE**) is prohibited.

<table>
<thead>
<tr>
<th>Present Open Mode</th>
<th>No exclusive mode</th>
<th>Concurrent Open Mode</th>
<th>TD_WEXCL</th>
<th>TD_REXCL</th>
<th>TD_EXCL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>U</td>
<td>W</td>
<td>R</td>
<td>U</td>
</tr>
<tr>
<td>No exclusive mode</td>
<td>R</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>TD_WEXCL</td>
<td>R</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>TD_REXCL</td>
<td>R</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>TD_EXCL</td>
<td>R</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

Table 5.1: Whether Concurrent Open of Same Device is Allowed or NOT

| R = TD_READ |
| W = TD_WRITE |
| U = TD_UPDATE |
| YES = Yes, can be opened |
| NO = No, cannot be opened (E_BUSY) |

When a physical device is opened, the logical devices belonging to it are all treated as having been opened in the same mode, and are processed as exclusive open.
5.2.2.2 tk_cls_dev - Close Device

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_cls_dev(ID dd, UINT option);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>option</td>
<td>Close Option</td>
<td>Close option</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

E_ID dd is invalid or not open
Other Error code returned by device driver

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Closes device descriptor dd. If a request is being processed, the processing is aborted and the device is closed.

option := [TD_EJECT]

#define TD_EJECT 0x0001 /* Eject media */

TD_EJECT

Eject media

If the same device has not been opened by another task, the media is ejected. In the case of devices that cannot eject their media, the request is ignored.
5.2.2.3  tk_rea_dev - Start Read Device

C Language Interface

#include <tk/tkernel.h>

ID reqid = tk_rea_dev(ID dd, W start, void *buf, SZ size, TMO tmout);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>start</td>
<td>Start Location</td>
<td>Read start location (≧ 0: Device-specific data, &lt; 0: Attribute data)</td>
</tr>
</tbody>
</table>

| void* | buf      | Buffer            | Buffer location for putting the read data |
|SZ    | size     | Read Size         | Read size |
|TMO   | tmout    | Timeout           | Request acceptance timeout (ms) |

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>reqid</th>
<th>Request ID</th>
</tr>
</thead>
</table>

or Error Code

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>dd is invalid or not open</td>
</tr>
<tr>
<td>E_OACV</td>
<td>Open mode is invalid (read not permitted)</td>
</tr>
<tr>
<td>E_LIMIT</td>
<td>Number of requests exceeds the limit</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>Busy processing other requests</td>
</tr>
<tr>
<td>E_ABORT</td>
<td>Processing aborted</td>
</tr>
<tr>
<td>Other</td>
<td>Error code returned by device driver</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Initiates reading device-specific data or attribute data from the specified device. This function initiates reading only, returning to its caller without waiting for the read operation to finish. The space specified in buf must be retained until the read operation completes. Read completion is waited for by tk_wai_dev. The time required for initiating read operation differs among device drivers; return of control is not necessarily immediate.

In the case of device-specific data, the start and size units are defined for each device. With attribute data, start is an attribute data number and size is in bytes. The attribute data of the data number specified in start is read. Normally size must be at least as large as the size of the attribute data to be read. Reading of multiple
attribute data in one operation is not possible. When size = 0 is specified, actual reading does not take place but the current size of data that can be read is checked.

Whether or not a new request can be accepted while a read or write operation is in progress depends on the device driver. If a new request cannot be accepted, the request is queued. The timeout for request waiting is set in tmout. The TMO_POL or TMO_FEVR attribute can be specified in tmout. Note that the timeout applies to the request acceptance. Once a request has been accepted, this function does not time out.

It is permissible to call this API to a driver that has TDA_DEV_D or TDA_TMO_U attribute. In that case, the parameters are converted appropriately by μT-Kernel/SM. For example, if the device driver has TDA_TMO_U attribute, the timeout interval (milliseconds) specified in tmout of this API is converted to time in microseconds, and then passed to the driver with TDA_TMO_U attribute.
5.2.2.4 tk_rea_dev_du - Read Device (64-bit, Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ID reqid = tk_rea_dev_du(ID dd, D start_d, void *buf, SZ size, TMO_U tmout_u);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>start_d</td>
<td>Start Location</td>
<td>Read start location (64 bit, ≥ 0: Device-specific data, &lt; 0: Attribute data)</td>
</tr>
<tr>
<td></td>
<td>buf</td>
<td>Buffer</td>
<td>Buffer location for putting the read data</td>
</tr>
<tr>
<td></td>
<td>sz</td>
<td>Read Size</td>
<td>Read size</td>
</tr>
<tr>
<td></td>
<td>tmout_u</td>
<td>Timeout</td>
<td>Request acceptance timeout (in microseconds)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>reqid</th>
<th>Request ID or Error Code</th>
</tr>
</thead>
</table>

Error Code

- **E_ID**: dd is invalid or not open
- **E_OACV**: Open mode is invalid (read not permitted)
- **E_LIMIT**: Number of requests exceeds the limit
- **E_TMOOUT**: Busy processing other requests
- **E_ABORT**: Processing aborted
- **Other**: Error code returned by device driver

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_LARGEDEV**: Support of large mass-storage devices (64-bit)
- **TK_SUPPORT_USEC**: Support of microsecond

Description

This API takes the parameters `start_d` (64 bits) and `tmout_u` (64-bit microseconds), instead of the parameters `start` and `tmout` of `tk_rea_dev`.

Its specification is the same as that of `tk_rea_dev`, except that the parameters are changed to `start_d` and `tmout_u`. For more details, see the description of `tk_rea_dev`.
Additional Notes

If the corresponding device driver does not have the `TDA_DEV_D` attribute, the error code `E_PAR` is returned when specifying a value that is out of the range of `W` for the start position `start_d`.

If the corresponding device driver does not have the `TDA_TMO_U` attribute (does not supports microseconds), it cannot handle the timeout in microseconds. In that case, the timeout (in microseconds) specified by this API in `tmout_u` is rounded to the time in milliseconds and passed to the device driver.

Thus, the appropriate conversion of parameters is executed by μT-Kernel/SM. The application does not have to know whether the device driver has the `TDA_DEV_D` attribute or not, i.e. whether the device driver supports 64 bits or not.
5.2.2.5  tk_srea_dev - Synchronous Read

C Language Interface

#include <tk/tkernel.h>

ER ercd = tk_srea_dev(ID dd, W start, void *buf, SZ size, SZ *asize);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>start</td>
<td>Start Location</td>
<td>Read start location (≧ 0:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Device-specific data, &lt; 0:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Attribute data)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void*</td>
<td>Buffer location for putting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>buf</td>
<td>the read data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SZ</td>
<td>Read Size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>size</td>
<td>Read size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SZ*</td>
<td>Pointer to the area to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>asize</td>
<td>return the read size</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>asize</td>
<td>Actual Size</td>
<td>Actually read size</td>
</tr>
</tbody>
</table>

Error Code

- E_ID: dd is invalid or not open
- E_OACV: Open mode is invalid (read not permitted)
- E_LIMIT: Number of requests exceeds the limit
- E_ABORT: Processing aborted
- Other: Error code returned by device driver

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Synchronous read. This is equivalent to the following.

```c
ER tk_srea_dev( ID dd, W start, void *buf, SZ size, SZ *asize )
{
    ER    er, ioer;
    er = tk_rea_dev(dd, start, buf, size, TMO_FEVR);
    if ( er > 0 ) {
        er = tk_wai_dev(dd, er, asize, &ioer, TMO_FEVR);
    }
}```
if ( er > 0 ) er = ioer;
}

return er;

This API can be used for a device driver that has the TDA_DEV_D attribute. In that case, the parameters are converted appropriately by μT-Kernel/SM.
5.2.2.6 tk_srea_dev_d - Synchronous Read (64-bit)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_srea_dev_d(ID dd, D start_d, void *buf, SZ size, SZ *asize);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>start_d</td>
<td>Start Location</td>
<td>Read start location (64 bit, ≥ 0: Device-specific data, &lt; 0: Attribute data)</td>
</tr>
<tr>
<td></td>
<td>void</td>
<td>Buffer</td>
<td>Buffer location for putting the read data</td>
</tr>
<tr>
<td></td>
<td>buf</td>
<td></td>
<td>Buffer location for putting the read data</td>
</tr>
<tr>
<td></td>
<td>size</td>
<td>Read Size</td>
<td>Read size</td>
</tr>
<tr>
<td></td>
<td>asize</td>
<td>Actual Size</td>
<td>Pointer to the area to return the read size</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>asize</td>
<td>Actual Size</td>
<td>Actually read size</td>
</tr>
</tbody>
</table>

Error Code

- **E_ID**: dd is invalid or not open
- **E_OACV**: Open mode is invalid (read not permitted)
- **E_LIMIT**: Number of requests exceeds the limit
- **E_ABORT**: Processing aborted
- **Other**: Error code returned by device driver

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_LARGEDEV**: Support of large mass-storage devices (64-bit)

Description

This API takes the 64-bit parameter `start_d`, instead of the parameter `start` of `tk_srea_dev`. Its specification is the same as that of `tk_srea_dev`, except that the parameter is changed to `start_d`. For more details, see the description of `tk_srea_dev`.
Additional Notes

If the corresponding device driver does not have the `TDA_DEV_D` attribute, the error code `E_PAR` is returned when specifying a value that is out of the range of `W` for the start position `start_d`.

Thus, the appropriate conversion of parameters is executed by μT-Kernel/SM. The application does not have to know whether the device driver has the `TDA_DEV_D` attribute or not, i.e. whether the device driver supports 64 bits or not.
5.2.2.7  tk_wri_dev - Start Write Device

C Language Interface

```
#include <tk/tkernel.h>
```

```c
ID reqid = tk_wri_dev(ID dd, W start, CONST void *buf, SZ size, TMO tmout);
```

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor write start location (≧ 0: Device-specific data, &lt; 0: Attribute data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>start</td>
<td>Start Location</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>buf</td>
<td>Buffer</td>
<td>Buffer holding data to be written</td>
</tr>
<tr>
<td></td>
<td>size</td>
<td>Write Size</td>
<td>Size of data to be written</td>
</tr>
<tr>
<td></td>
<td>tmout</td>
<td>Timeout</td>
<td>Request acceptance timeout (ms)</td>
</tr>
</tbody>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>reqid</th>
<th>Request ID</th>
<th>Request ID</th>
<th>or</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Request ID</td>
<td>Request ID</td>
<td>or</td>
<td>Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

**Error Code**

- **E_ID**  
  dd is invalid or not open
- **E_OACV**  
  Open mode is invalid (write not permitted)
- **E_RONLY**  
  Read-only device
- **E_LIMIT**  
  Number of requests exceeds the limit
- **E_TMOUT**  
  Busy processing other requests
- **E_ABORT**  
  Processing aborted
- **Other**  
  Error code returned by device driver

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

None.

**Description**

Initiates writing device-specific data or attribute data to a device. This function initiates writing only, returning to its caller without waiting for the write operation to finish. The space specified in buf must be retained until the write operation completes. Write completion is waited for by tk_wai_dev. The time required for initiating write operation differs among device drivers; return of control is not necessarily immediate.

In the case of device-specific data, the start and size units are defined for each device. With attribute data, start is an attribute data number and size is in bytes. The attribute data of the data number specified in start is written. Normally size must be at least as large as the size of the attribute data to be written. Multiple
attribute data cannot be written in one operation. When size = 0 is specified, actual writing does not take place but the current size of data that can be written is checked.

Whether or not a new request can be accepted while a read or write operation is in progress depends on the device driver. If a new request cannot be accepted, the request is queued. The timeout for request waiting is set in tmout. The TMO_POL or TMO_FEVR attribute can be specified in tmout. Note that the timeout applies to the request acceptance. Once a request has been accepted, this function does not time out.

It is permissible to call this API to a driver that has TDA_DEV_D or TDA_TMO_U attribute. In that case, the parameters are converted appropriately by μT-Kernel/SM. For example, if the device driver has TDA_TMO_U attribute, the timeout interval (milliseconds) specified in tmout of this API is converted to time in microseconds, and then passed to the driver with TDA_TMO_U attribute.
5.2.2.8  tk_wri_dev_du - Write Device (64-bit, Microseconds)

C Language Interface

#include <tk/tkernel.h>

ID reqid = tk_wri_dev_du(ID dd, D start_d, CONST void *buf, SZ size, TMO_U tmout_u);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>start_d</td>
<td>Start Location</td>
<td>Write start location (64 bit, ≥ 0: Device-specific data, &lt; 0: Attribute data)</td>
</tr>
<tr>
<td>CONST void*</td>
<td>buf</td>
<td>Buffer</td>
<td>Buffer holding data to be written</td>
</tr>
<tr>
<td>SZ</td>
<td>size</td>
<td>Write Size</td>
<td>Size of data to be written</td>
</tr>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout</td>
<td>Request acceptance timeout (in microseconds)</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>reqid</th>
<th>Request ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>or Error Code</td>
<td>Request ID</td>
</tr>
<tr>
<td></td>
<td>or Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

- E_ID: dd is invalid or not open
- E_OACV: Open mode is invalid (write not permitted)
- E_RONLY: Read-only device
- E_LIMIT: Number of requests exceeds the limit
- E_TMOUT: Busy processing other requests
- E_ABORT: Processing aborted
- Other: Error code returned by device driver

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- TK_SUPPORT_LARGEDEV: Support of large mass-storage devices (64-bit)
- TK_SUPPORT_USEC: Support of microsecond

Description

This API takes the parameters start_d (64 bits) and tmout_u (64-bit microseconds), instead of the parameters start and tmout of tk_wri_dev.

Its specification is the same as that of tk_wri_dev, except that the parameters are changed to start_d and
tmout_u. For more details, see the description of tk_wri_dev.

Additional Notes

If the corresponding device driver does not have the TDA_DEV_D attribute, the error code E_PAR is returned when specifying a value that is out of the range of W for the start position start_d.

If the corresponding device driver does not have the TDA_TMO_U attribute (does not support microseconds), it cannot handle the timeout in microseconds. In that case, the timeout (in microseconds) specified by this API in tmout_u is rounded to the time in milliseconds and passed to the device driver.

Thus, the appropriate conversion of parameters is executed by μT-Kernel/SM. The application does not have to know whether the device driver has the TDA_DEV_D attribute or not, i.e. whether the device driver supports 64 bits or not.
5.2.2.9  tk_swri_dev - Synchronous Write

C Language Interface

```
#include <tk/tkernel.h>

ER ercd = tk_swri_dev(ID dd, W start, CONST void *buf, SZ size, SZ *asize);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>start</td>
<td>Start Location</td>
<td>Write start location (≧ 0:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Device-specific data, &lt; 0:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Attribute data)</td>
</tr>
<tr>
<td></td>
<td>buf</td>
<td>Buffer</td>
<td>Buffer holding data to be</td>
</tr>
<tr>
<td></td>
<td>size</td>
<td>Write Size</td>
<td>written</td>
</tr>
<tr>
<td></td>
<td>asize</td>
<td>Actual Size</td>
<td>Pointer to the area to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>return the written size</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>asize</td>
<td>Actual Size</td>
<td>Actually written size</td>
</tr>
</tbody>
</table>

Error Code

- **E_ID**  dd is invalid or not open
- **E_OACV** Open mode is invalid (write not permitted)
- **E_RONLY** Read-only device
- **E_LIMIT** Number of requests exceeds the limit
- **E_ABORT** Processing aborted
- **Other** Error code returned by device driver

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task portion</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quasi-task portion</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task-independent</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Synchronous write. This is equivalent to the following.

```
ER tk_swri_dev( ID dd, W start, void *buf, SZ size, SZ *asize )
{
    ER er, ioor;
    er = tk_wri_dev(dd, start, buf, size, TMO_FEVR);
    if ( er > 0 ){
```
er = tk_wai_dev(dd, er, asize, &ioer, TMO_FEVR);
    if ( er > 0 ) er = ioer;

return er;
}

This API can be used for a device driver that has the TDA_DEV_D attribute. In that case, the parameters are converted appropriately by μT-Kernel/SM.
5.2.2.10  tk_swri_dev_d - Synchronous Write (64-bit)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_swri_dev_d(ID dd, D start_d, CONST void *buf, SZ size, SZ *asize);
```

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>start_d</td>
<td>Start Location</td>
<td>Write start location (64 bit, ≧ 0: Device-specific data, &lt; 0: Attribute data)</td>
</tr>
<tr>
<td>CONST void*</td>
<td>buf</td>
<td>Buffer</td>
<td>Buffer holding data to be written</td>
</tr>
<tr>
<td>SZ</td>
<td>size</td>
<td>Write Size</td>
<td>Size of data to be written</td>
</tr>
<tr>
<td>SZ*</td>
<td>asize</td>
<td>Actual Size</td>
<td>Pointer to the area to return the written size</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>asize</td>
<td>Actual Size</td>
<td>Actually written size</td>
</tr>
</tbody>
</table>

Error Code

- **E_ID**: dd is invalid or not open
- **E_OACV**: Open mode is invalid (write not permitted)
- **E_RONLY**: Read-only device
- **E_LIMIT**: Number of requests exceeds the limit
- **E_ABORT**: Processing aborted
- **Other**: Error code returned by device driver

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_LARGEDEV**: Support of large mass-storage devices (64-bit)

Description

This API takes the 64-bit parameter `start_d`, instead of the parameter `start` of `tk_swri_dev`.

Its specification is the same as that of `tk_swri_dev`, except that the parameter is changed to `start_d`. For more details, see the description of `tk_swri_dev`.
Additional Notes

If the corresponding device driver does not have the `TDA_DEV_D` attribute, the error code `E_PAR` is returned when specifying a value that is out of the range of `W` for the start position `start_d`.

Thus, the appropriate conversion of parameters is executed by μT-Kernel/SM. The application does not have to know whether the device driver has the `TDA_DEV_D` attribute or not, i.e. whether the device driver supports 64 bits or not.
5.2.2.11 tk_wai_dev - Wait for Request Completion for Device

C Language Interface

```c
#include <tk/tkernel.h>

ID creqid = tk_wai_dev(ID dd, ID reqid, SZ *asize, ER *ioer, TMO tmout);
```

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>reqid</td>
<td>Request ID</td>
<td>Request ID</td>
</tr>
<tr>
<td>SZ*</td>
<td>asize</td>
<td>Actually Read/Written Size</td>
<td>Pointer to the area to return the read/written size</td>
</tr>
<tr>
<td>ER*</td>
<td>ioer</td>
<td>I/O Error</td>
<td>Pointer to the area to return I/O error</td>
</tr>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
<td>Timeout (ms)</td>
</tr>
</tbody>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>creqid</th>
<th>Completed Request ID</th>
<th>Completed request ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>asize</td>
<td>Actually Read/Written Size</td>
<td>Actually read/written size</td>
</tr>
<tr>
<td>ER</td>
<td>ioer</td>
<td>I/O Error</td>
<td>I/O error</td>
</tr>
</tbody>
</table>

**Error Code**

- **E_ID**: `dd` is invalid or not opened, or `reqid` is invalid or not a request for `dd`
- **E_OBJ**: Another task is already waiting for request `reqid`
- **E_NOEXS**: No requests are being processed (only when `reqid` = 0)
- **E_TMOUT**: Timeout (processing continues)
- **E_ABORT**: Processing aborted
- **Other**: Error code returned by device driver

**Valid Context**

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

None.

**Description**

Waits for completion of request `reqid` for device `dd`. If `reqid` = 0 is set, this function waits for completion of any pending request to `dd`. This function waits for completion only of requests currently processing when the function is called. A request issued after `tk_wai_dev` was called is not waited for.

When multiple requests are being processed concurrently, the order of their completion is not necessarily the same as the order of request but is dependent on the device driver. Processing is, however, guaranteed to be performed in a sequence such that the result is consistent with the order of requesting. When processing a read operation from a disk, for example, the sequence might be changed as follows.
Block number request sequence
1 4 3 2 5

Block number processing sequence
1 2 3 4 5

Disk access can be made more efficient by changing the sequence as above with the aim of reducing seek time and spin wait time.

The timeout for waiting for completion is set in \texttt{tmout}. The \texttt{TMO\_POL} or \texttt{TMO\_FEVR} attribute can be specified for \texttt{tmout}. If a timeout error is returned (E\_TMOUT), \texttt{tk\_wai\_dev} must be called again to wait for completion since the request processing is still ongoing. When \texttt{reqid} > 0 and \texttt{tmout} = \texttt{TMO\_FEVR} are both set, the processing must be completed without timing out.

If the device driver returns a processing result error (such as I/O error) for the requested processing, the error code is stored in \texttt{ioer} instead of the return code. Specifically, the error code, which is stored in \texttt{error} of the request packet T\_DEVREQ by the wait-for-completion function (\texttt{waitfn}) called for processing \texttt{tk\_wai\_dev}, is returned to \texttt{ioer} as the processing result error.

On the other hand, the return code is used for errors when the wait request itself was not handled properly. When error is passed in the return code, \texttt{ioer} has no meaning. Note also that if an error is passed in the return code, \texttt{tk\_wai\_dev} must be called again to wait for completion since the processing is still ongoing. For more details, see Section 5.2.3.2.4, “\texttt{waitfn} - Wait-for-completion function”.

If a task exception is raised during completion waiting by \texttt{tk\_wai\_dev}, the request in \texttt{reqid} is aborted and processing is completed. The result of aborting the requested processing is dependent on the device driver. When \texttt{reqid} = 0 was set, however, requests are not aborted but are treated as timeout. In this case E\_ABORT rather than E\_TMOUT is returned.

It is not possible for multiple tasks to wait for completion of the same request ID at the same time. If there is a task waiting for request completion with \texttt{reqid} = 0 set, another task cannot wait for completion for the same \texttt{dd}. Similarly, if there is a task waiting for request completion with \texttt{reqid} > 0 set, another task cannot wait for completion specifying \texttt{reqid} = 0.

It is permissible to call this API to a driver with \texttt{TDA\_TMO\_U} attribute. In such instances, \texttt{\mu T-Kernel/SM} converts the parameter(s) appropriately. For example, if the device driver has \texttt{TDA\_TMO\_U} attribute, the timeout in milliseconds specified in \texttt{tmout} of this API is converted to timeout value in microseconds, and is passed to the driver with \texttt{TDA\_TMO\_U}.
5.2.2.12 tk_wai_dev_u - Wait Device (Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ID creqid = tk_wai_dev_u(ID dd, ID reqid, SZ *asize, ER *ioer, TMO_U tmout_u);
```

**Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>dd</td>
<td>Device Descriptor</td>
</tr>
<tr>
<td>ID</td>
<td>reqid</td>
<td>Request ID</td>
</tr>
<tr>
<td>SZ*</td>
<td>asize</td>
<td>Actually Read/Written Size</td>
</tr>
<tr>
<td>ER*</td>
<td>ioer</td>
<td>I/O Error</td>
</tr>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout (in microseconds)</td>
</tr>
</tbody>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>ID</th>
<th>creqid</th>
<th>Completed Request ID or Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ</td>
<td>asize</td>
<td>Actually Read/Written Size</td>
</tr>
<tr>
<td>ER</td>
<td>ioer</td>
<td>I/O Error</td>
</tr>
</tbody>
</table>

**Error Code**

- **E_ID**  
  dd is invalid or not opened, or reqid is invalid or not a request for dd
- **E_OBJ**  
  Another task is already waiting for request reqid
- **E_NOEXS**  
  No requests are being processed (only when reqid = 0)
- **E_TMOUT**  
  Timeout (processing continues)
- **E_ABORT**  
  Processing aborted
- **Other**  
  Error code returned by device driver

**Valid Context**

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_USEC**  
  Support of microsecond

**Description**

This API takes the parameter tmout_u (64-bit microseconds), instead of the parameter tmout of tk_wai_dev.

Its specification is the same as that of tk_wai_dev, except that the parameter changed to tmout_u. For more details, see the description of tk_wai_dev.
Additional Notes

If the corresponding device driver does not have the \texttt{TDA\_TMO\_U} attribute (does not supports microseconds), it cannot handle the timeout in microseconds. In that case, the timeout (in microseconds) specified by this API in \texttt{tmout\_u} is rounded to the time in milliseconds and passed to the device driver.

Thus, the appropriate conversion of parameters is executed by $\mu$ T-Kernel/SM. The application does not have to know whether the device driver has the \texttt{TDA\_TMO\_U} attribute or not, i.e., whether the device driver supports microseconds or not.
5.2.2.13  tk_sus_dev - Suspends Device

C Language Interface

#include <tk/tkernel.h>

INT dissus = tk_sus_dev(UINT mode);

Parameter

<table>
<thead>
<tr>
<th>UINT</th>
<th>mode</th>
<th>Mode</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>dissus</th>
<th>Suspend Disable Request Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>or Error Code Error code</td>
</tr>
</tbody>
</table>

Error Code

- E_BUSY: Suspend already disabled
- E_QOVR: Suspend disable request count limit exceeded

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- TK_SUPPORT_LOWPOWER: Support of power management functions

Description

Performs the processing specified in `mode`, then passes the resulting suspend disable request count in the return code.

\[
\text{mode} := (\text{TD_SUSPEND} | \text{TD_FORCE}) \| \text{TD_DISSUS} \| \text{TD_ENASUS} \| \text{TD_CHECK})
\]

- #define TD_SUSPEND 0x0001 /* suspend */
- #define TD_DISSUS 0x0002 /* disable suspension */
- #define TD_ENASUS 0x0003 /* enable suspension */
- #define TD_CHECK 0x0004 /* get suspend disable request count */
- #define TD_FORCE 0x8000 /* forced suspend specification */

**TD_SUSPEND**

Suspend

If suspending is enabled, suspends processing.

If suspending is disabled, returns E_BUSY.
**TD_SUSPEND**|**TD_FORCE**
- Forcibly suspend
  - Suspends even in suspend disabled state.

**TD_DISSUS**
- Disable suspension
  - Disables suspension.

**TD_ENASUS**
- Enable suspension
  - Enables suspension.

**TD_CHECK**
- Get suspend disable count
  - Gets only the number of times suspend disable has been requested.

Suspension is performed in the following steps.

1. Processing prior to start of suspension in each subsystem
   `tk_evt_ssy(0, TSEVT_SUSPEND_BEGIN, 0)`
2. Suspension processing in devices
3. Processing after completion of suspension in each subsystem
   `tk_evt_ssy(0, TSEVT_SUSPEND_DONE, 0)`
4. Suspended state
   `tk_set_pow(TPW_DOSUSPEND)`

Resumption from SUSPEND state is performed in the following steps.

1. Return from SUSPEND state
   - Return from `tk_set_pow(TPW_DOSUSPEND)`
2. Processing prior to start of resumption in each subsystem
   `tk_evt_ssy(0, TSEVT_RESUME_BEGIN, 0)`
3. Resumption processing in devices
4. Processing after completion of resumption in each subsystem
   `tk_evt_ssy(0, TSEVT_RESUME_DONE, 0)`

The number of suspend disable requests is counted. Suspension is enabled only if the same number of suspend enable requests is made. At system boot, the suspend disable count is 0 and suspension is enabled. The maximum suspend disable request count is implementation-dependent, but must be at least 255. When the upper limit is exceeded, E_QOVR is returned.
5.2.2.14  tk_get_dev - Get Device Name

C Language Interface

#include <tk/tkernel.h>

ID pdevid = tk_get_dev(ID devid, UB *devnm);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>devid</th>
<th>Device ID</th>
<th>Device ID of Physical Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB*</td>
<td>devnm</td>
<td>Device Name</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>pdevid</th>
<th>Device ID of Physical Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB</td>
<td>devnm</td>
<td>Device Name</td>
</tr>
</tbody>
</table>

Error Code

E_NOEXS  The device specified in devid does not exist

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Gets the device name of the device specified in devid and puts the result in devnm.

devid is the device ID of either a physical device or a logical device.

If devid is a physical device, the physical device name is put in devnm.

If devid is a logical device, the logical device name is put in devnm.

devnm requires a space of L_DEVNM + 1 bytes or larger.

The device ID of the physical device to which device devid belongs is passed in the return code.
5.2.2.15 tk_ref_dev - Get Device Information

C Language Interface

```c
#include <tk/tkernel.h>

ID devid = tk_ref_dev(CONST UB *devnm, T_RDEV *rdev);
```

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST UB*</td>
<td>devnm</td>
<td>Device Name</td>
</tr>
<tr>
<td>T_RDEV*</td>
<td>rdev</td>
<td>Packet to Return Device Information</td>
</tr>
</tbody>
</table>

Parameter Descriptions:

- **devnm**: Device Name
- **rdev**: Pointer to the area to return the device information

Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>devid</td>
<td>Device ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or Error Code</td>
</tr>
</tbody>
</table>

Parameter Descriptions:

- **devid**: Device ID
- **Error Code**: Error code

rdev Detail:

- **devatr**: Device Attribute
- **blksz**: Block Size of Device-specific Data
- **nsub**: Subunit Count
- **subno**: Subunit Number

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

- **E_NOEXS**: The device specified in `devnm` does not exist

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

- Gets device information about the device specified in `devnm`, and puts the result in `rdev`. If `rdev = NULL` is set, the device information is not stored.
- `nsub` indicates the number of physical device subunits belonging to the device specified in `devnm`.
- The device ID of the device specified in `devnm` is passed in the return code.
5.2.2.16  tk_oref_dev - Get Device Information

C Language Interface

#include <tk/tkernel.h>

ID devid = tk_oref_dev(ID dd, T_RDEV *rdev);

Parameter

<table>
<thead>
<tr>
<th>ID dd</th>
<th>Device Descriptor</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RDEV* rdev</td>
<td>Packet to Return Device Information</td>
<td>Pointer to the area to return the device information</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ID devid</th>
<th>Device ID</th>
<th>Device ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>or Error Code</td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

rdev Detail:

<table>
<thead>
<tr>
<th>ATR devatr</th>
<th>Device Attribute</th>
<th>Device attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZ blksz</td>
<td>Block Size of Device-specific Data</td>
<td>Block size of device-specific data (-1: unknown)</td>
</tr>
<tr>
<td>INT nsub</td>
<td>Subunit Count</td>
<td>Number of subunits</td>
</tr>
<tr>
<td>INT subno</td>
<td>Subunit Number</td>
<td>0: Physical device, 1 to nsub: Subunit number+1</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

E_ID dd is invalid or not open

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Gets device information about the device specified in dd, and puts the result in rdev. If rdev = NULL is set, the device information is not stored.

nsub indicates the number of physical device subunits belonging to the device specified in dd.

