

# FR ファミリ

μT-Kernel 仕様準拠

## SOFTUNE™ μT-REALOS/FR

### API リファレンス





**FR ファミリ**

**μT-Kernel 仕様準拠**

**SOFTUNE™ μT-REALOS/FR**  
**API リファレンス**



**富士通マイクロエレクトロニクス株式会社**



# はじめに

## 本書の目的と対象読者

本書は、SOFTUNE  $\mu$ T-REALOS/FR(以降、 $\mu$ T-REALOS とよびます)のアプリケーションプログラムを作成する方を対象としており、 $\mu$ T-REALOS API について記述しています。 $\mu$ T-REALOS API について確認したい時に、適宜本書を参照してください。また、本書をご覧になる前に、「SOFTUNE  $\mu$ T-REALOS/FR ユーザーズガイド」(以降、「ユーザーズガイド」とよびます)を一読されることをお勧めします。

$\mu$ T-REALOS は、32 ビット RISC コントローラ FR ファミリで動作する  $\mu$ T-Kernel 仕様のリアルタイム OS です。

$\mu$ T-Kernel 仕様は、T-Engine フォーラムが策定したオープンなリアルタイム OS の仕様です。 $\mu$ T-Kernel の仕様書は、T-Engine フォーラムのホームページ (<http://www.t-engine.org/>) から入手できます。 $\mu$ T-Kernel の著作権は、坂村健氏に属しています。 $\mu$ T-Kernel 仕様の著作権は、T-Engine フォーラムに属しています。本製品は、T-Engine フォーラム ([www.t-engine.org](http://www.t-engine.org)) の  $\mu$ T-License に基づき  $\mu$ T-Kernel ソースコードを利用しています。

## 商標

SOFTUNE は富士通マイクロエレクトロニクス株式会社の商標です。

REALOS は富士通マイクロエレクトロニクス株式会社の商標です。

TRON は、「The Real-time Operating system Nucleus」の略称です。

ITRON は、「Industrial TRON」の略称です。

$\mu$ ITRON は、「Micro Industrial TRON」の略称です。

T-Kernel および  $\mu$ T-Kernel は、コンピュータの仕様に対する名称であり、特定の商品ないしは商品群を指すものではありません。

その他の記載されている社名および製品名などの固有名詞は、各社の商標または登録商標です。

## 本書の全体構成

本書は、以下に示す 3 つの章と付録から構成されています。

### 第 1 章 概要

この章では、 $\mu$ T-REALOS API の概要、全体的な注意事項について説明します。

### 第 2 章 データ型

この章では、 $\mu$ T-REALOS API のデータ型について説明します。

### 第 3 章 システムコールインタフェース

この章では、 $\mu$ T-REALOS でサポートしている  $\mu$ T-Kernel 仕様のシステムコールのインタフェースを説明します。

### 付録

付録では、「エラーコード一覧」、「定数マクロ一覧」、「デバイスドライバインタフェース」、「 $\mu$ ITRON 仕様 OS から移行時の注意点」および「システムコール索引」について記載しています。

## 参考マニュアル

本システムを使用する際には、必要に応じて次に示すマニュアルを参照してください。

『SOFTUNE  $\mu$ T-REALOS/FR ユーザーズガイド』

『FR ファミリ SOFTUNE C/C++ コンパイラマニュアル V6』

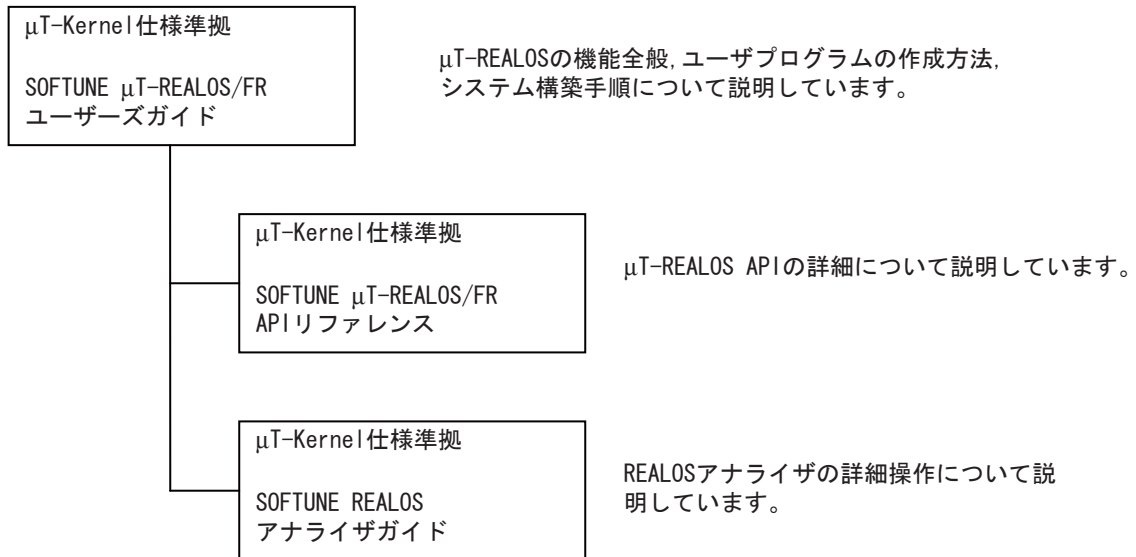
『FR ファミリ SOFTUNE アセンブルマニュアル V6』

『FR ファミリ SOFTUNE リンケージキットマニュアル V6』

## μT-REALOS のマニュアル体系

μT-REALOS のマニュアルは、以下の 3 分冊になっています。

μT-REALOS を初めてお使いになる方は、最初に『SOFTUNE μT-REALOS/FR ユーザーズガイド』をお読みください。



## 本書の読み方

### ● 記号の意味

本書では、システムコールのパラメータの記述形式を以下の表記に従って表現しています。

### 表記号の意味

記号	説明
[ ]	[ ] 内の要素は、記述が省略可能であることを示します。
	前後の要素の中からどれか一つの記述を選択することを示します。
	前後の要素を同時に選択できます。
: =	前の要素は後の要素の値をとります。



- 本資料の記載内容は、予告なしに変更することがありますので、ご用命の際は営業部門にご確認ください。
- 本資料に記載された動作概要や応用回路例は、半導体デバイスの標準的な動作や使い方を示したもので、実際に使用する機器での動作を保証するものではありません。したがって、これらを使用するにあたってはお客様の責任において機器の設計を行ってください。これらの使用に起因する損害などについては、当社はその責任を負いません。
- 本資料に記載された動作概要・回路図を含む技術情報は、当社もしくは第三者の特許権、著作権等の知的財産権やその他の権利の使用権または実施権の許諾を意味するものではありません。また、これらの使用について、第三者の知的財産権やその他の権利の実施ができることの保証を行うものではありません。したがって、これらの使用に起因する第三者の知的財産権やその他の権利の侵害について、当社はその責任を負いません。
- 本資料に記載された製品は、通常の産業用、一般事務用、パーソナル用、家庭用などの一般的用途に使用されることを意図して設計・製造されています。極めて高度な安全性が要求され、仮に当該安全性が確保されない場合、社会的に重大な影響を与えかつ直接生命・身体に対する重大な危険性を伴う用途（原子力施設における核反応制御、航空機自動飛行制御、航空交通管制、大量輸送システムにおける運行制御、生命維持のための医療機器、兵器システムにおけるミサイル発射制御をいう）、ならびに極めて高い信頼性が要求される用途（海底中継器、宇宙衛星をいう）に使用されるよう設計・製造されたものではありません。したがって、これらの用途にご使用をお考えのお客様は、必ず事前に営業部門までご相談ください。ご相談なく使用されたことにより発生した損害などについては、責任を負いかねますのでご了承ください。
- 半導体デバイスはある確率で故障が発生します。当社半導体デバイスが故障しても、結果的に人身事故、火災事故、社会的な損害を生じさせないように、お客様は、装置の冗長設計、延焼対策設計、過電流防止対策設計、誤動作防止設計などの安全設計をお願いします。
- 本資料に記載された製品を輸出または提供する場合は、外国為替及び外国貿易法および米国輸出管理関連法規等の規制をご確認の上、必要な手続きをおとりください。
- 本書に記載されている社名および製品名などの固有名詞は、各社の商標または登録商標です。

Copyright© 2008 FUJITSU MICROELECTRONICS LIMITED All rights reserved.

Copyright© 2006 T-Engine Forum All rights reserved.

このマニュアルはT-Engine フォーラムの許諾を得て  $\mu$ T-Kernel の仕様書に基づいて作成しています。



# 目次

<b>第 1 章</b>	<b>概要</b>	<b>1</b>
1.1	用語の説明	2
1.2	μT-REALOS API の概要	4
1.3	実装定義および実装依存仕様	5
1.4	拡張仕様	8
<b>第 2 章</b>	<b>データ型</b>	<b>9</b>
2.1	汎用的なデータ型と定数マクロ	10
2.2	μT-Kernel 固有の意味を持つデータ型と定数マクロ	12
<b>第 3 章</b>	<b>システムコールインタフェース</b>	<b>15</b>
3.1	システムコール一覧	16
3.2	システムコール説明内容	22
3.3	タスク管理機能のシステムコール	24
3.3.1	tk_cre_tsk(Create Task)	25
3.3.2	tk_del_tsk>Delete Task)	28
3.3.3	tk_sta_tsk (Start Task)	29
3.3.4	tk_ext_tsk(Exit Task)	30
3.3.5	tk_exd_tsk(Exit and Delete Task)	31
3.3.6	tk_ter_tsk(Terminate Task)	32
3.3.7	tk_chg_pri(Change Task Priority)	34
3.3.8	tk_get_reg(Get Task Registers)	37
3.3.9	tk_set_reg(Set Task Registers)	39
3.3.10	tk_ref_tsk(Refer Task Status)	41
3.4	タスク付属同期機能のシステムコール	45
3.4.1	tk_slp_tsk(Sleep Task)	46
3.4.2	tk_wup_tsk(Wakeup Task)	48
3.4.3	tk_can_wup(Cancel Wakeup Task)	50
3.4.4	tk_rel_wai(Release Wait)	52
3.4.5	tk_sus_tsk(Suspend Task)	54
3.4.6	tk_rsm_tsk(Resume Task)	56
3.4.7	tk_frsm_tsk(Force Resume Task)	58
3.4.8	tk_dly_tsk(Delay Task)	60
3.5	同期・通信機能のシステムコール	61
3.5.1	セマフォ機能のシステムコール	62
3.5.1.1	tk_cre_sem(Create Semaphore)	63
3.5.1.2	tk_del_sem>Delete Semaphore)	65
3.5.1.3	tk_sig_sem(Signal Semaphore)	66
3.5.1.4	tk_wai_sem(Wait on Semaphore)	68
3.5.1.5	tk_ref_sem(Refer Semaphore Status)	70
3.5.2	イベントフラグ機能のシステムコール	72
3.5.2.1	tk_cre_flg(Create Event Flag)	73
3.5.2.2	tk_del_flg>Delete Event Flag)	75
3.5.2.3	tk_set_flg(Set Event Flag)	76
3.5.2.4	tk_clr_flg(Clear Event Flag)	77

3.5.2.5	tk_wai_flg(Wait Event Flag).....	78
3.5.2.6	tk_ref_flg(Refer Event Flag Status) .....	81
3.5.3	メールボックス機能のシステムコール .....	83
3.5.3.1	tk_cre_mbx(Create Mailbox) .....	84
3.5.3.2	tk_del_mbx>Delete Mailbox) .....	86
3.5.3.3	tk_snd_mbx(Send Message to Mailbox) .....	87
3.5.3.4	tk_rcv_mbx(Receive Message from Mailbox) .....	89
3.5.3.5	tk_ref_mbx(Refer Mailbox Status) .....	91
3.6	拡張同期・通信機能のシステムコール.....	93
3.6.1	ミューテックス機能のシステムコール .....	94
3.6.1.1	tk_cre_mtx(Create Mutex).....	95
3.6.1.2	tk_del_mtx>Delete Mutex) .....	97
3.6.1.3	tk_loc_mtx(Lock Mutex) .....	98
3.6.1.4	tk_unl_mtx(Unlock Mutex).....	100
3.6.1.5	tk_ref_mtx(Refer Mutex Status) .....	102
3.6.2	メッセージバッファ機能のシステムコール .....	104
3.6.2.1	tk_cre_mbf(Create Message Buffer) .....	105
3.6.2.2	tk_del_mbf>Delete Message Buffer).....	108
3.6.2.3	tk_snd_mbf(Send Message to Message Buffer) .....	109
3.6.2.4	tk_rcv_mbf(Receive Message from Message Buffer) .....	111
3.6.2.5	tk_ref_mbf(Refer Message Buffer Status).....	113
3.6.3	ランデブポート機能のシステムコール .....	115
3.6.3.1	tk_cre_por(Create Port for Rendezvous) .....	116
3.6.3.2	tk_del_por>Delete Port for Rendezvous) .....	118
3.6.3.3	tk_cal_por(Call Port for Rendezvous) .....	119
3.6.3.4	tk_acp_por(Accept Port for Rendezvous) .....	122
3.6.3.5	tk_fwd_por(Forward Rendezvous to Another Port) .....	125
3.6.3.6	tk_rpl_rdv(Reply Rendezvous) .....	128
3.6.3.7	tk_ref_por(Refer Port Status) .....	130
3.7	メモリプール管理機能のシステムコール .....	132
3.7.1	固定長メモリプール機能のシステムコール .....	133
3.7.1.1	tk_cre_mpf(Create Fixed-size Memory Pool) .....	134
3.7.1.2	tk_del_mpf>Delete Fixed-size Memory Pool) .....	137
3.7.1.3	tk_get_mpf(Get Fixed-size Memory Block) .....	138
3.7.1.4	tk_rel_mpf(Release Fixed-size Memory Block).....	140
3.7.1.5	tk_ref_mpf(Refer Fixed-size Memory Pool Status) .....	142
3.7.2	可変長メモリプール機能のシステムコール .....	144
3.7.2.1	tk_cre_mpl(Create Variable-size Memory Pool).....	145
3.7.2.2	tk_del_mpl>Delete Variable-size Memory Pool) .....	148
3.7.2.3	tk_get_mpl(Get Variable-size Memory Block) .....	149
3.7.2.4	tk_rel_mpl(Release Variable-size Memory Block).....	151
3.7.2.5	tk_ref_mpl(Refer Variable-size Memory Pool Status) .....	153
3.8	時間管理機能のシステムコール .....	155
3.8.1	システム時刻管理機能のシステムコール.....	156
3.8.1.1	tk_set_tim(Set Time) .....	157
3.8.1.2	tk_get_tim(Get Time).....	158
3.8.1.3	tk_get_otm(Get Operating Time).....	159
3.8.1.4	isig_tim(Signal Time).....	160
3.8.2	周期ハンドラ機能のシステムコール .....	161

3.8.2.1	tk_cre_cyc(Create Cyclic Handler) .....	162
3.8.2.2	tk_del_cyc>Delete Cyclic Handler) .....	165
3.8.2.3	tk_sta_cyc(Start Cyclic Handler) .....	166
3.8.2.4	tk_stp_cyc(Stop Cyclic Handler) .....	167
3.8.2.5	tk_ref_cyc(Refer Cyclic Handler Status) .....	168
3.8.3	アラームハンドラ機能のシステムコール .....	170
3.8.3.1	tk_cre_alm(Create Alarm Handler) .....	171
3.8.3.2	tk_del_alm>Delete Alarm Handler) .....	173
3.8.3.3	tk_sta_alm(Start Alarm Handler) .....	174
3.8.3.4	tk_stp_alm(Stop Alarm Handler) .....	175
3.8.3.5	tk_ref_alm(Refer Alarm Handler Status) .....	176
3.9	割込み管理機能のシステムコール .....	178
3.9.1	tk_def_int(Define Interrupt Handler) .....	179
3.9.2	tk_ret_int(Return from Interrupt Handler) .....	181
3.9.3	DI .....	182
3.9.4	EI .....	183
3.9.5	isDI .....	184
3.10	システム状態管理機能のシステムコール .....	185
3.10.1	tk_rot_rdq(Rotate Ready Queue) .....	186
3.10.2	tk_get_tid(Get Task Identifier) .....	188
3.10.3	tk_dis_dsp(Disable Dispatch) .....	189
3.10.4	tk_ena_dsp(Enable Dispatch) .....	191
3.10.5	tk_ref_sys(Refer System Status) .....	192
3.10.6	tk_ref_ver(Refer Version Information) .....	194
3.11	サブシステム機能のシステムコール .....	196
3.11.1	tk_def_ssy(Define Subsystem) .....	197
3.11.2	tk_ref_ssy(Refer Subsystem Status) .....	200
3.12	デバイス管理機能のシステムコール .....	201
3.12.1	tk_def_dev(Define Device) .....	202
3.12.2	tk_ref_idv(Refer Initial Device Information) .....	205
3.12.3	tk_opn_dev(Open Device) .....	206
3.12.4	tk_cls_dev(Close Device) .....	208
3.12.5	tk_rea_dev(Read Device) .....	209
3.12.6	tk_srea_dev(Synchronous Read Device) .....	211
3.12.7	tk_wri_dev(Write Device) .....	213
3.12.8	tk_swri_dev(Synchronous Write Device) .....	215
3.12.9	tk_wai_dev(Wait Device) .....	217
3.12.10	tk_sus_dev(Suspend Device) .....	219
3.12.11	tk_get_dev(Get Device) .....	221
3.12.12	tk_ref_dev(Refer Device) .....	222
3.12.13	tk_oref_dev(Refer Device) .....	224
3.12.14	tk_lst_dev(List Device) .....	226
3.12.15	tk_evt_dev(Event Device) .....	228

<b>付録</b>	.....	<b>229</b>
付録 A エラーコード一覧.....		230
付録 B 定数マクロ一覧 .....		231
付録 C デバイスドライバインタフェース.....		235
付録 D $\mu$ ITRON 仕様 OS からの移行時の注意点 .....		249
付録 E システムコール索引 .....		250
 <b>索引</b>	.....	 <b>255</b>

# 第1章

---

## 概要

**μT-REALOS の API を使用するうえでの基本的な事項について説明します。**

- 1.1 用語の説明
- 1.2 μT-REALOS API の概要
- 1.3 実装定義および実装依存仕様
- 1.4 拡張仕様

## 1.1 用語の説明

μT-REALOS で使用する基本的な用語について説明します。

### ■ μT-REALOS の基本的な用語

表 1.1-1 に、μT-REALOS で使用する基本的な用語を示します。また、以下の用語は「ユーザーズガイド」でも説明していますので、併せて参照してください。

表 1.1-1 μT-REALOS の基本的な用語一覧 (1 / 2)

用語	概要
カーネル	OS 機能を提供するプログラムをカーネルとよびます。
ユーザプログラム	μT-REALOS の機能を使用したアプリケーションプログラムを指します。ユーザ作成である点を明示するため、本書は以降、ユーザプログラムと表記します。
タスク	ユーザプログラムを構成する最も基本的な単位です。ユーザプログラムの処理は、複数のタスクの協調動作により実現します。
準タスク	タスクコンテキストを持つハンドラを準タスクとよびます。μT-Kernel 仕様では、拡張 SVC ハンドラがこの準タスクに相当します。準タスクでは、待ち状態に入ることができ、準タスク内でディスパッチできます。また、システムコールの発行制約はタスクに準じます。
非タスク	タスク以外の OS で管理する実行単位です。準タスク、割り込みハンドラ、タイムイベントハンドラ が該当します。
タスク独立部	タスクコンテキストを持たないハンドラをタスク独立部とよびます。割り込みハンドラ、タイムイベントハンドラ、初期化ルーチンが該当します。タスク独立部では、tk_slp_tsk など、タスクが待ち状態になるシステムコールは使用できません。
ディスパッチ	CPU で実行するタスクの切換えをディスパッチとよびます。また、ディスパッチを実現するカーネル内の機構をディスパッチャとよびます。
プリエンプト	ほかのプログラムが保持する CPU の実行権を奪い取る操作を指します。
API	μT-REALOS( および μT-Kernel) が提供するユーザプログラムのプログラミング作法、機能の呼出し I/F の総称です。
システムコール	OS 機能を実現し、ユーザプログラムから直接呼び出される関数群をシステムコールとよびます。
オブジェクト	カーネルが操作対象とする資源をオブジェクトとよびます。具体的には、タスクや同期・通信機能を実現するセマフォ、メールボックスなどが該当します。
コンテキスト	プログラムを実行する環境 ( 実体はプログラムの実行が中断された時点の CPU レジスタの値をメモリに保存したもの ) をコンテキストとよびます。



表 1.1-1  $\mu$ T-REALOS の基本的な用語一覧 (2 / 2)

用語	概要
C 言語対応ルーチン	C 言語で記述したプログラムを作成した場合にコンパイラで付与されるルーチンで、プログラム内で使用するレジスタの退避 / 復帰、C 言語関数の引数を関数内で参照できるように前処理を行います。 なお、 $\mu$ T-Kernel 仕様書の「高級言語対応ルーチン」は、 $\mu$ T-REALOS では「C 言語対応ルーチン」に該当します ( $\mu$ T-REALOS では、高級言語は C 言語だけのサポートのため)。
現在優先度	タスクの現時点での優先度です。ディスパッチャはこの優先度を元にタスクを切り換えます。tk_chg_pri により、ユーザプログラムから変更できます。 通常、単に「優先度」とよぶ場合は、この現在優先度を指します。
起動時優先度	タスク起動時に最初に設定する優先度です。タスク起動時に、そのタスクの現在優先度、ベース優先度は起動時優先度と同じ値に設定します。起動時優先度は、タスク生成時に指定できますが、タスク生成後には変更できません。
ベース優先度	タスクの基本となる優先度です。tk_chg_pri により、ユーザプログラムから変更できます。 現在優先度は、ミューテックス機能を使用した場合に、OS 側が一時的に変更する場合がありますが、最終的にベース優先度に戻します。 ミューテックス機能を使用しない場合は、ベース優先度と現在優先度は常に同じ値になります。
優先順位	実行可能状態のタスクが複数存在する場合、それらのタスクに実行権を与える順番を指します。優先順位が高いタスクほど、先に実行権が与えられます。 優先度が高いタスクほど優先順位は高くなり、同一優先度のタスク間では、先に実行可能状態になったタスクほど優先順位が高くなります。
システムダウン	カーネル内部でユーザシステムの処理続行が不可能と判断した場合にはシステムダウンが発生します。システムダウンでは、カーネル内部の特定番地にジャンプして、そこで無限ループすることにより、ユーザシステムの処理を停止します。
コンフィギュレータ	$\mu$ T-REALOS 独自の機能で、カーネルコンフィギュレーション (最大のタスク優先度、オブジェクトの最大数、システムのスタックサイズなどをユーザ指定できる機能) を実行するためのツールです。 詳細は、「ユーザズガイド」の「3.13 カーネルコンフィギュレーション」および「5.3 コンフィギュレーションの設定」を参照してください。

## 1.2 $\mu$ T-REALOS API の概要

---

$\mu$ T-REALOS API を構成する要素について説明します。

---

### ■ データ型

$\mu$ T-Kernel 仕様では、char, int などの汎用的な型を別名で再定義することにより、 $\mu$ T-Kernel 独自の型を定義しています。これは、データの使用方法をより厳密にし、プログラムミスを防ぐことを目的としています。データ型の詳細については、「第 2 章 データ型」を参照してください。

### ■ システムコール

ユーザプログラムは、 $\mu$ T-REALOS の提供するシステムコールを C 言語関数の形で呼び出して、各プログラムの制御やシステムの状態を参照します。ほとんどのシステムコールがこの関数の復帰値としてエラーコードを返します。

#### ● システムコールの形式

エラーコード = システムコール関数名 ( パラメータ .. );

例 ) ercd = tk\_del\_tsk(task\_id);

各システムコールの詳細については、「第 3 章 システムコールインタフェース」を参照してください。

### ■ エラーコード

システムコールの復帰値は、エラーが発生した場合には負の値のエラーコード、処理を正常に終了した場合には E\_OK または正の値となります。

システムコールが返すエラーコードについては、「第 3 章 システムコールインタフェース」を参照してください。また、 $\mu$ T-REALOS で規定されたエラーコードの一覧とその意味については、「付録 A エラーコード一覧」を参照してください。

## 1.3 実装定義および実装依存仕様

実装定義および実装依存仕様について説明します。

### ■ 実装定義

μT-Kernel 仕様の OS において、各 OS の実装ごとに定義しなければならない仕様を実装定義とよびます。実装定義部分の仕様については、μT-Kernel 仕様の OS 間でも互換性が保証されないため注意してください。

μT-REALOS での実装定義を表 1.3-1 に示します。

表 1.3-1 μT-REALOS 実装定義一覧 (1 / 3)

分類	実装定義項目	内容
動作	システムコールをアセンブラで記述されたプログラムから呼び出す場合のインタフェース	アセンブラから C 関数を呼び出す時の手順に従います。詳細は「FR ファミリ SOFTUNE C/C++ コンパイラマニュアル」(以降、「コンパイラマニュアル」とよびます)の「第 4 章 fcc911s コマンドのオブジェクトプログラムの構成」を参照してください。
	システムコールの実装方法	関数呼出し(ソフトウェアトラップは未使用)です。
	特定のシステムコール* をタスク部以外から呼び出した場合の動作	表 3.1-1 および「第 3 章 システムコールインタフェース」を参照してください。
	呼び出した処理に復帰しないシステムコール(tk_ext_tsk, tk_exd_tsk, tk_ret_int) でエラーが発生した場合の動作	システムコールのエラールーチンが登録済みの場合には、tk_ret_int を除いて、そのエラールーチンを呼び出します。tk_ret_int については、その動作を保証しません。エラールーチンが登録されていない場合の動作は保証しません。
	システムコールの機能コード	μT-REALOS では機能コードは未使用です。
	メールボックスのメッセージヘッダ構造	「3.5.3.3 tk_snd_mbx(Send Message to Mailbox) 解説」を参照してください。
	ランデブ番号の付与方法	下位 16 ビットがタスク ID, 上位 16 ビットがランデブ受け順に採番する一意の番号となります。
	TA_ASM 属性のタイムイベントハンドラ	μT-REALOS ではサポートしていません。

表 1.3-1  $\mu$ T-REALOS 実装定義一覧 (2 / 3)

分類	実装定義項目	内容
動作	複数の割込みハンドラやタイムイベントハンドラの同時起動	<p>タイムイベントハンドラが同時に起動した場合、1つのタイムイベントハンドラの実行が終了してから、別のタイムイベントハンドラが実行されます。</p> <p>割込みハンドラ実行中に、より優先度の高い割込みが発生した場合、実行中の割込みハンドラの処理を中断し、優先度の高い割込みハンドラを実行します。その後、中断した割込みハンドラの処理に戻ります。</p> <p>タイムイベントハンドラ実行中に、タイマ割込みより優先度の高い割込みが発生した場合、実行中の割込みハンドラの処理を中断し、優先度の高い割込みハンドラを実行します。その後、中断したタイムイベントハンドラの処理に戻ります。</p> <p>タイマ割込みより優先度の低い割込みハンドラ実行中にタイムイベントハンドラが起動された場合、実行中の割込みハンドラの処理を中断し、タイムイベントハンドラを実行します。その後、中断した割込みハンドラの処理に戻ります。</p>
	TA_ASM 属性の割込みハンドラが起動した時点の状態 (システムコール呼出しの可否など)	タスク独立部として扱います。
	割込みハンドラ定義解除後に割込みが発生した場合の動作	システムダウンが発生します。
	TA_ASM 属性の割込みハンドラに入った時点のスタックやレジスタの状態、システムコールの呼出しの可否、システムコールを呼び出す方法、および割込みハンドラから OS を介さずに復帰する方法	「ユーザーズガイド 4.8 割込みハンドラ」を参照してください。
	T_REGS, T_EIT, T_CREGS の内容	「3.3.8 tk_get_reg(Get Task Registers)」および「3.3.9 tk_set_reg(Set Task Registers)」を参照してください。
	CPU 割込み制御機能 (DI, EI, isDI) のパラメータ (iststs) 内容	「3.9.3 DI」、「3.9.4 EI」および「3.9.5 isDI」を参照してください。
	割込みハンドラの第 2 引数	未使用です。
	デバイスの事象通知用メッセージバッファの使用	使用します。
	タイムイベントハンドラから tk_ret_int を呼び出した場合の E_CTX のエラーチェック	エラーチェックは行いません。
	E_OACV のエラー対象となるシステムオブジェクトの定義、およびエラー発生条件	E_OACV のエラー対象となるオブジェクトとしてデバイスを定義しています。また、本エラーの発生条件については、「3.12 デバイス管理機能のシステムコール」を参照してください。