The device ID of the device specified in dd is passed in the return code.
5.2.2.17 tk_lst_dev - Get Registered Device Information

C Language Interface

```c
#include <tk/tkernel.h>

INT remcnt = tk_lst_dev(T_LDEV *ldev, INT start, INT ndev);
```

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_LDEV*</td>
<td>ldev</td>
<td>List of Devices Location of registered device information (array)</td>
</tr>
<tr>
<td>INT</td>
<td>start</td>
<td>Starting Number Starting number</td>
</tr>
<tr>
<td>INT</td>
<td>ndev</td>
<td>Number of Devices Number to acquire</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>remcnt</td>
<td>Remaining Device Count Number of remaining registrations</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td>Error Code Error code</td>
</tr>
</tbody>
</table>

ldev Detail:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATR</td>
<td>devatr</td>
<td>Device Attribute Device attributes</td>
</tr>
<tr>
<td>SZ</td>
<td>blksz</td>
<td>Block Size of Device-specific Data Block size of device-specific data (-1: unknown)</td>
</tr>
<tr>
<td>INT</td>
<td>nsub</td>
<td>Subunit Count Number of subunits</td>
</tr>
<tr>
<td>UB</td>
<td>devnm[L_DEVNM]</td>
<td>Physical Device Name Physical device name</td>
</tr>
</tbody>
</table>

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

E_NOEXS start exceeds the registered number

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Gets information about registered devices. Registered devices are managed per physical device. The registered device information is therefore also obtained per physical device.

When the number of registered devices is N, number are assigned serially to devices from 0 to N - 1. Starting from the number specified in start in accordance with this scheme, the number of registrations specified in ndev is acquired and put in ldev. The space specified in ldev must be large enough to hold ndev registration information. The number of remaining registrations after start (N-start) is passed in the return code.
If the number of registrations from start is fewer than ndev, all remaining registrations are stored. A value passed in return code less than or equal to ndev means all remaining registrations were obtained. Note that this numbering changes as devices are registered and deleted. For this reason, accurate information may not be always obtained if the acquisition is carried out over multiple operations.
5.2.2.18  tk_evt_dev - Send Driver Request Event to Device

C Language Interface

#include <tk/tkernel.h>

INT retcode = tk_evt_dev(ID devid, INT evttyp, void *evtinf);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>devid</th>
<th>Device ID</th>
<th>Event destination device ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>evttyp</td>
<td>Event Type</td>
<td>Driver request event type</td>
</tr>
<tr>
<td>void*</td>
<td>evtinf</td>
<td>Event Information</td>
<td>Information for each event type</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>retcode</th>
<th>Return Code from eventfn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Return code passed by eventfn or Error Code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOEXS</td>
<td>The device specified in devid does not exist</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Internal device manager events (evttyp &lt; 0) cannot be specified</td>
</tr>
<tr>
<td>Other</td>
<td>Error code returned by device driver</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Sends a driver request event to the device (device driver) specified in devid.

The functioning of driver request events and the contents of evtinf are defined for each event type. For details on driver request event, see Section 5.2.3.2.6, “eventfn - Event function”.
5.2.3 Registration of Device Driver

5.2.3.1 Registration Method of Device Driver

Device driver registration is performed for each physical device.
5.2.3.1.1 tk_def_dev - Register Device

C Language Interface

#include <tk/tkernel.h>

ID devid = tk_def_dev(CONST UB *devnm, CONST T_DDEV *ddev, T_IDEV *idev);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devnm</td>
<td>Physical Device Name</td>
</tr>
<tr>
<td>ddev</td>
<td>Define Device</td>
</tr>
<tr>
<td>idev</td>
<td>Initial Device Information</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devid</td>
<td>Device ID or Error Code</td>
</tr>
</tbody>
</table>

idev Detail:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>evtmbfid</td>
<td>Event Notification Message Buffer ID</td>
</tr>
</tbody>
</table>

Error Code

- **E_LIMIT**: Number of registrations exceeds the system limit
- **E_NOEXS**: The device specified in devnm does not exist (when ddev = NULL)

Valid Context

<table>
<thead>
<tr>
<th>Section</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task portion</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Registers a device (device driver) with the device name set in devnm, and passes the device ID of the registered device in the return code. If a device with device name devnm is already registered, the registration is updated with new information, in which case the device ID does not change.

ddev specifies the device registration information. When ddev = NULL is specified, device devnm registration is deleted.

ddev is a structure in the following format:

```c
typedef struct t_ddev {
    void *exinf; /* extended information */
};
```
exinfo is used to store any desired information. The value is passed to the processing functions. Device management pays no attention to the contents.

drvatr sets device driver attribute information. The lower bits indicate system attributes, and the high bits are used for implementation-dependent attributes. The implementation-dependent attribute portion is used, for example, to define validity flags when implementation-dependent data is added to T_DDEV.

drvatr := [TDA_OPENREQ] | [TDA_TMO_U] | [TDA_DEV_D]

#define TDA_OPENREQ 0x0001 /* open/close each time */
#define TDA_TMO_U 0x0002 /* timeout in microseconds is used */
#define TDA_DEV_D 0x0004 /* 64 bit device */

drvatr can be specified by combining the following driver attributes.

TDA_OPENREQ
When a device is opened multiple times, normally openfn is called only the first time it is opened and closefn the last time it is closed. If TDA_OPENREQ is specified, then openfn/closefn will be called for all open/close operations even in case of multiple openings.

TDA_TMO_U
Indicates that timeout in microseconds is used.
In this case, the timeout tmout of driver processing functions is specified in the TMO_U format (microseconds).

TDA_DEV_D
Indicates that a 64-bit device is used. In this case, the type of the request packet devreq of driver processing functions is T_DEVREQ_D.
If TDA_TMO_U or TDA_DEV_D is specified, type of some parameters of driver processing functions is changed. If a combination of multiple driver attributes that change the type of parameters is specified in a driver processing function, the type of all specified parameters of that function is changed.

Device attributes are specified in devatr. The details of device attribute setting are as noted above.

The number of subunits is set in nsub. If there are no subunits, 0 is specified.

blksz sets the block size of device-specific data in bytes. In the case of a disk device, this is the physical block size. It is set to 1 byte for a serial port, etc. For a device with no device-specific data, it is set to 0. For an unformatted disk or other device whose block size is unknown, -1 is set. If blksz ≤ 0, device-specific data cannot be accessed. When device-specific data is accessed by tk_rea_dev or tk_wri_dev, size * blksz must be the size of the area being accessed, that is, the size of buf.

openfn, closefn, execfn, waitfn, abortfn, and eventfn set the entry address of driver processing functions. For more details on driver processing functions, see Section 5.2.3.2, "Device Driver Interface".

The device initialization information is returned in idev. This includes information set by default when the device driver is started, and can be used as necessary. When idev = NULL is set, device initialization information is not stored.
evtmbfid specifies the system default message buffer ID for event notification. If there is no system default event notification message buffer, 0 is set.

Notification like the following is made to each subsystem when a device is registered or deleted. devid is the device ID of the registered or deleted physical device.

Device registration or update:

```
tk_evt_ssy(0, TSEVTDEVICE_REGIST, devid)
```

Device deletion:

```
tk_evt_ssy(0, TSEVTDEVICE_DELETE, devid)
```
5.2.3.1.2 tk_ref_idv - Reference Device Initialization Information

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = tk_ref_idv(T_IDEV *idev);
```

Parameter

| T_IDEV* idev | Packet to Return Initial Device Information | Pointer to the area to return the device initialization information |

Return Parameter

| ER ercd | Error Code | Error code |

idev Detail:

| ID evtmbfid | Event Notification Message | Event notification message buffer ID Buffer ID |

(Other implementation-dependent parameters may be added beyond this point.)

Error Code

| E_MACV | Memory access privilege error |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Gets device initialization information. The contents are the same as the information obtained by `tk_def_dev`.

Additional Notes

The error code E_MACV is common to many system calls, and usually not included in the error code list of each system call. However, for this API, E_MACV is included in this error code list because it is the only typical error.
5.2.3.2 Device Driver Interface

The device driver interface consists of processing functions (driver processing functions) specified when registering a device.

Open function
ER `openfn`(ID devid, UINT omode, void *exinf);

Close function
ER `closefn`(ID devid, UINT option, void *exinf);

Execute function
ER `execfn`(T_DEVREQ *devreq, TMO tmout, void *exinf);

Wait-for-completion function
INT `waitfn`(T_DEVREQ *devreq, INT nreq, TMO tmout, void *exinf);

Abort function
ER `abortfn`(ID tskid, T_DEVREQ *devreq, INT nreq, void *exinf);

Event function
INT `eventfn`(INT evttyp, void *evtinf, void *exinf);

If TDA_TMO_U is specified for a driver attribute, the timeout specification tmout for the following driver processing functions is set to TMO_U type (in microseconds).

Execute function
ER `execfn`(T_DEVREQ *devreq, TMO_U tmout_u, void *exinf);

Wait-for-completion function
INT `waitfn`(T_DEVREQ *devreq, INT nreq, TMO_U tmout_u, void *exinf);

If TDA_DEV_D is specified for a driver attribute, the type of request packet devreq for the following driver processing functions is set to T_DEVREQ_D.

Execute function
ER `execfn`(T_DEVREQ_D *devreq_d, TMO tmout, void *exinf);

Wait-for-completion function
INT `waitfn`(T_DEVREQ_D *devreq_d, INT nreq, TMO tmout, void *exinf);

Abort function
ER `abortfn`(ID tskid, T_DEVREQ_D *devreq_d, INT nreq, void *exinf);

If TDA_TMO_U and TDA_DEV_D are specified set a driver attribute, a driver processing function is set to the one that has parameters with all the specified types of changes were applied.

Execute function
ER `execfn`(T_DEVREQ_D *devreq_d, TMO_U tmout_u, void *exinf);

Wait-for-completion function
INT `waitfn`(T_DEVREQ_D *devreq_d, INT nreq, TMO_U tmout_u, void *exinf);

Driver processing functions are called by device management and run as a quasi-task portion. These driver processing functions must be reentrant. Calling of these driver processing functions in a mutually exclusive manner is not guaranteed. If, for example, there are simultaneous requests from multiple devices for the same device, different tasks might call the same driver processing function at the same time. The device driver must perform mutual exclusion control in such cases as necessary.

I/O requests to a device driver are made by means of the following request packet associated with a request ID.
/*
 * Device request packet: For 32-bit
 * In: Input parameter to driver processing function (set in \uT-Kernel/SM device management)
 * Out: Output parameter from driver processing function (set in driver processing function)
 * X: Parameters other than input and output
 */
typedef struct t_devreq {
    struct t_devreq *next; /* In: Link to request packet (NULL: termination) */
    void *exinf; /* X: Extended information */
    ID devid; /* In: Target device ID */
    INT cmd:4; /* In: Request command */
    BOOL abort:1; /* In: TRUE if abort request */
    W start; /* In: Starting data number */
    SZ size; /* In: Request size */
    void *buf; /* In: IO buffer address */
    SZ asize; /* Out: Size of result */
    ER error; /* Out: Error result */
} T_DEVREQ;

/*
 * Device request packet: For 64-bit
 * In: Input parameter to driver processing function (set in \uT-Kernel/SM device management)
 * Out: Output parameter from driver processing function (set in driver processing function)
 * X: Parameters other than input and output
 */
typedef struct t_devreq_d {
    struct t_devreq_d *next; /* In: Link to request packet (NULL: termination) */
    void *exinf; /* X: Extended information */
    ID devid; /* In: Target device ID */
    INT cmd:4; /* In: Request command */
    BOOL abort:1; /* In: TRUE if abort request */
    D start_d; /* In: Starting data number, 64-bit */
    SZ size; /* In: Request size */
    void *buf; /* In: IO buffer address */
    SZ asize; /* Out: Size of result */
    ER error; /* Out: Error result */
} T_DEVREQ_D;

In: Input parameter to the driver processing function is set in \uT-Kernel/SM device management. Should not be changed on the device driver side. Parameters other than input parameters (In) are initially cleared to 0 by the device management. After that, device management does not modify them. Out: Output parameter returned from the driver execute function is set in the driver processing function.

next is used to link the request packet. In addition to usage for keeping track of request packets in device management, it is used also by the completion wait function (waitfn) and abort function (abortfn).

exinf can be used freely by the device driver. Device management does not pay attention to the contents.

The device ID of the device to which the request is issued is specified in devid.

The request command is specified in cmd as follows.

cmd := (TDC_READ || TDC_WRITE)
If abort processing is to be carried out, `abort` is set to `TRUE` right before calling the abort function (`abortfn`). `abort` is a flag indicating whether abort processing was requested, and does not indicate that processing was aborted. In some cases `abort` is set to `TRUE` even when the abort function (`abortfn`) is not called. Abort processing is performed when a request with `abort` set to `TRUE` is actually passed to the device driver.

`start`, `start_d`, and `size` are just set as `start`, `start_d`, and `size` specified in `tk_rea_dev`, `tk_rea_dev_du`, `tk_wri_dev`, and `tk_wri_dev_du`.

`buf` is just set as `buf` specified in `tk_rea_dev`, `tk_rea_dev_du`, `tk_wri_dev`, and `tk_wri_dev_du`. On systems that support virtual memory, the memory space specified in `buf` may be nonresident or belong to task space, so care must be taken to handle such cases.

The device driver sets in `asize` the value returned in `asize` by `tk_wai_dev`.

The device driver sets in `error` the error code passed by `tk_wai_dev` in its return code. `E_OK` indicates a normal result.

Difference between `T_DEVREQ` and `T_DEVREQ_D` is only the part of their names being `start` or `start_d`, and the data type.

The type of device request packet (`T_DEVREQ` or `T_DEVREQ_D`) is selected based on the driver attribute (`TDA_DEV_D`) at device registration. For this reason, `T_DEVREQ` and `T_DEVRE` do not co-exist in the request packet for one driver.
5.2.3.2.1 openfn - Open function

C Language Interface

ER ercd = openfn(ID devid, UINT omode, void *exinf);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>devid</th>
<th>Device ID</th>
<th>Device ID of the device to open</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>omode</td>
<td>Open Mode</td>
<td>Open mode (same as tk_opn_dev)</td>
</tr>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
<td>Extended information set at device registration</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

Error Code

<table>
<thead>
<tr>
<th>Other</th>
<th>Error code returned by the device driver</th>
</tr>
</thead>
</table>

Description

The open function openfn is called when tk_opn_dev is invoked.

The function openfn performs processing to enable use of a device. Details of the processing are device-dependent; if no processing is needed, it does nothing. The device driver does not need to remember whether a device is open or not, nor is it necessary to treat as an error the calling of another processing function simply because the device was not opened (openfn had not been called). If another processing function is called for a device that is not open, the necessary processing can be performed so long as there is no problem in device driver operation.

When openfn is used to perform device initialization or the like, in principle no processing should be performed that causes a wait. The processing and return from openfn must be as prompt as possible. In the case of a device such as a serial port for which it is necessary to set the communication mode, for example, the device can be initialized when the communication mode is set by tk_wri_dev. There is no need for openfn to initialize the device.

When the same device is opened multiple times, normally this function is called only for the first time. If, however, the driver attribute TDA_OPENREQ is specified in device registration, this function is called each time the device is opened.

The openfn function does not need to perform any processing with regard to multiple opening or open mode, which are handled by device management. Likewise, omode is simply passed as reference information; no processing relating to omode is required.

openfn runs as a quasi-task portion of the task that issued tk_opn_dev. That is, it is executed in the context of the quasi-task portion whose requesting task is the task that issued tk_opn_dev.
5.2.3.2.2 closefn - Close function

C Language Interface

ER ercd = closefn(ID devid, UINT option, void *exinf);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>devid</th>
<th>Device ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>option</td>
<td>Close Option</td>
</tr>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Device ID of the device to close</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close option (same as tk_cls_dev)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extended information set at device registration</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

Other Error code returned by the device driver

Description

The close function closefn is called when tk_cls_dev is invoked.

The closefn function performs processing to end use of a device. Details of the processing are device-dependent; if no processing is needed, it does nothing.

If the device is capable of ejecting media and TD_EJECT is set in option, media ejection is performed.

When closefn is used to perform device shutdown processing or media ejection, in principle no processing should be performed that causes a wait. The processing and return from closefn must be as prompt as possible. If media ejection takes time, it is permissible to return from closefn without waiting for the ejection to complete.

When the same device is opened multiple times, normally this function is called only the last time it is closed. If, however, the driver attribute TDA_OPENREQ is specified in device registration, this function is called each time the device is closed. In this case TD_EJECT is specified in option only for the last time.

The closefn function does not need to perform any processing with regard to multiple opening or open mode, which are handled by device management.

closefn runs as a quasi-task portion of the task that issued tk_cls_dev.
5.2.3.2.3 execfn - Execute function

C Language Interface

/* Execute function (32-bit request packet, millisecond timeout) */

ER ercd = execfn(T_DEVREQ *devreq, TMO tmout, void *exinf);

/* execute function (64-bit request packet, millisecond timeout) */

ER ercd = execfn(T_DEVREQ_D *devreq_d, TMO tmout, void *exinf);

/* execute function (32-bit request packet, microsecond timeout) */

ER ercd = execfn(T_DEVREQ *devreq, TMO_U tmout_u, void *exinf);

/* execute function (64-bit request packet, microsecond timeout) */

ER ercd = execfn(T_DEVREQ_D *devreq_d, TMO_U tmout_u, void *exinf);

Parameter

<table>
<thead>
<tr>
<th>T_DEVREQ*</th>
<th>devreq</th>
<th>Device Request Packet</th>
<th>Request packet (32-bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_DEVREQ_D*</td>
<td>devreq_d</td>
<td>Device Request Packet</td>
<td>Request packet (64-bit)</td>
</tr>
<tr>
<td>TMO</td>
<td>tmout</td>
<td>Timeout</td>
<td>Request acceptance timeout (ms)</td>
</tr>
<tr>
<td>TMO_U</td>
<td>tmout_u</td>
<td>Timeout</td>
<td>Request acceptance timeout (in microseconds)</td>
</tr>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
<td>Extended information set at device registration</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Error Code

Other          | Error code returned by the device driver |

Description

The execute function `execfn` is called when `tk_rea_dev` or `tk_wri_dev` is invoked. Initiates the processing requested in `devreq`. This function initiates the requested processing only, returning to its caller without waiting for the processing to complete. The time required to initiate processing depends on the device driver; this function does not necessarily complete immediately.

When new processing cannot be accepted, this function goes to WAITING state for request acceptance. If the new request cannot be accepted within the time specified in `tmout`, the function times out. The `TMO_POL` or `TMO_FEVR` attribute can be specified in `tmout`. If the function times out, `E_TMOUT` is passed in the `execfn` return code. The request packet error parameter does not change. Timeout applies to the request acceptance, not to the processing after acceptance.

When error is passed in the `execfn` return code, the request is considered not to have been accepted and the request packet is discarded.
If processing is aborted before the request is accepted (before the requested processing starts), E_ABORT is passed in the execfn return code. In this case, the request packet is discarded. If the abort occurs after the processing has been accepted, E_OK is returned for this function. The request packet is not discarded until waitfn is executed and processing completes.

When abort occurs, the important thing is to return from execfn as quickly as possible. If processing will end soon anyway without aborting, it is not necessary to abort.

execfn runs as a quasi-task portion of the task that issued tk_rea_dev, tk_wri_dev, tk_srea_dev, or tk_swri_dev.

In a device driver for which TDA_DEV_D is specified as an attribute at the time of registering the device, the execute function (64-bit request packet, millisecond timeout) execfn is called when tk_rea_dev or tk_wri_dev is invoked. In this case, the function specification is the same as that of 32-bit request packet, millisecond timeout execfn, except that the parameter request packet is a 64-bit T_DEVREQ_D* devreq_d.

In a device driver for which TDA_TMO_U is specified as an attribute at the time of registering the device, the execute function (32-bit request packet, microsecond timeout) execfn is called when tk_rea_dev or tk_wri_dev is invoked. In this case, the function specification is the same as that of 32-bit request packet, millisecond timeout execfn, except that the parameter timeout specification is a microsecond T_MO_U tmout_u.

In a device driver for which both TDA_DEV_D and TDA_TMO_U are specified as an attribute at the time of registering the device, the execute function (64-bit request packet, microsecond timeout) execfn is called when tk_rea_dev or tk_wri_dev is invoked. In this case, the function specification is the same as that of 32-bit request packet, millisecond timeout execfn, except that the parameter request packet is a 64-bit T_DEVREQ_D* devreq_d and the parameter timeout specification is a microsecond T_MO_U tmout_u.
5.2.3.2.4 waitfn - Wait-for-completion function

C Language Interface

*/ wait-for-completion function (32-bit request packet, millisecond timeout) */

INT creqno = waitfn(T_DEVREQ *devreq, INT nreq, TMO tmout, void *exinf);

*/ wait-for-completion function (64-bit request packet, millisecond timeout) */

INT creqno = waitfn(T_DEVREQ_D *devreq_d, INT nreq, TMO tmout, void *exinf);

*/ wait-for-completion function (32-bit request packet, microsecond timeout) */

INT creqno = waitfn(T_DEVREQ *devreq, INT nreq, TMO_U tmout_u, void *exinf);

*/ wait-for-completion function (64-bit request packet, microsecond timeout) */

INT creqno = waitfn(T_DEVREQ_D *devreq_d, INT nreq, TMO_U tmout_u, void *exinf);

Parameter

| T_DEVREQ* | devreq | Device Request Packet | Request packet list (32-bit) |
| T_DEVREQ_D* | devreq_d | Device Request Packet | Request packet list (64-bit) |
| INT | nreq | Number of Requests | Request packet count |
| TMO | tmout | Timeout | Timeout (ms) |
| TMO_U | tmout_u | Timeout | Timeout (in microseconds) |
| void* | exinf | Extended Information | Extended information set at device registration |

Return Parameter

| INT | creqno | Completed Request Packet Number | Completed request packet number |
| or | Error Code | Error code |

Error Code

Other

Error code returned by the device driver

Description

The wait-for-completion function waitfn is called when tk_wai_dev is invoked.

devreq is a list of request packets in a chain linked by devreq->next. This function waits for completion of any of the nreq request packets starting from devreq. The final next is not necessarily NULL, so the nreq must always be followed. The number of the completed request packet (which one after devreq) is passed in the return code. The first one is numbered 0 and the last one is numbered nreq - 1. Here completion means any of normal completion, abnormal (error) termination, or abort.

The timeout for waiting for completion is set in tmout. The TMO_POL or TMO_FEVR attribute can be specified for tmout. If the wait times out, the requested processing continues. The waitfn return code in case of timeout is
E_TMOUT. The request packet error parameter does not change. Note that if return from waitfn occurs while the requested processing continues, error must be returned in the waitfn return code; but the processing must not be completed when error is passed in the return code, and a value other than error must not be returned if processing is ongoing. As long as error is passed in the waitfn return code, the request is considered to be pending and no request packet is discarded. When the number of a request packet whose processing was completed is passed in the waitfn return code, the processing of that request is considered to be completed and that request packet is discarded.

I/O error and other device-related errors are stored in the request packet error parameter. Error is passed in the waitfn return code when completion waiting did not take place properly. The waitfn return code is set in the tk_wai_dev return code, whereas the request packet error value is returned in ioerr.

The abort processing when the abort function abortfn was executed during completion waiting by waitfn differs depending on whether to wait for completion of a single request (waitfn, nreq = 1) or multiple requests (waitfn, nreq > 1). When waiting for completion of a single request, the request currently processing is aborted. On the other hand, when waiting for completion of multiple requests, as a special handling, only the completion waiting by waitfn is released and the processing for the request itself is not aborted. It means that, even if the abort function abortfn is executed, the request packets’ abort remains FALSE and the processing for the requests continues. E_ABORT is passed in the return code from the released waitfn.

During a wait for request completion, an abort request may be set in the abort parameter of a request packet. In such a case, if it is a single request, the request abort processing must be performed. If the wait is for multiple requests it is also preferable that abort processing be executed, but it is also possible to ignore the abort flag.

When abort occurs, the important thing is to return from waitfn as quickly as possible. If processing will end soon anyway without aborting, it is not necessary to abort.

As a rule, E_ABORT is returned in the request packet error parameter when processing is aborted; but a different error code than E_ABORT may be returned as appropriate based on the device properties. It is also permissible to return E_OK on the basis that the processing right up to the abort is valid. If processing completes normally to the end, E_OK is returned even if there was an abort request.

waitfn runs as a quasi-task portion of the task that issued tk_wai_dev, tk_srea_dev, or tk_swri_dev.

In a device driver for which TDA_DEV_D is specified as an attribute at the time of registering the device, the wait-for-completion function (64-bit request packet, millisecond timeout) waitfn is called when tk_wai_dev is invoked. In this case, the function specification is the same as that of 32-bit request packet, millisecond timeout waitfn, except that the parameter request packet is a 64-bit T_DEVREQ_D* devreq_d.

In a device driver for which TDA_TMO_U is specified as an attribute at the time of registering the device, the wait-for-completion function (32-bit request packet, microsecond timeout) waitfn is called when tk_wai_dev is invoked. In this case, the function specification is the same as that of 32-bit request packet, millisecond timeout waitfn, except that the parameter timeout specification is a microsecond TMO_U tmout_u.

In a device driver for which TDA_DEV_D and TDA_TMO_U are specified as an attribute at the time of registering the device, the wait-for-completion function (64-bit request packet, microsecond timeout) waitfn is called when tk_wai_dev is invoked. In this case, the function specification is the same as that of 32-bit request packet, millisecond timeout waitfn, except that the parameter request packet is a 64-bit T_DEVREQ_D* devreq_d and the parameter timeout specification is a microsecond TMO_U tmout_u.
5.2.3.2.5 abortfn - Abort function

C Language Interface

/* abort function (32-bit request packet) */

ER ercd = abortfn(ID tskid, T_DEVREQ *devreq, INT nreq, void *exinf);

/* abort function (64-bit request packet) */

ER ercd = abortfn(ID tskid, T_DEVREQ_D *devreq_d, INT nreq, void *exinf);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Task ID of the task executing execfn or waitfn</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_DEVREQ*</td>
<td>devreq</td>
<td>Device Request Packet</td>
<td>Request packet list (32-bit)</td>
</tr>
<tr>
<td>T_DEVREQ_D*</td>
<td>devreq_d</td>
<td>Device Request Packet</td>
<td>Request packet list (64-bit)</td>
</tr>
<tr>
<td>INT</td>
<td>nreq</td>
<td>Number of Requests</td>
<td>Request packet count</td>
</tr>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
<td>Extended information set at device registration</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other</td>
<td>Error code returned by the device driver</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Error Code

Other Error code returned by the device driver

Description

The abort function abortfn is called when you want to promptly return from the currently running execute function execfn or wait-for-completion function waitfn. Normally this means the request being processed is aborted. If, however, the processing can be completed soon without aborting, it may not have to be aborted. The important thing is to return as quickly as possible from execfn or waitfn.

abortfn is called in the following cases.

- When a break function is executing after a task exception and the task that raised the exception requests abort processing, abortfn is used to abort the request being processed by that task.
- When a device is being closed by tk_cls_dev, and the device descriptor was processing a request, abortfn is used to abort the request being processed by the device descriptor.

_tskid indicates the task executing the request specified in devreq. In other words, it is the task executing execfn or waitfn. devreq and nreq are the same as the parameters that were passed to execfn or waitfn. In the case of execfn, nreq is always 1.

abortfn is called by a different task from the one executing execfn or waitfn. Since both tasks run concurrently, mutual exclusion control must be performed as necessary. It is possible that the abortfn function will be called immediately before calling execfn or waitfn, or during return from these functions. Measures must be taken to ensure proper operation in such cases. Before abortfn is called, the abort flag in the request packet whose
processing is to be aborted is set to TRUE, enabling execfn or waitfn to know whether there is going to be an abort request. Note also that abortfn can use tk_dis_wai for any object.

When waitfn is executing for multiple requests (nreq > 1), this is treated as a special case differing as follows from other cases.

- Only the completion wait is aborted (waited is released), not the requested processing.
- The abort flag is not set in the request packet (remains as abort = FALSE).

Aborting a request when execfn and waitfn are not executing is done not by calling abortfn but by setting the request packet abort flag. If execfn is called when the abort flag is set, the request is not accepted. If waitfn is called, abort processing is the same as if abortfn is called.

If a request for which processing was started by execfn is aborted before waitfn was called to wait for its completion, the completion of the aborted processing is notified when waitfn is called later. Even though processing was aborted, the request itself is not discarded until its completion has been checked by waitfn. abortfn initiates abort processing only, returning promptly without waiting for the abort to complete.

The abortfn that is executed on a task exception runs as a quasi-task portion of the task issuing tk_ras_tex that raised the task exception. The abortfn that is executed on a device close runs as a quasi-task portion of the task that issued tk_cls_dev.

In a device driver for which TDA_DEV_D is specified as an attribute at the time of registering the device, the abort function (64-bit request packet) abortfn is called when you want to promptly return from the currently running execute function execfn or wait-for-completion function waitfn. In this case, the function specification is the same as that of 32-bit request packet abortfn, except that the parameter request packet is a 64-bit T_DEVREQ_D* devreq_d.
5.2.3.2.6 eventfn - Event function

C Language Interface

```c
INT retcode = eventfn(INT evttyp, void *evtinf, void *exinf);
```

**Parameter**

<table>
<thead>
<tr>
<th>INT</th>
<th>evttyp</th>
<th>Event Type</th>
<th>Driver request event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>evtinf</td>
<td>Event Information</td>
<td>Information for each event type</td>
</tr>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
<td>Extended information set at device registration</td>
</tr>
</tbody>
</table>

**Return Parameter**

```
INT retcode or Error Code
```

**Return Code**

- Return code defined for each event type
- Error code

**Error Code**

Click Error code returned by the device driver

**Description**

When a state change occurs in the device or system which is caused by a factor other than normal device I/O processing by an application interface, requiring some processing by the device driver, a driver request event is raised and then the event function `eventfn` is called.

The driver request event is raised when suspending or resuming a device for power control (see `tk_sus_dev`) or when connecting a removable device such as USB.

For example, when the system is suspended by `tk_sus_dev`, the driver request event for the suspend (`TDV_SUSPEND`) is raised in the \( \mu \) T-Kernel (during `tk_sus_dev` processing) and the event function for each device is called with `evttyp = TDV_SUSPEND`. The event function called for each device performs necessary operations for suspend such as saving the state on receiving this driver request event.

The following driver request events are defined.

```c
#define TDV_SUSPEND (-1) /* suspend */
#define TDV_RESUME  (-2) /* resume */
#define TDV_CARDEVT 1 /* reserved */
#define TDV_USBEVT  2 /* USB event */
```

The driver request events with a negative value are called internally from the device management in the \( \mu \) T-Kernel/SM, for suspend or resume processing.

On the other hand, the driver request events with a positive value (TDV_USBEVT) are reference specifications which are not directly related to the \( \mu \) T-Kernel operation, and raised by calling `tk_evt_dev`. These driver request events are used as needed to implement a bus driver for USB or other device.

The processing performed by the event function is defined for each event type. For suspend and resume processings, see Section 5.2.3.4, “Device Suspend/Resume Processing”.

When a device event is called by `tk_evt_dev`, the `eventfn` return code is set transparently as the `tk_evt_dev` return code.

Requests to event functions must be accepted even if another request is processed, and must be processed as quickly as possible.

The `eventfn` runs as a quasi-task portion of the task that issued `tk_evt_dev` or `tk_sus_dev` that caused the event.
Additional Notes

The following behaviors are assumed for USB event.

Note that they describe implementation examples of device drivers that handle a device such as USB and are not part of the μT-Kernel specification.

When a USB device is connected, a class driver should dynamically be mapped to the USB device to perform an actual I/O processing.

For example, when a storage such as USB memory is connected, a device driver for the mass storage class handles the I/O for the device, or when a USB camera is connected, a device driver for the video class handles the I/O for the device. Which device driver should be used cannot be determined until the USB device is connected.

In this case, the driver request event for the USB connection and the event function for each device driver are used in order to map a class driver to the USB device. Specifically, when the USB bus driver (USB manager) monitoring the USB ports detects a newly connected USB device, it sends the driver request event for the USB connection (TDV_USBEVT) to each device driver which will be candidate of the class driver and then calls the event function for each device.

The event function for each device returns whether or not it can support the newly connected USB device in response to this TDV_USBEVT. The USB bus driver receives the return codes and determines the mapping to the actual class driver.
5.2.3.3 Device Event Notification

A device driver sends events that occur on each device to the specific message buffer (event notification message buffer) as device event notification messages. The event notification message buffer ID is referenced or set as an attribute data of TDN_EVENT for each device.

The system default event notification message buffer is used immediately after device registration. As a device is registered by tk_def_dev when a device driver is started, the system default event notification message buffer ID value is returned as this API’s return parameter, the value is held in the device driver and is used as the initial value of this attribute data, TDN_EVENT.

The system default event notification message buffer is created at system startup. Its size and maximum message length are defined by TDEvtMbfSz in the system configuration information.

The message formats used in device event notification are as follows: The content and size of the event notification message vary depending on the event type.

◇ Basic format of device event notification

```c
typedef struct t_devevt {
    TDEvtTyp   evttyp;    /* event type */
    /* Information specific to each event type is appended here. */
} T_DEVEVT;
```

◇ Format of device event notification with device ID

```c
typedef struct t_devevt_id {
    TDEvtTyp   evttyp;    /* event type */
    ID         devid;     /* Device ID */
    /* Information specific to each event type is appended here. */
} T_DEVEVT_ID;
```

◇ Format of device event notification with extended information

```c
typedef struct t_devevt_ex {
    TDEvtTyp   evttyp;    /* event type */
    ID         devid;     /* Device ID */
    UB         exdat[16]; /* Extended information */
    /* Information specific to each event type is appended here. */
} T_DEVEVT_EX;
```

The event type of a device event notification is classified as follows:

a. Basic event notification (event type: 0x0001 to 0x002F)
   Basic event notification from a device

b. System event notification (event type: 0x0030 to 0x007F)
   Event notification related to entire system such as power supply control

c. Event notification with extended information (event type: 0x0080 to 0x00FF)
   Event notification from a device with extended information

d. User-defined event notification (event type: 0x0100 to 0xFF)
   Notification of event that users can arbitrarily define

Typical event types are as follows:
typedef enum tdevttyp {
    TDE_unknown = 0, /* undefined */
    TDE_MOUNT = 0x01, /* media insert */
    TDE_EJECT = 0x02, /* Eject media */
    TDE_POWEROFF = 0x31, /* power switch off */
    TDE_POWERLOW = 0x32, /* low power alarm */
    TDE_POWERFAIL = 0x33, /* abnormal power */
    TDE_POWERSUS = 0x34 /* auto suspend */
} TDEvtTyp;

Measures must be taken so that if event notification cannot be sent because the message buffer is full, the lack of notification will not adversely affect operation on the receiving end. One option is to hold the notification until space becomes available in the message buffer, but in that case other device driver processing should not, as a rule, be allowed to fall behind as a result. Processing on the receiving end should be designed to avoid message buffer overflow as much as possible.
5.2.3.4  Device Suspend/Resume Processing

Device drivers perform suspend and resume operations in response to the issuing of suspend/resume events (TDV_SUSPEND/TDV_RESUME) to the event handling function (eventfn). Suspend and resume events are issued only to physical devices.

5.2.3.4.1  Device suspend processing

The event for starting suspend processing is as follows:

\[
\begin{align*}
evtyp &= \text{TDV SUSPEND} \\
evtinf &= \text{NULL (none)}
\end{align*}
\]

By issuing suspend event (TDV_SUSPEND), suspend processing takes place as follows.

1. If there is a request being processed at the time, the device driver waits for it to complete, pauses it or aborts it. Which of these options to take depends on the device driver implementation. Since the suspension must be effected as quickly as possible, however, pause or abort should be chosen if completion of the request will take time.
   Suspend events can be issued only for physical devices, but the same processing is applied to all logical devices included in the physical device.
   - Pause: Processing is suspended, then continues after the device resumes operation.
   - Abort: Processing is aborted just as when the abort function (abortfn) is executed, and is not continued after the device resumes operation.

2. New requests other than a resume event are not accepted.

3. The device power is cut off and other suspend operation is performed.

Abort should be avoided if possible because of its effects on applications. It should be used only in such cases as long input wait from a serial port, or when pause would be difficult. Normally it is best to wait for completion of a request or, if possible, choose pause (suspend and resume).

Requests arriving at the device driver in suspend state are made to wait until operation resumes, after which they are accepted for processing. If the request does not involve access to the device, however, or otherwise can be processed even during suspension, a request may be accepted without waiting for resumption.

5.2.3.4.2  Device resume processing

The event for starting resume processing is as follows:

\[
\begin{align*}
evtyp &= \text{TDV RESUME} \\
evtinf &= \text{NULL (none)}
\end{align*}
\]

By issuing resume event (TDV_RESUME), resume processing takes place as follows.

1. The device power is turned back on, the device states are restored and other device resume processing is performed.

2. Paused processing is resumed.

3. Accepting request is resumed.
5.3 Interrupt Management Functions

μT-Kernel/SM interrupt management functions are functions for disabling or enabling external interrupt, retrieving interrupt disable status, controlling interrupt controller, etc.

Interrupt handling is largely hardware-dependent, different on each system, and therefore difficult to standardize. The following are given as standard specification, but it may not be possible to follow these exactly on all systems. Implementors should comply with these specifications as much as possible; but where implementation is not feasible, full compliance is not mandatory. If functions not in the standard specification are added, however, the function names must be different from those given here. In any case, DI, EI, and isDI must be implemented in accordance with the standard specification.

Interrupt management functions are provided as library functions or C language macros. These can be called from a task-independent portion and while dispatching and interrupts are disabled.
5.3.1 CPU Interrupt Control

These functions control the external interrupt mask flag or interrupt mask level in the CPU. Generally speaking, interrupt controller is not touched.

**DI** disables all the external interrupts and **EI** enables them. After **DI** is issued and until **EI** is issued, the system is in external interrupt disabled state. In this state, an indivisible processing can be performed since no interruption occurs and no dispatching takes place.

There are a few restrictions about the API to control CPU interrupt and the external interrupt disabled state.

- CPU interrupt control API is usually implemented as C language compile time macro to set the external interrupt mask flag or interrupt mask level inside the CPU. Hence, this API can be invoked only in the privileged level that can access and control hardware directly. The precise meaning of the level is implementation-dependent.

- CPU interrupt control API only sets CPU’s external interrupt mask flag or interrupt mask level only. Hence, generally speaking, except for some implementations, the execution of these APIs will not cause delayed dispatching.

- There are restrictions on the available APIs in the external interrupt disabled state. API that puts the calling task into waiting state cannot be invoked. The system should return E_CTX. However, the proper error checking to return E_CTX is implementation-dependent. The following APIs of μT-Kernel/SM, Interrupt Management Functions and I/O Port Access Support Functions, can be invoked even when external interrupt is disabled. Whether other APIs can be invoked when external interrupt is disabled is implementation-dependent.

- System timer interrupt is disabled in the external interrupt disabled state. Hence, no timeout occurs, and no time event handler processing occurs.

---

**Additional Notes**

The APIs for controlling CPU interrupt is meant for device drivers to perform indivisible execution for low-level control such as hardware by disabling external interrupt temporarily. However, the disabling of external interrupt reduces the system responsiveness and the real-time performance suffers. So the indivisible operation should be finished quickly and external interrupt disabled state should be exited soon.

External interrupt disabled state entered by **DI** is very similar to task-independent portion. Even if an API that would usually cause dispatch such as **tk_wup_tsk** is invoked, dispatching does not occur. Afterward, when **EI** is issued to return to external interrupt enabled state, the delayed dispatching associated with **EI** do not occur generally (except for some implementations). As a result, after **EI** is issued, we may have an unexpected situation where a lower priority task continues to run even though a higher priority task in READY state exists.

To avoid the unexpected situation, when a program needs to issue an API that causes dispatching during the time interval that starts with **DI** and ends with **EI**, it is recommended to surround the interval of external interrupt disabled state by a pair of **tk_dis_dsp** and **tk_ena_dsp**. Namely, the APIs should be issued in the following order: **tk_dis_dsp** → **DI** → API that causes dispatching → **EI** → **tk_ena_dsp**. With this order of issuing the APIs, external interrupt and dispatching are disabled between **DI** and **EI**. Only dispatching is disabled between **tk_dis_dsp** and **tk_ena_dsp**. And at the timing of the last **tk_ena_dsp**, delayed dispatching does take place. Hence, the unexpected situation mentioned in the preceding sentences is corrected after all. Issuing the APIs in this order guarantees the same system behavior that is not implementation-dependent.
5.3.1.1 DI - Disable External Interrupts

C Language Interface

```c
#include <tk/tkernel.h>

DI (UINT intsts);
```

Parameter

| UINT  | intsts | Interrupt Status | Variable that stores the CPU external interrupt flag |

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Description

Controls the external interrupt flag in the CPU and disables all external interrupts. Also stores the flag state in `intsts` before disabling interrupt.

`intsts` is not a pointer. Write a variable directly. Generally, this API is defined as a C language macro.

Regarding the APIs that can be issued during external interrupt disabled state, see the explanation at the beginning of Section 5.3.1, “CPU Interrupt Control”. 
5.3.1.2 EI - Enable External Interrupt

C Language Interface

#include <tk/tkernel.h>

EI (UINT intsts);

Parameter

| UINT  | intsts | Interrupt Status | Variable that stores the CPU external interrupt flag |

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Description

Controls the external interrupt flag in the CPU and reverts the flag state to intsts. That is, this API reverts the flag state to the state before disabling external interrupts by the previously executed DI(intsts).

If the state before executing DI(intsts) was the external-interrupt-enabled, the subsequent EI(intsts) enables external interrupts. On the other hand, if the state was already interrupt-disabled at the time DI(intsts) was executed, interrupt is not enabled by EI(intsts). However, if 0 is specified in intsts, the external interrupt flag in the CPU is set to the interrupt-enable state.

intsts must be either the value saved by DI or 0. If any other value is specified, the subsequent correct behavior is not guaranteed.

The specifications pays attention to the execution efficiency to minimize overhead. Therefore, this API is usually implemented using assembly language or C language macro. This API controls the external interrupt mask flag in the CPU only and does nothing else. No error result is returned. Hence, except for some implementations, the execution of this API will not cause delayed dispatching, generally speaking.
5.3.1.3 isDI - Get Interrupt Disable Status

C Language Interface
#include <tk/tkernel.h>

BOOL disint = isDI(UINT intsts);

Parameter

| UINT  | intsts | Interrupt Status | Variable that stores the CPU external interrupt flag |

Return Parameter

| BOOL  | disint | Interrupt Disabled Status | External interrupt disabled status |

Error Codes
None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Description

Checks the external interrupt flag in the CPU that was stored in intsts by the previously executed DI, and returns TRUE (a non-zero value) if the flag status is determined as the interrupt-disabled, or FALSE otherwise. intsts must be the value saved by DI. If any other value is specified, the subsequent correct behavior is not guaranteed.

This specification pay attention to the execution efficiency to minimize overhead. Therefore, this API is usually implemented using assembly language or C language macro.

Sample Usage of isDI

```c
void foo()
{
    UINT intsts;
    DI(intsts);
    if ( isDI(intsts) ) {
        /* Interrupt was already disabled at the time the above DI() was called */
    } else {
        /* Interrupt was enabled at the time the above DI() was called */
    }
    EI(intsts);
}
```
5.3.1.4 SetCpuIntLevel - Set Interrupt Mask Level in CPU

C Language Interface

```c
#include <tk/tkernel.h>

void SetCpuIntLevel(INT level);
```

Parameter

<table>
<thead>
<tr>
<th>INT level</th>
<th>Interrupt Mask Level</th>
<th>Interrupt mask level</th>
</tr>
</thead>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

`TK_SUPPORT_CPUINTLEVEL` Support of CPU interrupt mask level

Description

Set interrupt mask level of CPU and disable interrupts that have lower interrupt priority than `level`. Interrupts that have interrupt priority that is equal to `level` or higher are enabled.

When `INTLEVEL_DI` is specified to `level`, the interrupt mask level within the interrupt controller is set to disable all external interrupts at all priority levels. Generally speaking, this is the same state of the system after `DI` is called.

When `INTLEVEL_EI` is specified to `level`, the mask level within the interrupt controller is set to enable all external interrupts at all priority levels. Generally speaking, this is the same state of the system after `EI(0)` is called.

While interrupts are disabled due to the execution of this API, dispatch may be delayed, as in the case of the interrupt handler’s being executed, until the interrupts are enabled again.

The range of value that can be specified by `level` and the concrete value of `INTLEVEL DI` are implementation-dependent. The ordering relation of the interrupt level as numeric value and the interrupt priority is implementation-dependent. Generally speaking, the specification about these is decided based on the CPU architecture.

The specifications pays attention to the execution efficiency to minimize overhead. Therefore, this API is usually implemented using assembly language or C language macro. This API controls the interrupt mask level in the CPU only and does nothing else. No error result is returned. Hence, except for some implementations, the execution of this API will not cause delayed dispatching, generally speaking.
Additional Notes

"Interrupt mask level" is defined to be the lower bound of interrupt priority level (interrupt level) for external interrupts that are enabled (masked). External interrupts with priorities equal to or higher than the interrupt mask level are enabled.

This API sets the interrupt mask level within CPU, and has a similar function as that of SetCtrlIntLevel which sets the interrupt mask level within the interrupt controller. The former affects the result of interrupt enable/disable setting done by DI, EI. The latter has nothing to do with this.

This API sets the interrupt mask level within CPU without regard to the previous setting. Note that there are both cases of either the increase of the disabled interrupts, or the decrease of disabled interrupts after the execution of this API.
5.3.1.5  GetCpuIntLevel - Get Interrupt MaskLevel in CPU

C Language Interface
#include <tk/tkernel.h>

INT level = GetCpuIntLevel(void);

Parameter
None.

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interrupt Mask Level</td>
</tr>
</tbody>
</table>

Error Codes
None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_CPUINTLEVEL  Support of CPU interrupt mask level

Description

Get the current value of interrupt mask level in CPU, and return it as the value of return parameter level.
The range of value that can be specified by level is implementation-dependent.

Additional Notes

See the explanation and additional notes in SetCpuIntLevel.
5.3.2 Control of Interrupt Controller

These functions control the interrupt controller. Generally they do not perform any operation with respect to the CPU interrupt flag.
5.3.2.1 EnableInt - Enable Interrupts

C Language Interface

```
#include <tk/tkernel.h>

void EnableInt(UINT intno);
void EnableInt(UINT intno, INT level);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT intno</td>
<td>Interrupt Number</td>
</tr>
<tr>
<td>INT level</td>
<td>Interrupt Priority Level</td>
</tr>
</tbody>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task portion</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_INTCTRL**
  Support of interrupt controller management

Additionally, the following service profile items are related to this API.

- **TK_HAS_ENAINTLEVEL**
  Interrupt priority level (level) can be specified as the 2nd argument

Description

Enable interrupt with interrupt number, `intno`. On a system where interrupt priority level can be specified, `level` is used to specify the interrupt priority level.

The interrupt number that can be specified in `intno` is limited to a number that can be usable by `tk_def_int` and at the same time, an interrupt number that is controlled by the interrupt controller. The subsequent correct behavior of the system as a whole when an invalid `intno` is specified is not guaranteed.

Either the support of `level` or the support without `level` is provided.

Additional Notes

This API does not check for error just as other interrupt-related APIs do not.
5.3.2.2 DisableInt - Disable Interrupts

C Language Interface

```c
#include <tk/tkernel.h>

void DisableInt(UINT intno);
```

**Parameter**

<table>
<thead>
<tr>
<th>UINT</th>
<th>intno</th>
<th>Interrupt Number</th>
<th>Interrupt number</th>
</tr>
</thead>
</table>

**Return Parameter**

None.

**Error Codes**

None.

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_INTCTRL**
  Support of interrupt controller management

**Description**

Disable interrupt with the interrupt number, `intno`. Generally speaking, an interrupt that is disabled will become pending and, once it is enabled by `EnableInt`, an interrupt is generated. If it is desired to cancel an interrupt condition that became pending because the interrupt was disabled, `ClearInt` must be called.

The interrupt number that can be specified in `intno` is limited to a number that can be usable by `tk_def_int` and at the same time, an interrupt number that is controlled by the interrupt controller. The subsequent correct behavior of the system as a whole when an invalid `intno` is specified is not guaranteed.

**Additional Notes**

This API does not check for error just as other interrupt-related APIs do not.
5.3.2.3 ClearInt - Clear Interrupt

C Language Interface

```c
#include <tk/tkernel.h>

void ClearInt(UINT intno);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT intno</td>
<td>Interrupt Number</td>
</tr>
</tbody>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_INTCTRL**
  Support of interrupt controller management

Description

If an interrupt with interrupt number, `intno`, has been generated, it is cleared.

The interrupt number that can be specified in `intno` is limited to a number that can be usable by `tk_def_int` and at the same time, an interrupt number that is controlled by the interrupt controller. The subsequent correct behavior of the system as a whole when an invalid `intno` is specified is not guaranteed.

Additional Notes

This API does not check for errors since it focuses on the execution efficiency to minimize overhead.
5.3.2.4  EndOfInt - Issue EOI to Interrupt Controller

C Language Interface

#include <tk/tkernel.h>

void EndOfInt(UINT intno);

Parameter

<table>
<thead>
<tr>
<th>UINT</th>
<th>intno</th>
<th>Interrupt Number</th>
<th>Interrupt number</th>
</tr>
</thead>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_INTCTRL  Support of interrupt controller management

Description

Issue EOI to Interrupt Controller. \texttt{intno} must identify an interrupt that is the target of EOI. Generally this must be executed at the end of an interrupt handler.

The interrupt number that can be specified in \texttt{intno} is limited to a number that can be usable by \texttt{tk_def_int} and at the same time, an interrupt number that is controlled by the interrupt controller. The subsequent correct behavior of the system as a whole when an invalid \texttt{intno} is specified is not guaranteed.

Additional Notes

This API does not check for errors since it focuses on the execution efficiency to minimize overhead.
5.3.2.5 CheckInt - Check Interrupt

C Language Interface

```c
#include <tk/tkernel.h>

BOOL rasint = CheckInt(UINT intno);
```

Parameter

| UINT    | intno   | Interrupt Number | Interrupt number |

Return Parameter

| BOOL    | rasint  | Interrupt Raised Status | External interrupt raised status |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- `TK_SUPPORT_INTCTRL`: Support of interrupt controller management

Description

Check to see if an interrupt with interrupt number, `intno`, has been generated. If an interrupt with the interrupt number, `intno`, has been generated, `TRUE` (a non-zero value) is returned, and if it has not, then `FALSE` is returned.
5.3.2.6 SetIntMode - Set Interrupt Mode

C Language Interface
#include <tk/tkernel.h>

void SetIntMode(UINT intno, UINT mode);

Parameter

<table>
<thead>
<tr>
<th>UINT intno</th>
<th>Interrupt Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT mode</td>
<td>Mode</td>
</tr>
</tbody>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_INTMODE Support of setting interrupt mode

Description

Set the interrupt mode of interrupt specified by intno to mode.

The interrupt number that can be specified in intno is limited to a number that can be usable by tk_def_int and at the same time, an interrupt number that is controlled by the interrupt controller. The subsequent correct behavior of the system as a whole when an invalid intno is specified is not guaranteed.

The settable modes and how to specify mode are implementation-dependent. The following is an example of settable modes:

\[
\text{mode} := (\text{IM\_LEVEL} \ || \ \text{IM\_EDGE}) \ || (\text{IM\_HI} \ || \ \text{IM\_LOW})
\]

#define IM_LEVEL 0x0002 /* Level trigger */
#define IM_EDGE 0x0000 /* Edge trigger */
#define IM_HI 0x0000 /* H level/Interrupt at rising edge */
#define IM_LOW 0x0001 /* L level/Interrupt at falling edge */

If invalid mode is specified, the subsequent correct behavior is not guaranteed.
Additional Notes

This API does not check for error just as other interrupt-related APIs do not.
5.3.2.7 SetCtrlIntLevel - Set Interrupt Mask Level in Interrupt Controller

C Language Interface

```
#include <tk/tkernel.h>

void SetCtrlIntLevel(INT level);
```

Parameter

| INT level | Interrupt Mask Level | Interrupt mask level |

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

```
TK_SUPPORT_CTRLINTLEVEL
```

Description

Set interrupt mask level of the interrupt controller and disable interrupts that have lower interrupt priority than `level`. Interrupts that have interrupt priority that is equal to `level` or higher are enabled.

When `INTLEVEL_DI` is specified to `level`, the interrupt mask level within the interrupt controller is set to disable all external interrupts at all priority levels.

When `INTLEVEL_EI` is specified to `level`, the mask level within the interrupt controller is set to enable all external interrupts at all priority levels.

While interrupts are disabled due to the execution of this API, dispatch may be delayed, as in the case of the interrupt handler’s being executed, until the interrupts are enabled again.

The range of value that can be specified by `level` and the concrete value of `INTLEVEL DI` are implementation-dependent. The ordering relation of the interrupt level as numeric value and the interrupt priority is implementation-dependent. Generally speaking, the specification about these is decided based on the CPU architecture.

Additional Notes

See the additional notes for `SetUpCpuIntLevel`.
This API sets the interrupt mask level within interrupt controller without regard to the previous setting. Note that there are both cases of either the increase of the disabled interrupts, or the decrease of disabled interrupts after the execution of this API.

This API does not check for errors since it focuses on the execution efficiency to minimize overhead.
5.3.2.8 GetCtrlIntLevel - Get Interrupt Mask Level in Interrupt Controller

C Language Interface

#include <tk/tkernel.h>

INT level = GetCtrlIntLevel(void);

Parameter

None.

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>level</th>
<th>Interrupt Mask Level</th>
<th>Interrupt mask level</th>
</tr>
</thead>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_CTRLINTLEVEL Support of interrupt controller mask level

Description

This returns the current interrupt mask level configured inside the interrupt controller, and return it in the return parameter level.

The range of value that can be specified by level is implementation-dependent.

Additional Notes

See the additional notes for SetCpuIntLevel.
5.4 I/O Port Access Support Functions

I/O port access support functions support accesses or operations to the I/O devices. These include functions that read from or write to the I/O port of the specified address using the unit of byte or word, and a function that realizes a wait for a short time (micro wait) which is used for I/O device operations.

I/O port access support functions are provided as library functions or C language macros. These can be called from a task-independent portion or while task dispatching and interrupts are disabled.

5.4.1 I/O Port Access

In a system with separate I/O space and memory space, I/O port access functions access I/O space. In a system with memory-mapped I/O only, I/O port access functions access memory space. Using these functions will improve software portability and readability even in a memory-mapped I/O system.
5.4.1.1 out_b - Write to I/O Port (In Unit of Byte)

C Language Interface
#include <tk/tkernel.h>

void out_b(INT port, UB data);

Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>port</th>
<th>I/O Port Address</th>
<th>I/O port address</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB</td>
<td>data</td>
<td>Write Data</td>
<td>Data to be written (in unit of byte)</td>
</tr>
</tbody>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_IOPORT  Support of I/O port access

Description

Writes data in byte (8-bit) to the I/O port pointed by the address port.
5.4.1.2 out_h - Write to I/O Port (In Unit of Half-word)

C Language Interface

#include <tk/tkernel.h>

void out_h(INT port, UH data);

Parameter

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>port</td>
<td>I/O Port Address</td>
<td>I/O port address</td>
</tr>
<tr>
<td>UH</td>
<td>data</td>
<td>Write Data</td>
<td>Data to be written (in unit of half-word)</td>
</tr>
</tbody>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_IOPORT Support of I/O port access

Description

Writes data in a half-word (16-bit) to the I/O port pointed by the address port.
5.4.1.3 out_w - Write to I/O Port (In Unit of Word)

C Language Interface
#include <tk/tkernel.h>

void out_w(INT port, UW data);

Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>port</th>
<th>I/O Port Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW</td>
<td>data</td>
<td>Write Data</td>
</tr>
</tbody>
</table>

I/O port address

Write Data

Data to be written (in unit of word)

Return Parameter
None.

Error Codes
None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_IOPORT Support of I/O port access

Description

Writes data in a word (32-bit) to the I/O port pointed by the address port.
5.4.1.4 out_d - Write to I/O Port (In Unit of Double-word)

C Language Interface

```c
#include <tk/tkernel.h>

void out_d(INT port, UD data);
```

Parameter

| INT    | port | I/O Port Address | I/O port address |
| UD     | data | Write Data       | Data to be written (in unit of double-word) |

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- `TK_SUPPORT_IOPORT` Support of I/O port access
- `TK_HAS_DOUBLEWORD` Support of 64-bit data types (D, UD, VD)

Description

Writes `data` in a double-word (64-bit) to the I/O port pointed by the address `port`.

Note that, in a system where I/O port cannot be accessed in double-word (64-bit) due to hardware constraint, `data` is separated into shorter units than double-word (64-bit) before they are written.

Rationale for the Specification

There are many systems where I/O port cannot be accessed in double-word (64-bit) due to hardware constraint such as 32-bit or less I/O data bus. In such systems, the strict specification of `out_d` and `in_d` cannot be implemented; that is, they cannot process `data` in one chunk of the specified bit width. In terms of the original purpose of this API, it is preferable not to implement the `out_d` and `in_d` or return an error at runtime. However, it is not practical to detect an error by determining the bus configuration at runtime, and it is often harmless to separate 64-bit data into 32-bit or narrower units before writing.

This is why the specification of `out_d` and `in_d` allow for the case where 64-bit data cannot be processed in one chunk. Therefore, whether `out_d` and `in_d` support the block access to 64-bit I/O port or not is implementation-
dependent. If the block access to 64-bit I/O port is needed, the system hardware configuration and handling of \texttt{out\_d} and \texttt{in\_d} should be checked.
5.4.1.5 in_b - Read from I/O Port (In Unit of Byte)

C Language Interface

#include <tk/tkernel.h>

UB data = in_b(INT port);

Parameter

| INT   | port | I/O Port Address | I/O port address |

Return Parameter

| UB    | data | Read Data | Data to be read (in unit of byte) |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_IOPORT  Support of I/O port access

Description

Reads data in a byte (8-bit) from the I/O port pointed by the address port and returns it in the return parameter data.
5.4.1.6  in_h - Read from I/O Port (In Unit of Half-word)

C Language Interface

```c
#include <tk/tkernel.h>

UH data = in_h(INT port);
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT port</td>
<td>I/O Port Address</td>
</tr>
</tbody>
</table>

**Return Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UH data</td>
<td>Read Data</td>
</tr>
</tbody>
</table>

**Error Codes**

None.