表 1.3-1 μT-REALOS 実装定義一覧 (3 / 3)

分類	実装定義項目	内容
数値	tk_wup_tsk の起床要求キューイング数最大値	32767(0x7fff)
	tk_sus_tsk の強制待ち要求のネスト数最大値	32767(0x7fff)
	セマフォカウントの最大値	2147483647(0x7fffffff)
	使用可能なサブシステム ID の最大値	「ユーザーズガイド 3.13 カーネルコンフィギュレーション」を参照してください。
	指定可能なサブシステムの優先度 (ssypri) の最大値	16
	tk_ref_ssy での ssypri, resblksz の値	不定値
	tk_def_int の引数, dintno の値	割込みベクタ番号
	デバイスのサスペンド禁止要求カウントの最大値	2147483647(0x7fffffff)

\*: タスク部以外からの呼出しが可能と明記されていないもの

## ■ 実装依存

μT-Kernel仕様において、ハードウェアの相違点によりOSの振舞いが変わる部分の仕様を実装依存とよびます。実装依存部の仕様については、μT-Kernel 仕様の OS 間でも互換性が保証されないため注意してください。

μT-REALOS の実装依存項目を表 1.3-2 に示します。

表 1.3-2 μT-REALOS 実装依存一覧

分類	実装依存項目	内容
定義	TA_ASM 属性が指定された場合のタスクの形式	μT-REALOS ではサポートしていません。
動作	E_MACV のエラー検出	μT-REALOS では、本エラーは未使用です。
	実装依存で発生する E_CTX エラー	「第 3 章 システムコールインタフェース」の各システムコールの説明において、発生する E_CTX のエラーについてすべて記載しています。

## 1.4 拡張仕様

μT-REALOS には、μT-Kernel 仕様を拡張している部分があります。これらの拡張仕様について説明します。

### ■ 拡張仕様

μT-REALOS では、μT-Kernel 仕様の機能はデバッガサポート機能を除きすべてサポートしていますが、そのほかに拡張している機能があります。

μT-Kernel 仕様を拡張している項目を表 1.4-1 に示します。

表 1.4-1 μT-REALOS 拡張定義一覧

拡張定義項目	定義内容	
初期化ルーチン	μT-REALOS 独自の拡張機能です。タスクの起動前に実行するプログラムを登録します。機能の詳細は、「ユーザーズガイド 2.2.1 タスク」を参照してください。	
	初期化ルーチンが登録できる最大数	1
エラールーチン	μT-REALOS 独自の拡張機能です。カーネルが何らかのエラーを検出したときに起動するプログラムを登録します。機能の詳細は、「ユーザーズガイド 2.3.1 タスク独立部」を参照してください。	
	エラールーチンが登録できる最大数	1
isig_tim	μT-REALOS 独自の拡張機能です。ユーザプログラムからシステム時刻の更新を行うための機能です。詳細は「3.8.1.4 isig_tim(Signal Time)」を参照してください。	
割込みハンドラ静的登録機能	μT-REALOS 独自の拡張機能で、割込みハンドラを静的に登録する機能です。機能の詳細は、「ユーザーズガイド 3.12 カーネルコンフィギュレーション」および「ユーザーズガイド 5.2 カーネルコンフィギュレーション」を参照してください。	

# 第2章

---

## データ型

**μT-REALOS で規定される C 言語のデータ型について説明します。**

2.1 汎用的なデータ型と定数マクロ

2.2 μT-Kernel 固有の意味を持つデータ型と定数マクロ

## 2.1 汎用的なデータ型と定数マクロ

μT-Kernel 仕様で規定されている汎用的なデータ型および定数マクロについて説明します。

### ■ 汎用的なデータ型

汎用的なデータ型と定数マクロは、「2.2 μT-Kernel 固有の意味を持つデータ型と定数マクロ」で説明するデータ型を定義するための基本的なデータ型と定数マクロです。表 2.1-1 および表 2.1-2 に、汎用的なデータ型の一覧を示します。

表 2.1-1 データ型一覧

型名	C 言語での型	意味
B	signed char	符号付き 8 ビット整数
H	signed short	符号付き 16 ビット整数
W	signed long	符号付き 32 ビット整数
UB	unsigned char	符号なし 8 ビット整数
UH	unsigned short	符号なし 16 ビット整数
UW	unsigned long	符号なし 32 ビット整数
VB	signed char	型が一定しない 8 ビットのデータ
VH	signed short	型が一定しない 16 ビットのデータ
VW	signed long	型が一定しない 32 ビットのデータ
VP	void*	型が一定しないデータへのポインタ
_B	volatile signed char	volatile 宣言付き
_H	volatile signed short	
_W	volatile signed long	
_UB	volatile unsigned char	
_UH	volatile unsigned short	
_UW	volatile unsigned long	
INT	signed int	CPU のビット幅の符号付き整数
UINT	unsigned int	CPU のビット幅の符号なし整数
ID	signed int	ID 一般
MSEC	signed long	時間一般 (ms)
(*FP)()	void	関数アドレス一般
(*FUNCP)()	signed in	関数アドレス一般
BOOL	signed int	ブール値
TC	unsigned short	TRON 文字コード



表 2.1-2 定数マクロ一覧

マクロ名	値	意味
LOCAL	static	ローカルシンボル定義
EXPORT	(なし)	グローバルシンボル定義
IMPORT	extern	グローバルシンボル参照
TRUE	1	ブール値の真
FALSE	0	ブール値の偽
TNULL	((TC)0)	TRON コード文字列の終端

(補足事項)

VB, VH, VW と B, H, W との相違点は、前者はビット数だけが分かっており、データ型の中身が分からないものを表すのに対して、後者は整数を表すことがはっきりしている点です。

タスクのスタックサイズなど、明らかに負の数にならないパラメータも原則として符号付き整数 (INT, W) のデータ型となっています。これは、整数はできる限り符号付きの数として扱う  $\mu$ T-Kernel 仕様のルールに基いたものです。また、タイムアウト (TMO tmount)のパラメータでは、これが符号付きの整数であることを利用してTMO\_FEVR(=-1)を特殊な意味に使っています。ビットパターンとして扱われるパラメータには符号なしのデータ型があります。

## 2.2 μT-Kernel 固有の意味を持つデータ型と定数マクロ

μT-Kernel 固有の意味を持つデータ型と定数マクロについて説明します。

### ■ μT-Kernel 固有の意味を持つデータ型の一覧

μT-Kernel 固有の意味を持つデータ型と定数マクロは、パラメータの意味を明確にするために定義されたデータ型と定数マクロです。

表 2.2-1 から表 2.2-3 に出現頻度の高いデータ型と定数マクロの一覧を示します。

なお、すべての定数マクロの一覧は「付録 B 定数マクロ一覧」を参照してください。

表 2.2-1 固有の意味を持つデータ型一覧 (構造体以外)

型名	データ型	意味
FN	INT	機能コード
RNO	INT	ランデブ番号
ATR	UW	オブジェクト / ハンドラ属性
ER	INT	エラーコード
PRI	INT	優先度
TMO	W	タイムアウト指定
RELTIM	UW	相対時間

表 2.2-2 固有の意味を持つデータ型一覧 (構造体)

型名	メンバ	データ型	意味
SYSTIM	hi	W	システム時刻 上位 32 ビット
	lo	UW	システム時刻 下位 32 ビット

表 2.2-3 固有の意味を持つ定数マクロ一覧

マクロ名	値	意味
NULL	0	無効ポインタ
TA_NULL	0	特別な属性を指定しない
TMO_POL	0	ポーリング
TMO_FEVR	(-1)	永久待ち
TSK_SELF	0	自タスク
TPRI_INI	0	起動時優先度
TPRI_RUN	0	実行状態のタスクの優先度

( 補足事項 )

複数のデータ型を複合的に使用するパラメータには、代表的なデータ型を使用します。  
例えば、`tk_cre_tsk` の復帰値はタスク ID かエラーコードですが、主となるのはタスク ID  
なのでデータ型は ID となります。



# 第3章

---

## システムコール インタフェース

μT-REALOS でサポートしている μT-Kernel 仕様のシステムコールのインタフェースを説明します。

- 3.1 システムコール一覧
- 3.2 システムコール説明内容
- 3.3 タスク管理機能のシステムコール
- 3.4 タスク付属同期機能のシステムコール
- 3.5 同期・通信機能のシステムコール
- 3.6 拡張同期・通信機能のシステムコール
- 3.7 メモリプール管理機能のシステムコール
- 3.8 時間管理機能のシステムコール
- 3.9 割込み管理機能のシステムコール
- 3.10 システム状態管理機能のシステムコール
- 3.11 サブシステム機能のシステムコール
- 3.12 デバイス管理機能のシステムコール

## 3.1 システムコール一覧

μT-REALOS でサポートしているシステムコールについて説明します。

### ■ システムコール一覧

表 3.1-1 に μT-REALOS が提供しているシステムコールの一覧を示します。

各システムコールの詳細については、「3.2 システムコール説明内容」以降を参照してください。

表 3.1-1 システムコール一覧 (1 / 6)

分類	名称	説明	呼出し可否		
			タスク部 <sup>*1</sup>	タスク独立部	ディスパッチ禁止状態
タスク管理機能	tk_cre_tsk	タスクを生成します。			
	tk_del_tsk	タスクを削除します。			
	tk_sta_tsk	タスクを起動します。			
	tk_ext_tsk	自タスクを終了します。		×	×
	tk_exd_tsk	自タスクを終了して削除します。		×	×
	tk_ter_tsk	他タスクを強制終了します。			
	tk_chg_pri	タスクの優先度を変更します。			
	tk_get_reg	タスクレジスタを獲得します。		×	
	tk_set_reg	タスクレジスタを設定します。		×	
	tk_ref_tsk	タスクの状態を参照します。			
タスク付属同期機能	tk_slp_tsk	自タスクを起床待ち状態へ移行します。		×	×
	tk_wup_tsk	他タスクを起床します。			
	tk_can_wup	タスクの起床要求をキャンセルします。			
	tk_rel_wai	他タスクの待ち状態を強制解除します。			
	tk_sus_tsk	タスクを強制待ち状態に移行します。			*2
	tk_rsm_tsk	強制待ち状態のタスクを再開します。			
	tk_frsm_tsk	強制待ち状態のタスクを強制再開します。			

表 3.1-1 システムコール一覧 (2 / 6)

分類	名称	説明	呼出し可否		
			タスク部 <sup>*1</sup>	タスク独立部	ディスパッチ禁止状態
タスク 付属 同期機能	tk_dly_tsk	自タスクを時間経過待ち状態へ移行します。		×	×
同期・ 通信機能	tk_cre_sem	セマフォを生成します。			
	tk_del_sem	セマフォを削除します。			
	tk_sig_sem	セマフォ資源を返却します。			
	tk_wai_sem	セマフォ資源を獲得します。		×	×
	tk_ref_sem	セマフォの状態を参照します。			
	tk_cre_flg	イベントフラグを生成します。			
	tk_del_flg	イベントフラグを削除します。			
	tk_set_flg	イベントフラグをセットします。			
	tk_clr_flg	イベントフラグをクリアします。			
	tk_wai_flg	イベントフラグを待ちます。		×	×
	tk_ref_flg	イベントフラグの状態を参照します。			
	tk_cre_mbx	メールボックスを作成します。			
	tk_del_mbx	メールボックスを削除します。			
	tk_snd_mbx	メールボックスへ送信します。			
	tk_rcv_mbx	メールボックスから受信します。		×	×
	tk_ref_mbx	メールボックスの状態を参照します。			
拡張同期 通信機能	tk_cre_mtx	ミューテックスを生成します。			
	tk_del_mtx	ミューテックスを削除します。			
	tk_loc_mtx	ミューテックスをロックします。		×	×
	tk_unl_mtx	ミューテックスのロックを解除します。		×	
	tk_ref_mtx	ミューテックスの状態を参照します。			
	tk_cre_mbf	メッセージバッファを生成します。			

表 3.1-1 システムコール一覧 (3 / 6)

分類	名称	説明	呼出し可否		
			タスク部 <sup>*1</sup>	タスク独立部	ディスパッチ禁止状態
拡張同期通信機能	tk_del_mbf	メッセージバッファを削除します。			
	tk_snd_mbf	メッセージバッファへ送信します。			
	tk_rcv_mbf	メッセージバッファから受信します。		×	×
	tk_ref_mbf	メッセージバッファの状態を参照します。			
	tk_cre_por	ランデブポートを生成します。		×	
	tk_del_por	ランデブポートを削除します。		×	
	tk_cal_por	ランデブポートに対するランデブを呼び出します。		×	×
	tk_acp_por	ランデブポートに対するランデブを受け付けます。		×	×
	tk_fwd_por	ランデブポートに対するランデブを回送します。		×	
	tk_rpl_rdv	ランデブポートに対するランデブに返答します。		×	
	tk_ref_por	ランデブポートの状態を参照します。			
メモリプール管理機能	tk_cre_mpf	固定長メモリプールを生成します。		×	×
	tk_del_mpf	固定長メモリプールを削除します。		×	×
	tk_get_mpf	固定長メモリブロックを獲得します。		×	×
	tk_rel_mpf	固定長メモリブロックを返却します。		×	×
	tk_ref_mpf	固定長メモリプールの状態を参照します。		×	×
	tk_cre_mpl	可変長メモリプールを生成します。		×	×
	tk_del_mpl	可変長メモリプールを削除します。		×	×
	tk_get_mpl	可変長メモリブロックを獲得します。		×	×



表 3.1-1 システムコール一覧 (4 / 6)

分類	名称	説明	呼出し可否		
			タスク部 <sup>*1</sup>	タスク独立部	ディスパッチ禁止状態
メモリプール管理機能	tk_rel_mpl	可変長メモリブロックを返却します。		×	×
	tk_ref_mpl	可変長メモリプールの状態を参照します。		×	×
時間管理機能	tk_set_tim	システム時刻を設定します。			
	tk_get_tim	システムの現在時刻を参照します。			
	tk_get_otm	システム稼働時間を参照します。			
	isig_tim	タイムティックを供給します。 <sup>*3</sup>	×		
	tk_cre_cyc	周期ハンドラを生成します。			
	tk_del_cyc	周期ハンドラを削除します。			
	tk_sta_cyc	周期ハンドラの動作を開始します。			
	tk_stp_cyc	周期ハンドラの動作を停止します。			
	tk_ref_cyc	周期ハンドラの状態を参照します。			
	tk_cre_alm	アラームハンドラを生成します。			
	tk_del_alm	アラームハンドラを削除します。			
	tk_sta_alm	アラームハンドラの動作を開始します。			
	tk_stp_alm	アラームハンドラの動作を停止します。			
	tk_ref_alm	アラームハンドラの状態を参照します。			

表 3.1-1 システムコール一覧 (5 / 6)

分類	名称	説明	呼出し可否		
			タスク部 <sup>*1</sup>	タスク独立部	ディスパッチ禁止状態
割込み管理機能	tk_def_int	割込みハンドラを定義します。			
	tk_ret_int	割込みハンドラから復帰します。	×		
	DI	すべての外部割込みを禁止します。			
	EI	すべての外部割込みを許可します。			
	isDI	外部割込み禁止状態を調べます。			
システム状態管理機能	tk_rot_rdq	タスクの優先順位を回転します。			
	tk_get_tid	実行状態タスクのタスク ID を参照します。			
	tk_dis_dsp	ディスパッチを禁止します。		×	
	tk_ena_dsp	ディスパッチを許可します。		×	
	tk_ref_sys	システム状態を参照します。			
	tk_ref_ver	バージョンを参照します。			
サブシステム管理機能	tk_def_ssy	サブシステムを定義します。			
	tk_ref_ssy	サブシステム定義情報を参照します。			
デバイス管理機能	tk_opn_dev	デバイスをオープンします。			
	tk_cls_dev	デバイスをクローズします。			
	tk_rea_dev	デバイスの読出しを開始します。			
	tk_srea_dev	デバイスの同期読出しを開始します。			
	tk_wri_dev	デバイスの書込みを開始します。			
	tk_swri_dev	デバイスの同期書込みを開始します。			
	tk_wai_dev	デバイスの要求完了を待ちます。			
	tk_sus_dev	デバイスをサスペンドします。			
	tk_get_dev	デバイスのデバイス名を獲得します。			

表 3.1-1 システムコール一覧 (6 / 6)

分類	名称	説明	呼出し可否		
			タスク部 <sup>*1</sup>	タスク独立部	ディスパッチ禁止状態
デバイス管理機能	tk_ref_dev	デバイスのデバイス情報を獲得します。			
	tk_oref_dev	デバイスのデバイス情報を獲得します。			
	tk_lst_dev	登録済みデバイスの一覧を獲得します。			
	tk_evt_dev	デバイスにドライバ要求イベントを送信します。			
	tk_def_dev	デバイスを登録します。			
	tk_ref_idv	デバイス初期情報を獲得します。			

○ : 呼出し可能    × : 呼出し不可 (システムコールがエラー復帰または動作を保証しない)

\*1: 「呼出し可否」の「タスク部」は「準タスク部」を含みます。

\*2: ディスパッチ禁止状態で実行状態のタスクを指定して tk\_sus\_tsk を呼び出した場合には E\_CTX エラーになります。

\*3:  $\mu$ T-REALOS 固有の機能 ( $\mu$ T-Kernel 仕様では規定されていません。 $\mu$ T-REALOS で追加した機能です。)

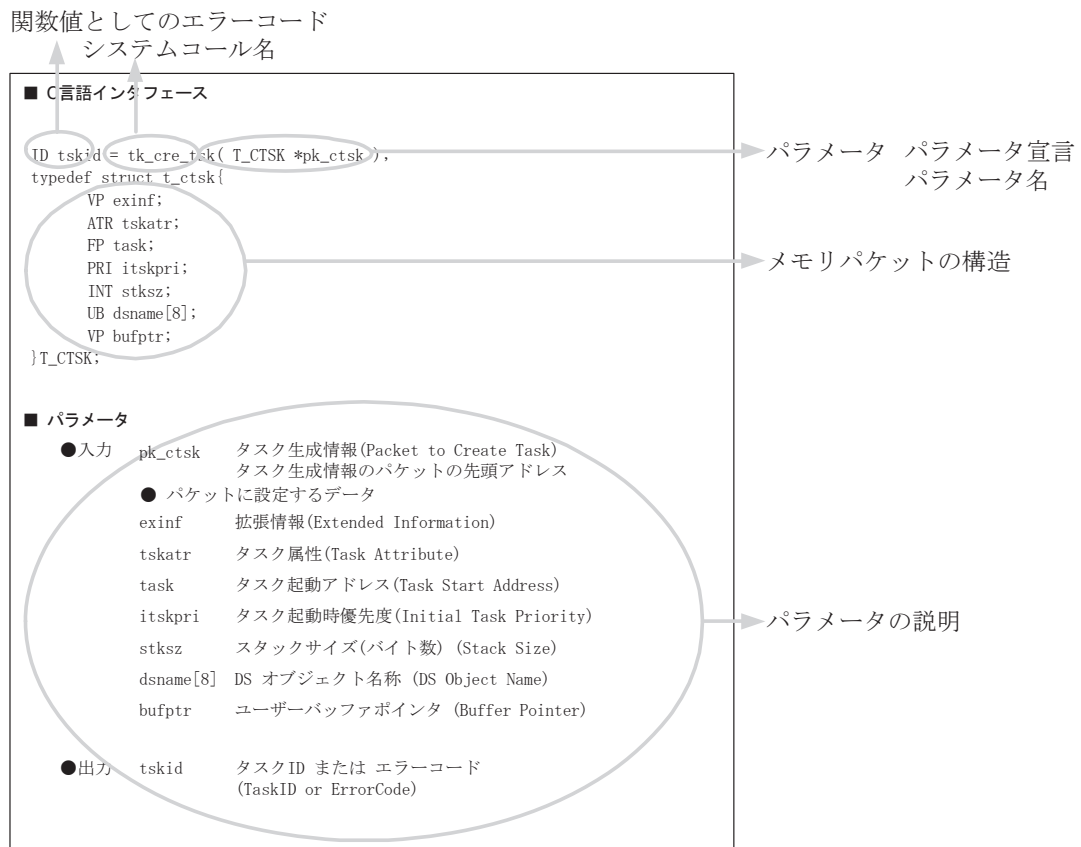
## 3.2 システムコール説明内容

「3.3 タスク管理機能のシステムコール」から「3.12 デバイス管理機能のシステムコール」までのシステムコールの内容説明について読み方を説明します。

### ■ システムコール説明内容

システムコール名		機能		英文	
機能		和文			
タスク部	○	タスク独立部	○	ディスパッチ禁止状態	○
↑		↑		↑	
タスク部から呼出し可能かどうかを表わす ○：呼出し可能 ×：呼出し禁止		タスク独立部から呼出し可能かどうかを表わす ○：呼出し可能 ×：呼出し禁止		ディスパッチ禁止状態から呼出し可能かどうかを表わす ○：呼出し可能 ×：呼出し禁止	

### 〈C言語インタフェース〉



## 〈エラーコード〉

### ■ エラーコード

E_NOMEM	-33	メモリ不足 (管理ブロックやスタック用の領域を確保できない)
E_LIMIT	-34	タスクの数がシステムの制限を超えた
E_RSATR	-11	予約属性 (tskatr が不正あるいは使用できない)
E_PAR	-17	パラメータエラー (pk_ctsk が不正, task, bufptr が不正, itskpri が不正)

このシステムコール発行時に発生する可能性のあるエラー

## 〈ディスパッチ要因〉

### ■ ディスパッチ要因

このシステムコールではディスパッチしません。

このシステムコールでディスパッチが発生する要因または発生の有無

## 〈解説〉

### ■ 解説

タスクを生成してタスクIDを割り当てます。具体的には、生成するタスクに対してTCB(Task Control Block)を割り付け、itskpri, task, stkszなどの情報をもとにその初期設定を行います。

対象タスクは生成後、休止状態(DORMANT)となります。

itskpriによって、タスクが起動するときの優先度の初期値を指定します。タスク優先度としては、1~140の値を指定することができ、数の小さい方が高い優先度となります。

exinfは、対象タスクに関する情報を入れておくためにユーザが自由に使用できます。ここで設定した情報は、タスクに起動パラメータとして渡されるほか、tk\_ref\_tskで取り出すことができます。

なお、ユーザの情報を入れるためにもっと大きな領域が必要な場合や、途中で内容を変更したい場合には、自分でそのためのメモリを確保し、そのメモリパケットのアドレスをexinfに入れます。OSではexinfの内容について関知しません。

tskatrは、下位側がシステム属性を表し、上位側が実装独自属性を表します。tskatrのシステム属性の部分では、次のように指定します。

このシステムコールの処理に対する解説、注意事項

## 3.3 タスク管理機能のシステムコール

---

タスク管理機能のシステムコールについて説明します。

---

### ■ タスク管理機能のシステムコール

タスク管理機能は、以下の 10 種類のシステムコールで構成されます。

- tk\_cre\_tsk(Create Task)
- tk\_del\_tsk>Delete Task)
- tk\_sta\_tsk(Start Task)
- tk\_ext\_tsk(Exit Task)
- tk\_exd\_tsk(Exit and Delete Task)
- tk\_ter\_tsk(Terminate Task)
- tk\_chg\_pri(Change Task Priority)
- tk\_get\_reg(Get Task Registers)
- tk\_set\_reg(Set Task Registers)
- tk\_ref\_tsk(Refer Task Status)

### 3.3.1 tk\_cre\_tsk(Create Task)

タスクを生成します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ID tskid = tk_cre_tsk ( T_CTSK *pk_ctsk ) ;

typedef struct t_ctsk {
    VP exinf;
    ATR tskatr;
    FP task;
    PRI itskpri;
    INT stksz;
    VP bufptr;
} T_CTSK;
```

■ パラメータ

● 入力

- pk\_ctsk      タスク生成情報 (Packet of Create Task)  
                 タスク生成情報のパケットの先頭アドレス  
                 パケットに設定するデータ
- exinf          拡張情報 (Extended Information)
- tskatr        タスク属性 (Task Attribute)  
                 tskatr := TA\_HLNG | [TA\_USERBUF]

属性	値	意味
TA_HLNG	0x00000001	対象タスクが C 言語で記述
TA_USERBUF	0x00000020	スタック領域としてユーザが指定した領域を使用する

- task            タスク起動アドレス (Task Start Address)
- itskpri        タスク起動時優先度 (Initial Task Priority)
- stksz          スタックサイズ ( バイト数 ) (Stack Size)
- bufptr        ユーザバッファのアドレス (Buffer Pointer)

● 出力

- tskid            タスク ID(TaskID) またはエラーコード (ErrorCode)

## ■ エラーコード

E_NOMEM	-33	メモリ不足 (タスクスタック用の領域を確保できない)
E_LIMIT	-34	タスクの数がシステムの制限より大きい
E_RSATR	-11	予約属性 (tskatr に未定義の値が設定された)
E_PAR	-17	パラメータエラー (itskpri が 0 以下, またはシステムの上限值より大きい, stksz が最小スタックサイズより小さい stksz が 8 の倍数でない (TA_USERBUF 設定時))

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

タスクを生成してタスク ID を割り当てます。具体的には, itskpri, tsk, stksz の情報を元に生成するタスクを初期設定します。タスクは生成後, 休止状態となります。

itskpri によって, タスクが起動するときの優先度の初期値を指定します。タスク優先度としては, 1 からシステムの最大優先度 (コンフィギュレータで設定した最大優先度) の値を指定でき, 数の小さい方が高い優先度となります。itskpri が 0 以下, またはシステムの最大優先度よりも大きい場合には E\_PAR のエラーで復帰します。

exinf は, 対象タスクに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報は, タスクに起動パラメータとして渡すほか, tk\_ref\_tsk で取り出せます。

なお, ユーザの情報を入れるためにもっと大きな領域が必要な場合や, 途中で内容を変更したい場合には, ユーザプログラムでそのためのメモリを確保して, そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容について関知しません。

tskatr は, タスクの属性を指定します。以下の値は無視します。また, tskatr に未定義の属性が指定された場合には, E\_RSATR のエラーで復帰します。

属性	値	意味
TA_ASM	0x00000000	対象タスクがアセンブラで記述
TA_RNG0	0x00000000	保護レベル 0 で実行
TA_RNG1	0x00000100	保護レベル 1 で実行
TA_RNG2	0x00000200	保護レベル 2 で実行
TA_RNG3	0x00000300	保護レベル 3 で実行

μT-Kernel 仕様では, タスクを C 言語で記述した場合に TA\_HLNG の設定を規定しています。ただし, μT-REALOS においては, タスクの記述は C 言語だけをサポートしているため, TA\_ASM が指定された場合 (TA\_HLNG の指定がない場合) でも, タスクは C 言語で記述されているものとして処理します。μT-Kernel 仕様の互換性の観点から TA\_HLNG を必ず指定してください。



各タスクにはスタックが1本あります。TA\_USERBUF が指定された場合に bufptr が有効になり、bufptr を先頭とする stksz バイトのメモリ領域をスタック領域として使用します。この場合、スタックは OS では用意されません。TA\_USERBUF が指定されなかった場合は、bufptr は無視され、スタック領域は OS が確保します。OS が確保する場合、stksz は 8 バイトの倍数に切り上げて獲得されます。カーネルのメモリプール領域にスタック領域を確保するだけの十分な空き領域がない場合には E\_NOMEM のエラーで復帰します。stksz が最小スタックサイズ (80 バイト) より小さい場合または tskatr に TA\_USERBUF が設定されている場合で、4 の倍数でない場合には E\_PAR のエラーで復帰します。

タスクがシステムの上限值 (コンフィギュレータで設定した最大タスク数) まで作成されている状態で、本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。また、pk\_ctk, tsk, bufptr が不正な場合でも、エラーチェックを行わず動作は保証されません。

## ■ 補足事項

μT-REALOS のタスクスタックはレジスタ退避用に最低 80 バイトの領域が必要です。また、タスクスタックサイズの求め方およびタスクの記述方法については、「ユーザーズガイド 4.9 タスク」を参照してください。

関数からの単純なリターン (return) でタスクを終了するとその後のシステムの動作は保証されません。必ず、tk\_ext\_tsk または tk\_exd\_tsk を呼び出してから終了してください。

### 3.3.2 tk\_del\_tsk(Delete Task)

---

タスクを削除します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tskid          タスク ID (Task ID)

##### ● 出力

ercd          エラーコード (ErrorCode)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が 0 以下または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが休止状態でない)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

tskid で示されたタスクを削除します。具体的には,tskid で指定されたタスクを休止状態から未登録状態に移行させて,スタック領域およびタスク ID 番号を未使用状態にします。

tskid で指定されたタスク ID が 0 以下または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスク ID のタスクが存在しない場合には E\_NOEXS のエラーで復帰します。

本システムコールで自タスクの指定はできません。自タスクが指定された場合には,自タスクが休止状態ではないため,E\_OBJ のエラーで復帰します。自タスクを削除するには tk\_exd\_tsk を呼び出してください。

### 3.3.3 tk\_sta\_tsk (Start Task)

タスクを起動します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

● 入力

tskid          タスク ID (Task ID)  
stacd          タスク起動コード (Start Code)

● 出力

ercd          エラーコード (ErrorCode)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が 0 以下または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが休止状態でない)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクを起動した場合には、起動したタスクにディスパッチします。

#### ■ 解説

tskid で示されたタスクを起動します。具体的には、休止状態から実行可能状態へと移行します。

tskid で指定されたタスク ID が 0 以下または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスク ID のタスクが存在しない場合には E\_NOEXS のエラーで復帰します。

stacd により、タスクの起動時にタスクに渡すパラメータを設定できます。このパラメータは、対象タスクから参照できて簡単なメッセージ通信の目的で使用できます。

タスク起動時のタスク優先度は、対象タスクが生成されたときに指定されたタスク起動時優先度となります。

本システムコールによる起動要求のキューイングは行いません。すなわち、休止状態以外のタスクを対象として、本システムコールが呼び出された場合には E\_OBJ のエラーで復帰します。

### 3.3.4 tk\_ext\_tsk(Exit Task)

---

自タスクを終了します。

---

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
void tk_ext_tsk ( void ) ;
```

#### ■ パラメータ

##### ● 入力

なし

##### ● 出力

なし

(注意事項) システムコールを呼び出したコンテキストには復帰しない

#### ■ エラーコード

(注意事項) 以下のエラーを検出する可能性があります, エラーを検出した場合でもシステムコールを呼び出したコンテキストには復帰しません。したがって, システムコールの復帰値として直接エラーコードを返せません。

E\_CTX        -25     コンテキストエラー  
(タスク独立部またはディスパッチ禁止状態で実行)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクが終了した場合には, 次の優先順位のタスクにディスパッチします。

#### ■ 解説

自タスクを正常終了させて, 休止状態へと移行させます。

本システムコールにより, タスクが休止状態になった場合にはタスク優先度は起動時優先度に戻ります。

#### ■ 補足事項

本システムコールによるタスクの終了時に, 終了するタスクがそれ以前に獲得した資源(メモリブロック, セマフォなど)を自動的に解放しません。このため, タスク終了により未使用となる資源をタスク終了前に解放してください。

本システムコールは呼出し元のコンテキストに復帰しないシステムコールです。タスク独立部, またはディスパッチ禁止状態で本システムコールが呼び出された場合にはOS内部でエラーを検出します。しかし, その場合でも呼出し元には復帰せず, その後の動作は保証されません。

### 3.3.5 tk\_exd\_tsk(Exit and Delete Task)

自タスクを終了して削除します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
void tk_exd_tsk ( void ) ;
```

#### ■ パラメータ

##### ● 入力

なし

##### ● 出力

なし

(注意事項) システムコールを呼び出したコンテキストには復帰しない

#### ■ エラーコード

(注意事項) 以下のエラーを検出する可能性があります, エラーを検出した場合でもシステムコールを呼び出したコンテキストには復帰しません。したがって, システムコールの復帰値として直接エラーコードを返せません。

E\_CTX        -25     コンテキストエラー  
(タスク独立部またはディスパッチ禁止状態で実行)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクが終了した場合には, 次の優先順位のタスクにディスパッチします。

#### ■ 解説

自タスクを正常終了させて, さらに自タスクを削除します。具体的には, 自タスクを未登録状態に移行させます。

#### ■ 補足事項

本システムコールによるタスクの終了時には, 終了するタスクがそれ以前に獲得した資源(メモリブロック, セマフォなど)を自動的に解放しません。このため, タスク終了により未使用となる資源をタスク終了前に解放してください。

本システムコールは呼出し元のコンテキストに復帰しないシステムコールです。タスク独立部, またはディスパッチ禁止状態で本システムコールが呼び出された場合にはOS内部でエラーを検出します。しかし, その場合でも呼出し元には復帰せず, その後の動作は保証されません。

### 3.3.6 tk\_ter\_tsk(Terminate Task)

---

他タスクを強制終了します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tskid          タスク ID (Task ID)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が 0 以下または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが休止状態または自タスク)

#### ■ ディスパッチ要因

セマフォ獲得待ち、メッセージバッファの送信待ちや、可変長メモリプールのメモリブロック獲得待ちの待ち行列で、待ち行列先頭のタスクが本システムコールにより強制終了させられた場合には、次のタスクの待ち状態を解除する場合があります。待ち状態が解除されたタスクの優先度が本システムコールを呼び出したタスクより高い場合には待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

tskid で示されたタスクを強制的に終了させます。具体的には、tskid で示された対象タスクを休止状態に移行させます。タスクが休止状態に戻るときは、優先度は起動時優先度に戻ります。

tskid で指定されたタスク ID が 0 以下か最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスク ID のタスクが存在しない場合には E\_NOEXS のエラーで復帰します。

対象タスクが待ち状態 (強制待ち状態を含む) の場合でも、対象タスクは待ち解除となって終了します。また、対象タスクが何らかの待ち行列 (セマフォ待ちなど) につながっていた場合には、本システムコールの実行によってその待ち行列から削除します。本システムコールでは自タスクの指定はできません。自タスクが指定された場合には E\_OBJ のエラーで復帰します。

本システムコールの対象タスクの状態と実行結果との関係についてまとめたものを表 3.3-1 に示します。

**表 3.3-1 tk\_ter\_tsk の対象タスクの状態と実行結果**

対象タスク状態	tk_ter_tsk の ercd	処理
実行できる状態 (RUNNING, READY) ( 自タスク以外 )	E_OK	強制終了処理
実行状態 (RUNNING)( 自タスク )	E_OBJ	何もしない
待ち状態 (WAITING, SUSPENDED, WAITING-SUSPENDED)	E_OK	強制終了処理
休止状態 (DORMANT)	E_OBJ	何もしない
未登録状態 (NON-EXISTENT)	E_NOEXS	何もしない

#### ■ 補足事項

tk\_ter\_tsk によるタスクの終了時に、終了するタスクがそれ以前に獲得した資源（メモリブロック、セマフォなど）は自動的に解放されません。このため、tk\_ter\_tsk でタスクを終了させる場合にはユーザプログラムによりそのタスクが獲得している資源を解放させてください。

tk\_ter\_tsk によるタスクの強制終了は対象タスクの実行状態とは関係なく行うため、システム全体に悪影響を及ぼす場合があります。このため、タスクを強制終了させる場合には注意してください。

### 3.3.7 tk\_chg\_pri(Change Task Priority)

タスクの優先度を変更します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tskid      タスク ID (Task ID)

1 から最大優先度の数値以外に以下のマクロが指定可能

名前	値	意味
TSK_SELF	0	自タスク

tskpri      優先度 (Task Priority)

1 から最大優先度の数値以外に以下のマクロが指定可能

名前	値	意味
TPRI_INI	0	起動時優先度

##### ● 出力

ercd      エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい, タスク独立部からの呼び出しで tskid に TSK_SELF(=0) が指定された)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_PAR	-17	パラメータエラー (tskpri が負またはシステムの最大優先度より大きい)
E_ILUSE	-28	不正使用 (上限優先度違反)



## ■ ディスパッチ要因

対象タスクの優先度を自タスクの優先度より高くした場合には、対象タスクにディスパッチします。自タスクの優先度をほかの実行可能状態のタスクの優先度より低くした場合には、ほかの実行可能状態のタスクのうち、一番優先順位の高いタスクにディスパッチします。

セマフォ獲得待ち、メッセージバッファの送信待ちや、可変長メモリプールのメモリブロック獲得におけるタスク優先度順待ち行列に存在するタスクを対象として優先度を変更した場合には待ち行列の並びを変更して対象タスクの待ち状態を解除する場合があります。対象タスクの優先度が本システムコールを呼び出したタスクより高い場合には対象タスクにディスパッチします。

## ■ 解説

`tskid` で指定されたタスクのベース優先度を `tskpri` で指定された値に変更します。ベース優先度と現在優先度が一致している場合には（ミューテックス機能を使用しない場合には、この条件は常に成り立ちます）、タスクの現在優先度も指定された値に変更してベース優先度と一致させます。`tskpri` はタスク優先度として 1 からシステムの最大優先度（コンフィギュレータで設定した最大優先度）までの値を指定でき、値の小さい方が高い優先度となります。

`tskpri` が負またはシステムの最大優先度よりも大きい場合には `E_PAR` のエラーで復帰します。`tskid` で指定されたタスク ID が負か最大タスク数（コンフィギュレータで設定した最大タスク数）より大きい場合には `E_ID` のエラーで復帰します。`tskid` で指定されたタスク ID のタスクが存在しない場合には `E_NOEXS` のエラーで復帰します。

`tskid` に `TSK_SELF(=0)` が指定されると自タスクを対象タスクとします。ただし、タスク独立部から呼び出したシステムコールで `tskid` に `TSK_SELF` が指定された場合には `E_ID` のエラーで復帰します。また、`tskpri` に `TPRI_INI(=0)` が指定されると、対象タスクのベース優先度をタスクの起動時優先度に変更します。

本システムコールで変更した優先度はタスクが終了するまで有効です。タスクが休止状態に戻るとき、タスク生成時に指定されたタスクの起動時優先度に初期化されます。ただし、既に休止状態であるときに変更した優先度は有効となり、次にタスクを起動したときは、その変更された優先度で起動します。

本システムコールを実行した結果、対象タスクの現在優先度がベース優先度と一致している場合には、以下の処理を行います。

- ・対象タスクが実行可能状態である場合には、タスクの優先順位を変更後の優先度に従って変化させます。変更後の優先度と同じ優先度を持つタスクの間では対象タスクの優先順位を最低とします。
- ・対象タスクが何らかのタスク優先度順の待ち行列につながれている場合にも、その待ち行列の中での順序を変更後の優先度に従って変化させます。変更後の優先度と同じ優先度を持つタスクの間では対象タスクを最後につなぎます。対象タスクが `TA_CEILING` 属性のミューテックスをロックしているか、ロックを待っている場合に `tskpri` で指定されたベース優先度が、それらのミューテックスのいずれかの上限優先度よりも高い場合には `E_ILUSE` のエラーで復帰します。ミューテックスについては、「ユーザーズガイド 3.5.1 ミューテックス機能」および本書の「3.6.1.1 `tk_cre_mtx(Create Mutex)`」から「3.6.1.5 `tk_ref_mtx(Refer Mutex Status)`」を参照してください。

### ■ 補足事項

本システムコールを呼び出した結果、対象タスクのタスク優先度順の待ち行列の中での順序が変化した場合には、対象タスクまたはその待ち行列で待っているほかのタスクの待ち状態を解除する場合があります(セマフォ獲得待ち、メッセージバッファの送信待ち行列、および可変長メモリプールの獲得待ち行列)。

対象タスクが TA\_INHERIT 属性のミューテックスのロック待ち状態である場合には、本システムコールでベース優先度を変更したことにより、推移的な優先度継承の処理が必要になる場合があります。

ミューテックス機能を使用しない場合に、対象タスクに自タスク、変更後の優先度に自タスクのベース優先度を指定して本システムコールが呼び出されると、自タスクの実行順位は同じ優先度のタスクの中で最低となります。そのため、本システムコールを使用して実行権を放棄できます。

### 3.3.8 tk\_get\_reg(Get Task Registers)

タスクレジスタを獲得します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

#### ■ C 言語インタフェース

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

typedef struct t_regs {
    VP pc;
    UW ps;
    VP rp;
    VW mdl;
    VW mdh;
    VW r[15];
} T_REGS;

typedef struct t_eit {
    VP pc;
    UW ps;
} T_EIT;

typedef struct t_cregs {
    VP usp;
} T_CREGS;
```

#### ■ パラメータ

● 入力

tskid           タスク ID (Task ID)

pk\_regs        汎用レジスタに設定されている値を格納したパケットアドレス  
                 (Packet of Registers)

● 出力

ercd           エラーコード (Error Code)

                パケットに設定するデータ

pc             PC レジスタ

ps             PS レジスタ

rp             RP レジスタ

mdl           MDL レジスタ

mdh           MDH レジスタ

r[15]          汎用レジスタ R0-R14

pk\_eit        CPU 例外時に保存するレジスタの値を格納したパケットアドレス  
                 (Packet of EIT)

パケットに設定するデータ

pc	PC レジスタ
ps	PS レジスタ
pk_cregs	制御レジスタに設定されている値を格納したパケットアドレス (Packet of Control Registers)

パケットに設定するデータ

usp	ユーザスタックポインタ
-----	-------------

### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが自タスク)
E_CTX	-25	コンテキストエラー (タスク独立部からの呼出し)

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

tskid のタスクの現在のレジスタの内容を参照します。

tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。また、自タスクに本システムコールを呼び出した場合には E\_OBJ のエラーで復帰します。

pk\_regs, pk\_eit, pk\_cregs にそれぞれ NULL が指定されると、対応するレジスタの参照は行いません。pk\_eit の pc, ps の内容は、pk\_regs の pc, ps と同じ値を返します。

参照されたレジスタの値は対象タスクの現在のレジスタの値であり、必ずしもタスク部実行中のものであるとは限りません。

タスク独立部から本システムコールを呼び出した場合には E\_CTX のエラーで復帰します。

pk\_regs, pk\_eit, pk\_cregs が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.3.9 tk\_set\_reg(Set Task Registers)

タスクレジスタを設定します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

#### ■ C 言語インタフェース

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

typedef struct t_regs {
    VP pc;
    UW ps;
    VP rp;
    VW mdl;
    VW mdh;
    VW r[15];
} T_REGS;

typedef struct t_eit {
    VP pc;
    UW ps;
} T_EIT;

typedef struct t_cregs {
    VP usp;
} T_CREGS;
```

#### ■ パラメータ

● 入力

tskid	タスク ID (Task ID)
pk_regs	汎用レジスタに設定する値を格納したパケットアドレス (Packet of Registers)
パケットに設定するデータ	
pc	PC レジスタ
ps	PS レジスタ
rp	RP レジスタ
mdl	MDL レジスタ
mdh	MDH レジスタ
r[15]	汎用レジスタ R0-R14
pk_eit	CPU 例外時に保存するレジスタに設定する値を格納したパケットアドレス (Packet of EIT)

パケットに設定するデータ

pc	PC レジスタ
ps	PS レジスタ
pk_cregs	制御レジスタに設定する値を格納したパケットアドレス (Packet of Control Registers)

パケットに設定するデータ

usp	ユーザスタックポインタ
-----	-------------

### ● 出力

ercd	エラーコード (Error Code)
------	---------------------

### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが自タスク)
E_CTX	-25	コンテキストエラー (タスク独立部からの呼出し)

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

tskid のタスクのレジスタを指定の内容に設定します。

tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。自タスクに本システムコールを呼び出した場合には E\_OBJ のエラーで復帰します。

pk\_regs, pk\_eit, pk\_cregs にそれぞれ NULL が指定されると、対応するレジスタの設定は行われません。pk\_regs と pk\_eit の pc, ps に異なる値が指定された場合には pk\_eit の pc, ps の値が有効となります。

設定するレジスタの値の正当性についてはチェックしません。このため、不当なレジスタの値を設定した場合にはシステムが誤動作、ハングアップする可能性があります。

タスク独立部から本システムコールを呼び出した場合には E\_CTX のエラーで復帰します。

pk\_regs, pk\_eit, pk\_cregs が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.3.10 tk\_ref\_tsk(Refer Task Status)

タスク状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

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

typedef struct t_rtsk {
    VP      exinf;
    PRI      tskpri;
    PRI      tskbpri;
    UINT     tsksstat;
    UW       tskswait;
    ID       wid;
    INT      wupcnt;
    INT      suscnt;
} T_RTsk;
```

■ パラメータ

● 入力

tskid           タスク ID (Task ID)  
1 から最大タスク数の数値以外に以下のマクロが指定可能

名前	値	意味
TSK_SELF	0	自タスク

\*pk\_rtsk   タスクの状態を返すパケットアドレス  
(Packet of Refer Task)

### 第3章 システムコールインタフェース

#### ● 出力

ercd エラーコード (Error Code)

パケットに設定するデータ

exinf 拡張情報 (Extended Information)

tskpri 現在の優先度 (Task Priority)

tskbpri ベース優先度 (Task Base Priority)

tskstat タスクの状態 (Task State)

状態	値	意味
TTS_RUN	0x00000001	実行状態 (RUNNING)
TTS_RDY	0x00000002	実行可能状態 (READY)
TTS_WAI	0x00000004	待ち状態 (WAITING)
TTS_SUS	0x00000008	強制待ち状態 (SUSPENDED)
TTS_WAS	0x0000000c	二重待ち状態 (WAITING-SUSPENDED)
TTS_DMT	0x00000010	休止状態 (DORMANT)

tskwait 待ち要因 (Task Wait Factor)

待ち要因	値	意味
TTW_SLP	0x00000001	tk_slp_tsk による待ち
TTW_DLY	0x00000002	tk_dly_tsk による待ち
TTW_SEM	0x00000004	tk_wai_sem による待ち
TTW_FLG	0x00000008	tk_wai_flg による待ち
TTW_MBX	0x00000040	tk_rcv_mbx による待ち
TTW_MTX	0x00000080	tk_loc_mtx による待ち
TTW_SMBF	0x00000100	tk_snd_mbf による待ち
TTW_RMBF	0x00000200	tk_rcv_mbf による待ち
TTW_CAL	0x00000400	ランデブ呼出し待ち
TTW_ACP	0x00000800	ランデブ受付け待ち
TTW_RDV	0x00001000	ランデブ終了待ち
(TTW_CAL/TTW_RDV)	0x00001400	ランデブ呼出しまたは終了待ち
TTW_MPF	0x00002000	tk_get_mpf による待ち
TTW_MPL	0x00004000	tk_get_mpl による待ち

wid 待ちオブジェクト ID (Waiting Object ID)

wupcnt 起床要求キューイング数 (Wakeup Count)

suscnt 強制待ち要求ネスト数 (Suspend Count)



## ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい, タスク独立部からの呼出しで tskid に TSK_SELF(=0) が指定された)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

tskid で示された対象タスクの状態を参照します。

tskid に TSK\_SELF(=0) を設定することによって自タスクを指定できます。タスク独立部から呼び出されたシステムコールで tskid に TSK\_SELF が指定された場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。

tskstat にはタスクの状態を設定します。割込みハンドラから割り込まれたタスクを対象として本システムコールを呼び出した場合には tskstat として実行状態(TTS\_RUN)を返します。

tskstat が TTS\_WAI(TTS\_WAS を含む) の場合には, tskwait, wid は表 3.3-2 の値をとります。

表 3.3-2 tskwait と wid の値

tskwait	wid
TTW_SLP	0
TTW_DLY	0
TTW_SEM	待ち対象の semid
TTW_FLG	待ち対象の flgid
TTW_MBX	待ち対象の mbxid
TTW_MTX	待ち対象の mtxid
TTW_SMBF	待ち対象の mbfid
TTW_RMBF	待ち対象の mbfid
TTW_CAL	待ち対象の porid
TTW_ACP	待ち対象の porid
TTW_RDV	0
(TTW_CAL/TTW_RDV)	0
TTW_MPF	待ち対象の mpfid
TTW_MPL	待ち対象の mplid

tskstat が TTS\_WAI(TTS\_WAS を含む) 以外の状態の場合には, tskwait, wid は共に "0" です。また, 休止状態のタスクでは, wupcnt, suscnt はすべて "0" です。

pk\_rtsk が不正な場合でも, エラーチェックを行わず, 動作は保証されません。

### ■ 補足事項

本システムコールで自タスクの ID を知ることはできません。自タスクの ID を知りたい場合には `tk_get_tid` を使用します。

また、タスク独立部から、タスク独立部が呼び出される前に実行中のタスクに対して、`tk_svs_tsk`, `tk_ter_tsk` などのタスク状態が変化するシステムコールを呼出し後、本システムコールを呼び出しても、`tskstat` は実行状態のままです。

## 3.4 タスク付属同期機能のシステムコール

---

タスク付属同期機能のシステムコールについて説明します。

---

### ■ タスク付属同期機能のシステムコール

タスク付属同期機能は、以下の8種類のシステムコールで構成されます。

- tk\_slp\_tsk(Sleep Task)
- tk\_wup\_tsk(Wakeup Task)
- tk\_can\_wup(Cancel Wakeup Task)
- tk\_rel\_wai(Release Wait)
- tk\_sus\_tsk(Suspend Task)
- tk\_rsm\_tsk(Resume Task)
- tk\_frsm\_tsk(Force Resume Task)
- tk\_dly\_tsk(Delay Task)

### 3.4.1 tk\_slp\_tsk(Sleep Task)

自タスクを起床待ち状態へ移行します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tmout      タイムアウト指定 (Timeout)  
0 から 0x7ffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

##### ● 出力

ercd      エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_PAR	-17	パラメータエラー (tmout    (-2))
E_RLWAI	-49	待ち状態強制解除 ( 待ちの間に tk_rel_wai を受け付ける )
E_TMOUT	-50	タイムアウト (tmout の時間経過により待ち状態が解除された )
E_CTX	-25	コンテキストエラー ( タスク独立部またはディスパッチ禁止状態で実行 )

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクが待ち状態になった場合に、次の優先順位のタスクにディスパッチします。

#### ■ 解説

起床要求キューイング数が"0"の場合には本システムコールでは、自タスクを実行状態から起床待ち状態(tk\_wup\_tsk を待つ状態)に移行します。

tmout で指定された時間が経過する前にこのタスクを対象とした tk\_wup\_tsk が呼び出された場合には本システムコールは正常終了します。一方、tmout で指定された時間が経過する間にtk\_wup\_tskまたはtk\_rel\_waiが呼び出されなかった場合にはE\_TMOUTのエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、tk\_wup\_tsk または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL は待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間 ( 時間の単位 ) はシステム時刻の基準時間 ( = 1 ms ) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

本システムコールで待ち状態のタスクに tk\_rel\_wai が呼び出された場合には本システムコールは E\_RLWAI のエラーとなり、待ち状態を解除します。タスク独立部またはディスパッチ禁止状態で本システムコールを呼び出した場合には E\_CTX のエラーで復帰します。

起床要求キューイング数が "0" より大きい場合には本システムコールは起床要求キューイング数を 1 減らして、待ち状態には移行せず、本システムコールを呼び出したタスクの処理を継続します。

## ■ 補足事項

本システムコールによって待ち状態になっているタスクに、ほかのタスクから tk\_sus\_tsk が呼び出された場合にはこのタスクは二重待ち状態になります。

### 3.4.2 tk\_wup\_tsk(Wakeup Task)

---

他タスクを起床します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tskid          タスク ID (Task ID)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが自タスクまたは休止状態)
E_QOVR	-43	キューイングまたはネストのオーバーフロー (起床要求キューイング数が 32767 より大きい)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクを起床させた場合には起床されたタスクにディスパッチします。

## ■ 解説

tskid で指定されたタスクが tk\_slp\_tsk の呼出しによる待ち状態であった場合にその待ち状態を解除します。

本システムコールでは、tskid に自タスクを指定できません。自タスクが指定された場合には E\_OBJ のエラーで復帰します。tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。

対象タスクが tk\_slp\_tsk を実行しておらず、待ち状態でない場合には本システムコールによる起床要求はキューイングします。カーネルでは、タスク単位に起床要求キューイング数を管理しており、その初期値 (tk\_sta\_tsk 実行時の値) は 0 です。起床待ち状態でないタスクに本システムコールを呼び出すことにより、対象タスクの起床要求キューイング数を 1 増やします。一方、tk\_slp\_tsk を呼び出すことにより、対象タスクの起床要求キューイング数を 1 減らします。起床要求キューイング数が 0 のタスクでは、tk\_slp\_tsk を呼び出したときにそのタスクは待ち状態になります。

起床要求キューイング数の最大値は 32767 です。この最大値を超えて本システムコールを呼び出した場合には E\_QOVR のエラーで復帰します。

### 3.4.3 tk\_can\_wup(Cancel Wakeup Task)

タスクの起床要求を無効化します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tskid      タスク ID (Task ID)

1 から最大タスク数の数値以外に以下のマクロが指定可能

名前	値	意味
TSK_SELF	0	自タスク

##### ● 出力

wupcnt      キューイングされていた起床要求回数 (Wakeup Count)  
またはエラーコード (Error Code)

#### ■ エラーコード

E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい, タスク独立部からの呼出しで tskid に TSK_SELF(=0) が指定された)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが休止状態)

#### ■ ディスパッチ要因

本システムコールはディスパッチしません。

#### ■ 解説

tskid で示されたタスクの起床要求キューイング数を復帰値 (wupcnt) として返し, 同時にその起床要求をすべてキャンセルします。すなわち, 対象タスクの起床要求キューイング数を "0" にします。

タスク独立部から呼び出した本システムコールで自タスクが指定された場合には E\_ID のエラーで復帰します。tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい, またはタスク独立部からの呼出しで TSK\_SELF が指定された場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。



## ■ 補足事項

本システムコールは、周期的にタスクを起床して動作させる場合に、時間内に処理が終わっているかどうかを判定するために使用できます。すなわち、前の起床要求に対する処理が終了して `tk_slp_tsk` を呼び出す前に、それを監視するタスクが `tk_can_wup` を呼び出し、その復帰値である `wupcnt` が 1 以上の値の場合には前の起床要求に対する処理が時間内に終了しなかったことを確認できます。したがって、処理の遅れに何らかの処置がとれます。

### 3.4.4 tk\_rel\_wai(Release Wait)

他タスクの待ち状態を解除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tskid          タスク ID (Task ID)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが実行状態, 実行可能状態, 強制待ち状態, または 休止状態)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクの待ち状態を解除した場合に, 待ち状態が解除されたタスクにディスパッチします。

メッセージバッファの送信待ちタスクまたは可変長メモリプールのメモリブロック獲得待ちタスクの待ち状態を解除してそのタスクの優先度が本システムコールを呼び出したタスクより高い場合には待ち状態が解除したタスクにディスパッチします。