**Valid Context**

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_IOPORT** Support of I/O port access

**Description**

Reads data in a half-word (16-bit) from the I/O port pointed by the address port and returns it in the return parameter data.
5.4.1.7  in_w - Read from I/O Port (In Unit of Word)

C Language Interface

#include <tk/tkernel.h>

UW data = in_w(INT port);

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>port</td>
</tr>
</tbody>
</table>

I/O Port Address  I/O port address

Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW</td>
<td>data</td>
</tr>
</tbody>
</table>

Read Data  Data to be read (in unit of word)

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

<table>
<thead>
<tr>
<th>Service Profile Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_IOPORT</td>
<td>Support of I/O port access</td>
</tr>
</tbody>
</table>

Description

Reads data in a word (32-bit) from the I/O port pointed by the address port and returns it in the return parameter data.
5.4.1.8  in_d - Read from I/O Port (In Unit of Double-word)

C Language Interface

```c
#include <tk/tkernel.h>

UD data = in_d(INT port);
```

Parameter

| INT  port | I/O Port Address | I/O port address |

Return Parameter

| UD data | Read Data | Data to be read (in unit of double-word) |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

| TK_SUPPORT_IOPORT | Support of I/O port access |
| TK_HAS_DOUBLEWORD | Support of 64-bit data types (D, UD, VD) |

Description

Reads data in a double-word (64-bit) from the I/O port pointed by the address `port` and returns it in the return parameter `data`.

Note that, in a system where I/O port cannot be accessed in one chunk of double-word (64-bit) due to hardware constraint, data is separated into shorter units than double-word (64-bit) before reading.

Rationale for the Specification

See Section 5.4.1.4, “out_d - Write to I/O Port (In Unit of Double-word)”. 
5.4.2 Micro Wait

5.4.2.1 WaitUsec - Micro Wait (Microseconds)

C Language Interface

#include <tk/tkernel.h>

void WaitUsec(UW usec);

Parameter

| UW usec | Micro Seconds | Wait time (in microseconds) |

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_MICROWAIT Support of micro wait

Description

Performs a micro wait for the specified interval (in microseconds).

This wait is usually implemented as a busy loop. This means that the micro wait occurs in the task RUNNING state rather than WAITING state.

The micro wait is easily influenced by the runtime environment, such as execution in RAM, execution in ROM, memory cache on or off, etc. The wait time is therefore not very accurate.
5.4.2.2 WaitNsec - Micro Wait (Nanoseconds)

C Language Interface

```c
#include <tk/tkernel.h>

void WaitNsec(UW nsec);
```

Parameter

<table>
<thead>
<tr>
<th>UW nsec</th>
<th>Nanoseconds</th>
<th>Wait time (in nanoseconds)</th>
</tr>
</thead>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_MICROWAIT** Support of micro wait

Description

Performs a micro wait for the specified interval (in nanoseconds).

This wait is usually implemented as a busy loop. This means that the micro wait occurs in the task RUNNING state rather than WAITING state.

The micro wait is easily influenced by the runtime environment, such as execution in RAM, execution in ROM, memory cache on or off, etc. The wait time is therefore not very accurate.
5.5 Power Management Functions

Power management functions are used to realize system power saving. Power management functions are called as a callback type function from within \( \mu \text{T-Kernel/OS} \).

Though \texttt{low_pow} and \texttt{off_pow} exist as part of APIs that are defined in the power management function, they are INTERNAL reference specification and should be used only internally inside the \( \mu \text{T-Kernel} \). Since device drivers, middleware, and applications do not call these APIs directly, it is allowed to modify the functions or their APIs in the original specification to realize more advanced power management function. If, however, the new functions implemented have only the equivalent or similar performance as the APIs being defined as the INTERNAL reference specification here, it is preferable to follow this INTERNAL reference specification in order to enhance the program reusability.

Calling method of APIs for these functions is also implementation-dependent. Simple system calls are possible, as is the use of a trap. These functions may be provided in programs other than the \( \mu \text{T-Kernel} \). Use of an extended SVC or other means that makes use of \( \mu \text{T-Kernel function} \) is not possible, however.
5.5.1 low_pow - Move System to Low-power Mode

C Language Interface

```c
void low_pow(void);
```

Parameter

None.

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_LOWPOWER**
  
  Support of power management functions

Description

It is called within the \(\mu\) T-Kernel task dispatcher, and performs processing that will put CPU hardware into low-power consumption mode.

After moving CPU to the low-power mode, `low_pow` waits for an external interrupt. When an external interrupt occurs, `low_pow` moves the CPU and its associated hardware back to the normal mode (non low-power mode) and then returns to the caller of it.

The detailed processing procedure for `low pow` is as follows:

1. Move CPU to the low-power mode. For example, lower the clock frequency.
2. Stop CPU, waiting for an external interrupt. For example, execute such a CPU instruction.
3. Resume CPU after an external interrupt (by hardware).
4. Move the CPU back to the normal mode. For example, restore the normal clock frequency.
5. Return to the caller. Caller is actually the internal dispatcher within \(\mu\) T-Kernel.

When implementing `low_pow`, the following points need to be noted:

- This function is called in interrupts disabled state.
• Interrupts must not be enabled.
• Since the processing speed affects the speed of response to an interrupt, it should be as fast as possible.

Additional Notes

The task dispatcher calls low_pow to lower the power consumption when it has no tasks to be executed.
5.5.2 off_pow - Move System to Suspend State

C Language Interface

void off_pow(void);

Parameter

None.

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_LOWPOWER Support of power management functions

Description

This is called during the processing of tk_set_pow with powmode = TPW_DOSUSPEND within μ T-Kernel, and it will move the CPU hardware and its peripherals to suspend state (state where the applied power is off).

After moving the hardware to the suspend state, off_pow waits for a resume factor (power on, etc.). When a resume factor occurs, off_pow releases the suspend state and then returns to the caller of it.

The detailed processing procedure for off_pow is as follows:

1. Move CPU to the suspend state and wait for a resume factor. For example, stop the clock.
2. Resume CPU on the occurrence of a resume factor (by hardware).
3. Move CPU or other hardware back to the normal state, if necessary. Release the suspend state.(may be processed by hardware together with the previous step)
4. Return to the caller. Caller is actually the processing portion of tk_set_pow in μ T-Kernel.

When implementing off_pow, the following points need to be noted:

- This function is called in interrupts disabled state.
- Interrupts must not be enabled.
Note that the device drivers perform the suspending and resuming of peripherals and other devices. For more details, see the description of `tk_sus_dev`. 
5.6 System Configuration Information Management Functions

System configuration information management functions maintain and manage various information related to system configuration.

A part of system configuration information including the information on the maximum number of tasks, timer interrupt intervals, etc. are defined as the standard definition. Other than these, any information arbitrarily defined in applications, subsystems, or device drivers can be used by adding it to the system configuration information.

The format of system configuration information consists of a name and defined data as a pair.

**Name**
The name is a string of up to 16 characters. A character encoding is US-ASCII. Characters that can be used (UB) are a to z, A to Z, 0 to 9 and '_' (underscore).

**Defined Data**
Data consists of numbers (integers) or character strings.
Characters that can be used (UB) are any characters other than 0x00 to 0x1F, 0x7F, or 0xFF (in character code).

<table>
<thead>
<tr>
<th>Name</th>
<th>Defined Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysVer</td>
<td>3 0</td>
</tr>
<tr>
<td>SysName</td>
<td>microT-Kernel Version 3.00</td>
</tr>
</tbody>
</table>

How the system configuration information is to be stored is not specified here, but it is generally put in memory (ROM/RAM). This functionality is therefore not intended for storing large amounts of information.

System configuration information can be retrieved by `tk_get_cfn` and `tk_get_cfs`.

However, system configuration information cannot be added or changed during system execution.
5.6.1 System Configuration Information Acquisition

There are `tk_get_cfn` and `tk_get_cfs` as API to retrieve system configuration information. These are callable from applications, subsystems, device drivers, etc. and are also used internally in the $\mu$ T-Kernel.
5.6.1.1  tk_get_cfn - Get Numbers

C Language Interface

#include <tk/tkernel.h>

INT ct = tk_get_cfn(CONST UB *name, W *val, INT max);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST UB* name</td>
<td>Name Name</td>
</tr>
<tr>
<td>W* val</td>
<td>Value Array storing numbers</td>
</tr>
<tr>
<td>INT max</td>
<td>Maximum Count Number of elements in val array</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Return Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT ct</td>
<td>Defined Numeric Information Count Number of defined numeric information</td>
</tr>
<tr>
<td>or Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

E_NOEXS No information is defined for the name specified in the name parameter

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

<table>
<thead>
<tr>
<th>Service Profile Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_SYSCONF</td>
<td>Support of system configuration information management functions</td>
</tr>
</tbody>
</table>

Description

Gets numeric information from system configuration information. This function gets up to max items of numerical information defined for the name specified in the name parameter and stores the acquired information in val. The number of defined numeric information is passed in the return code. If return code > max, this indicates that not all the information could be stored. By specifying max = 0, the number of defined numeric values can be found out without actually storing them in val.

E_NOEXS is returned if no information is defined for the name specified in the name parameter. The behavior if the information defined for name is a character string is indeterminate.

This function can be invoked from any protection level, without being limited to the protection level from which μT-Kernel/OS system call can be invoked.
5.6.1.2 tk_get_cfs - Get Character String

C Language Interface

#include <tk/tkernel.h>

INT rlen = tk_get_cfs(CONST UB *name, UB *buf, INT max);

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB*</td>
<td>name</td>
<td>Name</td>
</tr>
<tr>
<td>UB*</td>
<td>buf</td>
<td>Buffer Array storing character string</td>
</tr>
<tr>
<td>INT</td>
<td>max</td>
<td>Maximum Length</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>rlen</td>
<td>Size of Defined Character String Information</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

E_NOEXS No information is defined for the name specified in the name parameter

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

TK_SUPPORT_SYSCONF Support of system configuration information management functions

Description

Gets character string information from system configuration information. This function gets up to max characters of character string information defined for the name specified in the name parameter and stores the acquired information in buf. If the acquired character string is shorter than max characters, it is terminated by '\0' when stored. The length of the defined character string information (not including '\0') is passed in the return code. If return code > max, this indicates that not all the information could be stored. By specifying max = 0, the character string length can be found out without actually storing anything in buf.

E_NOEXS is returned if no information is defined for the name specified in the name parameter. The behavior if the information defined for name is a numeric string is indeterminate.

This function can be invoked from any protection level, without being limited to the protection level from which μT-Kernel/OS system call can be invoked.
### 5.6.2 Standard System Configuration Information

The following information is defined as standard system configuration information. A standard information name is prefixed by T.

<table>
<thead>
<tr>
<th>character string</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Numeric string information</td>
</tr>
<tr>
<td>S</td>
<td>Character string information</td>
</tr>
</tbody>
</table>

- **Product information**

<table>
<thead>
<tr>
<th>character string</th>
<th>Name of standard definition</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>TSysName</td>
<td>System name (product name)</td>
</tr>
</tbody>
</table>

- **Maximum number of objects**

<table>
<thead>
<tr>
<th>character string</th>
<th>Name of standard definition</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>TMaxTskId</td>
<td>Maximum number of tasks</td>
</tr>
<tr>
<td>N</td>
<td>TMaxSemId</td>
<td>Maximum number of semaphores</td>
</tr>
<tr>
<td>N</td>
<td>TMaxFlgId</td>
<td>Maximum number of event flags</td>
</tr>
<tr>
<td>N</td>
<td>TMaxMbxId</td>
<td>Maximum number of mailboxes</td>
</tr>
<tr>
<td>N</td>
<td>TMaxMtxId</td>
<td>Maximum number of mutexes</td>
</tr>
<tr>
<td>N</td>
<td>TMaxMbfId</td>
<td>Maximum number of message buffers</td>
</tr>
<tr>
<td>N</td>
<td>TMaxMpfId</td>
<td>Maximum number of fixed-size memory pools</td>
</tr>
<tr>
<td>N</td>
<td>TMaxMplId</td>
<td>Maximum number of variable-size memory pools</td>
</tr>
<tr>
<td>N</td>
<td>TMaxCycId</td>
<td>Maximum number of cyclic handlers</td>
</tr>
<tr>
<td>N</td>
<td>TMaxAlmId</td>
<td>Maximum number of alarm handlers</td>
</tr>
<tr>
<td>N</td>
<td>TMaxSsyId</td>
<td>Maximum number of subsystems</td>
</tr>
<tr>
<td>N</td>
<td>TMaxSsyPri</td>
<td>Maximum number of subsystem priorities</td>
</tr>
</tbody>
</table>

- **Other**

<table>
<thead>
<tr>
<th>character string</th>
<th>Name of standard definition</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>TSysStkSz</td>
<td>Default system stack size (in bytes)</td>
</tr>
<tr>
<td>N</td>
<td>TSVCLimit</td>
<td>Lowest protection level for system call invoking</td>
</tr>
<tr>
<td>N</td>
<td>TTimPeriod</td>
<td>Timer interrupt interval (in milliseconds)Timer interrupt interval (in microseconds)</td>
</tr>
</tbody>
</table>

The actual length of timer interrupt interval is a sum of time in milliseconds and time in microseconds. The interval in microseconds is assumed to be 0 when omitted.

For example, when timer interrupt interval should be 5 milliseconds, describe as "TTimPeriod 5" or "TTimPeriod 0 5000". When timer interrupt interval should be 1.5 milliseconds (1,500 microseconds), describe as "TTimPeriod 1 500" or "TTimPeriod 0 1500".
• device management function

<table>
<thead>
<tr>
<th>character string</th>
<th>Name of standard definition</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>TMaxRegDev</td>
<td>Maximum number of device registrations</td>
</tr>
<tr>
<td>N</td>
<td>TMaxOpnDev</td>
<td>Maximum device open count</td>
</tr>
<tr>
<td>N</td>
<td>TMaxReqDev</td>
<td>Maximum number of device requests</td>
</tr>
<tr>
<td>N</td>
<td>TDEvtMbfSz</td>
<td>Event notification message buffer size (in bytes) Maximum event notification message length (in bytes)</td>
</tr>
</tbody>
</table>

If TDEvtMbfSz is not defined or if the message buffer size is a negative value, an event notification message buffer is not used.

When multiple values are defined for any of the above numeric strings, they are stored in the same order as in the explanation.

Example of Storage Order of More than One Numeric Value

```
tk_get_cfn("TDEvtMbfSz", val, 2)
```

val[0] = Event notification message buffer size
val[1] = Maximum event notification message length
5.7 Memory Cache Control Functions

Memory cache control functions perform a cache control or mode setting.

The approach of cache control in μT-Kernel is as follows:

Basically, even if application and device driver programs are created without paying attention to the existence of cache, the appropriate cache control should be automatically performed during their execution. Especially, in consideration of program portability, functions with strong dependency on system including cache are better to be handled separately from application programs wherever possible. For this reason, it is the policy of individual systems based on μT-Kernel to make the μT-Kernel itself control the cache automatically.

Specifically, μT-Kernel sets the cache so that it is turned ON for space like memory to store usual programs or data, and OFF for space such as I/O. For this reason, ordinary application programs do not need to explicitly call a function for cache control. Appropriate cache control is automatically performed even if cache control is not explicitly performed from the program.

However, the cache control by μT-Kernel only (cache control by default setting) may not be enough for particular situations. For example, for I/O processing with DMA transfer or using memory space outside the kernel management, explicit cache control may be required. When executing a program by dynamically loading or generating (compiling) it, such cache control may be required so that data cache and instruction cache are appropriately synchronized. Memory cache control functions are assumed to be used in these situations.
5.7.1 SetCacheMode - Set Cache Mode

C Language Interface

#include <tk/tkernel.h>

SZ rlen = SetCacheMode(void *addr, SZ len, UINT mode);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Short Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>void*</td>
<td>Start Address</td>
</tr>
<tr>
<td>len</td>
<td>SZ</td>
<td>Length</td>
</tr>
<tr>
<td>mode</td>
<td>UINT</td>
<td>Mode</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Return Parameter</th>
<th>Short Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rlen</td>
<td>SZ</td>
<td>Result Length</td>
</tr>
</tbody>
</table>

or Error Code

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error (addr, len, or mode is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOSPT</td>
<td>Unsupported function (function specified in mode is unsupported)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>YES</th>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task portion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quasi-task portion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task-independent portion</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

<table>
<thead>
<tr>
<th>Service Profile Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_CACHECTRL</td>
<td>Support of memory cache control functions</td>
</tr>
<tr>
<td>TK_SUPPORT_SETCACHEMODE</td>
<td>Support of set cache mode function</td>
</tr>
</tbody>
</table>

Additionally, the following service profile items are related to this API.

<table>
<thead>
<tr>
<th>Service Profile Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_WBCACHE</td>
<td>Support for specifying write-back mode for cache mode(CM_WB)</td>
</tr>
<tr>
<td>TK_SUPPORT_WTCACHE</td>
<td>Support for specifying write-through mode for cache mode(CM_WT)</td>
</tr>
</tbody>
</table>

Description

Sets the cache mode for a memory area. Specifically, performs the setting specified in mode for the cache of the len bytes memory area from the address addr.
mode := ( CM_OFF || CM_WB || CM_WT ) | [CM_CONT]

CM_OFF Cache off
CM_WB Cache on (write back)
CM_WT Cache on (write through)
CM_CONT Applies the cache setting only for the contiguous address space

(physical address)

/* Implementation-dependent mode may be added */

Specify **CM_OFF** in `mode` to flush (writes back) the cache, invalidate it, and turn it off.
Specify **CM_WT** in `mode` to flush the cache and then set the write through cache mode.
Specify **CM_WB** in `mode` to set the write back cache mode. In this case, whether or not to flush the cache is implementation-dependent.
Specify **CM_CONT** in `mode` to apply the cache mode setting only for the contiguous address (physical address) space area starting from `addr`. If non-allocated area exists within the specified space, the processing is aborted immediately before the area and the size of the processed space is returned. If **CM_CONT** is not specified, then the whole area is the target of the cache mode processing, and the size of the area for which the processing has been performed is returned.

Some or all of the cache mode settings may be unusable depending on CPU or implementation. If an unusable mode is specified, E_NOSPT is returned without any processing.

`len` must be 1 or more. If a value of 0 or less is specified, the error code E_PAR is returned.

**Additional Notes**

Generally speaking, because the cache mode setting is performed in page units, the start address of the page including `addr` and subsequent addresses is taken as the setting target when `addr` is not at the start of the specified area. Note that unintended cache access may occur to an adjacent area when using this API. Care should be taken.

When you want more detailed cache mode settings depending on the hardware configuration or the cache function of CPU, add and use an implementation-dependent `mode`. For example, **NORMAL CACHE OFF (Weakly Order)**, **DEVICE CACHE OFF (Weakly Order)**, **STRONG ORDER**, or other cache mode may be specified.

When an unavailable `mode` is specified, it is implementation-dependent whether to generate an error as E_NOSPT or E_PAR.
5.7.2 ControlCache - Control Cache

C Language Interface

#include <tk/tkernel.h>

SZ rlen = ControlCache(void *addr, SZ len, UINT mode);

Parameter

| void* addr | Start Address | Start address |
|SZ len | Length | Memory area size (in bytes) |
|UINT mode | Mode | Control mode |

Return Parameter

| SZ rlen | Result Length | Size of the area for which the cache mode was set (in bytes) |
| or Error Code | Error code |

Error Code

- E_OK: Normal completion
- E_PAR: Parameter error (invalid addr, len or mode)
- E_NOSPT: Unsupported function (function specified in mode is unsupported)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- TK_SUPPORT_CACHETRL: Support of memory cache control functions

Description

Control the cache (flush or invalidate) of a memory area. Specifically, performs the control specified in mode for the cache of the len bytes memory area from the logical address addr.

mode := (CC_FLUSH | CC_INVALIDATE)

- CC_FLUSH: Flush (write back) cache
- CC_INVALIDATE: Invalidate cache
- ...

/* Implementation-dependent mode values may be added */

Both CC_FLUSH and CC_INVALIDATE can be set at the same time. This combination flushes the cache and then invalidates it.
If the processing is successful, the size of the processed space is returned.

A range that spans areas with different cache modes or attributes must not be specified. For example, a range that spans areas with cache on and cache off, or areas with different protection levels must not be specified. If such a range is specified, the subsequent correct behavior is not guaranteed.

The detail of the function varies depending on CPU, hardware, or implementation because the cache control depends heavily on the hardware. The cache control is basically applied on the specified area using the specified mode, but it may affect more area including the specified area. For example, there are the following cases:

- Only the exactly specified range is not always controlled (flushed or invalidated). An area including the specified range is controlled, but it is also possible to flush or invalidate the cache for other areas (for example, entire memory) depending on CPU, hardware, or implementation.
- Normally, no operation is performed when a cache-off area is specified. Even in this case, it is possible to flush or invalidate the cache for areas other than the specified range. (always flush the entire space, etc.)
- No operation is performed in a system without cache.

Generally, the cache control is performed in cache line size units. For this reason, note that unintended cache access may occur to adjacent area when using this API.
5.8 Physical Timer Functions

Physical timer functions are useful in the system equipped with more than one hardware timer when processing should be performed based on smaller unit of elapsed time than the timer interrupt interval (TTimPeriod).

A physical timer means a hardware counter that is monotonically incremented by one from 0 at a constant time interval. When a count value reaches a certain value (upper limit) specified for each physical timer, the handler (physical timer handler) specified for each physical timer is started and the count value is reset to 0.

More than one physical timer can be used depending on the number of hardware timers available in the system. The number of available physical timers is implementation-dependent. In the usual μT-Kernel implementation, one hardware timer is used to realize the time management functions. Therefore it is assumed that remaining hardware timers are used for the physical timers.

Positive integer of ascending order like 1, 2, ... is used as a physical timer number. For example, when there are four hardware timers, as one of them is used for the μT-Kernel time management functions, remaining three hardware timers are available with physical timer numbers assigned as 1, 2, and 3, respectively.

μT-Kernel/SM physical timer functions do not manage coordination between an individual physical timer and tasks that use the timer. If more than one task share one physical timer, coordination like mutual exclusion control must be performed on the application side.

Additional Notes
For the μT-Kernel time management functions, the kernel starts alarm handler or cyclic handler, processes timeout, and processes these requests, all in the handler that is started on the time interval specified by "timer interrupt interval" (TTimPeriod) in Section 5.6.2, "Standard System Configuration Information". On the other hand, the physical timer functions only standardize the primitive functions such as setting a hardware timer, reading a count value, and triggering interrupt. They do not handle simultaneous multiple requests like the time management functions do. Based on this observation, the physical timer functions carry the name of "physical timer" since they have lower abstraction level than conventional time management functions, and are closer to hardware layer.

Due to the above positioning, the physical timer functions are made to be as simple as possible and limited to a small specification, and are assumed to be realized by library functions which have small overhead. This policy is reflected in the specification of using the statically fixed physical timer numbers rather than dynamical ID numbers, and the specification of never performing the management of mapping with the requesting task or the requests from more than one task.

Physical timer functions standardize APIs that operate the timer (counter) device. However, the timer devices have direct relation with time related behaviors such as calling interrupt handler based on a small elapsed time, making such devises more closely connected with the kernel than other devices (storage and communication). For this reason, the physical timer is provided as more generic function by standardizing its specification as a part of the μT-Kernel/SM instead of standardizing it as part of device driver specification. Since the physical timer functions belong to the μT-Kernel/SM [Overall Note and Supplement] is applicable.

Hardware timer counter used as a physical timer is assumed to be 32-bit or less. Therefore, 32-bit UW is used for the data type that represents the count values or upper limits. In the future, 64-bit functions can be added.
5.8.1 Use Case of Physical Timer

Examples of effective use of physical timer functions are as follows:

(a) Example of processing to be realized
Assume that there are a cyclic processing X to be run every 2,500 microseconds and a cyclic processing Y to be run every 1,800 microseconds. Physical timers can achieve this efficiently.

(b) Implementation with physical timer functions
Two physical timers are used, and one is set to start a physical timer handler every 2,500 microseconds. For example, if the physical timer clock frequency is 10 MHz, as 1 clock corresponds to 0.1 microseconds (= 100 nanoseconds), set a physical timer upper limit (limit) to 24,999 (= 25,000 - 1) to make the physical timer handler start when the count value is changed from 24,999 to 0.
As this is a cyclic processing, mode of `StartPhysicalTimer` should be set to `TA_CYC_PTMR`.
Processing X is performed within this physical timer handler.
Similarly using another physical timer, the physical timer handler is set to start every 1,800 microseconds to perform the processing Y within this physical timer handler.
The timer interrupt interval (TTimPeriod) used by the μT-Kernel time management functions can be left at the default value (10 milliseconds) since it has no relationship with the physical timer functions.

(c) Implementation without physical timer functions
Instead of the physical timer handler, the μT-Kernel 3.0 system call (`tk_cre_cyc_u`) that can specify time in microseconds is used to define the cyclic handler that is invoked every 2,500 microseconds to perform the processing X within this cyclic handler. Similarly using another physical timer, a physical timer handler is invoked every 1,800 microseconds to perform the processing Y within this physical timer handler.
However, in this case, the timer interrupt interval used by μT-Kernel Time Management Function must be set with small enough interval so that the time of every 2,500 microseconds and every 1,800 microseconds can be processed precisely. Specifically, both processing every 2,500 microseconds and processing every 1,800 microseconds can be achieved with almost exact timing by using the timer interrupt interval of 100 microseconds which is a common divisor of 2,500 microseconds and 1,800 microseconds.

With the method (b) which uses the physical timer functions, the timer interrupt interval can be left as the default value (every 10 milliseconds) since the μT-Kernel time management functions are not used. Interrupts by the physical timer will occur every 2,500 and 1,800 microseconds, from which the physical timer handler is called to perform the processing X and Processing Y. No unnecessary interrupt related to timer will occur other than these.

On the other hand, for the method of (c) which does not use a physical timer, because the timer interrupt interval must be shortened, the overhead increases accordingly as the number of timer interrupts increases. For example, when comparing (b) and (c) in terms of the number of timer related interrupts that occur in 10 milliseconds period, (b) will have a total interrupt number of 10: 1 (= 10 milliseconds/10 milliseconds) for time management functions, 4 (= 10 milliseconds/2,500 microseconds) as physical timer interrupt for processing X, and 5 (= 10 milliseconds/1,800 microseconds) as physical timer interrupt for processing Y. For (c), timer interrupt number is 100 (10 milliseconds/100 microseconds) for time management functions. This is a trade-off situation with the accuracy of time. The smaller timer interval may be required depending on the difference between cycles or phases of processing X and processing Y, resulting in even larger overhead. In these cases, the physical timer functions are clearly effective.

However, the physical timer functions are highly effective only when the number of processings that depend on time is small and statically fixed, and enough number of hardware timers exist for them. Because the physical timer functions are, as its name shows, subject to the constraints of physical hardware resources, physical timer functions cannot be used effectively when the number of hardware timers is too small. Additionally, it will experience difficulty with the case where the number of time-dependent processings dynamically increases. In these cases, using the conventional time management functions such as the cyclic handler and alarm handler will achieve more flexible handling.
Though the application area of physical timer functions and time management functions in microseconds may overlap, they have different characteristics shown above. Therefore, it is recommended to use appropriate one depending on the hardware configuration and applications. The physical timer functions have been added for this reason.
5.8.2 StartPhysicalTimer - Start Physical Timer

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = StartPhysicalTimer(UINT ptmrno, UW limit, UINT mode);
```

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>ptmrno</td>
<td>Physical Timer Number</td>
</tr>
<tr>
<td>UW</td>
<td>limit</td>
<td>Limit</td>
</tr>
<tr>
<td>UINT</td>
<td>mode</td>
<td>Mode</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>ercd</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

- **E_OK**: Normal completion
- **E_PAR**: Parameter error (ptmrno, limit, or mode is invalid or cannot be used)

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- **TK_SUPPORT_PTIMER**: Support of physical timer function

Additionally, the following service profile items are related to this API.

- **TK_MAX_PTIMER**: Maximum number of physical timers

Description

Sets the count value of the physical timer specified by `ptmrno` to 0, and then starts counting. After this function is executed, the count value is incremented by one at a constant time interval that is the inverse of the timer clock frequency.

`limit` specifies the upper limit of the count value. When a time period equal to the inverse of the clock frequency has elapsed after the count value reaches the upper limit, the count value is reset to 0. At that timing, if a physical timer handler is defined for this physical timer, that handler will be started. The duration between when the counting is started by `StartPhysicalTimer` call and when the counter is reset to zero is \((\text{inverse of timer clock frequency}) \times (\text{upper limit} + 1)\).
If `limit` is set to 0, an E_PAR error will occur. `mode` specifies the following modes:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA_ALM_PTMR</td>
<td>0</td>
<td>The counting is stopped when the count value is reset to 0 from the upper limit value. Afterward, the count value remains as 0.</td>
</tr>
<tr>
<td>TA_CYC_PTMR</td>
<td>1</td>
<td>The count value starts to increase again, after it is reset to 0 from the upper limit value. Therefore, the cycle of increasing and resetting the count value repeats periodically.</td>
</tr>
</tbody>
</table>
5.8.3 StopPhysicalTimer - Stop Physical Timer

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = StopPhysicalTimer(UINT ptmrno);
```

Parameter

| UINT ptmrno | Physical Timer Number | Physical timer number |

Return Parameter

| ER ercd | Error Code |

Error Code

- E_OK: Normal completion
- E_PAR: Parameter error (ptmrno is invalid or cannot be used)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- TK_SUPPORT_PTIMER: Support of physical timer function

Additionally, the following service profile items are related to this API.

- TK_MAX_PTIMER: Maximum number of physical timers

Description

Stops the counting operation of the physical timer specified by ptmrno.

After executing this function, the last count value of the physical timer is retained. Therefore, if GetPhysicalTimerCount is executed after this function is executed, that function will return the physical timer count value just before this function is executed.