#### ■ 解説

tskid で指定されたタスクが待ち状態にある場合に, それを強制的に解除します。

tskid で指定されたタスクが待ち状態以外の場合には呼出し元に E\_OBJ のエラーで復帰します。tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。

本システムコールにより待ち状態が解除されたタスクは E\_RLWAI のエラーで復帰します。本システムコールでは待ち状態解除要求のキューイングは行いません。すなわち, 対象タスクが既に待ち状態の場合にはその待ち状態を解除しますが, 待ち状態以外の場合には E\_OBJ のエラーで復帰します。本システムコールで自タスクが指定された場合にも, 同様に E\_OBJ のエラーで復帰します。

本システムコールでは、強制待ち状態の解除は行いません。二重待ち状態のタスクを対象として本システムコールを呼び出すと、対象タスクは待ち状態が解除され、強制待ち状態となります。強制待ち状態も解除する必要がある場合には、別に `tk_rsm_tsk` または `tk_frsm_tsk` を呼び出します。本システムコールの対象タスクの状態と実行結果との関係についてまとめたものを表 3.4-1 に示します。

表 3.4-1 `tk_rel_wai` の対象タスクの状態と実行結果

対象タスクの状態	ercd	処理
実行できる状態 (RUNNING, READY) (自タスク以外)	E_OBJ	何もしない
実行状態 (RUNNING)(自タスク)	E_OBJ	何もしない
待ち状態 (WAITING)	E_OK	待ち解除 <sup>*1</sup>
強制待ち状態 (SUSPENDED)	E_OBJ	何もしない
二重待ち状態 (WAITING-SUSPENDED)	E_OK	強制待ち状態に移行 <sup>*2</sup>
休止状態 (DORMANT)	E_OBJ	何もしない
未登録状態 (NON-EXISTENT)	E_NOEXS	何もしない

\*1: 対象タスクには E\_RLWAI のエラーが返ります。対象タスクは、待ち解除の条件が満たされないまま、待ち状態が強制的に解除されます。

\*2: 強制待ち状態の解除後、E\_RLWAI のエラーで復帰します。

## ■ 補足事項

アラームハンドラなどを使用してあるタスクが待ち状態に入ってから一定時間後に本システムコールを呼び出すとタイムアウトに類似した機能を実現できます。

本システムコールと `tk_wup_tsk` とは以下の相違点があります。

- `tk_wup_tsk` は `tk_slp_tsk` による待ち状態だけを解除します。一方、本システムコールでは、それ以外の要因 (`tk_wai_flg`, `tk_wai_sem`, `tk_rcv_mbf` などのシステムコール) による待ち状態も解除します。
- 待ち状態に入ったシステムコールの復帰値は、`tk_wup_tsk` による待ち状態の解除が正常終了 (E\_OK) であるのに対して、本システムコールによる待ち状態の解除はエラー (E\_RLWAI) です。
- `tk_wup_tsk` の場合は、対象タスクがまだ `tk_slp_tsk` を実行していなくても要求がキューイングします。一方、本システムコールは、対象タスクが既に待ち状態以外の場合は E\_OBJ のエラーで復帰します。

### 3.4.5 tk\_sus\_tsk(Suspend Task)

---

他タスクを強制待ち状態へ移行します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tskid          タスク ID (Task ID)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが自タスクまたは休止状態)
E_CTX	-25	ディスパッチ禁止状態で実行状態のタスクが指定された
E_QOVR	-43	キューイングまたはネストのオーバフロー (強制待ち要求のネスト数が 32767 より大きい)

#### ■ ディスパッチ要因

ディスパッチ許可状態において、実行状態のタスクに対してタスク独立部から本システムコールを呼び出した場合に、次の優先順位のタスクにディスパッチします。

#### ■ 解説

tskid で指定されたタスクを強制待ち状態に移行してタスクの実行を中断させます。

tskid に自タスクが指定された場合には E\_OBJ のエラーで復帰します。tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。タスク独立部または自タスクからの本システムコールの呼出しにおいて、ディスパッチ禁止状態で tskid に実行状態のタスクが指定された場合には E\_CTX のエラーで復帰します。

強制待ち状態は、`tk_rsm_tsk`、`tk_frsm_tsk` の呼出しによって解除します。本システムコールの対象タスクが既に待ち状態であった場合には、本システムコールの実行により、対象タスクは待ち状態と強制待ち状態が複合した二重待ち状態となります。その後、このタスクの待ち解除の条件が満たされると対象タスクは強制待ち状態に移行します。一方、この二重待ち状態のタスクに `tk_rsm_tsk` または `tk_frsm_tsk` が呼び出されると対象タスクは前と同じ待ち状態に戻ります。

あるタスクに複数回の本システムコールが呼び出された場合、そのタスクは多重の強制待ち状態になります。これを強制待ち要求のネストとよびます。この場合、本システムコールが呼び出された回数 (`suscnt`) と同じ回数の `tk_rsm_tsk` システムコールを呼び出すか、`tk_frsm_tsk` を 1 回呼び出すことにより、対象タスクが元の状態に戻ります。したがって、本システムコール ~ `tk_rsm_tsk` システムコールの対をネストできます。強制待ち要求のネスト回数の最大値は 32767 です。

## ■ 補足事項

あるタスクが資源獲得のための待ち状態（セマフォ待ちなど）で、かつ強制待ち状態の場合でも、強制待ち状態でないときと同じ条件によって資源の割当て（セマフォの割当てなど）を行います。強制待ち状態であっても、資源割当ての遅延などを行うわけではなく、資源割当てや待ち状態の解除に関する条件や優先度は全く変わりません。

`tk_sus_tsk` によるタスクの強制待ち状態への移行は対象タスクの実行状態とは関係なく行うため、対象タスクが強制待ち状態に移行して処理が中断することにより、システム全体に悪影響を及ぼす場合があります。このため、タスクを強制待ち状態に移行させる場合には注意してください。

## 3.4.6 tk\_rsm\_tsk(Resume Task)

強制待ち状態のタスクを再開します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

### ■ C 言語インタフェース

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

### ■ パラメータ

#### ● 入力

tskid          タスク ID (Task ID)

#### ● 出力

ercd          エラーコード (Error Code)

### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが実行状態, 実行可能状態, 待ち状態, または休止状態)

### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクの強制待ち状態を解除した場合に, 強制待ち状態が解除されたタスクにディスパッチします。

### ■ 解説

tskid で指定されたタスクの強制待ち状態を解除します。すなわち, 対象タスクが以前に呼び出された tk\_sus\_tsk によって, 強制待ち状態に入りその実行が中断している場合に, その状態を解除し実行を再開させます。

tskid に自タスクが指定された場合には E\_OBJ のエラーで復帰します。tskid で指定されたタスクが強制待ち状態以外の場合も E\_OBJ のエラーで復帰します。tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。

対象タスクが待ち状態と強制待ち状態の複合した二重待ち状態の場合には, 本システムコールの実行により強制待ち状態だけが解除されて対象タスクは待ち状態となります。本システムコールでは, 強制待ち要求のネスト回数を 1 減らします。したがって, 対象タスクに 2 回以上の tk\_sus\_tsk が呼び出されている場合には, 本システムコールの終了後も対象タスクはまだ強制待ち状態のままです。

## ■ 補足事項

実行状態もしくは実行可能状態のタスクが `tk_sus_tsk` により強制待ち状態になった後に本システムコールや `tk_frsm_tsk` によって実行を再開した場合、そのタスクは同じ優先度のタスクの中で最低の優先順位となります。例えば、同じ優先度の `task_A` と `task_B` に以下のシステムコールを実行した場合には以下の動作をします。

```
tk_sta_tsk (tskid = task_A, stacd_A);  
tk_sta_tsk (tskid = task_B, stacd_B);  
/* この場合、優先順位はタスクを起動した順番、task_A task_B */  
tk_sus_tsk (tskid = task_A);  
tk_rsm_tsk (tskid = task_A);  
/* この場合に優先順位は task_B task_A となる */
```

### 3.4.7 tk\_frsm\_tsk(Force Resume Task)

強制待ち状態のタスクを強制再開します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

tskid          タスク ID (Task ID)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (tskid が負または最大タスク数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (tskid のタスクが存在しない)
E_OBJ	-41	オブジェクトの状態が不正 (対象タスクが実行状態, 実行可能状態, 待ち状態, または休止状態)

#### ■ ディスパッチ要因

システムコールを呼び出したタスクより優先度の高いタスクの強制待ち状態を解除した場合に, 強制待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

tskid で示されたタスクの強制待ち状態を解除します。すなわち, 対象タスクが以前に呼び出された tk\_sus\_tsk によって, 強制待ち状態に入りその実行が中断している場合に, その状態を解除して実行を再開させます。

tskid に自タスクが指定された場合には E\_OBJ のエラーで復帰します。tskid で指定された対象タスクが強制待ち状態以外の場合も E\_OBJ のエラーで復帰します。tskid で指定されたタスク ID が負または最大タスク数 (コンフィギュレータで設定した最大タスク数) より大きい場合には E\_ID のエラーで復帰します。tskid で指定されたタスクが存在しない場合には E\_NOEXS のエラーで復帰します。

対象タスクが待ち状態と強制待ち状態の複合した二重待ち状態の場合には, 本システムコールの実行により強制待ち状態だけが解除されて対象タスクは待ち状態となります。

対象タスクに2回以上の tk\_sus\_tsk が呼び出されている場合でも, それらの要求をすべて解除します。すなわち, 強制待ち状態は必ず解除されて対象タスクが二重待ち状態でない限り実行を再開できます。



## ■ 補足事項

実行状態もしくは実行可能状態のタスクが `tk_sus_tsk` により強制待ち状態になった後に `tk_rsm_tsk` や本システムコールによって実行を再開した場合、そのタスクは同じ優先度のタスクの中で最低の優先順位となります。例えば、同じ優先度の `task_A` と `task_B` に以下のシステムコールを実行した場合には以下の動作をします。

```
tk_sta_tsk (tskid = task_A, stacd_A);
tk_sta_tsk (tskid = task_B, stacd_B);
/* この場合、優先順位はタスクを起動した順番, task_A task_B */
tk_sus_tsk (tskid = task_A);
tk_frsm_tsk (tskid = task_A);
/* この場合に優先順位は task_B task_A となる */
```

### 3.4.8 tk\_dly\_tsk(Delay Task)

自タスクを時間経過待ち状態へ移行します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

dlytim      遅延時間 (Delay Time)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)
E_RLWAI	-49	待ち状態強制解除 (待ちの間に tk_rel_wai を受け付ける)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクが時間経過待ち状態になった場合に、次の優先順位のタスクにディスパッチします。

#### ■ 解説

自タスクの実行を一時的に停止し、時間経過待ちの状態 {WAITING} に入ります。

タスクの実行を停止する時間を dlytim により指定します。dlytim の基準時間 (時間の単位) はシステム時刻の基準時間 (= 1 ms) と同じです。dlytim が 0 の場合には何もせずに E\_OK で復帰します。本システムコールを呼び出したタスクが二重待ち状態になっている間も時間経過のカウントを行います。

tk\_rel\_wai により時間経過待ちを解除できます。本システムコールで待ち状態のタスクに tk\_rel\_wai が呼び出された場合には本システムコールは E\_RLWAI のエラーとなり、待ち状態を解除します。タスク独立部またはディスパッチ禁止状態で本システムコールを呼び出した場合には E\_CTX のエラーで復帰します。

## 3.5 同期・通信機能のシステムコール

---

同期・通信機能のシステムコールについて説明します。

---

### ■ 同期・通信機能のシステムコール

同期・通信機能は、以下の3種類のシステムコールで構成されます。

- セマフォ機能のシステムコール
- イベントフラグ機能のシステムコール
- メールボックス機能のシステムコール

### 3.5.1 セマフォ機能のシステムコール

---

セマフォ機能のシステムコールについて説明します。

---

#### ■ セマフォ機能のシステムコール

セマフォ機能は、以下の 5 種類のシステムコールで構成されます。

- tk\_cre\_sem(Create Semaphore)
- tk\_del\_sem>Delete Semaphore)
- tk\_sig\_sem(Signal Semaphore)
- tk\_wai\_sem(Wait on Semaphore)
- tk\_ref\_sem(Refer Semaphore Status)

### 3.5.1.1 tk\_cre\_sem(Create Semaphore)

セマフォを生成します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

```
typedef struct t_csem {  
    VP  exinf;  
    ATR sematr;  
    INT isemcnt;  
    INT maxsem;  
} T_CSEM;
```

#### ■ パラメータ

● 入力

pk\_csem     セマフォ生成情報を渡すパケットの先頭アドレス  
              (Packet of Create Semaphore)

              パケットに設定するデータ

exinf        拡張情報 (Extended Information)

sema<sup>tr</sup>      セマフォ属性 (Semaphore Attribute)  
              sema<sup>tr</sup>:= (TA\_TFIFO    TA\_TPRI) | (TA\_FIRST    TA\_CNT)

属性	値	意味
TA_TFIFO	0x00000000	待ちタスクを FIFO で管理
TA_TPRI	0x00000001	待ちタスクを優先度順で管理
TA_FIRST	0x00000000	待ち行列先頭のタスクを優先
TA_CNT	0x00000002	要求数の少ないタスクを優先

isemcnt     セマフォカウン트의初期値  
              (Initial Semaphore Count)

maxsem      セマフォカウン트의最大値  
              (Maximum Semaphore Count)

● 出力

semid        セマフォ ID (Semaphore ID)  
              またはエラーコード (Error Code)

### ■ エラーコード

E_LIMIT	-34	セマフォの数がシステムの上限より大きい
E_RSATR	-11	予約属性 (sematr に未定義の値が指定された)
E_PAR	-17	パラメータエラー (isemcnt が負または maxsem より大きい, maxsem が 0 以下)

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

セマフォを生成してセマフォID 番号を割り当てます。具体的にはセマフォカウン트의初期値が isemcnt, 最大値 (上限値) が maxsem のセマフォを生成します。maxsem に 0 以下の値, または isemcnt に負の値か imaxsem より大きい値が指定されると E\_PAR のエラーで復帰します。

exinf は対象セマフォに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報を tk\_ref\_sem で取り出せます。

なお, ユーザの情報を入れるためにもっと大きな領域が必要な場合や, 途中で内容を変更したい場合には, 自分でそのためのメモリを確保し, そのメモリバケットのアドレスを exinf に入れます。OS では exinf の内容について関知しません。

sematr はセマフォの属性を指定します。sematr に未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

TA\_TFIFO, TA\_TPRI では, タスクがセマフォの待ち行列に並ぶ場合の並び方を指定できます。属性が TA\_TFIFO の場合にはタスクの待ち行列は FIFO となり, TA\_TPRI の場合にはタスクの待ち行列はタスクの優先度順となります。

TA\_FIRST, TA\_CNT では, 資源獲得の優先順を指定します。TA\_FIRST および TA\_CNT の指定によって待ち行列の並び順は変わりません。待ち行列の並び順は TA\_TFIFO, TA\_TPRI だけで決定します。

TA\_FIRST では, 要求カウントに関係なく待ち行列の先頭のタスクから順に資源を割り当てます。待ち行列の先頭のタスクが要求分の資源を獲得できない限り, 待ち行列の後ろのタスクは資源を獲得できません。

TA\_CNT では, 要求カウント分の資源を獲得できるタスクから順に割り当てます。具体的には, 待ち行列の先頭のタスクから順に要求カウント数を検査し, 要求カウント数分の資源を割り当てられるタスクに割り当てます。要求カウント数の少ない順に割り当てる訳ではありません。

セマフォがシステムの上限值 (コンフィギュレータで設定した最大セマフォ数) まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。また, pk\_csem が不正な場合でも, エラーチェックを行わず, 動作は保証されません。

### 3.5.1.2 tk\_del\_sem(Delete Semaphore)

セマフォを削除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

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

■ パラメータ

● 入力

semid          セマフォ ID (Semaphore ID)

● 出力

ercd          エラーコード (Error Code)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (semid が 0 以下または最大セマフォ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (semid のセマフォが存在しない)

■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクが削除対象セマフォで待っている場合に、優先度の高いタスクにディスパッチします。

■ 解説

semid で指定されたセマフォを削除します。具体的には、対象セマフォを未生成状態にして ID 番号を解放します。

semid で指定されたセマフォ ID が 0 以下または最大セマフォ数 ( コンフィギュレータで設定した最大セマフォ数 ) より大きい場合には E\_ID のエラーで復帰します。 semid のセマフォが存在しない場合には E\_NOEXS のエラーで復帰します。

対象セマフォにおいて、待ち状態のタスクが存在していた場合にも、本システムコールは正常終了します。待ち状態のタスクは tk\_wai\_sem が E\_DLT のエラーで復帰すると待ち状態が解除されます。

### 3.5.1.3 tk\_sig\_sem(Signal Semaphore)

---

セマフォ資源を返却します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

semid      セマフォ ID (Semaphore ID)  
cnt        資源返却数 (Signal Count)

##### ● 出力

ercd       エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (semid が 0 以下または最大セマフォ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (semid のセマフォが存在しない)
E_QOVR	-43	キューイングまたはネストのオーバフロー (セマフォカウント値がセマフォカウントの最大値より大きい)
E_PAR	-17	パラメータエラー (cnt が 0 以下)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクのセマフォ待ち状態を解除した場合には待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

semid で指定されたセマフォに cnt 個の資源を返却する操作を行います。

semid で指定されたセマフォ ID が 0 以下、または最大セマフォ数 (コンフィギュレータで設定した最大セマフォ数) より大きい場合には E\_ID のエラーで復帰します。対象セマフォが存在しない場合には E\_NOEXS のエラーで復帰します。cnt が 0 以下の場合には E\_PAR のエラーで復帰します。

対象セマフォに既に待っているタスクがある場合には要求カウントを確認して、可能な場合には資源を割り当てます。資源が割り当てられたタスクを実行可能状態に移行します。複数のタスクが要求する資源数の総和以上に資源が返却された場合、複数のタスクに資源が割り当てられて実行可能状態になります。



本システムコール呼出し前のセマフォカウント値 (semcnt) と cnt の和がセマフォカウントの最大値 (maxcnt) より大きい場合は E\_QOVR のエラーで復帰します。この場合、資源の返却は一切行われずカウント値 (semcnt) も変化しません。

#### ■ 補足事項

セマフォカウント値 (semcnt) がセマフォカウントの初期値 (isemcnt) より大きい場合にも正常終了します。排他制御ではなく、同期の目的 (tk\_wup\_tsk ~ tk\_slp\_tsk と同様) でセマフォを使用する場合には、セマフォのカウント値 (semcnt) を初期値 (isemcnt) より大きくする場合があります。一方、排他制御の目的でセマフォを使用する場合は、セマフォカウントの初期値 (isemcnt) とセマフォカウントの最大値 (maxsem) を等しい値にしておくことによりカウント値の増加によるエラーをチェックできます。

### 3.5.1.4 tk\_wai\_sem(Wait on Semaphore)

セマフォ資源を獲得します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

semid      セマフォ ID (Semaphore ID)  
 cnt        資源要求数 (Require Count)  
 tmout      タイムアウト指定 (Timeout)  
 0 から 0x7fffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

##### ● 出力

ercd        エラーコード (Error Code)

#### ■ エラーコード

E\_OK        0      正常終了  
 E\_ID        -18    不正 ID 番号 (semid が 0 以下または最大セマフォ数より大きい)  
 E\_NOEXS    -42    オブジェクトが存在していない  
             (semid のセマフォが存在しない)  
 E\_PAR       -17    パラメータエラー (tmout    (-2), cnt    0)  
 E\_DLT       -51    待ちオブジェクトが削除された  
             ( 待ちの間に対象セマフォが待ちオブジェクトを削除 )  
 E\_RLWAI    -49    待ち状態強制解除 ( 待ちの間に tk\_rel\_wai を受け付ける )  
 E\_TMOUT    -50    ポーリング失敗またはタイムアウト  
 E\_CTX       -25    コンテキストエラー  
             ( タスク独立部またはディスパッチ禁止状態で実行 )

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクがセマフォ待ち状態になった場合に、次の優先順位のタスクにディスパッチします。

## ■ 解説

semid で指定されたセマフォから cnt 個の資源を獲得します。

semid が 0 以下または最大セマフォ数 ( コンフィギュレータで設定した最大セマフォ数 ) より大きい場合には E\_ID のエラーで復帰します。対象セマフォが存在しない場合には E\_NOEXS のエラーで復帰します。cnt が 0 以下の場合には E\_PAR のエラーで復帰します。

tmout により待ち時間の最大値 ( タイムアウト値 ) を指定できます。待ち解除の条件が満足されない ( tk\_sig\_sem が実行されない ) まま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、資源の獲得または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL は待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間 ( 時間の単位 ) はシステム時刻の基準時間 ( = 1 ms ) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

資源を獲得した場合には本システムコールの呼出しタスクは待ち状態に入らずに実行を継続します。この場合、そのセマフォのカウント値を cnt 分減算します。

資源が獲得できない場合には本システムコールを呼び出したタスクは待ち状態に入ります。すなわち、そのセマフォに対する待ち行列につながれます。この場合、そのセマフォのカウント値は不変です。

セマフォの待ち行列につながれた状態で対象セマフォが tk\_del\_sem により削除された場合には待ち状態が解除されて本システムコールは E\_DLT のエラーで復帰します。セマフォの待ち行列につながれたタスクを対象として tk\_rel\_wai を呼び出した場合には待ち状態が解除されて本システムコールは E\_RLWAI のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールを呼び出した場合には E\_CTX のエラーで復帰します。

### 3.5.1.5 tk\_ref\_sem(Refer Semaphore Status)

セマフォの状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

```
typedef struct t_rsem {
    VP  exinf;
    ID  wtsk;
    INT semcnt;
} T_RSEM;
```

#### ■ パラメータ

##### ● 入力

semid      セマフォ ID (Semaphore ID)

pk\_rsem    セマフォの状態を返すパケットアドレス  
(Packet of Refer Semaphore)

##### ● 出力

ercd      エラーコード (Error Code)

          パケットに設定するデータ

exinf      拡張情報 (Extended Information)

wtsk      待ちタスクの有無 (Wait Task)

semcnt    現在のセマフォカウント値 (Semaphore Count)

#### ■ エラーコード

E\_OK      0      正常終了

E\_ID      -18    不正 ID 番号 (semid が 0 以下または最大セマフォ数より大きい)

E\_NOEXS   -42    オブジェクトが存在していない  
(semid のセマフォが存在しない)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

semid で指定された対象セマフォの各種状態を参照して、復帰値として現在のセマフォカウント値 (semcnt), 待ちタスクの有無 (wtsk), 拡張情報 (exinf) を返します。

semid で指定されたセマフォ ID が 0 以下または最大セマフォ数 (コンフィギュレータで設定した最大セマフォ数) より大きい場合には E\_ID のエラーで復帰します。対象セマフォが存在しない場合には E\_NOEXS のエラーで復帰します。

wtsk は、このセマフォで待っているタスクの ID を示します。複数のタスクが待っている場合には待ち行列の先頭タスクの ID を返します。待ちタスクがない場合には wtsk は 0 となります。

pk\_rsem が不正な場合でも、エラーチェックを行わず、動作は保証されません。

## 3.5.2 イベントフラグ機能のシステムコール

---

イベントフラグ機能のシステムコールについて説明します。

---

### ■ イベントフラグ機能のシステムコール

イベントフラグ機能は、以下の 6 種類のシステムコールで構成されます。

- tk\_cre\_flg(Create Event Flag)
- tk\_del\_flg>Delete Event Flag)
- tk\_set\_flg(Set Event Flag)
- tk\_clr\_flg(Clear Event Flag)
- tk\_wai\_flg(Wait Event Flag)
- tk\_ref\_flg(Refer Event Flag Status)

3.5.2.1 tk\_cre\_flg(Create Event Flag)

イベントフラグを生成します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

ID flgid = tk\_cre\_flg ( T\_CFLG \*pk\_cflg ) ;

```
typedef struct t_cflg {
    VP      exinf;
    ATR      flgatr;
    UINT     iflgptn;
} T_CFLG;
```

■ パラメータ

● 入力

pk\_cflg    イベントフラグの状態を返すパケットアドレス  
          (Packet of Refer Event Flag)

          パケットに設定するデータ

exinf      拡張情報 (Extended Information)

flgatr      イベントフラグ属性 (Event Flag Attribute)

          flgatr := (TA\_TFIFO    TA\_TPRI) | (TA\_WMUL    TA\_WSGL)

属性	値	意味
TA_TFIFO	0x00000000	待ちタスクを FIFO で管理
TA_TPRI	0x00000001	待ちタスクを優先度順で管理
TA_WSGL	0x00000000	複数タスクの待ちを禁止
TA_WMUL	0x00000008	複数タスクの待ちを許可

iflgptn    イベントフラグの初期値  
          (Initial Event Flag Pattern)

● 出力

flgid      イベントフラグ ID (Event Flag ID)  
          またはエラーコード (Error Code)

■ エラーコード

E\_LIMIT    -34    イベントフラグの数がシステムの上限より大きい

E\_RSATR    -11    予約属性 (flgatr に未定義の値が指定された)

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

イベントフラグを生成してイベントフラグ ID 番号を割り当てます。具体的には、初期値が iflgptn のイベントフラグを生成します。

exinf は、対象イベントフラグに関する情報を入れておくためにユーザが自由に利用できます。ここで指定した情報を tk\_ref\_flg で取り出せます。

なお、ユーザの情報を入れるためにもっと大きな領域が必要な場合や、途中で内容を変更したい場合には、自分でそのためのメモリを確保し、そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容については関知しません。

flgatr はイベントフラグの属性を指定します。flgatr に未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

flgatr に TA\_WSGL が指定された場合には複数のタスクが同時に待ち状態になることを禁止します。TA\_WMUL が指定された場合には同時に複数のタスクが待ち状態となります。

TA\_TFIFO, TA\_TPRI では、タスクがイベントフラグの待ち行列に並ぶ場合の並び方を指定できます。属性が TA\_TFIFO の場合にはタスクの待ち行列は FIFO となり、属性が TA\_TPRI の場合にはタスクの待ち行列はタスクの優先度順となります。ただし、TA\_WSGL が指定された場合には待ち行列を作らないため、TA\_TFIFO, TA\_TPRI のどちらを指定しても動作に違いはありません。

複数のタスクが待っている場合には待ち行列の先頭から順に待ち条件が成立しているかを検査して、待ち条件が成立しているタスクの待ちを解除します。したがって、待ち行列先頭のタスクを必ずしも最初に待ち解除する訳ではありません。また、待ち条件が成立したタスクが複数ある場合には複数のタスクの待ちを解除します。

イベントフラグがシステムの上限值（コンフィギュレータで設定した最大イベントフラグ数）まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。

pk\_cflg が不正な場合でも、エラーチェックを行わず、動作は保証されません。



### 3.5.2.2 tk\_del\_flg(Delete Event Flag)

イベントフラグを削除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

● 入力

flgid          イベントフラグ ID (Event Flag ID)

● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (flgid が 0 以下または最大フラグ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (flgid のイベントフラグが存在しない)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクが削除したイベントフラグで待っていた場合には優先度の高いタスクにディスパッチします。

#### ■ 解説

flgid で示されたイベントフラグを削除します。具体的には、対象イベントフラグを未生成状態にして ID 番号を解放します。

flgid で指定されたイベントフラグ ID が 0 以下または最大イベントフラグ数 (コンフィギュレータで設定した最大イベントフラグ数) より大きい場合には E\_ID のエラーで復帰します。対象イベントフラグが存在しない場合には E\_NOEXS のエラーで復帰します。

対象イベントフラグにおいて条件成立を待っているタスクがある場合にも本システムコールは正常終了しますが、待ち状態のタスクが呼び出した tk\_wai\_flg は E\_DLT のエラーで復帰します。

### 3.5.2.3 tk\_set\_flg(Set Event Flag)

---

イベントフラグをセットします。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

flgid          イベントフラグ ID (Event Flag ID)  
setptn        セットするビットパターン (Set Bit Pattern)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (flgid が 0 以下または最大フラグ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (flgid のイベントフラグが存在しない)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクのイベントフラグ待ち状態を解除した場合には待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

本システムコールは、flgid で指定されたイベントフラグへ setptn で示されているビットをセットします。すなわち、flgid で指定されたイベントフラグの値に setptn の値で論理和をとります。

flgid で指定されたイベントフラグ ID が 0 以下または最大イベントフラグ数 (コンフィギュレータで設定した最大イベントフラグ数) より大きい場合には E\_ID のエラーで復帰します。対象イベントフラグが存在しない場合には E\_NOEXS のエラーで復帰します。

本システムコールによりイベントフラグの値を変更し、tk\_wai\_flg で待っていたタスクの待ち解除の条件を満たしていれば、そのタスクの待ち状態を解除します。その結果、待っていたタスクは実行状態または実行可能状態に移行します。ただし、待っていたタスクが二重待ち状態の場合には強制待ち状態へと移行します。

本システムコールで setptn の全ビットを 0 とした場合には対象イベントフラグに何の操作も行わずに正常終了します。

TA\_WMUL の属性のイベントフラグは、同一のイベントフラグに複数のタスクを同時に待てます。したがって、イベントフラグでもタスクが待ち行列を作ります。この場合、1 回の本システムコールで複数のタスクが待ち解除になる場合があります。

### 3.5.2.4 tk\_clr\_flg(Clear Event Flag)

イベントフラグをクリアします。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

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

■ パラメータ

● 入力

flgid	イベントフラグ ID (Event Flag ID)
clrptn	クリアするビットパターン (Clear Bit Pattern)

● 出力

ercd	エラーコード (Error Code)
------	---------------------

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (flgid が 0 以下または最大フラグ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (flgid のイベントフラグが存在しない)

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

本システムコールは, flgid で指定されたイベントフラグの値に clrptn が "0" になっているビットをクリアします。すなわち, flgid で指定されたイベントフラグの値に clrptn の値で論理積をとります。

flgid で指定されたイベントフラグ ID が 0 以下または最大イベントフラグ数 ( コンフィギュレータで設定した最大イベントフラグ数 ) より大きい場合には E\_ID のエラーで復帰します。対象イベントフラグが存在しない場合には E\_NOEXS のエラーで復帰します。

本システムコールでは , 対象イベントフラグを待っているタスクの待ち状態を解除しません。

本システムコールで clrptn の全ビットを 1 とした場合には , 対象イベントフラグに何の操作も行いません。ただし , この場合でも正常終了します。

### 3.5.2.5 tk\_wai\_flg(Wait Event Flag)

イベントフラグを待ちます。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

flgid イベントフラグ ID (Event Flag ID)  
 waiptn 待ちビットパターン (Wait Bit Pattern)  
 wfmode 待ちモード (Wait Event Flag Mode)  
 wfmode := (TWF\_ANDW TWF\_ORW) | [TWF\_CLR TWF\_BITCLR]

モード	値	意味
TWF_ANDW	0x00	AND 待ち
TWF_ORW	0x01	OR 待ち
TWF_CLR	0x10	全ビットクリア指定
TWF_BITCLR	0x20	条件ビットだけをクリア指定

tmout タイムアウト指定 (Timeout)  
 0 から 0x7fffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

##### ● 出力

ercd エラーコード (Error Code)  
 p\_flgptn 待ち解除時のビットパターン (Event Flag Bit Pattern)

#### ■ エラーコード

E\_OK 0 正常終了  
 E\_ID -18 不正 ID 番号 (flgid が 0 以下または最大フラグ数より大きい)  
 E\_NOEXS -42 オブジェクトが存在していない  
 (flgid のイベントフラグが存在しない)  
 E\_PAR -17 パラメータエラー  
 (waiptn = 0, wfmode に TWF\_ANDW, TWF\_ORW, TWF\_CLR, TWF\_BITCLR 以外の値が設定されている, tmout (-2))

E_OBJ	-41	オブジェクトの状態が不正 (TA_WSGL 属性のイベントフラグに対する複数タスクの待ち)
E_DLT	-51	待ちオブジェクトが削除された (待ちの間に対象イベントフラグが待ちオブジェクトを削除)
E_RLWAI	-49	待ち状態強制解除 (待ちの間に tk_rel_wai を受け付ける)
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)

## ■ ディスパッチ要因

本システムコールを呼び出したタスクがイベントフラグ待ち状態になった場合に、次の優先順位のタスクにディスパッチします。

## ■ 解説

本システムコールでは、wfmode で指定された待ち解除の条件に従って、flgid で指定されたイベントフラグがセットされるのを待ちます。

flgid で指定されたイベントフラグ ID が 0 以下または最大イベントフラグ数 (コンフィギュレータで設定した最大イベントフラグ数) より大きい場合には E\_ID のエラーで復帰します。対象イベントフラグが存在しない場合には E\_NOEXS のエラーで復帰します。

flgid で指定されたイベントフラグが既に wfmode で指定された待ち解除条件を満たしている場合には呼び出したタスクは待ち状態にならずに実行を続けます。wfmode には、以下の 4 種類の待ちモードを指定できます。

- TWF\_ORW : flgid で指定されたイベントフラグのうち、waipn で指定されたビットのいずれかがセットされるのを待ちます (OR 待ち)。
- TWF\_ANDW : flgid で指定されたイベントフラグのうち、waipn で指定されたビットのすべてがセットされるのを待ちます (AND 待ち)。
- TWF\_CLR : 条件が満足されてこのタスクが待ち解除となった場合、イベントフラグの値 (全部のビット) を "0" にクリアします (TWF\_CLR の指定あり)。条件が満足されてこのタスクが待ち解除となった場合にも、イベントフラグの値はそのままです (TWF\_CLR の指定なし)。
- TWF\_BITCLR: 条件が満足されてこのタスクが待ち解除となった場合にイベントフラグの待ち解除条件に一致したビットだけを "0" にクリアします (イベントフラグ値 &= 待ち解除条件)。

wfmode に TWF\_ANDW, TWF\_ORW, TWF\_CLR, TWF\_BITCLR 以外の値が指定されている場合には E\_PAR のエラーで復帰します。

p\_flgptn には、待ち状態を解除するときのイベントフラグの値 (flgptn) を設定する領域のアドレスを指定します。TWF\_CLR または TWF\_BITCLR 指定の場合には flgptn の値はイベントフラグがクリアされる前の値です。flgptn は待ち解除の条件を満たします。

なお、タイムアウト、tk\_rel\_wai または tk\_del\_flg により待ちが解除されると flgptn は不定となります。

tmout により待ち時間の最大値 (タイムアウト値) を指定できます。待ち解除の条件が満足されないまま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、条件成立または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL は待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

タイムアウトした場合は、TWF\_CLR または TWF\_BITCLR の指定があってもイベントフラグをクリアしません。

tmout の基準時間 ( 時間の単位 ) はシステム時刻の基準時間 ( = 1 ms ) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

既に待ちタスクの存在する TA\_WSGL 属性のイベントフラグに、別のタスクで本システムコールを実行できません。この場合は、後から本システムコールを実行したタスクが待ち状態に入るかどうか ( 待ち解除条件が満たされているかどうか ) にかかわらず、後から本システムコールを実行したタスクは E\_OBJ のエラーで復帰します。

一方、TA\_WMUL 属性のイベントフラグは、同一のイベントフラグに複数のタスクを同時に待てます。したがって、イベントフラグでもタスクが待ち行列を作ります。この場合には一回の tk\_set\_flg で複数のタスクが待ち解除になる場合があります。

TA\_WMUL 属性のイベントフラグに複数のタスクが待ち行列を作った場合には以下の動作をします。

- 待ち行列の順番は FIFO またはタスク優先度順です ( ただし、waipn や wfmode との関係により、必ずしも行列先頭のタスクから待ち解除になるとは限りません )。
- 待ち行列中にクリア指定のタスクがある場合には、そのタスクが待ち解除になるときにフラグをクリアします。
- クリア指定を行っていたタスクよりも後ろの待ち行列にあったタスクは、既にクリアされた後のイベントフラグを参照します。

tk\_set\_flg によって同じ優先度の複数のタスクが同時に待ち解除となる場合には、待ち解除後のタスクの優先順位は元のイベントフラグの待ち行列の順序を保持します。

waipn を 0 とした場合には E\_PAR のエラーで復帰します。イベントフラグ待ちのタスクが存在する状態で対象イベントフラグが tk\_del\_flg により削除された場合には待ち状態は解除されて本システムコールは E\_DLT のエラーで復帰します。また、イベントフラグ待ち状態のタスクに tk\_rel\_wai システムコールが呼び出された場合には待ち状態が解除されて E\_RLWAI のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

p\_flgptn が不正な場合でも、エラーチェックを行わず、動作は保証されません。

#### ■ 補足事項

本システムコールの待ち解除条件として全ビットの論理和を指定すると ( waipn = 0xffffffff, wfmode = TWF\_ORW ), tk\_set\_flg との組合せにより、CPU のビット幅分のビットパターンによるメッセージ転送ができます。ただし、この場合、全部のビットが 0 のメッセージは送れません。また、前のメッセージが本システムコールで読み出される前に tk\_set\_flg により次のメッセージが送られると、前のメッセージは消えてしまいます。すなわち、メッセージのキューイングはできません。waipn = 0 の指定は E\_PAR のエラーになるので、イベントフラグで待つタスクの waipn が 0 でないことを保証しています。したがって、tk\_set\_flg で全ビットをセットすると、どの条件でイベントフラグを待つタスクであっても、待ち行列の先頭にあるタスクは必ず待ち解除となります。

### 3.5.2.6 tk\_ref\_flg(Refer Event Flag Status)

イベントフラグの状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

```
typedef struct t_rflg {
    VP      exinf;
    ID      wtsk;
    UINT    flgptn;
} T_RFLG;
```

#### ■ パラメータ

##### ● 入力

flgid          イベントフラグ ID (Event Flag ID)

pk\_rflg       イベントフラグの状態を返すパケットの先頭アドレス  
(Packet of Refer Event Flag)

##### ● 出力

ercd          エラーコード (Error Code)

              パケットに返すデータ

exinf         拡張情報 (Extended Information)

wtsk          待ちタスクの有無 (Wait Task)

flgptn       イベントフラグのビットパターン  
(Event Flag Bit Pattern)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (flgid が 0 以下または最大フラグ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (flgid のイベントフラグが存在しない)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

flgid で示された対象イベントフラグの各種の状態を参照し、復帰値として現在のフラグ値 (flgptn)、待ちタスクの有無 (wtstk)、拡張情報 (exinf) を返します。

flgid で指定されたイベントフラグ ID が 0 以下または最大イベントフラグ数 (コンフィギュレータで設定した最大イベントフラグ数) より大きい場合 E\_ID のエラーで復帰します。対象イベントフラグが存在しない場合には E\_NOEXS のエラーで復帰します。

wtstk は、このイベントフラグで待っているタスクの ID を設定します。このイベントフラグで複数のタスクが待っている場合 (TA\_WMUL 属性の場合だけ) には待ち行列の先頭タスクの ID を返します。待ちタスクがない場合には wtstk の値は 0 となります。

pk\_rflg が不正な場合でも、エラーチェックを行わず、動作は保証されません。



### 3.5.3 メールボックス機能のシステムコール

---

メールボックス機能のシステムコールについて説明します。

---

#### ■ メールボックス機能のシステムコール

メールボックス機能は、以下の 5 種類のシステムコールで構成されます。

- tk\_cre\_mbx(Create Mailbox)
- tk\_del\_mbx>Delete Mailbox)
- tk\_snd\_mbx(Send Message to Mailbox)
- tk\_rcv\_mbx(Receive Message from Mailbox)
- tk\_ref\_mbx(Refer Mailbox Status)

### 3.5.3.1 tk\_cre\_mbx(Create Mailbox)

メールボックスを作成します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

```
typedef struct t_cmbx {
    VP exinf;
    ATR mbxatr;
} T_CMBX;
```

#### ■ パラメータ

##### ● 入力

pk\_cmbx    メールボックス生成情報 (Packet of Create Mailbox)

          パケットに設定するデータ

exinf        拡張情報 (Extended Information)

mbxatr       メールボックス属性 (Mailbox Attribute)

mbxatr:= (TA\_TFIFO    TA\_TPRI) | (TA\_MFIFO    TA\_MPRI)

属性	値	意味
TA_TFIFO	0x00000000	待ちタスクを FIFO で管理
TA_TPRI	0x00000001	待ちタスクを優先度順で管理
TA_MFIFO	0x00000000	メッセージを FIFO で管理
TA_MPRI	0x00000002	メッセージを優先度順で管理

##### ● 出力

mbxid        メールボックス ID (Mailbox ID)  
              またはエラーコード (Error Code)

#### ■ エラーコード

E\_LIMIT    -34    メールボックスの数がシステムの上限より大きい

E\_RSATR    -11    予約属性 (mbxatr に未定義の値が指定された)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません

## ■ 解説

メールボックスを生成しメールボックス ID 番号を割り当てます。

exinf は、対象メールボックスに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報を tk\_ref\_mbx で取り出せます。

なお、ユーザの情報を入れるためにもっと大きな領域が必要な場合や、途中で内容を変更したい場合には、自分でそのためのメモリを確保して、そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容については関知しません。

mbxatr はメールボックスの属性を指定します。未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

TA\_TFIFO, TA\_TPRI では、メッセージを受信するタスクがメールボックスの待ち行列に並ぶ際の並び方を指定できます。属性が TA\_TFIFO の場合にはタスクの待ち行列は FIFO となり、属性が TA\_TPRI の場合にはタスクの待ち行列はタスクの優先度順となります。

一方、TA\_MFIFO, TA\_MPRI では、メッセージがメッセージキュー(受信するのを待つメッセージの待ち行列)に入る際の並び方を指定できます。属性が TA\_MFIFO の場合にはメッセージキューは FIFO となり、属性が TA\_MPRI の場合にはメッセージキューはメッセージの優先度順となります。メッセージの優先度は、メッセージパケット内の msgpri で指定します(「3.5.3.3 tk\_snd\_mbx(Send Message to Mailbox)」を参照)。メッセージ優先度は正の値で、1 が最も優先度が高く、数値が大きくなるほど優先度は低くなります。PRI 型で表せる最大の正の値が最も低い優先度となります。同一優先度の場合は FIFO となります。

メールボックスがシステムの上限值(コンフィギュレータで設定した最大メールボックス数)まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。また、pk\_cmbx が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.5.3.2 tk\_del\_mbx(Delete Mailbox)

---

メールボックスを削除します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

mbxid          メールボックス ID (Mailbox ID)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mbxid が 0 以下または最大メールボックス数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mbxid のメールボックスが存在しない)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクが削除したメールボックスで待っていた場合に、待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

mbxid で示されたメールボックスを削除します。具体的には、対象メールボックスを未生成状態にして ID 番号を解放します。

mbxid で指定されたメールボックス ID が 0 以下または最大メールボックス数 (コンフィギュレータで設定した最大メールボックス数) より大きい場合には E\_ID のエラーで復帰します。対象メールボックスが存在しない場合には E\_NOEXS のエラーで復帰します。

対象メールボックスにおいてメッセージを待っているタスクがあった場合にも本システムコールは正常終了しますが、tk\_rcv\_mbx で待ち状態にあるタスクは E\_DLT のエラーで復帰します。また、対象メールボックスの中にメッセージが残っている場合でも、エラーとはならずメールボックスを削除します。

### 3.5.3.3 tk\_snd\_mbx(Send Message to Mailbox)

メールボックスへ送信します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

```
typedef struct t_msg {
    VP  msgque[1];
} T_MSG;
```

```
typedef struct t_msg_pri {
    T_MSG  msgque;
    PRI    msgpri;
} T_MSG_PRI;
```

#### ■ パラメータ

##### ● 入力

mbxid	メールボックス ID (Mailbox ID)
pk_msg	メッセージパケットの先頭アドレス (Packet of Message)
	パケットに設定するデータ
msgque	メッセージキュー用 OS 予約領域
msgpri	メッセージ優先度 (Message priority)

##### ● 出力

ercd	エラーコード (Error Code)
------	---------------------

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mbxid が 0 以下または最大メールボックス数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mbxid のメールボックスが存在しない)
E_PAR	-17	パラメータエラー (対象メールボックスがメッセージ優先順で msgpri が 0 以下)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクのメールボックス待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。

### ■ 解説

mbxid で示された対象メールボックスに ,pk\_msg を先頭アドレスとするメッセージパケットを送信します。

mbxid で指定されたメールボックス ID が 0 以下または最大メールボックス数 ( コンフィギュレータで設定した最大メールボックス数 ) より大きい場合には E\_ID のエラーで復帰します。対象メールボックスが存在しない場合には E\_NOEXS のエラーで復帰します。

メッセージパケットの内容をコピーせず, 受信時には先頭アドレス (pk\_msg の値) だけを渡します。対象メールボックスで既にメッセージを待っているタスクがある場合には, 待ち行列の先頭タスクの待ち状態が解除されて, 本システムコールで指定された pk\_msg がそのタスクに送信されて tk\_rcv\_mbx の復帰値となります。

一方, 対象メールボックスでメッセージを待っているタスクがない場合には, 送信されたメッセージはメールボックスの中のメッセージキュー ( メッセージの待ち行列 ) に入れます。どちらの場合にも, 本システムコールを呼び出したタスクは待ち状態とはなりません。pk\_msg はメッセージヘッダをも含めたメッセージパケットの先頭アドレスです。

メッセージヘッダは, TA\_MFIFO の場合には T\_MSG, TA\_MPRI の場合には T\_MSG\_PRI を使用してください。TA\_MPRI 属性のメールボックスに対して T\_MSG のメッセージヘッダを使用した場合には msgpri のフィールドがないため, 動作は保証されません。また, メッセージヘッダのサイズは固定長で, sizeof(T\_MSG) または sizeof(T\_MSG\_PRI) で獲得します。実際にメッセージを格納できるのは, メッセージヘッダより後ろの領域となります。メッセージ本体部分のサイズには制限はありません。

pk\_msg が不正な場合でも, エラーチェックを行わず, 動作は保証されません。

### ■ 補足事項

本システムコールによるメッセージ送信は, 受信側のタスクの状態とは無関係です。すなわち, 非同期のメッセージ送信を行います。待ち行列につながるのは, そのタスクが呼び出したメッセージであって, タスクそのものではありません。すなわち, メッセージの待ち行列 ( メッセージキュー ) や受信タスクの待ち行列は存在しますが, 送信タスクの待ち行列は存在しません。

### 3.5.3.4 tk\_rcv\_mbx(Receive Message from Mailbox)

メールボックスから受信します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

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

```
typedef struct t_msg {
    VP msgque[1];
} T_MSG;
```

```
typedef struct t_msg_pri {
    T_MSG msgque;
    PRI msgpri;
} T_MSG_PRI;
```

#### ■ パラメータ

##### ● 入力

mbxid      メールボックス ID (Mailbox ID)

ppk\_msg    メッセージパケットの先頭アドレスを返す変数の  
            アドレス (Packet of Message)

tmout      タイムアウト指定 (Timeout)

0 から 0x7ffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

##### ● 出力

ercd      エラーコード (Error Code)

            パケットに返すデータ

msgque    メッセージキュー用 OS 予約領域

msgpri    メッセージ優先度 (Message priority)

#### ■ エラーコード

E\_OK      0      正常終了

E\_ID      -18     不正 ID 番号  
            (mbxid が 0 以下または最大メールボックス数より大きい)

E_NOEXS	-42	オブジェクトが存在していない (mbxid のメールボックスが存在しない)
E_PAR	-17	パラメータエラー (tmout (-2))
E_DLT	-51	待ちオブジェクトが削除された (待ちの間に対象メールボックスが待ちオブジェクトを削除)
E_RLWAI	-49	待ち状態強制解除 (待ちの間に tk_rel_wai を受け付ける)
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクがメールボックス待ち状態になった場合、次の優先順位のタスクにディスパッチします。

#### ■ 解説

本システムコールでは、mbxid で示されたメールボックスからメッセージを受信します。mbxid で指定されたメールボックス ID が 0 以下または最大メールボックス数 (コンフィギュレータで設定した最大メールボックス数) より大きい場合には E\_ID のエラーで復帰します。対象メールボックスが存在しない場合には E\_NOEXS のエラーで復帰します。tmout により待ち時間の最大値 (タイムアウト値) を指定できます。タイムアウト指定が行われた場合に待ち解除の条件が満足されない (メッセージが到着しない) まま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。tmout の基準時間 (時間の単位) はシステム時刻の基準時間 (= 1 ms) と同じです。TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、メッセージの到着または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL は待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

対象メールボックスにまだメッセージが送信されていない場合 (メッセージキューが空の場合) には、本システムコールを呼び出したタスクは待ち状態となり、メッセージの到着を待つ待ち行列につながれます。一方、対象メールボックスに既にメッセージが入っている場合には、メッセージキューの先頭にあるメッセージを 1 つ取り出して、それを復帰値 ppk\_msg とします。

メッセージ待ち状態で、対象メールボックスが tk\_del\_mbx により削除された場合には待ち状態が解除されて本システムコールは E\_DLT のエラーで復帰します。メッセージ待ち状態のタスクに tk\_rel\_wai システムコールが呼び出された場合には待ち状態が解除されて E\_RLWAI のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

ppk\_msg が不正な場合でも、エラーチェックを行わず、動作は保証されません。

#### ■ 補足事項

ppk\_msg はメッセージヘッダを含めたメッセージパケットの先頭アドレスです。メッセージヘッダは TA\_MFIFO の場合には T\_MSG, TA\_MPRI の場合には T\_MSG\_PRI となります。



### 3.5.3.5 tk\_ref\_mbx(Refer Mailbox Status)

メールボックスの状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

```
typedef struct t_rmbx {
    VP      exinf;
    ID      wtsk;
    T_MSG   *pk_msg;
} T_RMBX;
```

#### ■ パラメータ

##### ● 入力

mbxid      メールボックス ID (Mailbox ID)

pk\_rmbx    メールボックスの状態を返すパケットアドレス  
(Packet of Refer Mailbox)

##### ● 出力

ercd      エラーコード (Error Code)

          パケットに返すデータ

extinf    拡張情報 (Extended Information)

wtsk      待ちタスクの有無 (Wait Task)

pk\_msg    次に受信するメッセージパケットの先頭アドレス  
(Packet of Message)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mbxid が 0 以下または最大メールボックス数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mbxid のメールボックスが存在しない)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

mbxid で示された対象メールボックスの各種状態を参照して復帰値として次に受信するメッセージ (メッセージキューの先頭のメッセージ), 待ちタスクの有無 (wtsk), 拡張情報 (exinf) を返します。

mbxid で指定されたメールボックス ID が 0 以下または最大メールボックス数 (コンフィギュレータで設定した最大メールボックス数) より大きい場合には E\_ID のエラーで復帰します。また, 対象メールボックスが存在しない場合には E\_NOEXS のエラーで復帰します。

wtsk は, このメールボックスで待っているタスクの ID を設定します。このメールボックスで複数のタスクが待っている場合には待ち行列の先頭のタスクの ID を返します。待ちタスクがない場合には wtsk が 0 となります。

pk\_msg は, 次に tk\_rcv\_mbx を実行した場合に受信するメッセージです。メッセージキューにメッセージがないときには pk\_msg は NULL となります。また, どんな場合でも, 「pk\_msg が NULL」または「wtsk が 0」の少なくとも一方は成り立ちます。

pk\_rmbx が不正な場合でも, エラーチェックを行わず, 動作は保証されません。

## 3.6 拡張同期・通信機能のシステムコール

---

拡張同期・通信機能のシステムコールについて説明します。

---

### ■ 拡張同期・通信機能のシステムコール

拡張同期・通信機能は、以下の 3 種類のシステムコールで構成されます。

- ミューテックス機能のシステムコール
- メッセージバッファ機能のシステムコール
- ランデブポート機能のシステムコール

## 3.6.1 ミューテックス機能のシステムコール

---

ミューテックス機能のシステムコールについて説明します。

---

### ■ ミューテックス機能のシステムコール

ミューテックス機能は、以下の 5 種類のシステムコールで構成されます。

- tk\_cre\_mtx(Create Mutex)
- tk\_del\_mtx>Delete Mutex)
- tk\_loc\_mtx(Lock Mutex)
- tk\_unl\_mtx(Unlock Mutex)
- tk\_ref\_mtx(Refer Mutex Status)