Executing this function for the physical timer that has already stopped counting does nothing. It does not generate any error.
Additional Notes

If the physical timer that is no longer used is kept running, it may not adversely affect the program operation, but clock signals will be used unnecessarily, which may not be desirable in terms of electric power saving. So, it is recommended to stop the physical timer no longer used by executing this function.

Use of this function is effective for the case TA_CYC_PTMR is specified for the physical timer and its use is ended. If TA_ALM_PTMR is specified as the mode, the physical timer automatically stopped counting after the count value is reset to 0 from the upper limit value, which results in the same state as that after this function being executed. In this case, it is not necessary to issue this function additionally. Issuing this function does not cause any problem, but nothing is changed.
5.8.4  GetPhysicalTimerCount - Get Physical Timer Count

C Language Interface

#include <tk/tkernel.h>

ER ercd = GetPhysicalTimerCount(UINT ptmrno, UW *p_count);

Parameter

<table>
<thead>
<tr>
<th>UINT</th>
<th>ptmrno</th>
<th>Physical Timer Number</th>
<th>Physical timer number</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW*</td>
<td>p_count</td>
<td>Pointer to Physical Timer Count</td>
<td>Pointer to the area to return the current physical timer count</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW</td>
<td>count</td>
<td>Physical Timer Count</td>
<td>Current count value</td>
</tr>
</tbody>
</table>

Error Code

- E_OK  Normal completion
- E_PAR Parameter error (ptmrno is invalid or cannot be used)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- TK_SUPPORT_PTIMER Support of physical timer function

Additionally, the following service profile items are related to this API.

- TK_MAX_PTIMER Maximum number of physical timers

Description

Gets the current count value of the physical timer specified by ptmrno, and returns it as the return parameter count.
5.8.5 DefinePhysicalTimerHandler - Define Physical Timer Handler

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = DefinePhysicalTimerHandler(UINT ptmrno, CONST T_DPTMR *pk_dptmr);
```

**Parameter**

<table>
<thead>
<tr>
<th>UINT</th>
<th>ptmrno</th>
<th>Physical Timer Number</th>
<th>Physical timer number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST T_DPTMR*</td>
<td>pk_dptmr</td>
<td>Packet to Define Physical Timer Handler</td>
<td>Packet to Define Physical Timer Handler definition</td>
</tr>
</tbody>
</table>

**pk_dptmr Detail**

| void* | exinf | Extended Information | Extended information |
| ATR       | ptmratr | Physical Timer Attribute | Physical timer handler attribute (TA_ASM || TA_HLNG) |
| FP        | ptmrhdr | Physical Timer Handler Address | Physical timer handler address |

**Return Parameter**

| ER | ercd | Error Code | Error code |

**Error Code**

- **E_OK** Normal completion
- **E_NOMEM** Insufficient memory (memory for control block cannot be allocated)
- **E_RSATR** Reserved attribute (ptmratr is invalid or cannot be used)
- **E_PAR** Parameter error (ptmrno, pk_dptmr, or ptmrhdr is invalid or cannot be used, or the physical timer handler for ptmrno cannot be defined)

**Valid Context**

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Related Service Profile Items**

Only when all the service profile items below are set to be effective, this API can be used.

| TK_SUPPORT_PTIMER | Support of physical timer function |

Additionally, the following service profile items are related to this API.

| TK_MAX_PTIMER | Maximum number of physical timers |
Description

If `pk_dptmr` is not `NULL`, this function defines the physical timer handler for the physical timer specified by `ptmrno`. The physical timer handler is a handler running as a task-independent portion, and is started when the physical timer count is reset to 0 from the upper limit value specified by `limit` of `StartPhysicalTimer`.

The programming format of physical timer handler is similar to that of cyclic handler or alarm handler. This means that if the `TA_HLNG` attribute is specified, the physical timer handler is started via a high-level language support routine and terminated by a return from the function. If the `TA_ASM` attribute is specified, the physical timer handler format is implementation-dependent. Regardless of which attribute is specified, `exinf` is passed as a startup parameter of physical timer handler.

If `pk_dptmr` is `NULL`, this function cancels the definition of the physical timer handler for the physical timer specified by `ptmrno`. The physical timer handlers for all the physical timers are undefined right after the system startup.

If the physical timer handler for the physical timer specified by `ptmrno` cannot be defined (if the `pk_rptmr->defhdr` in `GetPhysicalTimerConfig` returns `FALSE`), the E_PAR error occurs. If the physical timer specified by `ptmrno` does not exist or cannot be used, the E_PAR error also occurs.

Additional Notes

In a typical implementation, an interrupt handler to implement the function of physical timer is defined within μT-Kernel/SM, and is configured so that an interrupt to be raised when the physical timer counter value wraps around from the upper limit to zero. In this interrupt handler, the physical timer handler which is defined in this function is called as well as other processing for implementation of physical timer such as the support for `TA_ALM_PTMR` and `TA_CYC_PTMR`.
5.8.6 GetPhysicalTimerConfig - Get Physical Timer Configuration Information

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = GetPhysicalTimerConfig(UINT ptmrno, T_RPTMR *pk_rptmr);
```

Parameter

<table>
<thead>
<tr>
<th>UINT</th>
<th>ptmrno</th>
<th>Physical Timer Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RPTMR</td>
<td>pk_rptmr</td>
<td>Packet to Return Physical Timer Configuration Information</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
</table>

pk_rptmr Detail

<table>
<thead>
<tr>
<th>UW</th>
<th>ptmrclk</th>
<th>Physical Timer Clock Frequency</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>UW</th>
<th>maxcount</th>
<th>Maximum Count</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>BOOL</th>
<th>defhdr</th>
<th>Handler Support</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_PAR: Parameter error (ptmrno or pk_rptmr is invalid or cannot be used)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- TK_SUPPORT_PTIMER: Support of physical timer function

Additionally, the following service profile items are related to this API.

- TK_MAX_PTIMER: Maximum number of physical timers
Description

Gets the configuration information of the physical timer specified by ptmrno.

The retrievable configuration information includes the physical timer clock frequency ptmrclk, the maximum count value maxcount, and whether the support for physical timer handler exists defhdr.

ptmrclk indicates the clock frequency used to count up the target physical timer. If ptmrclk is set to 1, the clock is 1 Hz, and if it is set to MATH: $2^{32} - 1$, then the clock is MATH: $2^{32} - 1$ Hz (approximately 4 GHz). If the clock is long (less than 1 Hz), then ptmrclk is 0. If ptmrclk is other than 0, the physical timer count value is monotonically incremented by 1, from 0 to the upper limit value limit, at a constant time interval that is the inverse of ptmrclk.

maxcount is the maximum value that can be counted by the target physical timer, and also the maximum value that can be set as the upper limit value. Generally, maxcount is MATH: $2^{16} - 1$ for a 16-bit timer counter, and MATH: $2^{32} - 1$ for a 32-bit timer counter, but it may be other value depending on the hardware or system configuration.

If defhdr is TRUE, the physical timer handler, which is started when the target physical timer count reaches the upper limit value, can be defined. If defhdr is FALSE, the physical timer handler for this physical timer cannot be defined.

If the physical timer specified by ptmrno does not exist or cannot be used, the E_PAR error occurs. For the physical timer number, a positive integer value is assigned in ascending order, so if the system has N physical timers, the E_PAR error occurs when ptmrno is 0 or larger than N.

Additional Notes

As the substring “configuration” of this function name suggests, the values acquired by this function, ptmrclk, maxcount, and defhdr are assumed to be statically fixed by hardware or by the initialization done during the startup processing of the system, and are not expected to change during the subsequent execution of the system. Note, however, that there is a chance of adding dynamical reconfiguration feature to the core specification or implementation-defined feature: for example, changing the clock frequency of the physical timer. When such modifications are introduced, the information acquired by this function can be a value dynamically changed during the execution of the system. Such changes of use cases depend heavily on the operation methods or applications, and it was considered better to handle such differences in the upper library that use physical timer rather than in the base μT-Kernel specification. Hence, μT-Kernel specification does not define the possibility of dynamically changing nature of the configuration information acquired by this function. In a nutshell, whether the information acquired by this function may change during the execution of the system is implementation-dependent.
5.9 Utility Functions

Utility functions are used commonly from general programs such as applications, middleware, and device drivers on the μT-Kernel.

Utility functions are provided as library functions or C language macros.
5.9.1 Set Object Name

API for setting object name is provided as C language macros. It can be called from a task-independent portion and while task dispatching and interrupts are disabled.
5.9.1.1 SetOBJNAME - Set Object Name

C Language Interface

#include <tk/tkernel.h>

void SetOBJNAME(void *exinf, CONST UB *name);

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>exinf Extended Information Variable to set as extended information</td>
</tr>
<tr>
<td>CONST UB*</td>
<td>name Object Name Object name to be set</td>
</tr>
</tbody>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Interprets the ASCII string of four or less characters specified in name as a single 32-bit data to store it in exinf. This API is defined as a C language macro and exinf is not a pointer. Write a variable directly.

Additional Notes

This API can assign the ASCII string names (such as task name) to the kernel objects, and the names are stored in the extended information exinf of the kernel objects. It is possible to list the object names set by this API by printing the information in exinf as ASCII string using the debugger, etc. to investigate the state of the kernel objects.

Sample Usage of SetOBJNAME

```c
T_CTSK ctsk;
...
/* Set the object name "TEST" for the task ctsk */
SetOBJNAME(ctsk.exinf, "TEST");
task_id = tk_cre_tsk (&ctsk);
```
Note that you need to add '\0' which indicates the end of the string if you would like to manipulate the string by C language functions.

### 5.9.2 Fast Lock and Multi-lock Libraries

Fast lock and multi-lock libraries are for performing exclusion control faster between multiple tasks in the device drivers or subsystems. In order to perform the exclusion control, while semaphore or mutex can be used, fast lock is implemented as the μT-Kernel/SM library functions that processes the lock acquisition operation with specially higher speed when the task is not queued.

Fast lock and multi-lock libraries are for performing exclusion control quicker than semaphore and mutexes between multiple tasks in the device drivers or subsystems. Fast multi-lock is one object built by combining independent binary semaphores for mutual exclusion control. The number of binary semaphores is the number of the bits in UINT data type, and each binary semaphore is distinguished by the number from 0 to (bit width of UINT) - 1.

For example, when exclusion control is performed at ten locations, one fast multi-lock can be created and then the binary semaphores with lock numbers from 0 to 9 can be used to perform exclusion control while ten fast locks can be used. While using ten fast locks bring faster result, the total required resources is lower when the fast multi-lock is used.

**Additional Notes**

Fast lock function is implemented by using counters that show the lock states and a semaphore. Fast multi-lock function is implemented by using a counter that shows the lock states and event flags. When the invoking task is not queued at the lock acquisition, it performs faster than the usual semaphores or event flags because only counter operation is performed. On the other hand, when the invoking task is queued at lock acquisition, it is not necessarily faster than the usual semaphores or event flags because it uses usual semaphores and event flags to manage transitions to waiting state or queues. Fast lock and multi-lock are effective when possibility of being queued is low due to mutual exclusion control.
5.9.2.1 CreateLock - Create Fast Lock

C Language Interface

#include <tk/tkernel.h>

ER ercd = CreateLock(FastLock *lock, CONST UB *name);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastLock*</td>
<td>lock</td>
</tr>
<tr>
<td>CONST UB*</td>
<td>name</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>ercd</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_NOMEM</td>
<td>Insufficient memory (memory for control block cannot be allocated)</td>
</tr>
<tr>
<td>E_LIMIT</td>
<td>Number of fast locks exceeds the system limit</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Creates a fast lock. lock is a structure to control a fast lock. name is the name of the fast lock and can be NULL.

Fast lock is a binary semaphore used for mutual exclusion control and is implemented to be operated as fast as possible.
5.9.2.2 DeleteLock - Delete Fast Lock

C Language Interface

#include <tk/tkernel.h>

void DeleteLock(FastLock *lock);

Parameter

| FastLock* lock | Control Block of FastLock | Control block of fast lock |

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes a fast lock.

Error detection is omitted for faster operation.
5.9.2.3 Lock - Lock Fast Lock

C Language Interface

#include <tk/tkernel.h>

void Lock(FastLock *lock);

Parameter

| FastLock* lock | Control Block of FastLock | Control block of fast lock |

Return Parameter
None.

Error Codes
None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items
None.

Description

Locks a fast lock.

If the lock is already locked, the invoking task goes to the waiting state and is put in the task queue until it is unlocked. Tasks are queued in the priority order.

Error detection is omitted for faster operation.
5.9.2.4 Unlock - Unlock Fast Lock

C Language Interface
#include <tk/tkernel.h>

void Unlock(FastLock *lock);

Parameter

<table>
<thead>
<tr>
<th>FastLock*</th>
<th>lock</th>
<th>Control Block of FastLock</th>
<th>Control block of fast lock</th>
</tr>
</thead>
</table>

Return Parameter

None.

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Unlocks a fast lock.
If there are tasks waiting for the fast lock, the first task in the task queue newly acquires the lock.
Error detection is omitted for faster operation.
5.9.2.5 CreateMLock - Create Fast Multi-lock

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = CreateMLock(FastMLock *lock, CONST UB *name);
```

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastMLock* lock</td>
<td>Control Block of FastMLock</td>
</tr>
<tr>
<td>CONST UB* name</td>
<td>Name of FastMLock</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER ercd</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

- **E_OK**: Normal completion
- **E_NOMEM**: Insufficient memory (memory for control block cannot be allocated)
- **E_LIMIT**: Number of fast multi-locks exceeds the system limit

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Creates a fast multi-lock.

- **lock** is a structure to control a fast multi-lock. **name** is the name of the fast multi-lock and can be **NULL**.

Fast multi-lock is one object built by combining independent binary semaphores for mutual exclusion control, and is implemented for very fast execution. The number of binary semaphores is the number of the bits in UINT data type, and each binary semaphore is distinguished by the number from 0 to (bit width of UINT data type) - 1. For example, if UINT is 16 bits, a number from 0 to 15 can be used as lock number.

Porting Guideline

Be warned that the number of available lock numbers is now dependent on the bit width of UINT data type. For example, the number of binary semaphores can take the value from 0 to 15 in 16-bit environment.
5.9.2.6 DeleteMLock - Delete Fast Multi-lock

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = DeleteMLock(FastMLock *lock);
```

Parameter

| FastMLock* lock | Control Block of FastMLock | Control block of fast multi-lock |

Return Parameter

| ER ercd | Error Code | Error code |

Error Code

- E_OK Normal completion
- E_PAR Parameter error

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Deletes a fast multi-lock.
5.9.2.7 MLock - Lock Fast Multi-lock

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = MLock(FastMLock *lock, INT no);
```

Parameter

| FastMLock* lock | Control Block of FastMLock Control block of fast multi-lock |
| INT no | Lock Number | Lock number |

Return Parameter

| ER ercd | Error Code |
| Error code |

Error Code

- **E_OK**: Normal completion
- **E_PAR**: Parameter error
- **E_DLT**: Waiting object was deleted
- **E_RLWAI**: Waiting state was forcibly released
- **E_CTX**: Context error

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Locks a fast multi-lock.

- `no` is the lock number and is from 0 to (the bit width of UINT data type) - 1. For example, if UINT is 16 bits, a number from 0 to 15 can be used as lock number.

If the lock is already locked with the same lock number, the invoking task goes to the waiting state and is put in the task queue until it is unlocked with the same lock number. Tasks are queued in the priority order.

Porting Guideline

Be warned that the number of available lock numbers is now dependent on the bit width of UINT data type. For example, the number of binary semaphores can take the value from 0 to 15 in 16-bit environment.
5.9.2.8 MLockTmo - Lock Fast Multi-lock (with Timeout)

C Language Interface

#include <tk/tkernel.h>

ER ercd = MLockTmo(FastMLock *lock, INT no, TMO tmout);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastMLock* lock</td>
<td>Control Block of FastMLock</td>
</tr>
<tr>
<td>INT no</td>
<td>Lock Number</td>
</tr>
<tr>
<td>TMO tmout</td>
<td>Timeout</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER ercd</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Parameter error</td>
</tr>
<tr>
<td>E_DLT</td>
<td>Waiting object was deleted</td>
</tr>
<tr>
<td>E_RLWAI</td>
<td>Waiting state was forcibly released</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>Timeout</td>
</tr>
<tr>
<td>E_CTX</td>
<td>Context error</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Porting Guideline</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Locks a fast multi-lock with timeout.

This API is identical to MLock, except that it can specify the timeout interval in tmout. If the lock cannot be acquired before the timeout interval specified in tmout has elapsed, E_TMOUT is returned.

Porting Guideline

Be warned that the number of available lock numbers is now dependent on the bit width of UINT data type. For example, the number of binary semaphores can take the value from 0 to 15 in 16-bit environment.
5.9.2.9 MLockTmo_u - Lock Fast Multi-lock (with Timeout, Microseconds)

C Language Interface

```c
#include <tk/tkernel.h>

ER ercd = MLockTmo_u(FastMLock *lock, INT no, TMO_U tmout_u);
```

Parameter

- `FastMLock* lock`: Control Block of FastMLock
- `INT no`: Lock Number
- `TMO_U tmout_u`: Timeout (in microseconds)

Return Parameter

- `ER ercd`: Error Code

Error Code

- E_OK: Normal completion
- E_PAR: Parameter error
- E_DLT: Waiting object was deleted
- E_RLWAI: Waiting state was forcibly released
- E_TMOUT: Timeout
- E_CTX: Context error

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this API can be used.

- `TK_SUPPORT_USEC`: Support of microsecond

Description

Locks a fast multi-lock with timeout in microseconds.

This API is identical to `MLockTmo`, except that the timeout interval is specified with a 64-bit value in microseconds.

Porting Guideline

Be warned that the number of available lock numbers is now dependent on the bit width of UINT data type. For example, the number of binary semaphores can take the value from 0 to 15 in 16-bit environment.
5.9.2.10 MUnlock - Unlock Fast Multi-lock

C Language Interface