### 3.6.1.1 tk\_cre\_mtx(Create Mutex)

ミューテックスを生成します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ID mtxid = tk_cre_mtx ( T_CMTX *pk_cmtx )

typedef struct t_cmtx {
    VP      exinf;
    ATR     mtxatr;
    PRI     ceilpri;
} T_CMTX;
```

■ パラメータ

● 入力

pk\_cmtx    ミューテックス生成情報を設定する  
          パケットアドレス (Packet of Create Mutex)

          パケットに設定するデータ

exinf      拡張情報 (Extended Information)

mtxatr     ミューテックス属性 (Mutex Attribute)

          mtxatr:= (TA\_TFIFO    TA\_TPRI    TA\_INHERIT    TA\_CEILING)

属性	値	意味
TA_TFIFO	0x00000000	待ちタスクを FIFO で管理
TA_TPRI	0x00000001	待ちタスクを優先度順で管理
TA_INHERIT	0x00000002	優先度継承プロトコル
TA_CEILING	0x00000003	優先度上限プロトコル

ceilpri    ミューテックスの上限優先度 (Ceiling Priority)

● 出力

mtxid      ミューテックス ID (Mutex ID)  
          またはエラーコード (Error Code)

■ エラーコード

E\_LIMIT    -34    ミューテックスの数がシステムの上限より大きい

E\_RSATR    -11    予約属性 (mtxatr に未定義の値が指定された)

E\_PAR      -17    パラメータエラー (ceilpri が 0 以下またはシステムの最大優先度より大きい (mtxatr に TA\_CEILING を指定している場合))

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

ミューテックスを生成しミューテックス ID 番号を割り当てます。

`exinf` は、対象ミューテックスに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報を `tk_ref_mtx` で取り出せます。

なお、ユーザの情報を入れるためにもっと大きな領域が必要な場合や、途中で内容を変更したい場合には、自分でそのためのメモリを確保して、そのメモリパケットのアドレスを `exinf` に入れます。OS では `exinf` の内容については関知しません。

`mtxatr` はミューテックスの属性を指定します。未定義の属性が指定された場合には `E_RSATR` のエラーで復帰します。

`mtxatr` に `TA_TFIFO` が指定された場合にはミューテックスのタスクの待ち行列は FIFO となります。`TA_TPRI`, `TA_INHERIT`, `TA_CEILING` では、タスクの優先度順となります。`TA_INHERIT` では優先度継承プロトコル、`TA_CEILING` では優先度上限プロトコルを適用します。

`ceilpri` は、`mtxatr` に `TA_CEILING` が指定された場合にだけ有効となり、ミューテックスの上限優先度を設定します。`ceilpri` が 0 以下またはシステムの最大優先度（コンフィギュレータで設定した最大優先度）より大きい場合には `E_PAR` のエラーで復帰します。

ミューテックスがシステムの上限值（コンフィギュレータで設定した最大ミューテックス数）まで作成されている状態で本システムコールを呼び出した場合には `E_LIMIT` のエラーで復帰します。また、`pk_cmtx` が不正な場合でも、エラーチェックを行わず、動作は保証されません。

3.6.1.2 tk\_del\_mtx(Delete Mutex)

ミューテックスを削除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

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

■ パラメータ

● 入力

mtxid            ミューテックス ID (Mutex ID)

● 出力

ercd            エラーコード (Error Code)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mtxid が 0 以下または最大ミューテックス数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mtxid のミューテックスが存在しない)

■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクが削除したミューテックスで待っていた場合に、待ち状態が解除されたタスクにディスパッチします。

■ 解説

mtxid で示されたミューテックスを削除します。具体的には、対象ミューテックスを未生成状態にして ID 番号を解放します。

mtxid で指定されたミューテックス ID が 0 以下または最大ミューテックス数 (コンフィギュレータで設定した最大ミューテックス数) より大きい場合には E\_ID のエラーで復帰します。対象ミューテックスが存在しない場合には E\_NOEXS のエラーで復帰します。

対象ミューテックスにおいてロック待ちしているタスクがある場合にも、本システムコールは正常終了しますが、待ち状態にあったタスクは E\_DLT のエラーで復帰します。ミューテックスを削除すると、そのミューテックスをロックしているタスクにとっては、ロックしているミューテックスが減ります。したがって、削除されるミューテックスが TA\_INHERIT または TA\_CEILING 属性の場合には、ロックしていたタスクの現在優先度を変更する場合があります。

### 3.6.1.3 tk\_loc\_mtx(Lock Mutex)

ミューテックスをロックします。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

mtxid      ミューテックス ID (Mutex ID)

tmout      タイムアウト指定 (Timeout)

0 から 0x7ffffff の数値以外に以下のマクロ指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

##### ● 出力

ercd      エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mtxid が 0 以下または最大ミューテックス数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mtxid のミューテックスが存在しない)
E_PAR	-17	パラメータエラー (tmout    (-2))
E_DLT	-51	待ちオブジェクトが削除された ( 待ちの間に対象ミューテックスが待ちオブジェクトを削除 )
E_RLWAI	-49	待ち状態強制解除 ( 待ちの間に tk_rel_wai を受け付ける )
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CTX	-25	コンテキストエラー ( タスク独立部またはディスパッチ禁止状態で実行 )
E_ILUSE	-28	不正使用 ( 多重ロック , 上限優先度違反 )

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクがロック待ち状態になった場合 , 次の優先順位のタスクにディスパッチします。



## ■ 解説

mtxid のミューテックスをロックします。

mtxid で指定されたミューテックス ID が 0 以下または最大ミューテックス数 ( コンフィギュレータで設定した最大ミューテックス数 ) より大きい場合には E\_ID のエラーで復帰します。対象ミューテックスが存在しない場合には E\_NOEXS のエラーで復帰します。

tmout により待ち時間の最大値 ( タイムアウト値 ) を指定できます。待ち解除の条件が満足されない ( ロックが解除されない ) まま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、ロックの獲得または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL は待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間 ( 時間の単位 ) はシステム時刻の基準時間 ( = 1 ms ) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

ミューテックスがロックできると本システムコールの呼出しタスクは待ち状態に入らずに正常復帰します。この場合、そのミューテックスはロック状態になります。ロックできない場合には本システムコールを呼び出したタスクは待ち状態に入ります。すなわち、そのミューテックスに対する待ち行列につながれます。

自タスクが既に対象ミューテックスをロックしている場合には E\_ILUSE ( 多重ロック ) のエラーで復帰します。対象ミューテックスが TA\_CEILING 属性の場合には、自タスクのベース優先度が対象ミューテックスの上限優先度より高い場合には E\_ILSUE ( 上限優先度違反 ) のエラーで復帰します。

ミューテックス待ちの状態に対象ミューテックスが tk\_del\_mtx により削除された場合には待ち状態は解除されて本システムコールは E\_DLT のエラーで復帰します。ミューテックス待ち状態のタスクに tk\_rel\_wai が呼び出された場合には待ち状態は解除されて E\_RLWAI のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

## ■ 補足事項

- TA\_INHERIT 属性のミューテックスの場合

自タスクがロック待ち状態になる場合、そのミューテックスをロックしているタスクの現在優先度が自タスクより低い場合には、ロックしているタスクの現在優先度を自タスクと同じ現在優先度まで引き上げます。ロックを待っているタスクがロックを獲得せずに待ちを終了した場合 ( タイムアウトなど ) には、そのミューテックスをロック中のタスクの現在優先度を以下のうちの最も高い現在優先度まで引き下げます。

- そのミューテックスでロック待ちしているタスクの現在優先度のうちの最も高い優先度。
- ロック中のタスクのベース優先度。

- TA\_CEILING 属性のミューテックスの場合

自タスクがロックを獲得した場合に自タスクの現在優先度がミューテックスの上限優先度より低い場合には自タスクの現在優先度をミューテックスの上限優先度まで引き上げます。

### 3.6.1.4 tk\_unl\_mtx(Unlock Mutex)

---

ミューテックスをロック解除します。

---

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

mtxid          ミューテックス ID (Mutex ID)

##### ● 出力

ercd          エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mtxid が 0 以下または最大ミューテックス数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mtxid のミューテックスが存在しない)
E_ILUSE	-28	不正使用 ( 自タスクがロックしたミューテックスではない )
E_CTX	-25	コンテキストエラー ( タスク独立部で実行 )

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクのロック待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。

対象ミューテックスをロック解除した結果、自タスクの優先度が引き下げられた場合には、それより高い優先度のタスクにディスパッチする場合があります。

#### ■ 解説

mtxid のミューテックスのロックを解除します。ロック待ちしているタスクがある場合には待ち行列の先頭タスクの待ちを解除し、そのタスクをロック獲得状態にします。

mtxid で指定されたミューテックス ID が 0 以下または最大ミューテックス数 ( コンフィギュレータで設定した最大ミューテックス数 ) より大きい場合には E\_ID のエラーで復帰します。対象ミューテックスが存在しない場合には E\_NOEXS のエラーで復帰します。自タスクがロックしていないミューテックスが指定された場合には E\_ILUSE のエラーで復帰します。

**■ 補足事項**

ロック解除したミューテックスがTA\_INHERIT またはTA\_CEILING 属性の場合には以下のようにタスクの現在優先度を引き下げる必要があります。ロック解除により自タスクがロックしているミューテックスがすべてなくなった場合には、自タスクの現在優先度をベース優先度まで引き下げます。自タスクがロック中のミューテックスが残っている場合には自タスクの現在優先度を以下のうちの最も高い優先度まで引き下げます。

- ロックしているすべてのミューテックスのうちの最も高い優先度。
- ベース優先度。

ミューテックスをロックした状態でタスクを終了した（休止状態または未登録状態になった）場合、ロックしているすべてのミューテックスを自動的にロック解除します。

### 3.6.1.5 tk\_ref\_mtx(Refer Mutex Status)

---

ミューテックスの状態を参照します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

```
typedef struct t_rmtx {
    VP  exinf;
    ID  htsk;
    ID  wtsk;
} T_RMTX;
```

#### ■ パラメータ

##### ● 入力

mtxid          ミューテックス ID (Mutex ID)

pk\_rmtx        ミューテックスの状態を返すパケットアドレス  
(Packet of Refer Mutex Status)

##### ● 出力

ercd          エラーコード (Error Code)

                パケットに返すデータ

exinf          拡張情報 (Extended Information)

htsk          ロックしているタスクの ID (Hold Task ID)

wtsk          ロック待ちタスクの ID (Wait Task ID)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mtxid が 0 以下または最大ミューテックス数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mtxid のミューテックスが存在しない)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

mtxid で示された対象ミューテックスの各種の状態を参照して、復帰値としてロック中のタスク (htsk)、ロック待ちタスク (wtsk)、拡張情報 (exinf) を返します。

mtxid で指定されたミューテックス ID が 0 以下または最大ミューテックス数 (コンフィギュレータで設定した最大ミューテックス数) より大きい場合には E\_ID のエラーで復帰します。対象ミューテックスが存在しない場合には E\_NOEXS のエラーで復帰します。

htsk は、このミューテックスをロックしているタスクの ID を設定します。ロックしているタスクがない場合には htsk = 0 となります。

wtsk はこのミューテックスで待っているタスクの ID を設定します。複数のタスクが待っている場合には待ち行列の先頭タスクの ID を返します。待ちタスクがない場合には wtsk=0 となります。

pk\_rmtx が不正な場合でも、エラーチェックを行わず、動作は保証されません。

## 3.6.2 メッセージバッファ機能のシステムコール

---

メッセージバッファ機能のシステムコールについて説明します。

---

### ■ メッセージバッファ機能のシステムコール

メッセージバッファ機能は、以下の 5 種類のシステムコールで構成されます。

- tk\_cre\_mbf(Create Message Buffer)
- tk\_del\_mbf>Delete Message Buffer)
- tk\_snd\_mbf(Send Message to Message Buffer)
- tk\_rcv\_mbf(Receive Message from Message Buffer)
- tk\_ref\_mbf(Refer Message Buffer Status)

### 3.6.2.1 tk\_cre\_mbf(Create Message Buffer)

メッセージバッファを生成します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

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

```
typedef struct t_cmbf {  
    VP exinf;  
    ATR mbfatr;  
    INT bufsz;  
    INT maxmsz;  
    VP bufptr;  
} T_CMBF;
```

■ パラメータ

● 入力

pk\_cmbf      メッセージバッファ生成情報のパケット先頭アドレス  
                  (Packet of Create Message Buffer)

                パケットに設定するデータ

exinf          拡張情報 (Extended Information)

mbfatr         メッセージバッファ属性 (Message Buffer Attribute)  
                  mbfatr := (TA\_TFIFO    TA\_TPRI) | [TA\_USERBUF]

属性	値	意味
TA_TFIFO	0x00000000	待ちタスクを FIFO で管理
TA_TPRI	0x00000001	待ちタスクを優先度順で管理
TA_USERBUF	0x00000020	メッセージバッファ領域としてユーザが指定した領域を使用する

bufsz          メッセージバッファのサイズ ( バイト数 )(Buffer Size)

maxmsz         メッセージの最大長 ( バイト数 )  
                  (Maximum Message Size)

bufptr         ユーザバッファのアドレス (Buffer Pointer)

### ● 出力

mbfid      メッセージバッファ ID (MessageBuffer ID)  
 またはエラーコード (Error Code)

### ■ エラーコード

E_NOMEM	-33	メモリ不足 (メッセージバッファ領域を確保できない)
E_LIMIT	-34	メッセージバッファの数がシステムの上限より大きい
E_RSATR	-11	予約属性 (mtxatr に未定義の値が指定された)
E_PAR	-17	パラメータエラー (bufsz が負, maxmsz が 0 以下, bufsz が 4 の 倍数でない (TA_USERBUF が指定されている場合))

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

メッセージバッファを生成してメッセージバッファ ID 番号を割り当てます。

exinf は、対象メッセージバッファに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報を tk\_ref\_mbf で取り出せます。

なお、ユーザの情報を入れるためにもっと大きな領域が必要な場合や、途中で内容を変更したい場合には、自分でそのためのメモリを確保し、そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容については関知しません。

mbfatr はメッセージバッファの属性を指定します。未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

TA\_TFIFO, TA\_TPRI では、バッファが一杯の場合にメッセージを送信するタスクがメッセージバッファの待ち行列に並ぶ際の並び方を指定できます。属性が TA\_TFIFO の場合にはタスクの待ち行列は FIFO となり、属性が TA\_TPRI の場合にはタスクの待ち行列はタスクの優先度順となります。

なお、メッセージキューの順序は FIFO だけです。メッセージ受信待ちのタスクの待ち行列の順序は FIFO だけです。

TA\_USERBUF が指定された場合には bufptr が有効になり、bufptr を先頭とする bufsz バイトのメモリ領域をメッセージバッファ領域として使用します。この場合、メッセージバッファ領域は OS では用意しません。TA\_USERBUF が指定されなかった場合には bufptr は無視されてメッセージバッファ領域は OS が確保します。bufsz に "0" を指定することも可能で、この場合には、メッセージバッファを使用しない、完全に同期した通信になります。メッセージバッファ領域の先頭 4 バイトはカーネルで使用されます。また、bufsz が 8 バイトの倍数でない場合で mbfatr に TA\_USERBUF が指定されている場合、bufsz が 8 バイトの倍数になるように bufsz が切上げられます。bufsz が負の場合には E\_PAR のエラーで復帰します。メッセージバッファ領域が OS で確保できなかった場合には E\_NOMEM のエラーで復帰します。

maxmsz は、メッセージバッファで通信可能なメッセージの最大長をバイト数で設定します。maxmsz が 0 以下の場合には E\_PAR のエラーで復帰します。



メッセージバッファがシステムの上限值（コンフィギュレータで設定した最大メッセージバッファ数）まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。また、pk\_cmbf, bufptr が不正な場合でも、エラーチェックを行わずに動作は保証されません。

メッセージバッファは、可変長メッセージの送受信の管理を行うオブジェクトです。メールボックス (mbx) との相違点は、送信時と受信時に可変長のメッセージ内容をコピーする点です。また、バッファが一杯の場合に、メッセージ送信側も待ち状態に入る機能があります。

### 3.6.2.2 tk\_del\_mbf(Delete Message Buffer)

メッセージバッファを削除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

##### ● 入力

mbfid      メッセージバッファ ID  
(Message Buffer ID)

##### ● 出力

ercd      エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mbfid が 0 以下または最大メッセージバッファ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mbfid のメッセージバッファが存在しない)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクが削除したメッセージバッファで待っていた場合に、待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

mbfid で示されたメッセージバッファを削除します。具体的には、対象メッセージバッファを未生成状態にして ID 番号およびメッセージを入れるバッファ領域を解放します。

mbfid で指定されたメッセージ ID が 0 以下または最大メッセージバッファ数 (コンフィギュレータで設定した最大メッセージバッファ数) より大きい場合には E\_ID のエラーで復帰します。対象メッセージバッファが存在しない場合には E\_NOEXS のエラーで復帰します。

対象メッセージバッファにおいてメッセージ受信またはメッセージ送信を待っているタスクがあった場合にも、本システムコールは正常終了します。ただし、待ち状態にあったタスクは E\_DLT のエラーで復帰します。また、対象メッセージバッファの中にメッセージが残っている場合でもエラーとはならず、メッセージバッファの削除が行われて中にあるメッセージは消滅します。

### 3.6.2.3 tk\_snd\_mbf(Send Message to Message Buffer)

メッセージバッファへ送信します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_snd_mbf ( ID mbfid, VP msg, INT msgsz, TMO tmout ) ;
```

#### ■ パラメータ

##### ● 入力

mbfid	メッセージバッファ ID (Message Buffer ID)
msgsz	送信メッセージのサイズ ( バイト数 ) (Message Size)
msg	送信メッセージの先頭アドレス (Message)
tmout	タイムアウト指定 (Timeout)
0 から 0x7fffffff の数値以外に以下のマクロが指定可能	

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

##### ● 出力

ercd エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mbfid が 0 以下または最大メッセージバッファ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mbfid のメッセージバッファが存在しない)
E_PAR	-17	パラメータエラー (msgsz 0, msgsz > maxmsz, tmout (-2))
E_DLT	-51	待ちオブジェクトが削除された ( 待ちの間に対象メッセージバッファが待ちオブジェクト を削除 )
E_RLWAI	-49	待ち状態強制解除 ( 待ちの間に tk_rel_wai を受け付ける )
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CTX	-25	コンテキストエラー ( タスク独立部またはディスパッチ禁止状態で実行 (tmout が TMO_POL 以外 ) )

### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクのメッセージバッファ受信待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。または、システムコールを呼び出したタスクがメッセージバッファ送信待ち状態になった場合に、次の優先順位のタスクにディスパッチします。

### ■ 解説

本システムコールでは、mbfid で示されたメッセージバッファにmsg のアドレスに入っているメッセージを送信します。

mbfid で指定されたメッセージ ID が 0 以下または最大メッセージバッファ数（コンフィギュレータで設定した最大メッセージバッファ数）より大きい場合には E\_ID のエラーで復帰します。対象メッセージバッファが存在しない場合には E\_NOEXS のエラーで復帰します。

メッセージのサイズは msgsz で指定します。すなわち、msg 以下の msgsz バイトを mbfid で指定されたメッセージバッファのメッセージキューにコピーします。メッセージキューは、リングバッファによって実現しています。

msgsz が 0 以下、msgsz が k\_cre\_mbf で指定された maxmsz より大きい場合には E\_PAR のエラーで復帰します。

tmout により待ち時間の最大値（タイムアウト値）を指定できます。タイムアウト指定が行われた場合には待ち解除の条件が満足されない（バッファに十分な空き領域ができない）まま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、バッファに空きができるまで、または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL はバッファに十分な空きがない場合でも待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間（時間の単位）はシステム時刻の基準時間（= 1 ms）と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

バッファの空き領域が少なく、msg のメッセージをメッセージキューに入れられない場合には本システムコールを呼び出したタスクはメッセージ送信待ち状態となり、バッファの空きを待つための待ち行列（送信待ち行列）につながれます。待ち行列の順序は tk\_cre\_mbf 呼出し時の指定により FIFO またはタスク優先度順となります。

タスク独立部やディスパッチ禁止状態から実行した場合には E\_CTX のエラーで復帰します。メッセージ送信待ち状態で対象メッセージバッファが tk\_del\_mbf により削除された場合には、待ち状態は解除されて本システムコールは E\_DLT のエラーで復帰します。メッセージ送信待ち状態のタスクに tk\_rel\_wai が呼び出された場合には待ち状態は解除されて E\_RLWAI のエラーで復帰します。

msg が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.6.2.4 tk\_rcv\_mbf(Receive Message from Message Buffer)

メッセージバッファから受信します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
INT msgsz = tk_rcv_mbf ( ID mbfid, VP msg, TMO tmout ) ;
```

#### ■ パラメータ

● 入力

mbfid	メッセージバッファ ID (Message Buffer ID)
msg	受信メッセージを入れるアドレス (Message)
tmout	タイムアウト指定 (Timeout) 0 から 0x7fffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

● 出力

msgsz	受信したメッセージのサイズ ( バイト数 ) (Message size) またはエラーコード (Error Code)
-------	--

#### ■ エラーコード

E_ID	-18	不正 ID 番号 (mbfid が 0 以下または最大メッセージバッファ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mbfid のメッセージバッファが存在しない)
E_PAR	-17	パラメータエラー (tmout (-2))
E_DLT	-51	待ちオブジェクトが削除された ( 待ちの間に対象メッセージバッファが待ちオブジェクト を削除 )
E_RLWAI	-49	待ち状態強制解除 ( 待ちの間に tk_rel_wai を受け付ける )
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CTX	-25	コンテキストエラー ( タスク独立部またはディスパッチ禁止状態で実行 )

### ■ ディスパッチ要因

本システムコールを呼び出したタスクがメッセージバッファ受信待ち状態になった場合に、次の優先順位のタスクにディスパッチします。または、システムコールを呼び出したタスクより優先度の高いタスクのメッセージバッファ送信待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。

### ■ 解説

本システムコールでは、mbfid で示されたメッセージバッファからメッセージを受信して msg で指定された領域に入れます。すなわち、mbfid で指定されたメッセージバッファのメッセージキューの先頭メッセージの内容を msg 以下の msgsz バイトにコピーします。

mbfid で指定されたメッセージ ID が 0 以下または最大メッセージバッファ数 (コンフィギュレータで設定した最大メッセージバッファ数) より大きい場合には E\_ID のエラーで復帰します。対象メッセージバッファが存在しない場合には E\_NOEXS のエラーで復帰します。

tmout により待ち時間の最大値 (タイムアウト値) を指定できます。タイムアウト指定が行われた場合には待ち解除の条件が満足されない (メッセージが到着しない) まま tmout の時間が経過すると、E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、メッセージが到着するまで、または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL はメッセージがない場合でも待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間 (時間の単位) はシステム時刻の基準時間 (= 1 ms) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

mbfid で示されたメッセージバッファにまだメッセージが送信されていない場合 (メッセージキューが空の場合) には、本システムコールを呼び出したタスクは待ち状態となり、メッセージの到着を待つ待ち行列 (受信待ち行列) につながれます。受信待ちタスクの待ち行列は FIFO だけです。

メッセージ受信待ち状態で対象メッセージバッファが tk\_del\_mbf により削除された場合には、待ち状態が解除されて本システムコールは E\_DLT のエラーで復帰します。メッセージ受信待ち状態のタスクに tk\_rel\_wai が呼び出された場合には、待ち状態が解除されて E\_RLWAI のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

msg が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.6.2.5 tk\_ref\_mbf(Refer Message Buffer Status)

メッセージバッファの状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

typedef struct t_rmbf {
    VP  exinf;
    ID  wtsk;
    ID  stsk;
    INT msgsz;
    INT frbufsz;
    INT maxmsz;
} T_RMBF;
```

#### ■ パラメータ

● 入力

mbfid      メッセージバッファ ID  
            (Message Buffer ID)

pk\_rmbf    メッセージバッファの状態を返すパケットアドレス  
            (Packet of Refer Message Buffer)

● 出力

ercd      エラーコード (Error Code)

            パケットに返すデータ

exinf      拡張情報 (Extended Information)

wtsk      受信待ちタスクの有無 (Wait Task)

stsk      送信待ちタスクの有無 (Send Task)

msgsz      メッセージサイズ (Message Size)

frbufsz    空きバッファのサイズ ( バイト数 ) (Free Buffer Size)

maxmsz    メッセージの最大長 ( バイト数 )  
            (Maximum Message Size)

### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mbfid が 0 以下または最大メッセージバッファ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mbfid のメッセージバッファが存在しない)

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

mbfid で示された対象メッセージバッファの各種状態を参照して、復帰値として拡張情報 (exinf), 受信待ちタスクの有無 (wtsk), 送信待ちタスクの有無 (stsk), 次に受信するメッセージのサイズ (msgsz), 空きバッファのサイズ (frbufsz), メッセージの最大長 (maxmsz) を返します。

mbfid で指定されたメッセージ ID が 0 以下または最大メッセージバッファ数 (コンフィギュレータで設定した最大メッセージバッファ数) より大きい場合には E\_ID のエラーで復帰します。対象メッセージバッファが存在しない場合には E\_NOEXS のエラーで復帰します。

wtsk は、このメッセージバッファで受信待ちしているタスクの ID を設定します。また、stsk は送信待ちしているタスクの ID を設定します。このメッセージバッファで複数のタスクが待っている場合には待ち行列の先頭タスクの ID を返します。待ちタスクがない場合には 0 となります。

msgsz には、メッセージキューの先頭メッセージ (次に受信するメッセージ) のサイズが返ります。メッセージキューにメッセージがない場合には msgsz は 0 となります。

なお、サイズが 0 のメッセージを送れません。どんな場合でも、msgsz = 0 と wtsk = 0 の少なくとも一方は成り立ちます。

frbufsz はメッセージキューを構成するリングバッファの空き領域のサイズを示すものです。この値により、送信可能なメッセージ量を知ることができます。maxmsz は tk\_cre\_mbf で指定されたメッセージの最大長を返します。

pk\_rmbf が不正な場合でも、エラーチェックを行わず、動作は保証されません。



### 3.6.3 ランデブポート機能のシステムコール

---

ランデブポート機能のシステムコールについて説明します。

---

#### ■ ランデブポート機能のシステムコール

ランデブポート機能は、以下の7種類のシステムコールで構成されます。

- tk\_cre\_por(Create Port for Rendezvous)
- tk\_del\_por>Delete Port for Rendezvous)
- tk\_cal\_por(Call Port for Rendezvous)
- tk\_acp\_por(Accept Port for Rendezvous)
- tk\_fwd\_por(Forward Port for Rendezvous)
- tk\_rpl\_rdv(Reply Rendezvous)
- tk\_ref\_por(Refer Port Status)

### 3.6.3.1 tk\_cre\_por(Create Port for Rendezvous)

ランデブポートを生成します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

#### ■ C 言語インタフェース

```
ID porid = tk_cre_por ( T_CPOR *pk_cpor ) ;
```

```
typedef struct t_cpor {
    VP  exinf;
    ATR poratr;
    INT maxcmsz;
    INT maxrmsz;
} T_CPOR;
```

#### ■ パラメータ

##### ● 入力

pk\_cpor   ランデブポートの生成情報のパケット先頭アドレス(Packet of Create Port)  
           パケットに設定するデータ

exinf     拡張情報 (Extended Information)

poratr     ランデブポート属性 (Port Attribute)  
           poratr := (TA\_TFIFO   TA\_TPRI)

属性	値	意味
TA_TFIFO	0x00000000	待ちタスクを FIFO で管理
TA_TPRI	0x00000001	待ちタスクを優先度順で管理

maxcmsz   呼出し時のメッセージの最大長 ( バイト数 )  
           (Maximum Call Message Size)

maxrmsz   返答時のメッセージの最大長 ( バイト数 )  
           (Maximum Reply Message Size)

##### ● 出力

porid     ランデブポート ID (Port ID)  
           またはエラーコード (Error Code)

## ■ エラーコード

E_LIMIT	-34	ランデブポートの数がシステムの上限より大きい
E_RSATR	-11	予約属性 (poratr に未定義の値が指定された)
E_PAR	-17	パラメータエラー (maxcmsz が負, maxrmsz が負)
E_CTX	-25	コンテキストエラー (タスク独立部で実行)

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

ランデブポートを生成しランデブポート ID 番号を割り当てます。ランデブポートは、ランデブを実現するための基本となるオブジェクトです。

exinf は、対象ランデブポートに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報を tk\_ref\_por で取り出せます。

なお、ユーザの情報を入れるためにもっと大きな領域が必要な場合や、途中で内容を変更したい場合には、自分でそのためのメモリを確保し、そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容については関知しません。

poratr はランデブポートの属性を指定します。未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

TA\_TFIFO, TA\_TPRI では、ランデブ呼出し待ちのタスクの待ち行列の並び順を指定します。ランデブ受付け待ちのタスクの待ち行列は FIFO だけです。

maxcmsz は呼出し時のメッセージの最大長をバイト数で指定します。maxcmsz が負の場合には E\_PAR のエラーで復帰します。

maxrmsz は、返答時のメッセージの最大長をバイト数で指定します。maxrmsz が負の場合には E\_PAR のエラーで復帰します。

ランデブポートがシステムの上限值 (コンフィギュレータで設定した最大ランデブポート数) まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。タスク独立部で本システムコールを発行した場合には E\_CTX のエラーで復帰します。また、pk\_cpor が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.6.3.2 tk\_del\_por(Delete Port for Rendezvous)

ランデブポートを削除します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_del_por ( ID porid ) ;
```

#### ■ パラメータ

##### ● 入力

porid      ランデブポート ID (Port ID)

##### ● 出力

ercd      エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (porid が 0 以下または最大ランデブポート数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (porid のランデブポートが存在しない)
E_CTX	-25	コンテキストエラー (タスク独立部で実行)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクが削除したランデブポートで待っていた場合に、待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

porid で示されたランデブポートを削除します。具体的には、対象ランデブポートを未生成状態にして ID 番号を解放します。

porid で指定されたランデブポート ID が 0 以下または最大ランデブポート数 (コンフィギュレータで設定した最大ランデブポート数) より大きい場合には E\_ID のエラーで復帰します。対象ランデブポートが存在しない場合には E\_NOEXS のエラーで復帰します。

対象ランデブポートにおいてランデブ受付け (tk\_acp\_por) や呼出し (tk\_cal\_por) を待っているタスクがあった場合にも本システムコールは正常終了しますが、待ち状態にあったタスクには E\_DLT のエラーで復帰します。tk\_del\_por によりランデブポートが削除されても、既にランデブ成立済みのタスクに影響しません。ランデブ受付け側タスク (待ち状態ではない) には何も通知されず、ランデブ呼出し側タスク (ランデブ終了待ち状態) の状態も変化しません。ランデブ受付け側タスクが tk\_rpl\_rdv を呼び出すときに、ランデブ成立に使ったランデブポートが既に削除されていても tk\_rpl\_rdv を正常に実行します。

タスク独立部で本システムコールを発行した場合には E\_CTX のエラーで復帰します。

3.6.3.3 tk\_cal\_por(Call Port for Rendezvous)

ランデブポートに対するランデブを呼び出します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

■ C 言語インタフェース

INT rmsgsz = tk\_cal\_por ( ID porid, UINT calptn, VP msg, INT cmsgsz, TMO tmout ) ;

■ パラメータ

● 入力

porid	ランデブポート ID (Port ID)
calptn	呼出し側選択条件を表すビットパターン (Call Bit Pattern)
msg	メッセージを入れるアドレス (Message)
cmsgsz	呼出しメッセージのサイズ ( バイト数 ) (Call Message Size)
tmout	タイムアウト指定 (Timeout)
0 から 0x7ffffff の数値以外に以下のマクロが指定可能	

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

● 出力

rmsgsz	返答メッセージのサイズ ( バイト数 )(Reply Message Size) またはエラーコード (Error Code)
--------	---

■ エラーコード

E_ID	-18	不正 ID 番号 (porid が 0 以下または最大ランデブポート数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (porid のランデブポートが存在しない)
E_PAR	-17	パラメータエラー (cmsgsz < 0, cmsgsz > maxcmsz, calptn = 0, tmout (-2))
E_DLT	-51	待ちオブジェクトが削除された ( 待ちの間に対象ランデブポートが待ちオブジェクトを削除 )
E_RLWAI	-49	待ち状態強制解除 ( 待ちの間に tk_rel_wai を受け付ける )
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CTX	-25	コンテキストエラー ( タスク独立部またはディスパッチ禁止状態で実行 )

## ■ ディスパッチ要因

本システムコールを呼び出したタスクがランデブ呼出し待ち状態になった場合に、次の優先順位のタスクにディスパッチします。または、システムコールを呼び出したタスクより優先度の高いタスクのランデブ受付け待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。

## ■ 解説

porid で指定されたランデブポートに対するランデブを呼び出します。

porid で指定されたランデブポート ID が 0 以下または最大ランデブポート数 ( コンフィギュレータで設定した最大ランデブポート数 ) より大きい場合には E\_ID のエラーで復帰します。対象ランデブポートが存在しない場合には E\_NOEXS のエラーで復帰します。

呼出しメッセージのサイズを cmsgsz で指定します。本システムコールで指定した msg 領域 ( 呼出し側タスクの msg 領域 ) の cmsgsz バイトのデータを tk\_acp\_por で指定された msg 領域 ( 受付け側タスクの msg 領域 ) にコピーします。

cmsgsz が tk\_cre\_por で指定された maxcmsz よりも大きい場合には E\_PAR のエラーで復帰します。このエラーはランデブ呼出し待ち状態に入る前にチェックされ、エラーの場合には本システムコールを実行したタスクはランデブ呼出し待ち状態には入りません。cmsgsz が負か、cmsgsz が maxcmsz より大きい場合には E\_PAR のエラーで復帰します。

tmout により待ち時間の最大値 ( タイムアウト値 ) を指定できます。タイムアウト指定が行われた場合に待ち解除の条件が満足されない ( ランデブが成立しない ) まま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、ランデブが成立するまで、または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL は対象ランデブポートにランデブ受付け待ちタスクがない、またはランデブ成立条件が満たされない場合でも待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間 ( 時間の単位 ) はシステム時刻の基準時間 ( = 1 ms ) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

本システムコールの具体的な動作は以下のようになります。

porid で指定されたランデブポートにおいてランデブ受付け待ち状態のタスクがあり、そのタスクと本システムコール呼出しタスクとの間でランデブ成立条件が満たされた場合にはランデブ成立となります。この場合、ランデブ受付け待ちだったタスクは実行可能状態となり、本システムコール呼出しタスクはランデブ終了待ちの状態になります。ランデブ終了待ちの状態になったタスクは、ランデブの相手のタスク ( ランデブ受付けタスク ) が tk\_rpl\_rdv システムコールを実行することにより待ち状態を解除します。この時点で本システムコールが終了します。

porid で指定されたランデブポートに受付け待ちタスクがなかった場合や、受付け待ちタスクがあってもランデブ成立条件が満たされなかった場合には、本システムコール呼出しタスクはこのランデブポートの呼出し側待ち行列に並びランデブ呼出し待ちの状態となります。ランデブ呼出し待ち行列の順序は、tk\_cre\_por の指定により FIFO またはタスク優先度順となります。

ランデブ成立条件は、受付け側タスクの acpptn と呼出し側タスクの calptn との論理積が 0 かどうかによって判定します。論理積が 0 以外の場合にランデブ成立となります。calptn が 0 の場合は、ランデブが成立しなくなるので E\_PAR のエラーで復帰します。

ランデブ成立時には、呼出し側タスクから受付け側タスクにメッセージ（呼出しメッセージ）を送れます。

逆に、ランデブ終了時には、受付け側タスクから呼出し側タスクにメッセージ（返答メッセージ）を送れます。tk\_rpl\_rdv で指定した応答メッセージ（呼出し側タスクの応答メッセージ）の内容を、本システムコールで指定したメッセージ領域（呼出し側タスクの msg 領域）にコピーします。また、返答メッセージのサイズ rmsgsz は、本システムコールの復帰値となります。結局、本システムコールの msg パラメータで指定されたメッセージ領域を tk\_rpl\_rdv 実行のときに送られてくるメッセージによって破壊します。

なお、ランデブが回送された場合は、本システムコールで指定された msg のアドレスから最大で maxrmsz の領域をバッファとして使用するため、その内容を破壊する可能性があります。したがって、本システムコールで要求したランデブが回送される可能性がある場合は、期待する返答メッセージのサイズにかかわらず、msg 以下に少なくとも maxrmsz のサイズの領域を確保してください（詳細は「3.6.3.5 tk\_fwd\_por(Forward Rendezvous to Another Port)」を参照）。

ランデブ呼出し待ち、またはランデブ終了待ちの状態で対象ランデブポートが tk\_del\_por により削除された場合、待ち状態は解除されて本システムコールは E\_DLT のエラーで復帰します。ランデブ呼出し待ち、またはランデブ終了待ちの状態のタスクに tk\_rel\_wai システムコールが呼び出された場合、待ち状態は解除されて E\_RLWAI のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

msg が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.6.3.4 tk\_acp\_por(Accept Port for Rendezvous)

ランデブポートに対するランデブを受け付けます。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
INT cmsgsz = tk_acp_por ( ID porid, UINT acpptn, RNO *p_rdvno, VP msg, TMO tmout );
```

#### ■ パラメータ

##### ● 入力

porid	ランデブポート ID (Port ID)
acpptn	受け付け側選択条件を表すビットパターン (Accept Bit Pattern)
p_rdvno	ランデブ番号を入れるアドレス (Rendezvous Number)
msg	メッセージを入れるアドレス (Message)
tmout	タイムアウト指定 (Timeout)
0 から 0x7ffffff の数値以外に以下のマクロが指定可能	

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

##### ● 出力

cmsgsz	呼出しメッセージのサイズ (バイト数) またはエラーコード (Call Message Size or Error Code)
--------	--

#### ■ エラーコード

E_ID	-18	不正 ID 番号 (porid が 0 以下または最大ランデブポート数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (porid のランデブポートが存在しない)
E_PAR	-17	パラメータエラー (acpptn = 0, tmout (-2))
E_DLT	-51	待ちオブジェクトが削除された ( 待ちの間に対象ランデブポートが待ちオブジェクトを削除 )
E_RLWAI	-49	待ち状態強制解除 ( 待ちの間に tk_rel_wai を受け付ける )
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CTX	-25	コンテキストエラー ( タスク独立部またはディスパッチ禁止状態で実行 )



## ■ ディスパッチ要因

本システムコールを呼び出したタスクがランデブ受け待ち状態になった場合に、次の優先順位のタスクにディスパッチします。または、システムコールを呼び出したタスクより優先度の高いタスクのランデブ呼出し待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。

## ■ 解説

ランデブポートに対するランデブを受付けます。

porid で指定されたランデブポート ID が 0 以下または最大ランデブポート数 ( コンフィギュレータで設定した最大ランデブポート数 ) より大きい場合には E\_ID のエラーで復帰します。対象ランデブポートが存在しない場合には E\_NOEXS のエラーで復帰します。

tmout により待ち時間の最大値 ( タイムアウト値 ) を指定できます。タイムアウト指定が行われた場合に待ち解除の条件が満足されない ( ランデブが成立しない ) まま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、ランデブの成立まで、または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL は、対象ランデブポートにランデブ呼出し待ちタスクがないか、ランデブ成立条件が満たされない場合でも待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間 ( 時間の単位 ) はシステム時刻の基準時間 ( = 1 ms ) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

本システムコールの具体的な動作は以下のようになります。

porid で指定されたランデブポートの呼出し側待ち行列に入っているタスクと、このタスクとの間でランデブ成立条件が満たされた場合はランデブ成立となります。この場合、呼出し側待ち行列にあったタスクは行列から外れ、ランデブ呼出し待ち ( ランデブ成立待ち ) の状態からランデブ終了待ちの状態に変わります。本システムコールの呼出しタスクは実行を継続します。

porid で指定されたランデブポートの呼出し側待ち行列にタスクがなかった場合や、タスクがあってもランデブ成立条件が満たされなかった場合、本システムコールの呼出しタスクはそのランデブポートに対するランデブ受け待ち状態になります。

この場合、既に別のタスクがランデブ受け待ち状態であったとしても、エラーとはならず、本システムコールを呼び出したタスクはランデブ受け待ち行列につながれます。また、1つのランデブポートを使って複数のタスクが同時にランデブを行えます。そのため、porid で指定されたランデブポートで別のタスクがランデブを行っている間に ( 前に成立したランデブに関する tk\_rpl\_rdv が実行される前に ) 次のランデブを行っても正常終了します。

ランデブ成立条件は、受け側タスクの acptn と呼出し側タスクの calptn との論理積が 0 かどうかによって判定します。論理積が 0 以外の場合にランデブ成立となります。先頭のタスクが条件を満たさない場合には、待ち行列の次のタスクについて順にチェックします。calptn と acptn に 0 以外の同じ値を指定すると条件がない ( 無条件 ) のと同じになります。また、acptn が 0 の場合は、ランデブが成立しなくなるので E\_PAR のエラーで復帰します。ランデブ成立までの処理に関しては、ランデブ呼出し側とランデブ受け側で完全に対称です。

ランデブ成立時には、呼出し側タスクから受付け側タスクに呼出しメッセージを送れます。呼出し側タスクが指定した呼出しメッセージの内容を受付け側タスクが本システムコールで指定した `msg` 以下の領域にコピーします。また、呼出しメッセージのサイズ `cmsgsz` は、本システムコールの復帰値となります。

ランデブ受付け側のタスクが同時に複数のランデブを行うこともできます。具体的には、本システムコールによりあるランデブを受け付けたタスクが、`tk_rpl_rdv` を実行する前にもう一度本システムコールを実行しても構いません。また、この場合の本システムコールは、前と異なるランデブポートを対象としたものであっても、前と同じランデブポートを対象としたものであっても構いません。特殊な例ですが、ランデブ中のタスクが同じランデブポートにもう一度本システムコールを実行してランデブが成立した場合には同一のタスクが同一のランデブポートに複数の(多重の)ランデブを行っている状態になります。もちろん、この場合にランデブの相手(呼出し側タスク)は異なっています。

本システムコールの復帰値として返すランデブ番号、`p_rdvno` は、同時に成立している複数のランデブを区別するための情報であり、ランデブ終了時に `tk_rpl_rdv` のパラメータとして使用します。また、ランデブ回送時には `tk_fwd_por` のパラメータとして使用します。ランデブ番号は、下位 16 ビットがランデブを受け付けたタスクのタスク ID、上位 16 ビットはランデブ受付け順に採番されます。

ランデブ受付け待ち、またはランデブ終了待ちの状態を対象ランデブポートが `tk_del_por` により削除された場合には待ち状態は解除されて本システムコールは `E_DLT` のエラーで復帰します。ランデブ受付け待ち、またはランデブ終了待ちの状態のタスクに `tk_rel_wai` システムコールが呼び出された場合には待ち状態は解除されて `E_RLWAI` のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には `E_CTX` のエラーで復帰します。

`p_rdvno`, `msg` が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.6.3.5 tk\_fwd\_por(Forward Rendezvous to Another Port)

ランデブポートに対するランデブを回送します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_fwd_por ( ID porid, UINT calptn, RNO rdvno, VP msg, INT cmsgsz ) ;
```

#### ■ パラメータ

##### ● 入力

porid	回送先のランデブポート ID (Port ID)
calptn	呼出し側選択条件を表すビットパターン (Call Bit Pattern)
rdvno	回送前のランデブ番号 (Rendezvous Number)
msg	呼出しメッセージを入れるアドレス (Message)
cmsgsz	呼出しメッセージのサイズ (バイト数) (Call Message Size)

##### ● 出力

ercd	エラーコード (Error Code)
------	---------------------

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (porid が 0 以下または最大ランデブポート数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (porid のランデブポートが存在しない)
E_PAR	-17	パラメータエラー (cmsgsz < 0, cmsgsz > 回送後の maxcmsz, cmsgsz > 回送前の maxrmsz, calptn = 0)
E_OBJ	-41	オブジェクトの状態が不正 (rdvno に含まれるタスク ID が負または最大タスク数より大きい, 呼出し側タスクの状態がランデブ終了待ち (TTW_RDV) 以外, rdvno が tk_acp_por の復帰値と一致しない, 回送後の maxrmsz > 回送前の maxrmsz)
E_CTX	-25	コンテキストエラー (タスク独立部で実行)

#### ■ ディスパッチ要因

システムコールを呼び出したタスクより優先度の高いタスクのランデブ受付け待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。

## ■ 解説

いったん受け付けたランデブを別のランデブポートに回送します。

porid で指定されたランデブポート ID が 0 以下または最大ランデブポート数 ( コンフィギュレータで設定した最大ランデブポート数 ) より大きい場合には E\_ID のエラーで復帰します。対象ランデブポートが存在しない場合には E\_NOEXS のエラーで復帰します。

本システムコールは、現在ランデブ中の状態 ( tk\_acp\_por を実行した後の状態 ) のタスク ( タスク X とする ) から呼び出してください。例えば、ランデブ相手の呼出し側タスクをタスク Y, tk\_acp\_por の復帰値として返されたランデブ番号を rdvno とします。その状態で本システムコールを実行すると、タスク X とタスク Y との間のランデブ状態が解除され、その後 porid で指定された別のランデブポート ( ランデブポート B ) にタスク Y がランデブ呼出しを行ったのと同じ状況になります。

本システムコールの具体的な動作は以下のようになります。

1. rdvno で指定されたランデブを解除します。
2. タスク Y を porid のランデブポートにランデブ呼出し待ちの状態にします。この場合、ランデブ成立のための呼出し側選択条件を表すビットパターン calptn は、タスク Y が tk\_cal\_por で指定したものではなく、タスク X が本システムコールで指定したものを使用します。タスク Y から見ると、ランデブ終了待ちの状態からランデブ呼出し待ちの状態に戻ります。
3. その後、porid のランデブポートに対するランデブが受け付けられると、それを受け付けたタスクとタスク Y との間でランデブ成立となります。もちろん、porid のランデブポートに既にランデブ受け付け待ちタスクが存在し、ランデブ成立条件が満たされていると、本システムコールの実行により即座にランデブ成立となることもあります。ここで、ランデブ成立時に受け付け側に送られるメッセージは、タスク Y が tk\_cal\_por で指定したものではなく、タスク X が本システムコールで指定したものを使用します ( calptn と同様 )。
4. 新しいランデブが終了したときに tk\_rpl\_rdv で返すメッセージは、タスク X が本システムコールで指定した msg の領域ではなく、タスク Y が tk\_cal\_por で指定した msg の領域にコピーします。

基本的には、「 tk\_cal\_por ( porid=portA, calptn=ptnA, msg=mesA ) の後で tk\_fwd\_por ( porid=portB, calptn=ptnB, msg=mesB ) が実行された状態」と、「 tk\_cal\_por ( porid=portB, calptn=ptnB, msg=mesB ) が実行された状態」とは全く同じ状態になります。ランデブ回送の履歴を覚えておく必要はありません。

本システムコールによってランデブ呼出し待ち状態に戻ったタスクに tk\_ref\_tsk を実行した場合、tskwait は TTW\_CAL となります。また、wid も回送先のランデブポートの ID 番号になります。

本システムコールの実行は即座に終了します。本システムコールで待ち状態になりません。また、本システムコールの実行が終わった後の本システムコール呼出しタスクは、回送前のランデブが成立したランデブポート、回送後のランデブポート ( porid のランデブポート )、それらのうえでランデブを行ったタスクのいずれとも無関係になります。

cmsgsz が、回送後のランデブポートの maxcmsz よりも大きい場合には E\_PAR のエラーで復帰します。このエラーはランデブの回送を行う前にチェックします。エラーの場合にはランデブの回送は行わず、rdvno で指定されたランデブも解除しません。

cmsgsz が 0 以下, cmsgsz が回送後の maxcmsz より大きい, または cmsgsz が回送前の maxrmsz より大きい場合には E\_PAR のエラーで復帰します。

本システムコールで指定された送信メッセージは, 本システムコール実行時にほかの領域 (例えば tk\_cal\_por で指定したメッセージ領域) にコピーします。したがって, 回送されたランデブが成立する前に本システムコールの msg で指定されたメッセージ領域の内容が変更されても, 回送されたランデブには影響しません。

本システムコールでランデブを回送する場合には回送後のランデブポート (porid のランデブポート) の maxrmsz を回送前のランデブの成立したランデブポートの maxrmsz と等しいか, それよりも小さくしてください。回送後のランデブポートの maxrmsz が回送前のランデブポートの maxrmsz よりも大きかった場合には回送先のランデブポートが不適当なので E\_OBJ のエラーで復帰します。ランデブ呼出し側では, 回送前のランデブポートの maxrmsz に合わせて返答メッセージ受信領域を用意しているため, ランデブの回送によって返答メッセージの最大サイズが大きくなると, 呼出し側に予期せぬ大きさの返答メッセージを返す可能性があり問題を起こします。maxrmsz の大きなランデブポートにランデブを回送できないのは, この理由によります。また, 本システムコールで送信するメッセージのサイズ cmsgsz についても, 回送前のランデブの成立したランデブポートの maxrmsz と等しいか, それよりも小さくしてください。これは, 本システムコールの実装方法として, tk\_cal\_por で指定されたメッセージ領域を送信メッセージのバッファとして使うことを想定しているためです。cmsgsz が回送前のランデブポートの maxrmsz より大きかった場合には E\_PAR のエラーで復帰します。

ディスパッチ禁止中あるいは割込み禁止中のタスクから本システムコール, tk\_rpl\_rdv を呼び出した場合も, これらのシステムコールは正常に動作します。この機能は, 本システムコールや tk\_rpl\_rdv と不可分に何らかの処理を行う場合に使用できます。

本システムコールにより, ランデブ終了待ち状態であったタスク Y がランデブ呼出し待ちの状態に戻った場合, 次にランデブが成立するまでのタイムアウトは常に永久待ち (TMO\_FEVR) として扱われます。

回送先のランデブポートは, 前のランデブに使用していたランデブポート (rdvno のランデブが成立したランデブポート) と同じランデブポートであっても構いません。この場合は, 本システムコールによって, いったん受け付けたランデブの受け付け処理をとりやめます。ただし, この場合でも, 呼出しメッセージや calptn は, 呼出し側タスクが tk\_cal\_por で指定したのではなく, 受け付け側タスクが本システムコールで指定したものに変わります。また, いったん回送されてきたランデブをさらに回送することもできます。

calptn が 0 の場合には E\_PAR のエラーで復帰します。rdvno に含まれるタスク ID が負または最大タスク数より大きい, 呼出し側タスクが待ち状態でない, 呼出し側タスクの待ち状態がランデブ終了待ち (TTW\_RDV) でない, rdvno が tk\_acp\_por の復帰値と一致しない, または回送後の maxrmsz が回送前の maxrmsz より大きい場合には E\_OBJ のエラーで復帰します。

msg が不正な場合でも, エラーチェックを行わず, 動作は保証されません。

### 3.6.3.6 tk\_rpl\_rdv(Reply Rendezvous)

---

ランデブポートに対するランデブに返答します。

---

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_rpl_rdv ( RNO rdvno, VP msg, INT rmsgsz ) ;
```

#### ■ パラメータ

##### ● 入力

rdvno	ランデブ番号 (Rendezvous Number)
msg	メッセージを入れるアドレス (Message)
rmsgsz	返答メッセージのサイズ (バイト数) (Reply Message Size)

##### ● 出力

ercd	エラーコード (Error Code)
------	---------------------

#### ■ エラーコード

E_OK	0	正常終了
E_PAR	-17	パラメータエラー (rmsgsz < 0, rmsgsz > maxrmsz)
E_OBJ	-18	オブジェクトの状態が不正 (rdvno に含まれるタスク ID が 0 以下または最大タスク数より大きい, 呼出し側タスクが待ち状態でない, 呼出し側タスクの待ち状態がランデブ終了待ち (TTW_RDV) でない, rdvno が tk_acp_por の復帰値と一致しない)
E_CTX	-25	コンテキストエラー (タスク独立部から実行)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクのランデブ終了待ち状態を解除した場合に, 待ち状態が解除されたタスクにディスパッチします。

## ■ 解説

ランデブの相手のタスク（呼出し側タスク）に返答を返してランデブを終了します。

本システムコールは、ランデブ中の状態 (`tk_acp_por` を実行した後の状態) のタスク（タスク X とする）から呼び出してください。例えば、ランデブ相手の呼出し側タスクをタスク Y、`tk_acp_por` の復帰値として返されたランデブ番号を `rdvno` とします。その状態で本システムコールを実行すると、タスク X とタスク Y との間のランデブ状態が解除されてランデブ終了待ち状態にあった呼出し側タスク Y は実行可能状態に戻ります。

本システムコールでランデブを終了させる場合には受付け側タスク X から呼出し側タスク Y に返答メッセージを送れます。受付け側タスクが指定した返答メッセージの内容を呼出し側タスクが `tk_cal_por` で指定した `msg` 以下の領域にコピーします。また、返答メッセージのサイズ `rmsgsz` は `tk_cal_por` の復帰値となります。

`rmsgsz` が負または `tk_cre_por` で指定された `maxrmsz` よりも大きい場合には `E_PAR` のエラーで復帰します。このエラーが検出された場合、ランデブは終了せずに `tk_cal_por` を実行したタスクのランデブ終了待ち状態を解除しません。

`rdvno` に含まれるタスク ID (`rdvno` の下位 16 ビット) が 0 以下または最大タスク数（コンフィギュレータで設定した最大ランデブポート数）より大きい、呼出し側タスクが待ち状態でない、呼出し側タスクの待ち状態がランデブ終了待ち (`TTW_RDV`) でない、`rdvno` が `tk_acp_por` の復帰値と一致しない場合には `E_OBJ` のエラーで復帰します。

`msg` が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.6.3.7 tk\_ref\_por(Refer Port Status)

ランデブポートの状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

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

```
typedef struct t_rpor {
    VP  exinf;
    ID  wtsk;
    ID  atsk;
    INT maxcmsz;
    INT maxrmsz;
} T_RPOR;
```

#### ■ パラメータ

##### ● 入力

porid      ランデブポート ID (Port ID)

pk\_rpor    ランデブポート状態を返すパケットの先頭アドレス  
(Packet of Refer Port)

##### ● 出力

ercd      エラーコード (Error Code)

          パケットに返すデータ

exinf      拡張情報 (Extended Information)

wtsk      呼出し待ちタスクの有無 (Wait Task)

atsk      受付け待ちタスクの有無 (Accept Task)

maxcmsz    呼出し時のメッセージの最大長 ( バイト数 )  
(Maximum Call Message Size)

maxrmsz    返答時のメッセージの最大長 ( バイト数 )  
(Maximum Receive Message Size)

#### ■ エラーコード

E\_OK      0      正常終了

E\_ID      -18    不正 ID 番号  
(porid が 0 以下または最大ランデブポート数より大きい)

E\_NOEXS   -42    オブジェクトが存在していない  
(porid のランデブポートが存在しない)



## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

porid で指定された対象ランデブポートの各種の状態を参照し、復帰値として受付け待ちタスクの有無 (atsk)、呼出し待ちタスクの有無 (wtsk)、メッセージの最大長 (maxcmsz, maxrmsz)、拡張情報 (exinf) を返します。

porid で指定されたランデブポート ID が 0 以下または最大ランデブポート数 (コンフィギュレータで設定した最大ランデブポート数) より大きい場合には E\_ID のエラーで復帰します。対象ランデブポートが存在しない場合には E\_NOEXS のエラーで復帰します。

wtsk は、このランデブポートでランデブ呼出し待ちになっているタスクの ID を設定します。ランデブ呼出し待ちタスクがない場合には wtsk が 0 となります。一方、atsk は、このランデブポートでランデブ受付け待ちになっているタスクの ID を設定します。ランデブ受付け待ちタスクがない場合には atsk が 0 となります。

このランデブポートで複数のタスクが呼出し待ち状態または受付け待ち状態になっている場合には、それぞれ呼出し待ち行列または受付け待ち行列の先頭タスクの ID を返します。

pk\_rpor が不正な場合でも、エラーチェックを行わず、動作は保証されません。

## ■ 補足事項

本システムコールでは、現在ランデブ中のタスクに関する情報を知ることはできません。

## 3.7 メモリプール管理機能のシステムコール

---

メモリプール管理機能のシステムコールについて説明します。

---

### ■ メモリプール管理機能のシステムコール

メモリプール機能は、以下の 2 種類の機能のシステムコールで構成されます。

- 固定長メモリプール機能のシステムコール
- 可変長メモリプール機能のシステムコール

### 3.7.1 固定長メモリプール機能のシステムコール

---

固定長メモリプール機能のシステムコールについて説明します。

---

#### ■ 固定長メモリプール機能のシステムコール

固定長メモリプール機能は、以下の5種類のシステムコールで構成されます。

- tk\_cre\_mpf(Create Fixed-size Memory Pool)
- tk\_del\_mpf>Delete Fixed-size Memory Pool)
- tk\_get\_mpf(Get Fixed-size Memory Block)
- tk\_rel\_mpf(Release Fixed-size Memory Block)
- tk\_ref\_mpf(Refer Fixed-size Memory Pool Status)

### 3.7.1.1 tk\_cre\_mpf(Create Fixed-size Memory Pool)

固定長メモリプールを生成します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
ID mpfid = tk_cre_mpf ( T_CMPF *pk_cmpf ) ;
```