```c
#include <tk/kernel.h>

ER ercd = MUnlock(FastMLock *lock, INT no);
```

Parameter

<table>
<thead>
<tr>
<th>FastMLock* lock</th>
<th>Control Block of FastMLock</th>
<th>Control block of fast multi-lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT no</td>
<td>Lock Number</td>
<td>Lock number</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

- **E_OK**: Normal completion

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Related Service Profile Items

None.

Description

Unlocks a fast multi-lock.

- `no` is the lock number and is from 0 to (the bit width of UINT data type) - 1. For example, if UINT is 16 bits, a number from 0 to 15 can be used as lock number.

- If there are tasks in the waiting state for the same lock number, the first task in the task queue newly acquires the lock.

Porting Guideline

Be warned that the number of available lock numbers is now dependent on the bit width of UINT data type. For example, the number of binary semaphores can take the value from 0 to 15 in 16-bit environment.
Chapter 6

μT-Kernel/DS Functions

This chapter describes details of the functions provided by μT-Kernel/DS (Debugger Support).

μT-Kernel/DS provides functions enabling a debugger to reference μT-Kernel internal states and run a trace. The functions provided by μT-Kernel/DS are only for debugger use and not for use by applications or other programs.

Overall Note and Supplement

• Except where otherwise noted, μT-Kernel/DS system calls (td_...) can be called from a task independent portion and while dispatching and interrupts are disabled. There may be some limitations, however, imposed by particular implementations.

• When μT-Kernel/DS system calls (td_...) are invoked in interrupts disabled state, they are processed without enabling interrupts. Other kernel states likewise remain unchanged during this processing. Changes in kernel states may occur if a service call is invoked while interrupts or dispatching are enabled, since the kernel continues operating.

• μT-Kernel/DS system calls (td_...) cannot be invoked from a lower protection level than that at which μT-Kernel/OS system calls can be invoked (lower than TSVCLimit(E_OACV)).

• Error codes such as E_PAR, E_MACV, and E_CTX that can be returned in many situations are not described here always unless there is some special reason for doing so.
6.1 Kernel Internal State Acquisition Functions

Kernel internal state reference functions are functions for enabling a debugger to get T-Kernel internal states. They include functions for getting a list of objects, getting task precedence, getting the order in which tasks are queued, getting the status of objects, system, and task registers, and getting time.
6.1.1 td_lst_tsk - Reference Task ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_tsk(ID list[], INT nent);

Parameter

| ID list[] | List Location of task ID list |
| INT nent | Number of List Entries Maximum number of entries in list |

Return Parameter

| INT ct | Count Number of used tasks |
| or Error Code |
| Error code |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Gets the list of the IDs of the currently used tasks, and puts in list up to nent IDs. The number of the used tasks is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.2  td_lst_sem - Reference Semaphore ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_sem(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>list[]</th>
<th>List Location of semaphore ID list</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

| INT     | ct       | Count Number of used semaphores |
|         |          | or Error Code Error code        |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT        Support of μT-Kernel/DS

Description

Gets the list of the IDs of the currently used semaphores, and puts in list up to nent IDs. The number of the used semaphores is passed in the return code. If return code > nent, this means not all semaphore IDs could be retrieved.
6.1.3  td_lst_flg - Reference Event Flag ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_flg(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>List</th>
<th>Location of event flag ID list</th>
</tr>
</thead>
<tbody>
<tr>
<td>list[]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Maximum number of entries in list</td>
</tr>
<tr>
<td></td>
<td>Number of List Entries</td>
<td></td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>Count or Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Number of used event flags</td>
</tr>
<tr>
<td>or</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of \(\mu\) T-Kernel/DS

Description

Gets the list of the IDs of the currently used event flags, and puts in list up to nent IDs. The number of the used event flags is passed in the return code. If return code > nent, this means not all event flag IDs could be retrieved.
6.1.4 td_lst_mbx - Reference Mailbox ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_mbx(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>list[]</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
</tr>
</tbody>
</table>

Location of mailbox ID list

Maximum number of entries in list

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>Error Code</td>
<td></td>
</tr>
</tbody>
</table>

Number of used mailboxes

Error code

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Gets the list of the IDs of the currently used mailboxes, and puts in list up to nent IDs. The number of the used mailboxes is passed in the return code. If return code > nent, this means not all mailbox IDs could be retrieved.
6.1.5  td_lst_mtx - Reference Mutex ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_mtx(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>List Location of mutex ID list</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INT</th>
<th>Number of List Entries</th>
</tr>
</thead>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>Count Number of used mutexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>Error Code Error code</td>
</tr>
</tbody>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Gets the list of the IDs of the currently used mutexes, and puts in list up to nent IDs. The number of the used mutexes is passed in the return code. If return code > nent, this means not all mutex IDs could be retrieved.
6.1.6   td_lst_mbf - Reference Message Buffer ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_mbf(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>list[]</th>
<th>List Location of message buffer ID list</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count Number of used message buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>or Error Code</td>
</tr>
</tbody>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TPK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

Gets the list of the IDs of the currently used message buffers, and puts in list up to nent IDs. The number of the used message buffers is passed in the return code. If return code > nent, this means not all message buffer IDs could be retrieved.
6.1.7  td_lst_mpf - Reference Fixed-size Memory Pool ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_mpf(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>list[]</th>
<th>List</th>
<th>Location of fixed-size memory pool ID list</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
<th>Number of used fixed-size memory pools</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td></td>
<td></td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

Gets the list of the IDs of the currently used fixed-size memory pools, and puts in list up to nent IDs. The number of the used fixed-size memory pools is passed in the return code. If return code > nent, this means not all fixed-size memory pool IDs could be retrieved.
6.1.8  td_lst_mpl - Reference Variable-size Memory Pool ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_mpl(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>List</th>
<th>Location of variable-size memory pool ID list</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>ct</th>
<th>Count</th>
<th>Number of used variable-size memory pools</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td></td>
<td></td>
<td>Error code</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td>Error Code</td>
<td></td>
</tr>
</tbody>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μ T-Kernel/DS

Description

Gets the list of the IDs of the currently used variable-size memory pools, and puts in list up to nent IDs. The number of the used variable-size memory pools is passed in the return code. If return code > nent, this means not all variable-size memory pool IDs could be retrieved.
6.1.9  td_lst_cyc - Reference Cyclic Handler ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_cyc(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID list[]</td>
<td>List Location of cyclic handler ID list</td>
</tr>
<tr>
<td>INT nent</td>
<td>Number of List Entries Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT ct</td>
<td>Count Number of used cyclic handlers or Error Code</td>
</tr>
</tbody>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Context Type</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT       Support of μT-Kernel/DS

Description

Gets the list of the IDs of the currently used cyclic handlers, and puts in list up to nent IDs. The number of the used cyclic handlers is passed in the return code. If return code > nent, this means not all cyclic handler IDs could be retrieved.
6.1.10  td_lst_alm - Reference Alarm Handler ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_alm(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID list[]</th>
<th>List Location of alarm handler ID list</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT nent</td>
<td>Number of List Entries Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

| INT ct | Count Number of used alarm handlers |
| or Error Code | Error code |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

Gets the list of the IDs of the currently used alarm handlers, and puts in list up to nent IDs. The number of the used alarm handlers is passed in the return code. If return code > nent, this means not all alarm handler IDs could be retrieved.
6.1.11 td_lst_ssy - Reference Subsystem ID List

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_lst_ssy(ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID list[]</td>
<td>List Location of subsystem ID list</td>
</tr>
<tr>
<td>INT nent</td>
<td>Number of List Entries Maxmum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT ct</td>
<td>Count Number of used subsystems</td>
</tr>
<tr>
<td>or Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Context Type</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS
TK_SUPPORT_SUBSYSTEM Support of subsystem management functions

Description

 Gets the list of the IDs of the currently used subsystems, and puts in list up to nent IDs. The number of the used subsystems is passed in the return code. If return code > nent, this means not all subsystem IDs could be retrieved.
6.1.12  td_rdy_que - Reference Task Precedence

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_rdy_que(PRI pri, ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>PRI</th>
<th>pri</th>
<th>Task Priority</th>
<th>Task priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>list[]</td>
<td>Task ID List</td>
<td>Location of task ID list</td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
<th>Number of tasks with priority pri in a run state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>or Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

E_PAR Parameter error (pri is invalid or cannot be used)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Gets a list of IDs of the tasks in a run state (READY state or RUNNING state) whose task priority is pri, arranged in the order from the highest to the lowest precedence.

This function stores in list up to nent task IDs, arranged in the order of precedence starting from the highest-precedence task ID at the head of the list.

The number of tasks in a run state with priority pri is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.13  td_sem_que - Reference Semaphore Queue

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_sem_que(ID semid, ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>semid</th>
<th>Semaphore ID</th>
<th>Target semaphore ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>list[]</td>
<td>Task ID List</td>
<td>Location of waiting task IDs</td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
<th>Number of waiting tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>or</td>
<td>Error Code Error code</td>
</tr>
</tbody>
</table>

Error Code

- 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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Gets the list of the IDs of the queued tasks waiting for a semaphore specified in semid. This function stores in list up to nent task IDs, arranged in the order in which tasks are queued, starting from the first task in the semaphore queue. The number of the tasks in the semaphore queue is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.14  td_flg_que - Reference Event Flag Queue

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_flg_que(ID flgid, ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>flgid</th>
<th>EventFlag ID</th>
<th>Target event flag ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>list[]</td>
<td>Task ID List</td>
<td>Location of waiting task IDs</td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
<th>Number of waiting tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>Error Code</td>
<td>Error code</td>
<td></td>
</tr>
</tbody>
</table>

Error Code

- 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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

Gets the list of the IDs of the queued tasks waiting for an event flag specified in flgid. This function stores in list up to nent task IDs, arranged in the order in which tasks are queued, starting from the first task in the event flag queue. The number of the tasks in the event flag queue is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.15  td_mbx_que - Reference Mailbox Queue

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_mbx_que(ID mbxid, ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbxid</th>
<th>Mailbox ID</th>
<th>Target mailbox ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>list[]</td>
<td>Task ID List</td>
<td>Location of waiting task IDs</td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
<th>Number of waiting tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>or</td>
<td>Error Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

- 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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

Gets the list of the IDs of the queued tasks waiting for a mailbox specified in mbxid. This function stores in list up to nent task IDs, arranged in the order in which tasks are queued, starting from the first task in the mailbox queue. The number of the tasks in the mailbox queue is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.16  td_mtx_que - Reference Mutex Queue

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_mtx_que(ID mtxid, ID list[], INT nent);

Parameter

| ID  | mtxid | Mutex ID       | Target mutex ID          |
| ID  | list[]| Task ID List   | Location of waiting task IDs |
| INT | nent  | Number of List Entries | Maximum number of entries in list |

Return Parameter

| INT | ct | Count | Number of waiting tasks |
|     | or | Error Code | Error code |

Error Code

| 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) |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

| TK_SUPPORT_DBGSPT | Support of μT-Kernel/DS |

Description

Gets the list of the IDs of the queued tasks waiting for a mutex specified in mtxid. This function stores in list up to nent task IDs, arranged in the order in which tasks are queued, starting from the first task in the mutex queue. The number of the tasks in the mutex queue is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.17  td_smbf_que - Reference Message Buffer Send Queue

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_smbf_que(ID mbfid, ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbfid</th>
<th>Message Buffer ID</th>
<th>Target message buffer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>list[]</td>
<td>Task ID List</td>
<td>Location of waiting task IDs</td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
<th>Number of waiting tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>Error Code</td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_ID</th>
<th>Invalid ID number (mbfid is invalid or cannot be used)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the message buffer specified in mbfid does not exist)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

Gets the list of the IDs of the queued tasks waiting for sending a message to a message buffer specified in mbfid. This function stores in list up to nent task IDs, arranged in the order in which tasks are queued, starting from the first task in the message buffer send queue. The number of the tasks in the message buffer send queue is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.18  td_rmbf_que - Reference Message Buffer Receive Queue

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_rmbf_que(ID mbfid, ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbfid</th>
<th>Message Buffer ID</th>
<th>Target message buffer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>list[]</td>
<td>Task ID List</td>
<td>Location of waiting task IDs</td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
<th>Number of waiting tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>Error Code</td>
<td>Error code</td>
<td></td>
</tr>
</tbody>
</table>

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)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Gets the list of the IDs of the queued tasks waiting for receiving a message from a message buffer specified in mbfid. This function stores in list up to nent task IDs, arranged in the order in which tasks are queued, starting from the first task in the message buffer receive queue. The number of the tasks in the message buffer receive queue is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.19  td_mpf_que - Reference Fixed-size Memory Pool Queue

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_mpf_que(ID mpfid, ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mpfid</th>
<th>Memory Pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>list[]</td>
<td>Task ID List</td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
</tr>
</tbody>
</table>

Target fixed-size memory pool ID
Location of waiting task IDs
Maximum number of entries in list

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>or Error Code</td>
</tr>
</tbody>
</table>

Number of waiting tasks
Error code

Error Code

E_ID  Invalid ID number (mpfid is invalid or cannot be used)
E_NOEXS  Object does not exist (the fixed-size memory pool specified in mpfid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

Gets the list of the IDs of the queued tasks waiting for allocation in a fixed-size memory pool specified in mpfid. This function stores in list up to nent task IDs, arranged in the order in which tasks are queued, starting from the first task in the fixed-size memory pool queue. The number of the tasks in the fixed-size memory pool queue is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.20  td_mpl_que - Reference Variable-size Memory Pool Queue

C Language Interface

#include <tk/dbgspt.h>

INT ct = td_mpl_que(ID mplid, ID list[], INT nent);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mplid</th>
<th>Memory Pool ID</th>
<th>Target variable-size memory pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>list[]</td>
<td>Task ID List</td>
<td>Location of waiting task IDs</td>
</tr>
<tr>
<td>INT</td>
<td>nent</td>
<td>Number of List Entries</td>
<td>Maximum number of entries in list</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>INT</th>
<th>ct</th>
<th>Count</th>
<th>Number of waiting tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>or</td>
<td>Error Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

E_ID  Invalid ID number (mplid is invalid or cannot be used)
E_NOEXS  Object does not exist (the variable-size memory pool specified in mplid does not exist)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

Gets the list of the IDs of the queued tasks waiting for allocation in a variable-size memory pool specified in mplid. This function stores in list up to nent task IDs, arranged in the order in which tasks are queued, starting from the first task in the variable-size memory pool queue. The number of the tasks in the variable-size memory pool queue is passed in the return code. If return code > nent, this means not all task IDs could be retrieved.
6.1.21  td_ref_tsk - Reference Task Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_tsk(ID tskid, TD_RTSK *rtsk);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Target task ID (TSK_SELF can be specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RTSK</td>
<td>rtsk</td>
<td>Packet to Return Task Status</td>
<td>Pointer to the area to return the task status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

rtsk Detail:

void* exinf | Extended Information | Extended information |
PRI tskpri  | Task Priority        | Current priority     |
PRI tskbpri | Task Base Priority   | Base priority        |
UINT tskstat| Task State           | Task States          |
UW tskwait  | Task Wait Factor     | Wait factor          |
ID wid      | Waiting Object ID    | Waiting object ID    |
INT wupcnt  | Wakeup Count         | Wakeup request queuing count |
INT suscnt  | Suspend Count        | Suspend request nesting count |
UW waitmask | Wait Mask            | Disabled wait factors |
UINT texmask| Task Exception Mask  | Allowed task exceptions |
UINT tskevent| Task Event          | Raised task event    |
FP task     | Task Start Address   | Task start address   |
SZ stksz    | User Stack Size      | User stack size (in bytes) |
SZ sstksz   | System Stack Size    | System stack size (in bytes) |
void* istack| Initial User Stack Pointer | User stack pointer initial value |
void* isstack| Initial System Stack Pointer | System stack pointer initial value |

Error Code

E_OK Normal completion
E_ID Bad identifier
E_NOEXS Object does not exist

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

\[
\begin{align*}
\text{TK\_SUPPORT\_DBGSPT} & \quad \text{Support of } \mu\text{T-Kernel/DS} \\
\text{TK\_SUPPORT\_DISWAI} & \quad \text{Information about disabled wait factors (}\text{waitmask}\text{) is obtainable} \\
\text{TK\_SUPPORT\_TASKEXCEPTION} & \quad \text{Task exception information (}\text{texmask}\text{) can be acquired.} \\
\text{TK\_SUPPORT\_TASKEVENT} & \quad \text{Generated task event(}\text{tskevent}\text{) can be acquired} \\
\text{TK\_HAS\_SYSSTACK} & \quad \text{Task can have a system stack independent of user-stack, and} \\
& \quad \text{information can be acquired of the system stack as well as user stack(}\text{sstksz} \text{ and } \text{isstack})
\end{align*}
\]

Description

Gets the state of the task designated in tskid. This function is similar to \texttt{tk\_ref\_tsk}, with the task start address and stack information added to the state information obtained.

The stack area extends from the stack pointer initial value toward the low addresses for the number of bytes designated as the stack size.

- \[\text{nstksz} \leq \text{user stack area} < \text{nstksz}\]
- \[\text{issksz} \leq \text{system stack area} < \text{issksz}\]

Note that the stack pointer initial value (\text{nstksz}, \text{issksz}) is not the same as its current position. The stack area may be used even before a task is started. Calling \texttt{td\_get\_reg} gets the stack pointer current position.
6.1.22  td_ref_tex - Reference Task Exception Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_tex(ID tskid, TD_RTEX *pk_rtex);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Target task ID (TSK_SELF can be specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RTEX*</td>
<td>pk_rtex</td>
<td>Packet to Return Task Exception Status</td>
<td>Pointer to the area to return the task exception status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

pk_rtex Detail:

<table>
<thead>
<tr>
<th>UINT</th>
<th>pendtx</th>
<th>Pending Task Exception</th>
<th>Pending task exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>texmask</td>
<td>Task Exception Mask</td>
<td>Allowed task exceptions</td>
</tr>
</tbody>
</table>

Error Code

E_OK  Normal completion
E_ID  Bad identifier
E_NOEXS Object does not exist

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS
TK_SUPPORT_TASKEXCEPTION Support of task exception handling functions

Description

Gets the task exception status. This is similar to tk_ref_tex.
6.1.23  td_ref_sem - Reference Semaphore Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_sem(ID semid, TD_RSEM *rsem);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>semid</th>
<th>Semaphore ID</th>
<th>Target semaphore ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>rsem</td>
<td>Packet to Return Semaphore Status</td>
<td>Pointer to the area to return the semaphore status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
<td>Extended information</td>
</tr>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Waiting Task ID</td>
<td>Waiting task ID</td>
</tr>
<tr>
<td>INT</td>
<td>semcnt</td>
<td>Semaphore Count</td>
<td>Current semaphore resource count</td>
</tr>
</tbody>
</table>

Error Code

E_OK     Normal completion
E_ID     Bad identifier
E_NOEXS  Object does not exist

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORTDBGSPRT  Support of μT-Kernel/DS

Description

References the semaphore status. This is similar to tk_ref_sem.
6.1.24  td_ref_flg - Reference Event Flag Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_flg(ID flgid, TD_RFLG *rflg);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>flgid</th>
<th>EventFlag ID</th>
<th>Target event flag ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RFLG*</td>
<td>rflg</td>
<td>Packet to Return EventFlag Status</td>
<td>Pointer to the area to return the event flag status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

rflg Detail:

| void* | exinf | Extended Information | Extended information |
| ID    | wtsk  | Waiting Task ID      | Waiting task ID      |
| UINT  | flgptn | EventFlag Bit Pattern | The current event flag bit pattern |

Error Code

- E_OK Normal completion
- E_ID Bad identifier
- E_NOEXS Object does not exist

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

References the event flag status. This is similar to tk_ref_flg.
6.1.25  td_ref_mbx - Reference Mailbox Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_mbx(ID mbxid, TD_RMBX*rmbx);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbxid</th>
<th>Mailbox ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RMBX*</td>
<td>rmbx</td>
<td>Packet to Return Mailbox Status</td>
</tr>
<tr>
<td>Target mailbox ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pointer to the area to return the mailbox status</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
</tr>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Waiting Task ID</td>
</tr>
<tr>
<td>T_MSG*</td>
<td>pk_msg</td>
<td>Packet of Message</td>
</tr>
<tr>
<td>Extended information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Waiting task ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Next message to be received</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Error Code

| E_OK | Normal completion |
| E_ID | Bad identifier |
| E_NOEXS | Object does not exist |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPRT  Support of μT-Kernel/DS

Description

References the mailbox status. This is similar to tk_ref_mbx.
6.1.26  td_ref_mtx - Refer Mutex Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_mtx(ID mtxid, TD_RMTX *rmtx);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Mutex ID</td>
</tr>
<tr>
<td>mtxid</td>
<td>Target mutex ID</td>
</tr>
<tr>
<td>TD_RMTX*</td>
<td>Packet to Return Mutex Status</td>
</tr>
<tr>
<td>rmtx</td>
<td>Pointer to the area to return the mutex status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>Error Code</td>
</tr>
<tr>
<td>ercd</td>
<td>Error code</td>
</tr>
</tbody>
</table>

rmtx Detail:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void*</td>
<td>Extended Information</td>
</tr>
<tr>
<td>exinf</td>
<td>ID of task locking the mutex</td>
</tr>
<tr>
<td>ID</td>
<td>Locking Task ID</td>
</tr>
<tr>
<td>htsk</td>
<td>ID of tasks waiting to lock the mutex</td>
</tr>
<tr>
<td>ID</td>
<td>Lock Waiting Task ID</td>
</tr>
<tr>
<td>wtsk</td>
<td></td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Bad identifier</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Portion</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

References the mutex status. This is similar to tk_ref_mtx.
6.1.27 td_ref_mbf - Reference Message Buffer Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_mbf(ID mbfid, TD_RMBF *rmbf);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mbfid</th>
<th>Message Buffer ID</th>
<th>Target message buffer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RMBF*</td>
<td>rmbf</td>
<td>Packet to Return Message Buffer Status</td>
<td>Pointer to the area to return the message buffer status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

rmbf Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>wtsk</td>
<td>Waiting Task ID</td>
<td>Receive waiting task ID</td>
</tr>
<tr>
<td>ID</td>
<td>stsk</td>
<td>Send Waiting Task ID</td>
<td>Send waiting task ID</td>
</tr>
<tr>
<td>INT</td>
<td>msgsz</td>
<td>Message Size</td>
<td>Size of the next message to be received (in bytes)</td>
</tr>
<tr>
<td>SZ</td>
<td>frbufsz</td>
<td>Free Buffer Size</td>
<td>Free buffer size (in bytes)</td>
</tr>
<tr>
<td>INT</td>
<td>maxmsz</td>
<td>Maximum Message Size</td>
<td>Maximum message size (in bytes)</td>
</tr>
</tbody>
</table>

Error Code

E_OK Normal completion
E_ID Bad identifier
E_NOEXS Object does not exist

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

References the message buffer status. This is similar to tk_ref_mbf.
6.1.28  td_ref_mpf - Reference Fixed-size Memory Pool Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_mpf(ID mpfid, TD_RMPF *rmpf);

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Memory Pool ID</td>
</tr>
<tr>
<td>TD_RMPF</td>
<td>Packet to Return Memory Pool Status</td>
</tr>
<tr>
<td>mpfid</td>
<td>Target fixed-size memory pool ID</td>
</tr>
<tr>
<td>rmpf</td>
<td>Pointer to the area to return the memory pool status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>Error Code</td>
</tr>
<tr>
<td>ercd</td>
<td>Error code</td>
</tr>
</tbody>
</table>

rmpf Detail:

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wtsk</td>
<td>Waiting Task ID</td>
</tr>
<tr>
<td>frbcnt</td>
<td>Free Block Count</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Bad identifier</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Portion</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task portion</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quasi-task portion</td>
<td></td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Task-independent portion</td>
<td></td>
<td></td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

| TK_SUPPORT_DBGSPT | Support of \(\mu\)-T-Kernel/DS |

Description

References the fixed-size memory pool status. This is similar to tk_ref_mpf.
6.1.29  td_ref_mpl - Reference Variable-size Memory Pool Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_mpl(ID mplid, TD_RMPL *rmpl);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>mplid</th>
<th>Memory Pool ID</th>
<th>Target variable-size memory pool ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RMP</td>
<td>r mpl</td>
<td>Packet to Return Memory Pool Status</td>
<td>Pointer to the area to return the memory pool status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

rmpl Detail:

<table>
<thead>
<tr>
<th>ID</th>
<th>exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>w tsk</td>
<td>Waiting Task ID</td>
<td>Waiting task ID</td>
</tr>
<tr>
<td>SZ</td>
<td>frsz</td>
<td>Free Memory Size</td>
<td>Free memory size (in bytes)</td>
</tr>
<tr>
<td>SZ</td>
<td>maxsz</td>
<td>Max Memory Size</td>
<td>Maximum memory space size (in bytes)</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>Bad identifier</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSP

Description

References the variable-size memory pool status. This is similar to tk_ref_mpl.
6.1.30 td_ref_cyc - Reference Cyclic Handler Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_cyc(ID cycid, TD_RCYC *rcyc);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>cycid</th>
<th>Cyclic Handler ID</th>
<th>Target cyclic handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RCYC*</td>
<td>rcyc</td>
<td>Packet to Return Cyclic Handler Status</td>
<td>Pointer to the area to return the cyclic handler status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

rcyc Detail:

<table>
<thead>
<tr>
<th>type</th>
<th>exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELTIM</td>
<td>lfttim</td>
<td>Left Time</td>
<td>Time remaining until the next handler starts (ms)</td>
</tr>
<tr>
<td>UINT</td>
<td>cycstat</td>
<td>Cyclic Handler Status</td>
<td>Cyclic handler activation state</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_ID</td>
<td>Bad identifier</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th></th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORTDBGSPTR Support of μT-Kernel/DS

Description

References the cyclic handler status. This is similar to tk_ref_cyc.

The time remaining lfttim returned in the cyclic handler status information (TD_RCYC) obtained by td_ref_cyc is a value rounded to milliseconds. To know the value in microseconds, call td_ref_cyc_u.
6.1.31 td_ref_cyc_u - Reference Cyclic Handler Status (Microseconds)

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_cyc_u(ID cycid, TD_RNCYC_U *rcyc_u);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>cycid</th>
<th>Cyclic Handler ID</th>
<th>Target cyclic handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RNCYC_U</td>
<td>rcyc_u</td>
<td>Packet to Return Cyclic Handler Status</td>
<td>Pointer to the area to return the cyclic handler status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

rcyc_u Detail:

| void* | exinf | Extended Information | Extended information |
| RELTIM_U | lfttim_u | Left Time | Time remaining until the next handler starts (in microseconds) |
| UINT | cycstat | Cyclic Handler Status | Cyclic handler activation state |

Error Code

| E_OK       | Normal completion     |
| E_ID       | Bad identifier        |
| E_NOEXS    | Object does not exist |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

| TK_SUPPORT_DBGSPRT | Support of μT-Kernel/DS |
| TK_SUPPORT_USEC   | Support of microsecond |

Description

This system call takes 64-bit lfttim_u in microseconds instead of the return parameter lfttim of td_ref_cyc. The specification of this system call is same as that of td_ref_cyc, except that the return parameter is replaced with lfttim_u. For more details, see the description of td_ref_cyc.
6.1.32  td_ref_alm - Reference Alarm Handler Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_alm(ID almid, TD_RALM *ralm);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>almid</th>
<th>Alarm Handler ID</th>
<th>Target alarm handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RALM*</td>
<td>ralm</td>
<td>Packet to Return Alarm Handler Status</td>
<td>Pointer to the area to return the alarm handler status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ralm</td>
<td>Detail:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void*</td>
<td>exinf</td>
<td>Extended Information</td>
<td>Extended information</td>
</tr>
<tr>
<td>RELTIM</td>
<td>lfttim</td>
<td>Left Time</td>
<td>Time remaining until the handler starts (ms)</td>
</tr>
<tr>
<td>UINT</td>
<td>almstat</td>
<td>Alarm Handler Status</td>
<td>Alarm handler activation state</td>
</tr>
</tbody>
</table>

Error Code

| E_OK | Normal completion |
| E_ID | Bad identifier |
| E_NOEXS | Object does not exist |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS

Description

References the alarm handler status. This is similar to tk_ref_alm.

The time remaining lfttim returned in the alarm handler status information (TD_RALM) obtained by td_ref_alm is a value rounded to milliseconds. To know the value in microseconds, call td_ref_alm_u.
6.1.33  td_ref alm_u - Reference Alarm Handler Status (Microseconds)

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref alm_u(ID almid, TD_RALM_U *ralm_u);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>almid</th>
<th>Alarm Handler ID</th>
<th>Target alarm handler ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RALM_U</td>
<td>ralm_u</td>
<td>Packet to Return Alarm Handler Status</td>
<td>Pointer to the area to return the alarm handler status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

ralm_u Detail:

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Extended Information</th>
<th>Extended information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELTIM_U</td>
<td>lfttim_u</td>
<td>Left Time</td>
<td>Time remaining until the handler starts (in microseconds)</td>
</tr>
<tr>
<td>UINT</td>
<td>almstat</td>
<td>Alarm Handler Status</td>
<td>Alarm handler activation state</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>Bad identifier</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

<table>
<thead>
<tr>
<th>TK_SUPPORT_DBGSPRT</th>
<th>Support of μT-Kernel/DS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_USEC</td>
<td>Support of microsecond</td>
</tr>
</tbody>
</table>

Description

This system call takes 64-bit lfttim_u in microseconds instead of the return parameter lfttim of td_ref alm. The specification of this system call is same as that of td_ref alm, except that the return parameter is replaced with lfttim_u. For more details, see the description of td_ref alm.
6.1.34  td_ref_sys - Reference System Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_sys(TD_RSYS *pk_rsys);

Parameter

<table>
<thead>
<tr>
<th>TD_RSYS*</th>
<th>pk_rsys</th>
<th>Packet to Return System Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Pointer to the area to return the system status</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error code</td>
</tr>
</tbody>
</table>

pk_rsys Detail:

<table>
<thead>
<tr>
<th>UINT</th>
<th>sysstat</th>
<th>System State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>runtskid</td>
<td>Running Task ID</td>
</tr>
<tr>
<td>ID</td>
<td>schedtskid</td>
<td>Scheduled Task ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID of the task currently in RUNNING state</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID of the task scheduled to run next</td>
</tr>
</tbody>
</table>

Error Code

| E_OK             | Normal completion |

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSP - Support of μT-Kernel/DS

Description

Gets the system status. This is similar to tk_ref_sys.
6.1.35  td_ref_ssy - Reference Subsystem Status

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_ssy(ID ssid, TD_RSSY *rssy);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>ssid</th>
<th>Subsystem ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD_RSSY</td>
<td>rssy</td>
<td>Packet to Return Subsystem Status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Target subsystem ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pointer to the area to return the subsystem definition information</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>ercd</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

rssy Detail:

<table>
<thead>
<tr>
<th>PRI</th>
<th>ssypri</th>
<th>Subsystem Priority</th>
</tr>
</thead>
</table>

Error Code

- E_OK: Normal completion
- E_ID: Bad identifier
- E_NOEXS: Object does not exist

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

<table>
<thead>
<tr>
<th>Service Profile Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_DBGSPRT</td>
<td>Support of μ T-Kernel/DS</td>
</tr>
<tr>
<td>TK_SUPPORT_SUBSYSTEM</td>
<td>Support of subsystem management functions</td>
</tr>
</tbody>
</table>

Description

References the subsystem status. This is similar to tk_ref_ssy.
6.1.36  td_get_reg - Get Task Register

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_get_reg(ID tskid, T_REGS *pk_regs, T_EIT *pk_eit, T_CREGS *pk_cregs);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID</th>
<th>Target task ID (TSK_SELF cannot be specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_REGS*</td>
<td>pk_regs</td>
<td>Packet of Registers</td>
<td>Pointer to the area to return the general register values</td>
</tr>
<tr>
<td>T_EIT*</td>
<td>pk_eit</td>
<td>Packet of EIT Registers</td>
<td>Pointer to the area to return the values of registers saved when an exception occurs</td>
</tr>
<tr>
<td>T_CREGS*</td>
<td>pk_cregs</td>
<td>Packet of Control Registers</td>
<td>Pointer to the area to return the control register values</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

The contents of T_REGS, T_EIT, and T_CREGS are defined for each CPU and implementation.

Error Code

<table>
<thead>
<tr>
<th>E_OK</th>
<th>Normal completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>Invalid ID number (tskid is invalid or cannot be used)</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist (the task specified in tskid does not exist)</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>Invalid object state (issued for a RUNNING state task)</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

<table>
<thead>
<tr>
<th>TK_SUPPORTDBGSP</th>
<th>Support of μT-Kernel/DS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK_SUPPORT_REGOPS</td>
<td>Support for task-register manipulation functions</td>
</tr>
</tbody>
</table>

Description

Gets the register values of the task designated in tskid. This is similar to tk_get_reg.
Registers cannot be referenced for the task currently in RUNNING state. Except when a task-independent portion is executing, the current RUNNING state task is the invoking task.

If NULL is set in pk_regs, pk_eit, or pk_cregs, the corresponding registers are not referenced.

The contents of T_REGS, T_EIT, and T_CREGS are implementation-dependent.
6.1.37  td_set_reg - Set Task Registers

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_set_reg(ID tskid, CONST T_REGS *pk_regs, CONST T_EIT *pk_eit, CONST T_CREGS *pk_cregs);

Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>tskid</td>
<td>Task ID Target task ID (TSK_SELF cannot be specified)</td>
</tr>
<tr>
<td>CONST T_REGS*</td>
<td>pk_regs</td>
<td>Packet of Registers General registers</td>
</tr>
<tr>
<td>CONST T_EIT*</td>
<td>pk_eit</td>
<td>Packet of EIT Registers Registers saved when EIT occurs</td>
</tr>
<tr>
<td>CONST T_CREGS*</td>
<td>pk_cregs</td>
<td>Packet of Control Registers Control registers</td>
</tr>
</tbody>
</table>

The contents of T_REGS, T_EIT, and T_CREGS are defined for each CPU and implementation.

Return Parameter

<table>
<thead>
<tr>
<th>ID</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ER</td>
<td>Error Code</td>
<td>Error code</td>
</tr>
</tbody>
</table>

Error Code

- E_OK  Normal completion
- E_ID  Invalid ID number (tskid is invalid or cannot be used)
- E_NOEXS Object does not exist (the task specified in tskid does not exist)
- E_OBJ  Invalid object state (issued for a RUNNING state task)

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- TK_SUPPORT_DBGPT Support of μT-Kernel/DS
- TK_SUPPORT_REGOPS Support for task-register manipulation functions

Description

Sets registers of the task designated in tskid. This is similar to tk_set_reg.

Registers cannot be set for the task currently in RUNNING state. Except when a task-independent portion is executing, the current RUNNING state task is the invoking task.

If NULL is set in pk_regs, pk_eit, or pk_cregs, the corresponding registers are not set.

The contents of T_REGS, T_EIT, and T_CREGS are implementation-dependent.
6.1.38  td_get_utc - Get System Time

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_get_utc(SYSTIM *tim, UW *ofs);

Parameter

| SYSTIM*  tim | Time          | Pointer to the area to return the current time (ms) |
| UW*  ofs    | Offset        | Pointer to the area to return the return parameter ofs |

Return Parameter

| ER  ercd  | Error Code | Error code |
| SYSTIM  tim | Time       | Current time (in milliseconds) |
| UW  ofs   | Offset     | Elapsed time from tim (in nanoseconds) |

tim Detail:

| W  hi     | High 32 bits | Higher 32 bits of current time of the system time |
| UW  lo    | Low 32 bits  | Lower 32 bits of current time of the system time |

Error Code

E_OK  Normal completion

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

| TK_SUPPORT_DBGSPH | Support of μT-Kernel/DS |
| TK_SUPPORT_RTC    | Support of UNIX time   |

Description

Gets the current time as total elapsed milliseconds since 0:00:00, January 1, 1970 (UTC). The value returned in tim is the same as that obtained by tk_get_utc. tim is the resolution of timer interrupt intervals (cycles),
but even more precise time information is obtained in `ofs` as the elapsed time from `tim` in nanoseconds. The resolution of `ofs` is implementation-dependent, but generally is the resolution of hardware timer.

Since `tim` is a cumulative time counted based on timer interrupts, in some cases time is not refreshed, when a timer interrupt cycle arrives while interrupts are disabled and the timer interrupt handler is not started (is delayed). In such cases, the time as updated by the previous timer interrupt is returned in `tim`, and the elapsed time from the previous timer interrupt is returned in `ofs`. Accordingly, in some cases `ofs` will be longer than the timer interrupt cycle. The length of elapsed time that can be measured by `ofs` depends on the hardware, but preferably it should be possible to measure at least up to twice the timer interrupt cycle ($0 \leq ofs < \text{twice the timer interrupt cycle}$).

Note that the time returned in `tim` and `ofs` is the time at some point between the calling of and return from `td_get_utc`. It is neither the time at which `td_get_utc` was called nor the time of return from `td_get_utc`. In order to obtain more accurate information, this function should be called in interrupts disabled state.
6.1.39  td_get_utc_u - Get System Time (Microseconds)

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_get_utc_u(SYSTIM_U *tim_u, UW *ofs);

Parameter

| SYSTIM_U* tim_u | Time          | Pointer to the area to return the current time (in microseconds) |
| UW* ofs         | Offset        | Pointer to the area to return the return parameter ofs |

Return Parameter

| ER ercd | Error Code | Error code |
| SYSTIM_U tim_u | Time | Current time (in microseconds) |
| UW ofs | Offset | Elapsed time from tim_u (in nanoseconds) |

Error Code

E_OK  Normal completion

Valid Context

| Task portion | Quasi-task portion | Task-independent portion |
| YES | YES | YES |

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPRT  Support of μT-Kernel/DS
TK_SUPPORT_UTC  Support of UNIX time
TK_SUPPORT_USEC  Support of microsecond

Description

This system call takes 64-bit tim_u in microseconds instead of the return parameter tim of td_get_utc.

The specification of this system call is same as that of td_get_utc, except that the return parameter is replaced with tim_u. For more details, see the description of td_get_utc.
6.1.40  td_get_tim - Get System Time (TRON)

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_get_tim(SYSTIM *tim, UW *ofs);

Parameter

SYSTIM* tim  Time  Pointer to the area to return the current time (ms)
UW*  ofs  Offset  Pointer to the area to return the return parameter ofs

Return Parameter

ER  ercd  Error Code  Error code
SYSTIM  tim  Time  Current time (in milliseconds)
UW  ofs  Offset  Elapsed time from tim (in nanoseconds)

tim Detail:

W  hi  High 32 bits  Higher 32 bits of current time of the system time
UW  lo  Low 32 bits  Lower 32 bits of current time of the system time

Error Code

E_OK  Normal completion

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT  Support of μT-Kernel/DS
TK_SUPPORT_TRONTIME  Support of TRON time

Description

Gets the current time as total elapsed milliseconds since 0:00:00 (GMT), January 1, 1985. The value returned in tim is the same as that obtained by tk_get_tim. tim is the resolution of timer interrupt intervals (cycles),
but even more precise time information is obtained in \( \text{ofs} \) as the elapsed time from \( \text{tim} \) in nanoseconds. The resolution of \( \text{ofs} \) is implementation-dependent, but generally is the resolution of hardware timer.

Since \( \text{tim} \) is a cumulative time counted based on timer interrupts, in some cases time is not refreshed, when a timer interrupt cycle arrives while interrupts are disabled and the timer interrupt handler is not started (is delayed). In such cases, the time as updated by the previous timer interrupt is returned in \( \text{tim} \), and the elapsed time from the previous timer interrupt is returned in \( \text{ofs} \). Accordingly, in some cases \( \text{ofs} \) will be longer than the timer interrupt cycle. The length of elapsed time that can be measured by \( \text{ofs} \) depends on the hardware, but preferably it should be possible to measure at least up to twice the timer interrupt cycle (\( 0 \leq \text{ofs} < \text{twice the timer interrupt cycle} \)).

Note that the time returned in \( \text{tim} \) and \( \text{ofs} \) is the time at some point between the calling of and return from \text{td_get_tim}. It is neither the time at which \text{td_get_tim} was called nor the time of return from \text{td_get_tim}. In order to obtain more accurate information, this function should be called in interrupts disabled state.

**Additional Notes**

\text{td_get_tim} is very similar to \text{td_get_utc}. However, it uses the time system with a different epoch. \text{td_get_tim} is an API to keep compatibility with legacy \( \mu \) T-Kernel or T-Kernel specifications.
6.1.41 td_get_tim_u - Get System Time (TRON, Microseconds)

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_get_tim_u(SYSTIM_U *tim_u, UW *ofs);

Parameter

<table>
<thead>
<tr>
<th>SYSTIM_U*</th>
<th>tim_u</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW*</td>
<td>ofs</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Parameter

<table>
<thead>
<tr>
<th>SYSTIM_U*</th>
<th>tim_u</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW*</td>
<td>ofs</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Parameters:

- tim_u: Pointer to the area to return the current time (in microseconds)
- ofs: Pointer to the area to return the return parameter

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTIM_U</td>
<td>tim_u</td>
<td>Time</td>
</tr>
<tr>
<td>UW</td>
<td>ofs</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Error Code

- E_OK: Normal completion

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

- TK_SUPPORT_DBGSPT: Support of μT-Kernel/DS
- TK_SUPPORT_TRONTIME: Support of TRON time
- TK_SUPPORT_USEC: Support of microsecond

Description

This system call takes 64-bit tim_u in microseconds instead of the return parameter tim of td_get_tim. The specification of this system call is same as that of td_get_tim, except that the return parameter is replaced with tim_u. For more details, see the description of td_get_tim.

Additional Notes

td_get_tim_u is very similar to td_get_utc_u. However, it uses the time system with a different epoch. td_get_tim_u is an API to keep compatibility with legacy μT-Kernel or T-Kernel specifications.
6.1.42 td_get_otm - Get Operating Time

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_get_otm(SYSTIM *tim, UW *ofs);

Parameter

| SYSTIM* tim | Time | Pointer to the area to return the operating time (ms) |
| UW* ofs | Offset | Pointer to the area to return the return parameter ofs |

Return Parameter

| ER ercd | Error Code |
| SYSTIM tim | Time | Operating code (in milliseconds) |
| UW ofs | Offset | Elapsed time from tim (in nanoseconds) |

tim Detail:

| UW hi | High 32 bits |
| UW lo | Lower 32 bits of the system operating time |

| UW hi | Higher 32 bits of the system operating time |
| UW lo | Lower 32 bits of the system operating time |

Error Code

E_OK Normal completion

Valid Context


<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Gets the system operating time (uptime, as elapsed milliseconds since the system was booted). The value returned in tim is the same as that obtained by tk_get_otm. tim is the resolution of timer interrupt intervals (cycles), but even more precise time information is obtained in ofs as the elapsed time from tim in nanoseconds. The resolution of ofs is implementation-dependent, but generally is the resolution of hardware timer.
Since `tim` is a cumulative time counted based on timer interrupts, in some cases time is not refreshed, when a timer interrupt cycle arrives while interrupts are disabled and the timer interrupt handler is not started (is delayed). In such cases, the time as updated by the previous timer interrupt is returned in `tim`, and the elapsed time from the previous timer interrupt is returned in `ofs`. Accordingly, in some cases `ofs` will be longer than the timer interrupt cycle. The length of elapsed time that can be measured by `ofs` depends on the hardware, but preferably it should be possible to measure at least up to twice the timer interrupt cycle (0 ≤ `ofs` < twice the timer interrupt cycle).

Note that the time returned in `tim` and `ofs` is the time at some point between the calling of and return from `td_get_otm`. It is neither the time at which `td_get_otm` was called nor the time of return from `td_get_otm`. In order to obtain more accurate information, this function should be called in interrupts disabled state.
6.1.43  td_get_otm_u - Get Operating Time (Microseconds)

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_get_otm_u(SYSTIM_U *tim_u, UW *ofs);

Parameter

<table>
<thead>
<tr>
<th>SYSTIM_U*</th>
<th>tim_u</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>UW*</td>
<td>ofs</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Pointer to the area to return the operating time (in microseconds)
Pointer to the area to return the return parameter ofs

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTIM_U</td>
<td>tim_u</td>
<td>Time</td>
</tr>
<tr>
<td>UW</td>
<td>ofs</td>
<td>Offset</td>
</tr>
</tbody>
</table>

Error code
Operating time (in microseconds)
Elapsed time from tim_u (in nanoseconds)

Error Code

E_OK Normal completion

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS
TK_SUPPORT_USEC Support of microsecond

Description

This system call takes 64-bit tim_u in microseconds instead of the return parameter tim of td_get_otm.
The specification of this system call is same as that of td_get_otm, except that the return parameter is replaced with tim_u. For more details, see the description of td_get_otm.
6.1.44  td_ref_dsname - Refer to DS Object Name

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_ref_dsname(UINT type, ID id, UB *dsname);

Parameter

<table>
<thead>
<tr>
<th>UINT</th>
<th>type</th>
<th>Object Type</th>
<th>Target object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>id</td>
<td>Object ID</td>
<td>Object ID</td>
</tr>
<tr>
<td>UB*</td>
<td>dsname</td>
<td>DS Object Name</td>
<td>Pointer to the area to return the DS object name</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>ER</th>
<th>ercd</th>
<th>Error Code</th>
<th>Error code</th>
</tr>
</thead>
</table>

dsname Detail:
DS object name, set at object creation or by td_set_dsname

Error Code

- E_OK  Normal completion
- E_PAR Invalid object type
- E_NOEXS Object does not exist
- E_OBJ  DS object name is not used

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DSNAME  Support for DS object name

Description

References the DS object name (dsname), which is set at object creation. The object is specified by object type (type) and object ID (id).

Object types (type) are as follows:

- TN_TSK 0x01  Task
- TN_SEM 0x02  Semaphore
DS object name is valid if `TA_DSNAME` is set as object attribute. If DS object name is changed by `td_set_dsname`, then `td_ref_dsname` references the new name.

DS object name needs to satisfy the following conditions. However, character code range is not checked by μT-Kernel.

Available characters (UB)
   a to z, A to Z, 0 to 9, _

Name length
   Up to 8 bytes (not including '¥0')

Additional Notes

The DS object name that is read is terminated with a '¥0' character. Hence, `dsname` must have a area of 9 or more bytes.
6.1.45  td_set_dsname - Set DS Object Name

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_set_dsname(UINT type, ID id, CONST UB *dsname);

Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>type</td>
<td>Object Type</td>
</tr>
<tr>
<td>ID</td>
<td>id</td>
<td>Object ID</td>
</tr>
<tr>
<td>CONST UB*</td>
<td>dsname</td>
<td>DS Object Name</td>
</tr>
</tbody>
</table>

Return Parameter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>ercd</td>
<td>Error Code</td>
</tr>
</tbody>
</table>

Error Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>E_PAR</td>
<td>Invalid object type</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>Object does not exist</td>
</tr>
<tr>
<td>E_OBJ</td>
<td>DS object name is not used</td>
</tr>
</tbody>
</table>

Valid Context

<table>
<thead>
<tr>
<th>Context</th>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DSNAME Support for DS object name

Description

Re-sets DS object name (dsname), which is set at object creation. The object is specified by object type (type) and object ID (id).

Object types (type) are as same as that of td_ref_dsname.

DS object name needs to satisfy the following conditions. However, character code range is not checked by μT-Kernel.

Available characters (UB)

- a to z, A to Z, 0 to 9, _

Name length

- Up to 8 bytes (not including ’\0’)

-
DS object name is valid if TA_DSNAME is set as object attribute. td_set_dsname returns E_OBJ error if TA_DSNAME attribute is not specified.
6.2 Trace Functions

Trace functions are functions for enabling a debugger to trace program execution. Execution trace is performed by setting hook routines.

- Return from a hook routine must be made after states have returned to where they were when the hook routine was called. Restoring of registers, however, can be done in accordance with the C language function saving rules.

- In a hook routine, limitations on states must not be loosened to make them less restrictive than when the routine was called. For example, if the hook routine was called during interrupts disabled state, interrupts must not be enabled.

- A hook routine was called at protection level 0.

- A hook routine inherits the stack at the time of the hook. Using too much stack may therefore cause a stack overflow. The extent to which the stack can be used is not definite, since it differs with the situation at the time of the hook. Switching to a separate stack in the hook routine is a safer option.
6.2.1  td_hok_svc - Define System Call/Extended SVC Hook Routine

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_hok_svc(CONST TD_HSVC *hsvc);

Parameter

| CONST TD_HSVC* | hsvc | SVC Hook Routine | Hook routine definition information |

hsvc Detail:

| FP | enter | Hook Routine before Calling | Hook routine before calling |
| FP | leave | Hook Routine after Calling  | Hook routine after calling  |

Return Parameter

| ER | ercd | Error Code | Error code |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Sets hook routines before and after the issuing of a system call or extended SVC. Setting NULL in hsvc cancels a hook routine.

The target of trace functions are the system calls of μT-Kernel/OS (tk_~) and extended SVCs. Note, however, generally speaking tk_ret_int is not the target of trace function. This is implementation-dependent.

System calls of μ T-Kernel/DS (td_~) are not the target of trace functions.

A hook routine runs as a quasi-task portion of the task that called a system call or extended SVC for which a hook routine is set. Therefore, for example, the invoking task in a hook routine is the same as the task that invoked the system call or extended SVC.
Since task dispatching and interrupts can occur inside system call processing, `enter()` and `leave()` are not necessarily called in succession as a pair in every case. If a system call is one that does not return, `leave()` will not be called.

```c
void *enter(FN fncd, TD_CALINF *calinf, ...);
```

<table>
<thead>
<tr>
<th>FN</th>
<th>fncd</th>
<th>Function Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&lt; 0 System call</td>
</tr>
<tr>
<td></td>
<td></td>
<td>≧ 0 Extended SVC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TD_CALINF*</th>
<th>calinf</th>
<th>Caller information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Parameters (variable number)</td>
</tr>
</tbody>
</table>

Return

Any value passed to `leave()`

```c
typedef struct td_calinf {
  Information to determine the caller for the system call or extended SVC;
  it is preferable to include the information for the stack back-trace.
  The contents are implementation-dependent,
  but generally consist of register values such as stack pointer and program counter.
} TD_CALINF;
```

`enter` is called right before a system call or extended SVC.

The value passed in the return code is passed transparently to the corresponding `leave()`. This makes it possible to pair `enter()` and `leave()` calls or to pass any other information.

```c
exinf = enter(fncd, &calinf, ...)
ret = system call or extended SVC execution
leave(fncd, ret, exinf)
```

- For system call
  The parameters are the same as the system call parameters.

```c
tk_wai_sem(ID semid, INT cnt, TMO tmout)
enter(TFN_WAI_SEM, &calinf, semid, cnt, tmout)
```

- For extended SVC
  The parameters are as in the packet passed to the extended SVC handler.

```c
fncd is likewise the same as that passed to the extended SVC handler.
```

```c
enter (FN fnccd, TD_CALINF *calinf, void *pk_para);
void leave(FN fnccd, INT ret, void *exinf);
```

<table>
<thead>
<tr>
<th>FN</th>
<th>fncd</th>
<th>Function Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&lt; 0 System call</td>
</tr>
<tr>
<td></td>
<td></td>
<td>≧ 0 Extended SVC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INT</th>
<th>ret</th>
<th>Return code of the system call or extended SVC</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>void*</th>
<th>exinf</th>
<th>Any value returned by <code>enter()</code></th>
</tr>
</thead>
</table>

`enter` is called right after returning from a system call or extended SVC.

When a hook routine is set after a system call or extended SVC is called (while the system call or extended SVC is executing), in some cases `leave()` only may be called without calling `enter()`.

If, on the other hand, a hook routine is canceled after a system call or extended SVC is called, there may be cases when `enter()` is called but not `leave()`. 
6.2.2  td_hok_dsp - Define Task Dispatch Hook Routine

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_hok_dsp(CONST TD_HDSP *hdsp);

Parameter

<table>
<thead>
<tr>
<th>CONST TD_HDSP*</th>
<th>hdsp</th>
<th>Dispatcher Hook Routine</th>
<th>Hook routine definition information</th>
</tr>
</thead>
</table>

hdsp Detail:

| FP | exec | Hook Routine when Execution Starts | Hook routine when execution starts |
| FP | stop | Hook Routine when Execution Stops | Hook routine when execution stops |

Return Parameter

| ER | ercd | Error Code | Error code |

Error Codes

None.

Valid Context

<table>
<thead>
<tr>
<th>Task portion</th>
<th>Quasi-task portion</th>
<th>Task-independent portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

| TK_SUPPORT_DBGSPT | Support of μT-Kernel/DS |

Description

Sets hook routines in the task dispatcher. Setting NULL in hdsp cancels a hook routine.

A hook routine is called while dispatching is disabled. A hook routine shall not invoke system calls of μT-Kernel/OS (tk_~) and extended SVCs. A hook routine can invoke system calls of μT-Kernel/DS (td_~).

void exec(ID tskid);

ID tskid  Task ID of the started or resumed task


`exec()` is called when the designated task starts execution or resumes. At the time `exec()` is called, the task designated in `tskid` is already in RUNNING state. However, execution of the `tskid` task program code occurs after the return from `exec()`.

```c
void stop(ID tskid, UINT tskstat);
```

<table>
<thead>
<tr>
<th>ID</th>
<th>tskid</th>
<th>Task ID of the executed or stopped task</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>tskstat</td>
<td>State of the task designated in <code>tskid</code></td>
</tr>
</tbody>
</table>

`stop()` is called when the designated task executes or stops. `tskstat` indicates the task state after stopping, as one of the following states:

```
TTS_RDY    READY state
TTS_WAI    WAITING state
TTS_SUS    SUSPENDED state
TTS_WAS    WAITING-SUSPENDED state
TTS_DMT    DORMANT state
0          NON-EXISTENT state
```

At the time `stop()` is called, the task designated in `tskid` has already entered the state indicated in `tskstat`. 
6.2.3  td_hok_int - Define Interrupt Handler Hook Routine

C Language Interface

#include <tk/dbgspt.h>

ER ercd = td_hok_int(CONST TD_HINT *hint);

Parameter

| CONST TD_HINT* hint | Interrupt Handler Hook Routine | Hook routine definition information |

hint Detail:

| FP enter | Hook Routine before Calling Handler | Hook routine before calling handler |
| FP leave | Hook Routine after Calling Handler  | Hook routine after calling handler |

Return Parameter

| ER ercd | Error Code | Error code |

Error Codes

None.

Valid Context

| Task portion | Quasi-task portion | Task-independent portion |
| YES | YES | YES |

Related Service Profile Items

Only when all the service profile items below are set to be effective, this system call can be used.

TK_SUPPORT_DBGSPT Support of μT-Kernel/DS

Description

Sets hook routines before and after an interrupt handler is called. Hook routine setting cannot be done individually for different exception or interrupt factors. One pair of hook routines is set in common for all exception and interrupt factors.

Setting hint to NULL cancels the hook routines.

The hook routines are called as task-independent portion (part of the interrupt handler). Accordingly, the hook routines can call only those system calls that can be invoked from a task-independent portion.
Note that hook routines can be set only for interrupt handlers defined by `tk_def_int` with the `TA_HLNG` attribute. A `TA_ASM` attribute interrupt handler cannot be hooked by a hook routine. Hooking of a `TA_ASM` attribute interrupt handler is possible only by directly manipulating the exception/interrupt vector table. The actual methods are implementation-dependent.

```c
void *enter(UINT intno);
void *leave(UINT intno);
```

Parameters passed to `enter()` and `leave()` are the same as those of exception handler and interrupt handler. Depending on the implementation, other information about the interrupt may be passed in addition to `intno`.

A hook routine is called as follows from a high-level language support routine.

```c
enter(intno);
inthdr(intno); /* Interrupt or exception handler */
leave(intno);
```

`enter()` is called in interrupts disabled state, and interrupts must not be enabled. Since `leave()` assumes the status on return from `inthdr()`, the interrupts disabled or enabled status is indeterminate.

`enter()` can obtain the same amount of information which the function `inthdr()` can obtain. If the function `inthdr()` cannot obtain a piece of information, that information cannot be acquired by `enter()`, either. The specification guarantees that `enter()` and `inthdr()` can access information by means of `intno`, but whether other information can be acquired is implementation dependent. Note that during the execution of the function `leave()`, the states such as interrupt mask status may have changed, it may be impossible to obtain the same amount of information obtained by `enter()` or `inthdr()`.
Chapter 7

Appendix
7.1 System Configuration

It is permitted to change \( \mu T \)-Kernel or remove unnecessary functions from it when an implementation of \( \mu T \)-Kernel is embedded into a real-world product, etc. The reference implementation of \( \mu T \)-Kernel 3.0 contains features to change and set the parameters such as the number of resources and limit values of certain parameters. Changing and setting system parameters is called system configuration.

This section lists the system configuration items that are provided by the \( \mu T \)-Kernel 3.0 reference implementation.

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFN_MAX_PRI</td>
<td>Maximum task priority (it is reflected in service profile item, TK_MAX_TSKPRI)</td>
</tr>
<tr>
<td>CFN_SYSTEMAREA_TOP</td>
<td>Lowest address of a region dynamically managed by the memory management function of ( \mu T )-Kernel</td>
</tr>
<tr>
<td>CFN_SYSTEMAREA_END</td>
<td>Highest address of a region dynamically managed by the memory management function of ( \mu T )-Kernel</td>
</tr>
<tr>
<td>CFN_TIMER_PERIOD</td>
<td>Timer interrupt interval (in milliseconds)</td>
</tr>
<tr>
<td>CFN_MAX_TSKID</td>
<td>Maximum number of tasks</td>
</tr>
<tr>
<td>CFN_MAX_SEMID</td>
<td>Maximum number of semaphores</td>
</tr>
<tr>
<td>CFN_MAX_FLGID</td>
<td>Maximum number of event flags</td>
</tr>
<tr>
<td>CFN_MAX_MBXID</td>
<td>Maximum number of mailboxes</td>
</tr>
<tr>
<td>CFN_MAX_MTXID</td>
<td>Maximum number of mutexes</td>
</tr>
<tr>
<td>CFN_MAX_MBFID</td>
<td>Maximum number of messages buffers.</td>
</tr>
<tr>
<td>CFN_MAX_MPFID</td>
<td>Maximum number of fixed-size memory pools</td>
</tr>
<tr>
<td>CFN_MAX_MPLID</td>
<td>Maximum number of variable-size memory pools</td>
</tr>
<tr>
<td>CFN_MAX_CYCID</td>
<td>Maximum number of cyclic handlers</td>
</tr>
<tr>
<td>CFN_MAX_ALMID</td>
<td>Maximum number of alarm handlers</td>
</tr>
<tr>
<td>CFN_MAX_SSYID</td>
<td>Maximum number of subsystems</td>
</tr>
<tr>
<td>CFN_MAX_SSYPRI</td>
<td>Maximum number of subsystem priorities</td>
</tr>
<tr>
<td>CFN_MAX_REGDEV</td>
<td>Maximum number of registered devices</td>
</tr>
<tr>
<td>CFN_MAX_OPNDEV</td>
<td>Maximum number of open devices</td>
</tr>
<tr>
<td>CFN_MAX_REQDEV</td>
<td>Maximum number of pending device requests</td>
</tr>
</tbody>
</table>

Additional Notes
System configuration is not within the scope of \( \mu T \)-Kernel 3.0. But if similar function is to be provided, it is desirable to use naming conventions that are compatible with these item names.
7.2 Keywords

1. Keywords related to the name of the OS
   - μT-Kernel
   - TRON
   - IEEE 2050-2018

2. Keywords related to the application of the OS
   - embedded
   - real time
   - IoT edgenode
   - small scale
   - 16-bit CPU
   - single chip microcomputer
   - single chip MicroController Unit (MCU)
   - power saving

3. Keywords related to the fundamental concepts used in the OS
   - real-time operating system (RTOS)
   - Application Programming Interface (API)
   - system call
   - kernel
   - task
   - dispatching (task dispatching)
   - scheduling (task scheduling)
   - priority-based
   - task portion
   - non-task portion
   - taskindependent portion
   - quasi-task portion
   - service profile

4. Keywords related to the individual functions of OS
   - semaphore
   - event flag
   - mailbox
   - message buffer
   - mutex
   - memory pool
   - interrupt handler
   - cyclic handler
   - alarm handler
   - device management
   - power management
   - physical timer
   - fast lock
   - fast multi-lock
Chapter 8

Reference
8.1 List of C Language Interface

8.1.1 μT-Kernel/OS

8.1.1.1 Task Management Functions

- ID tskid = tk_cre_tsk (CONST T_CTSK *pk_ctsk);
- ER ercd = tk_del_tsk (ID tskid);
- ER ercd = tk_sta_tsk (ID tskid, INT stacd);
- void tk_ext_tsk (void);
- void tk_exd_tsk (void);
- ER ercd = tk_ter_tsk (ID tskid);
- ER ercd = tk_chg_pri (ID tskid, PRI tskpri);
- ER ercd = tk_get_reg (ID tskid, T_REGS *pk_regs, T_EIT *pk_eit, T_CREGS *pk_cregs);
- ER ercd = tk_set_reg (ID tskid, CONST T_REGS *pk_regs, CONST T_EIT *pk_eit, CONST T_CREGS *pk_cregs);
- ER ercd = tk_get_cpr (ID tskid, INT copno, T_COPREGS *pk_copregs);
- ER ercd = tk_set_cpr (ID tskid, INT copno, CONST T_COPREGS *pk_copregs);
- ER ercd = tk_ref_tsk (ID tskid, T_RTSK *pk_rtsk);

8.1.1.2 Task Synchronization Functions

- ER ercd = tk_slp_tsk (TMO tmout);
- ER ercd = tk_slp_tsk_u (TMO_U tmout_u);
- ER ercd = tk_wup_tsk (ID tskid);
- INT wupcnt = tk_can_wup (ID tskid);
- ER ercd = tk_rel_wai (ID tskid);
- ER ercd = tk_sus_tsk (ID tskid);
- ER ercd = tk_rsm_tsk (ID tskid);
- ER ercd = tk_frsm_tsk (ID tskid);
- ER ercd = tk_dly_tsk (RELTIM dlytim);
- ER ercd = tk_dly_tsk_u (RELTIM_U dlytim_u);
- ER ercd = tk_sig_tev (ID tskid, INT tskevt);
- INT tevptn = tk_wai_tev (INT waiptn, TMO tmout);
- INT tevptn = tk_wai_tev_u (INT waiptn, TMO_U tmout_u);
- INT tskwait = tk_dis_wai (ID tskid, UW waitmask);
- ER ercd = tk_ena_wai (ID tskid);
8.1.1.3 Task Exception Handling Functions

- \( \text{ER ercd} = \text{tk_def_tex} ( \text{ID tskid}, \text{CONST T_DTEX }'\text{pk_dtex}', ) \)
- \( \text{ER ercd} = \text{tk_ena_tex} ( \text{ID tskid}, \text{UINT texptn} ) \)
- \( \text{ER ercd} = \text{tk_dis_tex} ( \text{ID tskid}, \text{UINT texptn} ) \)
- \( \text{ER ercd} = \text{tk_ras_tex} ( \text{ID tskid}, \text{INT texcd} ) \)
- \( \text{INT texcd} = \text{tk_end_tex} ( \text{BOOL enatex} ) \)
- \( \text{ER ercd} = \text{tk_ref_tex} ( \text{ID tskid}, \text{T_RTEX }'\text{pk_rtex}', ) \)

8.1.1.4 Synchronization and Communication Functions

- \( \text{ID semid} = \text{tk_cre_sem} ( \text{CONST T_CSEM }'\text{pk_csem}', ) \)
- \( \text{ER ercd} = \text{tk_del_sem} ( \text{ID semid} ) \)
- \( \text{ER ercd} = \text{tk_sig_sem} ( \text{ID semid}, \text{INT cnt} ) \)
- \( \text{ER ercd} = \text{tk_wai_sem} ( \text{ID semid}, \text{INT cnt}, \text{TMO tmout} ) \)
- \( \text{ER ercd} = \text{tk_wai_sem}_u ( \text{ID semid}, \text{INT cnt}, \text{TMO_U tmout}_u ) \)
- \( \text{ER ercd} = \text{tk_ref_sem} ( \text{ID semid}, \text{T_RSEM }'\text{pk_rsem}', ) \)
- \( \text{ID flgid} = \text{tk_cre_flg} ( \text{CONST T_CFLG }'\text{pk_cflg}', ) \)
- \( \text{ER ercd} = \text{tk_del_flg} ( \text{ID flgid} ) \)
- \( \text{ER ercd} = \text{tk_set_flg} ( \text{ID flgid}, \text{UINT setptn} ) \)
- \( \text{ER ercd} = \text{tk_clr_flg} ( \text{ID flgid}, \text{UINT clrptn} ) \)
- \( \text{ER ercd} = \text{tk_wai_flg} ( \text{ID flgid}, \text{UINT waiptn}, \text{UINT wmode}, \text{UINT }'\text{p_flgptn}, \text{TMO tmout} ) \)
- \( \text{ER ercd} = \text{tk_wai_flg}_u ( \text{ID flgid}, \text{UINT waiptn}, \text{UINT wmode}, \text{UINT }'\text{p_flgptn}, \text{TMO_U tmout}_u ) \)
- \( \text{ER ercd} = \text{tk_ref_flg} ( \text{ID flgid}, \text{T_RFLG }'\text{pk_rflg}', ) \)
- \( \text{ID mbxid} = \text{tk_cre_mbx} ( \text{CONST T_CMBX }'\text{pk_cmbx}', ) \)
- \( \text{ER ercd} = \text{tk_del_mbx} ( \text{ID mbxid} ) \)
- \( \text{ER ercd} = \text{tk_snd_mbx} ( \text{ID mbxid}, \text{T_MSG }'\text{pk_msg}', ) \)
- \( \text{ER ercd} = \text{tk_rcv_mbx} ( \text{ID mbxid}, \text{T_MSG }'\text{ppk_msg}, \text{TMO tmout} ) \)
- \( \text{ER ercd} = \text{tk_rcv_mbx}_u ( \text{ID mbxid}, \text{T_MSG }'\text{ppk_msg}, \text{TMO_U tmout}_u ) \)
- \( \text{ER ercd} = \text{tk_ref_mbx} ( \text{ID mbxid}, \text{T_RMBX }'\text{pk_rmbx}', ) \)
8.1.1.5 Extended Synchronization and Communication Functions

- ID mtxid = tk_cre_mtx (CONST T_CMTX *pk_cmtx);
- ER ercd = tk_del_mtx (ID mtxid);
- ER ercd = tk_loc_mtx (ID mtxid, TMO tmout);
- ER ercd = tk_loc_mtx_u (ID mtxid, TMO_U tmout_u);
- ER ercd = tk_unl_mtx (ID mtxid);
- ER ercd = tk_ref_mtx (ID mtxid, T_RMTX *pk_rmtx);
- ID mbfid = tk_cre_mbf (CONST T_CMBF *pk_cmbf);
- ER ercd = tk_del_mbf (ID mbfid);
- ER ercd = tk_snd_mbf (ID mbfid, CONST void *msg, INT msgsz, TMO tmout);
- ER ercd = tk_snd_mbf_u (ID mbfid, CONST void *msg, INT msgsz, TMO_U tmout_u);
- INT msgsz = tk_rcv_mbf (ID mbfid, void *msg, TMO tmout);
- INT msgsz = tk_rcv_mbf_u (ID mbfid, void *msg, TMO_U tmout_u);
- ER ercd = tk_ref_mbf (ID mbfid, T_RMBF *pk_rmbf);

8.1.1.6 Memory Pool Management Functions

- ID mpfid = tk_cre_mpf (CONST T_CMPF *pk_cmpf);
- ER ercd = tk_del_mpf (ID mpfid);
- ER ercd = tk_get_mpf (ID mpfid, void *p_blf, TMO tmout);
- ER ercd = tk_get_mpf_u (ID mpfid, void *p_blf, TMO_U tmout_u);
- ER ercd = tk_rel_mpf (ID mpfid, void *blf);
- ER ercd = tk_ref_mpf (ID mpfid, T_RMPF *pk_rmpf);
- ID mplid = tk_cre_mpl (CONST T_CMPL *pk_cmpl);
- ER ercd = tk_del_mpl (ID mplid);
- ER ercd = tk_get_mpl (ID mplid, SZ blksz, void *p_blk, TMO tmout);
- ER ercd = tk_get_mpl_u (ID mplid, SZ blksz, void *p_blk, TMO_U tmout_u);
- ER ercd = tk_rel_mpl (ID mplid, void *blk);
- ER ercd = tk_ref_mpl (ID mplid, T_RMPL *pk_rmpl);
8.1.1.7 Time Management Functions

- ER ercd = tk_set_utc (CONST SYSTIM *pk_tim);
- ER ercd = tk_set_utc_u (SYSTIM_U tim_u);
- ER ercd = tk_set_tim (CONST SYSTIM *pk_tim);
- ER ercd = tk_set_tim_u (SYSTIM_U tim_u);
- ER ercd = tk_get_utc (SYSTIM *pk_tim);
- ER ercd = tk_get_utc_u (SYSTIM_U tim_u, UW ofs);
- ER ercd = tk_get_tim (SYSTIM_U *tim_u);
- ER ercd = tk_get_tim_u (SYSTIM_U *tim_u, UW ofs);
- ER ercd = tk_get_otm (SYSTIM_U *tim_u);
- ID cycid = tk_cre_cyc (CONST T_CCYC *pk_ccyc);
- ID cycid = tk_cre_cyc_u (CONST T_CCYC_U *pk_ccyc_u);
- ER ercd = tk_del_cyc (ID cycid);
- ER ercd = tk_sta_cyc (ID cycid);
- ER ercd = tk_stp_cyc (ID cycid);
- ER ercd = tk_ref_cyc (ID cycid, T_RCYC *pk_rcyc);
- ER ercd = tk_ref_cyc_u (ID cycid, T_RCYC_U *pk_rcyc_u);
- ID almid = tk_cre_alm (CONST T_CALM *pk_calm);
- ER ercd = tk_del_alm (ID almid);
- ER ercd = tk_sta_alm (ID almid, RELTIM almtim);
- ER ercd = tk_sta_alm_u (ID almid, RELTIM_U almtim_u);
- ER ercd = tk_stp_alm (ID almid);
- ER ercd = tk_ref_alm (ID almid, T_RALM *pk_ralm);
- ER ercd = tk_ref_alm_u (ID almid, T_RALM_U *pk_ralm_u);

8.1.1.8 Interrupt Management Functions

- ER ercd = tk_def_int (UINT intno, CONST T_DINT *pk_dint);
- void tk_ret_int (void);

8.1.1.9 System Management Functions

- ER ercd = tk_rot_rdq (PRI tskpri);
- ID tskid = tk_get_tid (void);
- ER ercd = tk_dis_dsp (void);
- ER ercd = tk_ena_dsp (void);
- ER ercd = tk_ref_sys (T_RSYS *pk_rsys);
- ER ercd = tk_set_pow (UINT powmode);
- ER ercd = tk_ref_ver (T_RVER *pk_rver);
8.1.1.10 Subsystem Management Functions

- ER ercd = tk_def_ssy ( ID ssid, CONST T_DSSY *pk_dssy );
- ER ercd = tk_evt_ssy ( ID ssid, INT evttyp, ID resid, INT info );
- ER ercd = tk_ref_ssy ( ID ssid, T_RSSY *pk_rssy );

8.1.2 μT-Kernel/SM

8.1.2.1 System Memory Management Functions

- void* Kmalloc ( size_t size );
- void* Kcalloc ( size_t nmemb, size_t size );
- void* Krealloc ( void *ptr, size_t size );
- void Kfree ( void *ptr );

8.1.2.2 Device Management Functions

- ID dd = tk_opn_dev ( CONST UB *devnm, UINT omode );
- ER ercd = tk_cls_dev ( ID dd, UINT option );
- ID reqid = tk_rea_dev ( ID dd, W start, void *buf, SZ size, TMO tmout );
- ID reqid = tk_rea_dev_du ( ID dd, D start_d, void *buf, SZ size, TMO_U tmout_u );
- ER ercd = tk_srea_dev ( ID dd, W start, void *buf, SZ size, SZ *asize );
- ER ercd = tk_srea_dev_d ( ID dd, D start_d, void *buf, SZ size, SZ *asize );
- ID reqid = tk_wri_dev ( ID dd, W start, CONST void *buf, SZ size, TMO tmout );
- ID reqid = tk_wri_dev_du ( ID dd, D start_d, CONST void *buf, SZ size, TMO_U tmout_u );
- ER ercd = tk_swri_dev ( ID dd, W start, CONST void *buf, SZ size, SZ *asize );
- ER ercd = tk_swri_dev_d ( ID dd, D start_d, CONST void *buf, W size, W *asize );
- ID creqid = tk_wai_dev ( ID dd, ID reqid, SZ *asize, ER *ioer, TMO tmout );
- ID creqid = tk_wai_dev_u ( ID dd, ID reqid, SZ *asize, ER *ioer, TMO_U tmout_u );
- INT dissus = tk_sus_dev ( UINT mode );
- ID devid = tk_get_dev ( ID devid, UB *devnm );
- ID devid = tk_ref_dev ( CONST UB *devnm, T_RDEV *rdev );
- ID devid = tk_oref_dev ( ID dd, T_RDEV *rdev );
- INT remcnt = tk_lst_dev ( T_LDEV *ldev, INT start, INT ndev );
- INT retcode = tk_evt_dev ( ID devid, INT evttyp, void *evtinf );
- ID devid = tk_def_dev ( CONST UB *devnm, CONST T_DDEV *ddev, T_IDEV *idev );
- ER ercd = tk_ref_idv ( T_IDEV *idev );
- ER ercd = openfn ( IDdevid, UINTomode, void *exinf );
• ER ercd = closefn ( IDdevid, UINToption, void *exinf);
• ER ercd = execfn ( T_DEVREQ *devreq, TMOtmout, void *exinf);
• ER ercd = execfn ( T_DEVREQ_D *devreq_d, TMOtmout, void *exinf);
• ER ercd = execfn ( T_DEVREQ *devreq, TMO_Utmout_u, void *exinf);
• ER ercd = execfn ( T_DEVREQ_D *devreq_d, TMO_Utmout_u, void *exinf);
• INT creqno = waitfn ( T_DEVREQ *devreq, INTnreq, TMOtmout *exinf);
• INT creqno = waitfn ( T_DEVREQ_D *devreq_d, INTnreq, TMOtmout *exinf);
• INT creqno = waitfn ( T_DEVREQ *devreq, INTnreq, TMO_Utmout_u *exinf);
• ER ercd = abortfn ( IDtskid, T_DEVRQ *devreq, INTnreq, void *exinf);
• ER ercd = abortfn ( IDtskid, T_DEVRQ_D *devreq_d, INTnreq, void *exinf);
• INT retcode = eventfn ( INTevttyp, void *evtinf, void *exinf);

8.1.2.3 Interrupt Management Functions

• DI ( UINT intsts );
• EI ( UINT intsts );
• BOOL disint = isDI ( UINT intsts );
• void SetCpuIntLevel ( INT level );
• INT level = GetCpuIntLevel ( void );
• void EnableInt ( UINT intno );
• void EnableInt ( UINT intno, INT level );
• void DisableInt ( UINT intno );
• void ClearInt ( UINT intno );
• void EndOfInt ( UINT intno );
• BOOL rasint = CheckInt ( UINT intno );
• void SetIntMode ( UINT intno, UINT mode );
• void SetCtrlIntLevel ( INT level );
• INT level = GetCtrlIntLevel ( void );
8.1.2.4 I/O Port Access Support Functions

- void out_b ( INT port, UB data );
- void out_h ( INT port, UH data );
- void out_w ( INT port, UW data );
- void out_d ( INT port, UD data );
- UB data = in_b ( INT port );
- UH data = in_h ( INT port );
- UW data = in_w ( INT port );
- UD data = in_d ( INT port );
- void WaitUsec ( UW usec );
- void WaitNsec ( UW nsec );

8.1.2.5 Power Management Functions

- void low_pow ( void );
- void off_pow ( void );

8.1.2.6 System Configuration Information Management Functions

- INT ct = tk_get_cfn ( CONST UB *name, W *val, INT max );
- INT rlen = tk_get_cfs ( CONST UB *name, UB *buf, INT max );

8.1.2.7 Memory Cache Control Functions

- SZ rlen = SetCacheMode ( void *addr, SZ len, UINT mode );
- SZ rlen = ControlCache ( void *addr, SZ len, UINT mode );

8.1.2.8 Physical Timer Functions

- ER ercd = StartPhysicalTimer ( UINT ptmrno, UW limit, UINT mode );
- ER ercd = StopPhysicalTimer ( UINT ptmrno );
- ER ercd = GetPhysicalTimerCount ( UINT ptmrno, UW *p_count );
- ER ercd = DefinePhysicalTimerHandler ( UINT ptmrno, CONST T_DPTMR *pk_dptmr );
- ER ercd = GetPhysicalTimerConfig ( UINT ptmrno, T_RPTMR *pk_rptmr );
8.1.2.9 Utility Functions

- void SetOBJNAME (void *exinf, CONST UB *name);
- ER ercd = CreateLock (FastLock *lock, CONST UB *name);
- void DeleteLock (FastLock *lock);
- void Lock (FastLock *lock);
- void Unlock (FastLock *lock);
- ER ercd = CreateMLock (FastMLock *lock, CONST UB *name);
- ER ercd = DeleteMLock (FastMLock *lock);
- ER ercd = MLock (FastMLock *lock, INT no);
- ER ercd = MLockTmo (FastMLock *lock, INT no, TMO tmout);
- ER ercd = MLockTmo_u (FastMLock *lock, INT no, TMO_U tmout_u);
- ER ercd = MUnlock (FastMLock *lock, INT no);

8.1.3 μT-Kernel/DS

8.1.3.1 Kernel Internal State Acquisition Functions

- INT ct = td_lst_tsk (ID list[], INT nent);
- INT ct = td_lst_sem (ID list[], INT nent);
- INT ct = td_lst_flg (ID list[], INT nent);
- INT ct = td_lst_mbx (ID list[], INT nent);
- INT ct = td_lst_mtx (ID list[], INT nent);
- INT ct = td_lst_mbf (ID list[], INT nent);
- INT ct = td_lst_mpf (ID list[], INT nent);
- INT ct = td_lst_mpl (ID list[], INT nent);
- INT ct = td_lst_cyc (ID list[], INT nent);
- INT ct = td_lst_alm (ID list[], INT nent);
- INT ct = td_lst_ssy (ID list[], INT nent);
- INT ct = td_rdy_que (PRI pri, ID list[], INT nent);
- INT ct = td_sem_que (ID semid, ID list[], INT nent);
- INT ct = td_flg_que (ID flgid, ID list[], INT nent);
- INT ct = td_mbx_que (ID mbxid, ID list[], INT nent);
- INT ct = td_mtx_que (ID mtxid, ID list[], INT nent);
- INT ct = td_smbf_que (ID mbfid, ID list[], INT nent);
- INT ct = td_rmbf_que (ID mbfid, ID list[], INT nent);
- INT ct = td_mpf_que (ID mpfid, ID list[], INT nent);
• INT ct = td_mpl_que ( ID mplid, ID list[], INT nent );
• ER ercd = td_ref_tsk ( ID tskid, TD_RTSK *rtsk );
• ER ercd = td_ref_tex ( ID tskid, TD_RTEX *pk_rtex );
• ER ercd = td_ref_sem ( ID semid, TD_RSEM *rsem );
• ER ercd = td_ref_flg ( ID flgid, TD_RFLG *rflg );
• ER ercd = td_ref_mbx ( ID mbxid, TD_RMBX *rmbx );
• ER ercd = td_ref_mtx ( ID mtxid, TD_RMTX *rmtx );
• ER ercd = td_ref_mbf ( ID mbfid, TD_RMBF *rmbf );
• ER ercd = td_ref_mpf ( ID mpfid, TD_RMPF *rpmf );
• ER ercd = td_ref_mpl ( ID mplid, TD_RMPL *rmpl );
• ER ercd = td_ref_cyc ( ID cycid, TD_RCYC *rcyc );
• ER ercd = td_ref_cyc_u ( ID cycid, TD_RCYC_U *rcyc_u );
• ER ercd = td_ref_alm ( ID almid, TD_RALM *ralm );
• ER ercd = td_ref_alm_u ( ID almid, TD_RALM_U *ralm_u );
• ER ercd = td_ref_sys ( TD_RSYS *pk_rsys );
• ER ercd = td_ref_ssy ( ID ssid, TD_RSSY *rssy );
• ER ercd = td_get_reg ( ID tskid, T_REGS *pk_regs, T_EIT *pk_eit, T_CREGS *pk_cregs );
• ER ercd = td_set_reg ( ID tskid, CONST T_REGS *pk_regs, CONST T_EIT *pk_eit, CONST T_CREGS *pk_cregs );
• ER ercd = td_get_utc ( SYSTIM *tim, UW *ofs );
• ER ercd = td_get_utc_u ( SYSTIM_U *tim_u, UW *ofs );
• ER ercd = td_get_tim ( SYSTIM *tim, UW *ofs );
• ER ercd = td_get_tim_u ( SYSTIM_U *tim_u, UW *ofs );
• ER ercd = td_get_otm ( SYSTIM *tim, UW *ofs );
• ER ercd = td_get_otm_u ( SYSTIM_U *tim_u, UW *ofs );
• ER ercd = td_ref_dsname ( UINT type, ID id, UB *dsname );
• ER ercd = td_set_dsname ( UINT type, ID id, CONST UB *dsname );

8.1.3.2 Trace Functions
• ER ercd = td_hok_svc ( CONST TD_HSVC *hsvc );
• ER ercd = td_hok_dsp ( CONST TD_HDSP *hdsp );
• ER ercd = td_hok_int ( CONST TD_HINT *hint );
8.2 List of Error Codes

8.2.1 Normal Completion Error Class (0)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OK</td>
<td>0</td>
<td>Normal completion</td>
</tr>
</tbody>
</table>

8.2.2 Normal completion Internal Error Class (5 to 8)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_SYS</td>
<td>ERCD(-5, 0)</td>
<td>System error</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOCOP</td>
<td>ERCD(-6, 0)</td>
<td>Unavailable co-processor</td>
</tr>
</tbody>
</table>

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

8.2.3 Unsupported Error Class (9 to 16)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOSPT</td>
<td>ERCD(-9, 0)</td>
<td>Unsupported function</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_RSFN</td>
<td>ERCD(-10, 0)</td>
<td>Reserved function code number</td>
</tr>
</tbody>
</table>

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 it is attempted to execute an undefined extended SVC handler (a positive function code).

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_RSATR</td>
<td>ERCD(-11, 0)</td>
<td>Reserved attribute</td>
</tr>
</tbody>
</table>

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.

8.2.4 Parameter Error Class (17 to 24)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_PAR</td>
<td>ERCD(-17, 0)</td>
<td>Parameter error</td>
</tr>
</tbody>
</table>
Checking for this error may be omitted if system-dependent optimization is implemented.

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ID</td>
<td>ERC(-18, 0)</td>
<td>Invalid ID number</td>
</tr>
</tbody>
</table>

E_ID is an error that is returned only for objects having an ID number. Error code E_PAR is returned when a static error is detected for such as reserved number or out of range in the case of interrupt number.

### 8.2.5 Call Context Error Class (25 to 32)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_CTX</td>
<td>ERC(-25, 0)</td>
<td>Context error</td>
</tr>
</tbody>
</table>

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

This error must be returned whenever there is a semantic context error in issuing a system call, such as calling from a task-independent portion a system call that may put the invoking task in WAITING state. Due to implementation limitations, there may be other system calls that, when called from a given context (such as an interrupt handler), will cause this error to be returned.

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_MACV</td>
<td>ERC(-26, 0)</td>
<td>Memory cannot be accessed; memory access privilege error</td>
</tr>
</tbody>
</table>

Error detection is implementation-dependent.

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OACV</td>
<td>ERC(-27, 0)</td>
<td>Object access privilege error</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ILUSE</td>
<td>ERC(-28, 0)</td>
<td>System call illegal use</td>
</tr>
</tbody>
</table>

### 8.2.6 Resource Constraint Error Class (33 to 40)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOMEM</td>
<td>ERC(-33, 0)</td>
<td>Insufficient memory</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_LIMIT</td>
<td>ERC(-34, 0)</td>
<td>System limit exceeded</td>
</tr>
</tbody>
</table>
This error code is returned, for example, when it is attempted to create more object(s) than the system allows.

### 8.2.7 Object State Error Class (41 to 48)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_OBJ</td>
<td>ERCD(-41, 0)</td>
<td>Invalid object state</td>
</tr>
<tr>
<td>E_NOEXS</td>
<td>ERCD(-42, 0)</td>
<td>Object does not exist</td>
</tr>
<tr>
<td>E_QOVR</td>
<td>ERCD(-43, 0)</td>
<td>Queuing or nesting overflow</td>
</tr>
</tbody>
</table>

### 8.2.8 Wait Error Class (49 to 56)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_RLWAI</td>
<td>ERCD(-49, 0)</td>
<td>Waiting state was forcibly released</td>
</tr>
<tr>
<td>E_TMOUT</td>
<td>ERCD(-50, 0)</td>
<td>Polling failed or timeout</td>
</tr>
<tr>
<td>E_DLT</td>
<td>ERCD(-51, 0)</td>
<td>Waiting object was deleted</td>
</tr>
<tr>
<td>E_DISWAI</td>
<td>ERCD(-52, 0)</td>
<td>Wait released due to disabling of wait</td>
</tr>
</tbody>
</table>

### 8.2.9 Device Error Class (57 to 64) (μT-Kernel/SM)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_IO</td>
<td>ERCD(-57, 0)</td>
<td>I/O error</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_NOMDA</td>
<td>ERCD(-58, 0)</td>
<td>No media</td>
</tr>
</tbody>
</table>

### 8.2.10 Status Error Class (65 to 72) (μT-Kernel/SM)

<table>
<thead>
<tr>
<th>Error code name</th>
<th>Error Codes</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_BUSY</td>
<td>ERCD(-65, 0)</td>
<td>Busy</td>
</tr>
<tr>
<td>E_ABORT</td>
<td>ERCD(-66, 0)</td>
<td>Processing was aborted</td>
</tr>
<tr>
<td>E_RONLY</td>
<td>ERCD(-67, 0)</td>
<td>Write protected</td>
</tr>
</tbody>
</table>
8.3 List of APIs and Service Profile Items

8.3.1 μT-Kernel/OS

8.3.1.1 Task Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_cre_tsk</td>
<td>Always</td>
<td>TK_SUPPORT_ASM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_USERBUF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_AUTOBUF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_FPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_COPn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_HAS_SYSSTACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_DSNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_MAX_TSKPRI</td>
</tr>
<tr>
<td>tk_del_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_sta_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_ext_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_exd_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_ter_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_chg_pri</td>
<td>Always</td>
<td>TK_MAX_TSKPRI</td>
</tr>
<tr>
<td>tk_get_reg</td>
<td>TK_SUPPORT_REGOPS</td>
<td>None</td>
</tr>
<tr>
<td>tk_set_reg</td>
<td>TK_SUPPORT_REGOPS</td>
<td>None</td>
</tr>
<tr>
<td>tk_get_cpr</td>
<td>TK_SUPPORT_COPn</td>
<td>None</td>
</tr>
<tr>
<td>tk_set_cpr</td>
<td>TK_SUPPORT_COPn</td>
<td>None</td>
</tr>
<tr>
<td>tk_ref_tsk</td>
<td>Always</td>
<td>TK_SUPPORT_DISWAI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_TASKEXCEPTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_TASKEVENT</td>
</tr>
</tbody>
</table>

8.3.1.2 Task Synchronization Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_slp_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_slp_tsk_u</td>
<td>TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_wup_tsk</td>
<td>Always</td>
<td>TK_WAKEUP_MAXCNT</td>
</tr>
<tr>
<td>tk_can_wup</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_rel_wai</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_sus_tsk</td>
<td>Always</td>
<td>TK_SUSPEND_MAXCNT</td>
</tr>
<tr>
<td>tk_rsm_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_frsn_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_dly_tsk</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_dly_tsk_u</td>
<td>TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_sig_tev</td>
<td>TK_SUPPORT_TASKEVENT</td>
<td>None</td>
</tr>
<tr>
<td>tk_wai_tev</td>
<td>TK_SUPPORT_TASKEVENT</td>
<td>None</td>
</tr>
<tr>
<td>tk_wai_tev_u</td>
<td>TK_SUPPORT_TASKEVENT &amp; &amp;</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_USEC</td>
</tr>
<tr>
<td>tk_dis_wai</td>
<td>TK_SUPPORT_DISWAI</td>
<td>None</td>
</tr>
</tbody>
</table>
### Task Exception Handling Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_ena_wai</td>
<td>TK_SUPPORT_DISWAI</td>
<td>None</td>
</tr>
</tbody>
</table>

### Synchronization and Communication Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_cre_sem</td>
<td>Always</td>
<td>TK_SUPPORT_DISWAI TK_SUPPORT_DSNAME TK_SEMAPHORE_MAXCNT</td>
</tr>
<tr>
<td>tk_del_sem</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_sig_sem</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_wai_sem</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_wai_sem_u</td>
<td>TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_ref_sem</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_cre_flg</td>
<td>Always</td>
<td>TK_SUPPORT_DISWAI TK.Support_DSNAME</td>
</tr>
<tr>
<td>tk_del_flg</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_set_flg</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_clr_flg</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_wai_flg</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_wai_lst_u</td>
<td>TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_ref_lst</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_cre_mbx</td>
<td>Always</td>
<td>TK_SUPPORT_DISWAI TK_SUPPORT_DSNAME</td>
</tr>
<tr>
<td>tk_del_mbx</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_snd_mbx</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_rcv_mbx</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_rcv_mbx_u</td>
<td>TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_ref_mbx</td>
<td>Always</td>
<td>None</td>
</tr>
</tbody>
</table>

### Extended Synchronization and Communication Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_cre_mtx</td>
<td>Always</td>
<td>TK_SUPPORT_DISWAI TK_SUPPORT_DSNAME</td>
</tr>
</tbody>
</table>
## 8.3.1.6 Memory Pool Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_del_mpf</td>
<td>Always</td>
<td>TK_SUPPORT_USERBUF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_AUTOBUF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_DISWAI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TK_SUPPORT_DSNNAME</td>
</tr>
<tr>
<td>tk_del_mpf</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_snd_mpf</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_snd_mpl_u</td>
<td>TK_SUPPORT_UAEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_rcv_mpf</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_rcv_mpl_u</td>
<td>TK_SUPPORT_UAEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_ref_mpf</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_ref_mpl</td>
<td>Always</td>
<td>None</td>
</tr>
</tbody>
</table>

## 8.3.1.7 Time Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_set_utc</td>
<td>TK_SUPPORT_UTC</td>
<td>None</td>
</tr>
<tr>
<td>tk_set_utc_u</td>
<td>TK_SUPPORT_UTC &amp; TK_SUPPORT_UAEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_set_tim</td>
<td>TK_SUPPORT_TRONTIME</td>
<td>None</td>
</tr>
<tr>
<td>tk_set_tim_u</td>
<td>TK_SUPPORT_TRONTIME &amp; TK_SUPPORT_UAEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_get_utc</td>
<td>TK_SUPPORT_UTC</td>
<td>None</td>
</tr>
</tbody>
</table>
### 8.3.1.8 Interrupt Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tk_def_int</code></td>
<td>Always</td>
<td>TK_SUPPORT_ASM</td>
</tr>
<tr>
<td><code>tk_ref_int</code></td>
<td>Always</td>
<td>TK_SUPPORT_ASM</td>
</tr>
</tbody>
</table>

### 8.3.1.9 System Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tk_rot_rdq</code></td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td><code>tk_get_tid</code></td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td><code>tk_dis_dsp</code></td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td><code>tk_ena_dsp</code></td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td><code>tk_ref_sys</code></td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td><code>tk_set_pow</code></td>
<td>TK_SUPPORT_LOWPOWER</td>
<td>None</td>
</tr>
<tr>
<td><code>tk_ref_ver</code></td>
<td>Always</td>
<td>None</td>
</tr>
</tbody>
</table>

### 8.3.1.10 Subsystem Management Functions
### μT-Kernel 3.0 Specification

#### 8.3.2 μT-Kernel/SM

#### 8.3.2.1 System Memory Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kmalloc</td>
<td>TK_SUPPORT_MEMLIB</td>
<td>None</td>
</tr>
<tr>
<td>Kcalloc</td>
<td>TK_SUPPORT_MEMLIB</td>
<td>None</td>
</tr>
<tr>
<td>Krealloc</td>
<td>TK_SUPPORT_MEMLIB</td>
<td>None</td>
</tr>
<tr>
<td>Kfree</td>
<td>TK_SUPPORT_MEMLIB</td>
<td>None</td>
</tr>
</tbody>
</table>

#### 8.3.2.2 Device Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>tk_opn_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_cls_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_rea_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_rea_dev_du</td>
<td>TK_SUPPORT_LARGEDEV &amp; &amp; TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_srea_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_srea_dev_d</td>
<td>TK_SUPPORT_LARGEDEV</td>
<td>None</td>
</tr>
<tr>
<td>tk_wri_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_wri_dev_du</td>
<td>TK_SUPPORT_LARGEDEV &amp; &amp; TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_swri_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_swri_dev_d</td>
<td>TK_SUPPORT_LARGEDEV</td>
<td>None</td>
</tr>
<tr>
<td>tk_wai_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_wai_dev_u</td>
<td>TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>tk_sus_dev</td>
<td>TK_SUPPORT_LOWPOWER</td>
<td>None</td>
</tr>
<tr>
<td>tk_get_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_ref_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_oref_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_lst_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_evt_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_def_dev</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>tk_ref_idv</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>openfn</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>closefn</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>execfn</td>
<td>Always</td>
<td>TK_SUPPORT_LARGEDEV, TK_SUPPORT_USEC</td>
</tr>
<tr>
<td>waitfn</td>
<td>Always</td>
<td>TK_SUPPORT_LARGEDEV, TK_SUPPORT_USEC</td>
</tr>
</tbody>
</table>
### 8.3.2.3 Interrupt Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>EI</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>isDI</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>SetCpuIntLevel</td>
<td>TK_SUPPORT_CPUINTELLEVEL</td>
<td>None</td>
</tr>
<tr>
<td>GetCpuIntLevel</td>
<td>TK_SUPPORT_CPUINTELLEVEL</td>
<td>None</td>
</tr>
<tr>
<td>EnableInt</td>
<td>TK_SUPPORT_INTCTRL</td>
<td>TK_HAS_ENAINTLEVEL</td>
</tr>
<tr>
<td>DisableInt</td>
<td>TK_SUPPORT_INTCTRL</td>
<td>None</td>
</tr>
<tr>
<td>ClearInt</td>
<td>TK_SUPPORT_INTCTRL</td>
<td>None</td>
</tr>
<tr>
<td>EndOfInt</td>
<td>TK_SUPPORT_INTCTRL</td>
<td>None</td>
</tr>
<tr>
<td>CheckInt</td>
<td>TK_SUPPORT_INTCTRL</td>
<td>None</td>
</tr>
<tr>
<td>SetIntMode</td>
<td>TK_SUPPORT_INTMODE</td>
<td>None</td>
</tr>
<tr>
<td>SetCtrlIntLevel</td>
<td>TK_SUPPORT_CTRLINTELLEVEL</td>
<td>None</td>
</tr>
<tr>
<td>GetCtrlIntLevel</td>
<td>TK_SUPPORT_CTRLINTELLEVEL</td>
<td>None</td>
</tr>
</tbody>
</table>
8.3.2.4  I/O Port Access Support Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out_b</code></td>
<td><code>TK_SUPPORT_IOPORT</code></td>
<td>None</td>
</tr>
<tr>
<td><code>out_h</code></td>
<td><code>TK_SUPPORT_IOPORT</code></td>
<td>None</td>
</tr>
<tr>
<td><code>out_w</code></td>
<td><code>TK_SUPPORT_IOPORT</code></td>
<td>None</td>
</tr>
<tr>
<td><code>out_d</code></td>
<td><code>TK_SUPPORT_IOPORT &amp;&amp; TK_HAS_DOUBLEWORD</code></td>
<td>None</td>
</tr>
<tr>
<td><code>in_b</code></td>
<td><code>TK_SUPPORT_IOPORT</code></td>
<td>None</td>
</tr>
<tr>
<td><code>in_h</code></td>
<td><code>TK_SUPPORT_IOPORT</code></td>
<td>None</td>
</tr>
<tr>
<td><code>in_w</code></td>
<td><code>TK_SUPPORT_IOPORT</code></td>
<td>None</td>
</tr>
<tr>
<td><code>in_d</code></td>
<td><code>TK_SUPPORT_IOPORT &amp;&amp; TK_HAS_DOUBLEWORD</code></td>
<td>None</td>
</tr>
<tr>
<td><code>WaitUsec</code></td>
<td><code>TK_SUPPORT_MICROWAIT</code></td>
<td>None</td>
</tr>
<tr>
<td><code>WaitNsec</code></td>
<td><code>TK_SUPPORT_MICROWAIT</code></td>
<td>None</td>
</tr>
</tbody>
</table>

8.3.2.5  Power Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>low_pow</code></td>
<td><code>TK_SUPPORT_LOWPOWER</code></td>
<td>None</td>
</tr>
<tr>
<td><code>off_pow</code></td>
<td><code>TK_SUPPORT_LOWPOWER</code></td>
<td>None</td>
</tr>
</tbody>
</table>

8.3.2.6  System Configuration Information Management Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tk_get_cfn</code></td>
<td><code>TK_SUPPORT_SYSCONF</code></td>
<td>None</td>
</tr>
<tr>
<td><code>tk_get_cfs</code></td>
<td><code>TK_SUPPORT_SYSCONF</code></td>
<td>None</td>
</tr>
</tbody>
</table>

8.3.2.7  Memory Cache Control Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SetCacheMode</code></td>
<td><code>TK_SUPPORT_CACHECRTL &amp;&amp; TK_SUPPORT_SETCACHEMODE</code></td>
<td><code>TK_SUPPORT_WBCACHE</code> <code>TK_SUPPORT_WTCACHE</code></td>
</tr>
<tr>
<td><code>ControlCache</code></td>
<td><code>TK_SUPPORT_CACHECRTL</code></td>
<td>None</td>
</tr>
</tbody>
</table>

8.3.2.8  Physical Timer Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>StartPhysicalTimer</code></td>
<td><code>TK_SUPPORT_PTIMER</code></td>
<td><code>TK_MAX_PTIMER</code></td>
</tr>
<tr>
<td><code>StopPhysicalTimer</code></td>
<td><code>TK_SUPPORT_PTIMER</code></td>
<td><code>TK_MAX_PTIMER</code></td>
</tr>
<tr>
<td><code>GetPhysicalTimerCount</code></td>
<td><code>TK_SUPPORT_PTIMER</code></td>
<td><code>TK_MAX_PTIMER</code></td>
</tr>
</tbody>
</table>
### 8.3.2.9 Utility Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetOBJNAME</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>CreateLock</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>DeleteLock</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>Lock</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>Unlock</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>CreateMLock</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>DeleteMLock</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>MLock</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>MLockTmo</td>
<td>Always</td>
<td>None</td>
</tr>
<tr>
<td>MLockTmo_u</td>
<td>TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>MUnlock</td>
<td>Always</td>
<td>None</td>
</tr>
</tbody>
</table>

### 8.3.3 μT-Kernel/DS

#### 8.3.3.1 Kernel Internal State Acquisition Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>td_lst_tsk</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_sem</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_flg</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_mbx</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_mtx</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_mbf</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_mpf</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_mpl</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_cyc</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_alm</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_lst_ssy</td>
<td>TK_SUPPORT_SUBSYSTEM</td>
<td></td>
</tr>
<tr>
<td>td_rdy_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_sem_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_flq_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_mbx_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_mtx_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_smbf_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_rmbf_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_mpfl_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_mpl_que</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>None</td>
</tr>
<tr>
<td>td_ref_tsk</td>
<td>TK_SUPPORT_DBGSPT</td>
<td>TK_SUPPORT_DISWAI\ TK_SUPPORT_TASKEXCEPTION</td>
</tr>
</tbody>
</table>
### Trace Functions

<table>
<thead>
<tr>
<th>API name</th>
<th>Availability</th>
<th>Other related service profile items</th>
</tr>
</thead>
<tbody>
<tr>
<td>td_hok_svc</td>
<td>TK_SUPPORT_DBGSP &amp;;&amp; TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>td_hok_dsp</td>
<td>TK_SUPPORT_DBGSP &amp;;&amp; TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
<tr>
<td>td_hok_int</td>
<td>TK_SUPPORT_DBGSP &amp;;&amp; TK_SUPPORT_USEC</td>
<td>None</td>
</tr>
</tbody>
</table>