```
typedef struct t_cmpf {
    VP  exinf;
    ATR mpfatr;
    INT mpfcnt;
    INT blfsz;
    VP  bufptr;
} T_CMPF;
```

#### ■ パラメータ

##### ● 入力

pk\_cmpf    固定長メモリプール生成情報  
(Packet of Create Memory Pool)

          パケットに設定するデータ

exinf      拡張情報 (Extended Information)

mpfatr     固定長メモリプール属性  
(Fixed-size Memory Pool Attribute)  
mpfatr:= (TA\_TFIFO    TA\_TPRI) | [TA\_USERBUF]

属性	値	意味
TA_TFIFO	0x00000000	待ちタスクを FIFO で管理
TA_TPRI	0x00000001	待ちタスクを優先度順で管理
TA_USERBUF	0x00000020	スタック領域としてユーザが指定した領域を使用する

mpfcnt     固定長メモリプール全体のブロック数  
(Fixed-size Memory Pool Block Count)

blfsz      メモリブロックサイズ (バイト数)  
(Fixed-size Memory Pool Block Size)

bufptr     ユーザバッファのアドレス (Buffer Pointer)

● 出力

mpfid            固定長メモリプール ID (Fixed-size Memory Pool ID)  
                 またはエラーコード (Error Code)

■ エラーコード

E\_NOMEM        -33      メモリ不足  
                 (メモリプール用の領域を確保できない)

E\_LIMIT        -34      固定長メモリプールの数がシステムの上限より大きい

E\_RSATR        -11      予約属性 (mpfatr に未定義の値が指定された)

E\_PAR           -17      パラメータエラー (mpfcnt, blfsz が 0 以下, bufsz が 4 の倍数で  
                 ない (TA\_USERBUF が指定されている場合))

E\_CTX           -25      コンテキストエラー  
                 (タスク独立部またはディスパッチ禁止状態で実行)

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

固定長メモリプールを生成して固定長メモリプール ID を割り当てます。具体的には, mpfcnt, blfsz の情報を元にメモリプールとして使用するメモリ領域を確保します。ここで生成されたメモリプールに tk\_get\_mpf を呼び出すことにより, blfsz のサイズ (バイト数) をもつメモリブロックを獲得できます。

exinf は, 対象メモリプールに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報を tk\_ref\_mpf で取り出せます。

なお, ユーザの情報を入れるためにもっと大きな領域が必要な場合や, 途中で内容を変更したい場合には, 自分でそのためのメモリを確保して, そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容については関知しません。

mpfatr に以下の属性を指定した場合には無視されます。また, 未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

属性	値	意味
TA_RNG0	0x00000000	保護レベル 0 で実行
TA_RNG1	0x00000100	保護レベル 1 で実行
TA_RNG2	0x00000200	保護レベル 2 で実行
TA_RNG3	0x00000300	保護レベル 3 で実行

TA\_TFIFO, TA\_TPRI では, タスクがメモリ獲得のためにメモリプールの待ち行列に並ぶときの並び方を指定できます。属性が TA\_TFIFO の場合にはタスクの待ち行列は FIFO となり, 属性が TA\_TPRI の場合にはタスクの待ち行列はタスクの優先度順となります。

TA\_USERBUF が指定された場合には bufptr が有効になり , bufptr を先頭とする mpfcnt\*blfsz バイトのメモリ領域をメモリプール領域として使用します。この場合 , メモリプール領域を OS では用意しません。TA\_USERBUF を指定しなかった場合は , bufptr は無視され , メモリプール領域を OS が確保します。固定長メモリプールの領域が OS で確保できなかった場合には E\_NOMEM のエラーで復帰します。mpfcnt, blfsz が 0 以下の場合には E\_PAR のエラーで復帰します。

固定長メモリプールがシステムの上限值 ( コンフィギュレータで設定した最大固定長メモリプール数 ) まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。また , pk\_cmpf, bufptr が不正な場合でも , エラーチェックを行わず , 動作は保証されません。

#### ■ 補足事項

固定長メモリプールの場合にブロックサイズを変えるには別のメモリプールを用意してください。すなわち , 何種類かのメモリブロックサイズが必要となる場合にはサイズごとに複数のメモリプールを設けてください。

### 3.7.1.2 tk\_del\_mpf(Delete Fixed-size Memory Pool)

固定長メモリプールを削除します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
ER ercd = tk_del_mpf ( ID mpfid ) ;
```

#### ■ パラメータ

● 入力

mpfid            固定長メモリプール ID  
                  (Fixed-size Memory Pool ID)

● 出力

ercd            エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mpfid が 0 以下または最大固定長メモリプール数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mpfid の固定長メモリプールが存在しない)
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクが削除した固定長メモリプールで待っていた場合に、待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

mpfid で指定された固定長メモリプールを削除します。具体的には、対象メモリプールを未生成状態にして ID 番号およびメモリプール本体の領域を解放します。

mpfid で指定された固定長メモリプール ID が 0 以下または最大固定長メモリプール数 (コンフィギュレータで設定した最大固定長メモリプール数) より大きい場合には E\_ID のエラーで復帰します。対象固定長メモリプールが存在しない場合には E\_NOEXS のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

このメモリプールからメモリを獲得しているタスクが存在しても、そのチェックや通知は行いません。すべてのメモリブロックが返却されていなくても、本システムコールは正常終了します。

対象メモリプールにおいてメモリ獲得を待っているタスクがあった場合にも、本システムコールは正常終了しますが、待ち状態にあったタスクには E\_DLT のエラーで復帰します。

### 3.7.1.3 tk\_get\_mpf(Get Fixed-size Memory Block)

固定長メモリブロックを獲得します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
ER ercd = tk_get_mpf ( ID mpfid, VP *p_blf, TMO tmout ) ;
```

#### ■ パラメータ

##### ● 入力

mpfid      固定長メモリプール ID  
(Fixed-size Memory Pool ID)

tmout      タイムアウト指定 (Timeout)  
0 から 0x7fffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

##### ● 出力

ercd      エラーコード (Error Code)

p\_blf      メモリブロックの先頭アドレス  
(Fixed-size Block Start Address)

#### ■ エラーコード

E\_OK      0      正常終了

E\_ID      -18      不正 ID 番号  
(mpfid が 0 以下または最大固定長メモリプール数より大きい)

E\_NOEXS      -42      オブジェクトが存在していない  
(mpfid の固定長メモリプールが存在しない)

E\_PAR      -17      パラメータエラー (tmout    (-2))

E\_DLT      -51      待ちオブジェクトが削除された  
( 待ちの間に対象メモリプールが待ちオブジェクトを削除 )

E\_RLWAI      -49      待ち状態強制解除 ( 待ちの間に tk\_rel\_wai を受け付ける )

E\_TMOUT      -50      ポーリング失敗またはタイムアウト

E\_CTX      -25      コンテキストエラー  
( タスク独立部またはディスパッチ禁止状態で実行 )



## ■ ディスパッチ要因

本システムコールを呼び出したタスクがメモリプール待ち状態になった場合に、次の優先順位のタスクにディスパッチします。

## ■ 解説

mpfid で指定された固定長メモリプールからメモリブロックを獲得します。獲得したメモリブロックの先頭アドレスを p\_blf に返します。獲得するメモリブロックのサイズは、固定長メモリプール生成時に blfsz で指定された値になります。獲得したメモリのゼロクリアは行わず、内容は不定となります。

mpfid で指定された固定長メモリプール ID が 0 以下または最大固定長メモリプール数 (コンフィギュレータで設定した最大固定長メモリプール数) より大きい場合には E\_ID のエラーで復帰します。対象固定長メモリプールが存在しない場合には E\_NOEXS のエラーで復帰します。

tmout により待ち時間の最大値 (タイムアウト値) を指定できます。タイムアウト指定が行われた場合に待ち解除の条件が満足されない (空きメモリができない) まま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、メモリが獲得できるまで、または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL はメモリが獲得できなかった場合も待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間 (時間の単位) はシステム時刻の基準時間 (= 1 ms) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

指定されたメモリプールからメモリブロックが獲得できない場合には tk\_get\_mpf 呼出しタスクがそのメモリプールのメモリ獲得待ち行列につながれ、メモリを獲得できるようになるまで待ちます。

メモリブロック獲得待ちを行う場合の待ち行列の順序は、メモリプールの属性によって、FIFO またはタスク優先度順のいずれかとなります。

固定長メモリプール待ち状態で対象固定長メモリプールが tk\_del\_mpf により削除された場合、待ち状態は解除されて本システムコールは E\_DLT のエラーで復帰します。固定長メモリプール待ち状態のタスクに tk\_rel\_wai システムコールが呼び出された場合には待ち状態は解除されて E\_RLWAI のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

p\_blf が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.7.1.4 tk\_rel\_mpf(Release Fixed-size Memory Block)

固定長メモリブロックを返却します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
ER ercd = tk_rel_mpf ( ID mpfid, VP blf ) ;
```

#### ■ パラメータ

##### ● 入力

mpfid      固定長メモリプール ID  
(Fixed-size Memory Pool ID)

blf        メモリブロックの先頭アドレス  
(Fixed-size Block Start Address)

##### ● 出力

ercd       エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mpfid が 0 以下または最大固定長メモリプール数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mpfid の固定長メモリプールが存在しない)
E_PAR	-17	パラメータエラー (blf がメモリプール領域の範囲外, (blf - メモリプール領域 先頭アドレス) がメモリブロックサイズの倍数でない)
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクのメモリプール待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。

## ■ 解説

blf で指定されたメモリブロックを mpfid で指定された固定長メモリプールへ返却します。

mpfid で指定された固定長メモリプール ID が 0 以下または最大固定長メモリプール数 (コンフィギュレータで設定した最大固定長メモリプール数) より大きい場合には E\_ID のエラーで復帰します。対象固定長メモリプールが存在しない場合には E\_NOEXS のエラーで復帰します。

本システムコールの実行により, mpfid のメモリプールでメモリを待っていた別のタスクがメモリを獲得し, そのタスクの待ち状態を解除する場合があります。

メモリブロックは, そのメモリブロックを獲得した固定長メモリプールに必ず返却してください。メモリブロックの返却を行うメモリプールが, メモリブロックの獲得を行ったメモリプールと異なっていることが検出された場合 (blf がメモリプール領域の範囲外 (blf - メモリプール領域先頭アドレス) がメモリブロックサイズの倍数でない) には E\_PAR のエラーで復帰します。

### 3.7.1.5 tk\_ref\_mpf(Refer Fixed-size Memory Pool Status)

固定長メモリプールの状態を参照します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

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

```
typedef struct t_rmpf {
    VP  exinf;
    ID  wtsk;
    INT frbcnt;
} T_RMPF;
```

#### ■ パラメータ

##### ● 入力

mpfid      固定長メモリプール ID  
(Fixed-size Memory Pool ID)

pk\_rmpf    メモリプールの状態を返すパケットアドレス  
(Packet of Refer Fixed-size Memory Pool)

##### ● 出力

ercd      エラーコード (Error Code)

          パケットに設定するデータ

exinf      拡張情報 (Extended Information)

wtsk      待ちタスクの有無 (Wait Task)

frbcnt    空き領域のブロック数 (Free Block Count)

#### ■ エラーコード

E\_OK      0      正常終了

E\_ID      -18    不正 ID 番号  
(mpfid が 0 以下または最大固定長メモリプール数より大きい)

E\_NOEXS   -42    オブジェクトが存在していない  
(mpfid の固定長メモリプールが存在しない)

E\_CTX     -25    コンテキストエラー  
(タスク独立部またはディスパッチ禁止状態で実行)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

mpfid で示された対象固定長メモリプールの各種状態を参照して復帰値として現在の空きブロック数 frbent, 待ちタスクの有無 (wtsk), 拡張情報 (exinf) を返します。

mpfid で指定された固定長メモリプール ID が 0 以下または最大固定長メモリプール数 (コンフィギュレータで設定した最大固定長メモリプール数) より大きい場合には E\_ID のエラーで復帰します。対象固定長メモリプールが存在しない場合には E\_NOEXS のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

wtsk は、この固定長メモリプールで待っているタスクの ID を指定します。この固定長メモリプールで複数のタスクが待っている場合には待ち行列の先頭タスクの ID を返します。待ちタスクがない場合は wtsk = 0 となります。どんな場合でも frbent = 0 と wtsk = 0 の少なくとも一方は成り立ちます。

pk\_rmpf が不正な場合でも、エラーチェックを行わず、動作は保証されません。

## ■ 補足事項

tk\_ref\_mpl の frsz ではメモリの空き領域の合計サイズがバイト数で返るのに対して、本システムコールの frbent では空きブロックの数が返ります。

## 3.7.2 可変長メモリプール機能のシステムコール

---

可変長メモリプール機能のシステムコールについて説明します。

---

### ■ 可変長メモリプール機能のシステムコール

可変長メモリプール機能は、以下の 5 種類のシステムコールで構成されます。

- tk\_cre\_mpl(Create Variable-size Memory Pool)
- tk\_del\_mpl>Delete Variable-size Memory Pool)
- tk\_get\_mpl(Get Variable-size Memory Block)
- tk\_rel\_mpl(Release Variable-size Memory Block)
- tk\_ref\_mpl(Refer Variable-size Memory Pool Status)

### 3.7.2.1 tk\_cre\_mpl(Create Variable-size Memory Pool)

可変長メモリプールを生成します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

■ C 言語インタフェース

```
ID mplid = tk_cre_mpl ( T_CMPL *pk_cmpl ) ;
```

```
typedef struct t_cmpl {  
    VP exinf;  
    ATR mplatr;  
    INT mplsz;  
    VP bufptr;  
} T_CMPL;
```

■ パラメータ

● 入力

- pk\_cmpl     可変長メモリプール生成情報  
              (Packet of Create Memory Pool)  
  
              パケットに設定するデータ
- exinf       拡張情報 (Extended Information)
- mplatr      メモリプール属性 (Memory Pool Attribute)  
              mplatr:= (TA\_TFIFO    TA\_TPRI) | [TA\_USERBUF]

属性	値	意味
TA_TFIFO	0x0000000	待ちタスクを FIFO で管理
TA_TPRI	0x0000001	待ちタスクを優先度順で管理
TA_USERBUF	0x0000020	スタック領域としてユーザが指定した領域を使用する

- mplsz       メモリプール全体のサイズ ( バイト数 )  
              (Memory Pool Size)
- bufptr      ユーザバッファのアドレス (Buffer Pointer)

● 出力

- mplid       可変長メモリプール ID (Memory Pool ID)  
              またはエラーコード (Error Code)

## ■ エラーコード

E_NOMEM	-33	メモリ不足 (メモリプール用の領域を確保できない)
E_LIMIT	-34	可変長メモリプールの数がシステムの上限より大きい
E_RSATR	-11	予約属性 (mplatr に未定義の値が指定された)
E_PAR	-17	パラメータエラー (mplsz が 0 以下または 4 の倍数でない (TA_USERBUF が指定されている場合))
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

可変長メモリプールを生成して可変長メモリプール ID を割り当てます。具体的には，mplsz の情報を元に，メモリプールとして使用するメモリ領域を確保します。

exinf は，対象メモリプールに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報を tk\_ref\_mpl で取り出せます。

なお，ユーザの情報を入れるためにもっと大きな領域が必要な場合や，途中で内容を変更したい場合には，自分でそのためのメモリを確保して，そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容については関知しません。

mplatr は可変長メモリプールの属性を指定します。mplatr に以下の属性を指定した場合は無視されます。mplatr に未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

属性	値	意味
TA_RNG0	0x00000000	保護レベル 0 で実行
TA_RNG1	0x00000100	保護レベル 1 で実行
TA_RNG2	0x00000200	保護レベル 2 で実行
TA_RNG3	0x00000300	保護レベル 3 で実行

TA\_TFIFO, TA\_TPRI では，タスクがメモリ獲得のためにメモリプールの待ち行列に並ぶときの並び方を指定できます。属性が TA\_TFIFO の場合にはタスクの待ち行列は FIFO となり，属性が TA\_TPRI の場合にはタスクの待ち行列はタスクの優先度順となります。

TA\_USERBUF が指定された場合には bufptr が有効になり，bufptr を先頭とする mplsz バイトのメモリ領域をメモリプール領域として使用します。この場合，メモリプール領域を OS では用意しません。TA\_USERBUF を指定しなかった場合は，bufptr は無視されてメモリプール領域は OS が確保します。可変長メモリプールの領域が OS で確保できなかった場合には E\_NOMEM のエラーで復帰します。mplsz が 0 以下の場合には E\_PAR のエラーで復帰します。



可変長メモリプールがシステムの上限值（コンフィギュレータで設定した最大可変長メモリプール数）まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。また、pk\_cmpl, bufptr が不正な場合でも、エラーチェックを行わず、動作は保証されません。

タスクがメモリ獲得待ちの行列を作った場合は、待ち行列先頭のタスクに優先してメモリを割り当てます。待ち行列の2番目以降により少ないメモリサイズを要求しているタスクがあった場合も、そのタスクが先にメモリを獲得できません。例えば、ある可変長メモリプールに要求メモリサイズ = 400 のタスク A と要求メモリサイズ = 100 のタスク B がこの順で待っており、別のタスクの tk\_rel\_mpl によりメモリサイズ = 200 の連続空きメモリ領域ができたとします。この場合、要求サイズの少ないタスク B は先にメモリを獲得できません。

## ■ 補足事項

以下のように待ち行列の順序が変化した場合、新たに待ち行列の先頭になったタスクにメモリ割当てが試みられます。その結果、メモリを割り当てられているとそのタスクの待ちを解除します。したがって、tk\_rel\_mpl によるメモリの解放がなくても、メモリの獲得を行い、待ちを解除する場合があります。

- メモリ獲得待ち行列の先頭タスクの待ちが強制解除された。
- メモリ獲得待ち行列の先頭タスクが強制終了した。
- 優先度順のメモリ獲得待ち行列で、先頭タスク以外のタスクの優先度が変更されて先頭タスクの優先度よりも高くなった。

### 3.7.2.2 tk\_del\_mpl(Delete Variable-size Memory Pool)

可変長メモリプールを削除します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

```
ER ercd = tk_del_mpl ( ID mplid ) ;
```

#### ■ パラメータ

##### ● 入力

**mplid**      可変長メモリプール ID  
(Memory Pool ID)

##### ● 出力

**ercd**      エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mplid が 0 以下または最大可変長メモリプール数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mplid の可変長メモリプールが存在しない)
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクが削除した可変長メモリプールで待っていた場合に、待ち状態が解除されたタスクにディスパッチします。

#### ■ 解説

mplid で指定された可変長メモリプールを削除します。具体的には、対象メモリプールを未生成状態にして ID 番号およびメモリプール本体の領域を解放します。

mplid で指定された可変長メモリプール ID が 0 以下または最大可変長メモリプール数 (コンフィギュレータで設定した最大可変長メモリプール数) より大きい場合には E\_ID のエラーで復帰します。対象可変長メモリプールが存在しない場合には E\_NOEXS のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

このメモリプールからメモリを獲得しているタスクが存在しても、そのチェックや通知は行いません。すべてのメモリブロックが返却されていなくても、本システムコールは正常終了します。

対象メモリプールにおいてメモリ獲得を待っているタスクがあった場合にも本システムコールは正常終了しますが、待ち状態にあったタスクには E\_DLT のエラーで復帰します。

3.7.2.3 tk\_get\_mpl(Get Variable-size Memory Block)

可変長メモリブロックを獲得します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

■ C 言語インタフェース

```
ER ercd = tk_get_mpl ( ID mplid, W blkksz, VP *p_blk, TMO tmout ) ;
```

■ パラメータ

● 入力

mplid	可変長メモリプール ID (Memory Pool ID)
blkksz	メモリブロックサイズ ( バイト数 ) (Block Size)
tmout	タイムアウト指定 (Timeout) 0 から 0x7fffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

● 出力

ercd	エラーコード (Error Code)
p_blk	メモリブロックの先頭アドレス (Block Start Address)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mplid が 0 以下または最大可変長メモリプール数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mplid の固定長メモリプールが存在しない)
E_PAR	-17	パラメータエラー (tmout (-2), blkksz が 0 以下またはメモリプールの領域サイズより大きい)
E_DLT	-51	待ちオブジェクトが削除された ( 待ちの間に対象メモリプールが待ちオブジェクトを削除 )
E_RLWAI	-49	待ち状態強制解除 ( 待ちの間に tk_rel_wai を受け付ける )
E_TMOUT	-50	ポーリング失敗またはタイムアウト
E_CTX	-25	コンテキストエラー ( タスク独立部またはディスパッチ禁止状態で実行 )

## ■ ディスパッチ要因

本システムコールを呼び出したタスクがメモリプール待ち状態になった場合に、次の優先順位のタスクにディスパッチします。

## ■ 解説

mplid で指定された可変長メモリプールから blksiz で指定されたサイズ ( バイト数 ) のメモリブロックを獲得します。

獲得したメモリブロックの先頭アドレスを p\_blk に返します。獲得したメモリのゼロクリアは行わず、内容は不定となります。メモリが獲得できない場合には本システムコールを呼び出したタスクは待ち状態に入ります。

mplid で指定された可変長メモリプール ID が 0 以下または最大可変長メモリプール数 ( コンフィギュレータで設定した最大可変長メモリプール数 ) より大きい場合には E\_ID のエラーで復帰します。対象可変長メモリプールが存在しない場合には E\_NOEXS のエラーで復帰します。blksiz が 0 以下の場合には E\_PAR のエラーで復帰します。

tmout により待ち時間の最大値 ( タイムアウト値 ) を指定できます。タイムアウト指定が行われた場合、待ち解除の条件が満足されない ( 空きメモリができない ) まま tmout の時間が経過すると E\_TMOUT のエラーで復帰します。

TMO\_FEVR は、タイムアウトまでの時間が無限大であることを示します。この場合は、メモリが獲得できるまで、または tk\_rel\_wai が呼び出されるまで待ち状態になります。TMO\_POL はメモリが獲得できなかった場合も待ち状態には移行せずに E\_TMOUT のエラーで復帰します。

tmout の基準時間 ( 時間の単位 ) はシステム時刻の基準時間 ( = 1 ms ) と同じです。また、tmout が -2 以下の場合には E\_PAR のエラーで復帰します。

メモリブロック獲得待ちを行う場合の待ち行列の順序はメモリプールの属性によって、FIFO またはタスク優先度順のいずれかとなります。

可変長メモリプール待ち状態で、対象メッセージバッファが tk\_del\_mpl により削除された場合、待ち状態は解除されて本システムコールは E\_DLT のエラーで復帰します。可変長メモリプール待ち状態のタスクに tk\_rel\_wai システムコールが呼び出された場合、待ち状態は解除されて E\_RLWAI のエラーで復帰します。タスク独立部から、またはディスパッチ禁止状態で本システムコールが呼び出された場合には E\_CTX のエラーで復帰します。

p\_blk が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.7.2.4 tk\_rel\_mpl(Release Variable-size Memory Block)

可変長メモリブロックを返却します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

■ C 言語インタフェース

```
ER ercd = tk_rel_mpl ( ID mplid, VP blk ) ;
```

■ パラメータ

● 入力

mplid	可変長メモリプール ID (Memory Pool ID)
blk	メモリブロックの先頭アドレス (Block Start Address)

● 出力

ercd	エラーコード (Error Code)
------	---------------------

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mplid が 0 以下または最大可変長メモリプール数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mplid の可変長メモリプールが存在しない)
E_PAR	-17	パラメータエラー (blk がメモリプール領域の範囲外)
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)

■ ディスパッチ要因

本システムコールを呼び出したタスクより優先度の高いタスクのメモリプール待ち状態を解除した場合に、待ち状態が解除されたタスクにディスパッチします。

■ 解説

blk で指定されたアドレスのメモリブロックを mplid で指定された可変長メモリプールへ返却します。

mplid で指定された可変長メモリプール ID が 0 以下または最大可変長メモリプール数 (コンフィギュレータで設定した最大可変長メモリプール数) より大きい場合には E\_ID のエラーで復帰します。対象可変長メモリプールが存在しない場合には E\_NOEXS のエラーで復帰します。

tk\_rel\_mpl の実行により、mplid のメモリプールでメモリを待っていた別のタスクがメモリを獲得し、そのタスクの待ち状態を解除する場合があります。

メモリブロックは、メモリブロックの獲得を行った可変長メモリプールに必ず返却してください。メモリブロックの返却を行うメモリプールがメモリブロックの獲得を行ったメモリプールと異なっていることが検出された場合には E\_PAR のエラーで復帰します。

#### ■ 補足事項

複数のタスクが待っている可変長メモリプールにメモリを返却する場合、複数のタスクの要求メモリサイズの総和以上のメモリを返却した場合、複数のタスクが同時に待ち解除となります。この場合の待ち解除後のタスクの優先順位は、同じ優先度のタスクの間では待ち行列に並んでいたときと同じ順序となります。

### 3.7.2.5 tk\_ref\_mpl(Refer Variable-size Memory Pool Status)

可変長メモリプールの状態を参照します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	×
------	--	--------	---	------------	---

#### ■ C 言語インタフェース

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

typedef struct t_rmpl {
    VP  exinf;
    ID  wtsk;
    INT frsz;
    INT maxsz;
} T_RMPL;
```

#### ■ パラメータ

● 入力

- mplid**            可変長メモリプール ID  
                    (Memory Pool ID)
- pk\_rmpl**          メモリプールの状態を返すパケットアドレス  
                    (Packet of Refer Memory Pool)

● 出力

- ercd**            エラーコード (Error Code)  
                    パケットに返すデータ
- exinf**          拡張情報 (Extended Information)
- wtsk**          待ちタスクの有無  
                    (Wait Task)
- frsz**          空き領域の合計サイズ ( バイト数 ) (Free Memory Size)
- maxsz**        最大の空き領域のサイズ ( バイト数 )  
                    (Max Memory Size)

### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (mplid が 0 以下または最大可変長メモリプール数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (mplid の可変長メモリプールが存在しない)
E_CTX	-25	コンテキストエラー (タスク独立部またはディスパッチ禁止状態で実行)

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

mplid で示された対象可変長メモリプールの各種状態を参照して復帰値として現在の空き領域の合計サイズ frsz, 獲得可能な最大の空き領域のサイズ maxsz, 待ちタスクの有無 (wtsk), 拡張情報 (exinf) を返します。

mplid で指定された可変長メモリプール ID が 0 以下または最大可変長メモリプール数 (コンフィギュレータで設定した最大可変長メモリプール数) より大きい場合には E\_ID のエラーで復帰します。対象可変長メモリプールが存在しない場合には E\_NOEXS のエラーで復帰します。

wtsk は、この可変長メモリプールで待っているタスクの ID を設定します。この可変長メモリプールで複数のタスクが待っている場合には待ち行列の先頭タスクの ID を返します。待ちタスクがない場合には wtsk = 0 となります。

pk\_rmpl が不正な場合でも、エラーチェックを行わず、動作は保証されません。



## 3.8 時間管理機能のシステムコール

---

時間管理機能のシステムコールについて説明します。

---

### ■ 時間管理機能のシステムコール

時間管理機能は、以下の 3 種類のシステムコールで構成されます。

- システム時刻管理機能のシステムコール
- 周期ハンドラ機能のシステムコール
- アラームハンドラ機能のシステムコール

### 3.8.1 システム時刻管理機能のシステムコール

---

システム時刻管理機能のシステムコールについて説明します。

---

#### ■ システム時刻管理機能のシステムコール

システム時刻管理機能は、以下の 4 種類のシステムコールで構成されます。

- tk\_set\_tim(Set Time)
- tk\_get\_tim(Get Time)
- tk\_get\_otm(Get Operating Time)
- isig\_tim(Signal Time)

3.8.1.1 tk\_set\_tim(Set Time)

システム時刻を設定します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_set_tim ( SYSTIM *pk_tim ) ;

typedef struct systim {
    W    hi;
    UW   lo;
} SYSTIM;
```

■ パラメータ

● 入力

- pk\_tim      現在時刻を示すパケットアドレス  
              (Packet of Current Time)
- パケットに設定するデータ
- hi            システム設定用の現在時刻 ( 上位 32 ビット )
- lo            システム設定用の現在時刻 ( 下位 32 ビット )

● 出力

- ercd          エラーコード (Error Code)

■ エラーコード

- E\_OK          0          正常終了
- E\_PAR        -17        パラメータエラー (pk\_tim.hi が負)

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

システム時刻の値を systim で指定された値に設定します。システム時刻は , 1985 年 1 月 1 日 0 時 (GMT) からの通算の ms とします。

hi が負の場合には E\_PAR のエラーで復帰します。

pk\_tim が不正な場合でも , エラーチェックを行わず , 動作は保証されません。

■ 補足事項

システムの動作中に本システムコールを使ってシステム時刻を変更した場合にも , RELTIM や TMO で指定された相対時間は変化しません。例えば , 60 秒後にタイムアウトするように指定された場合にタイムアウト待ちの間に本システムコールで時間を 60 秒進めても , そこでタイムアウトすることではなく , 60 秒後にタイムアウトします。したがって , 本システムコールによってタイムアウトするシステム時刻は変化します。

### 3.8.1.2 tk\_get\_tim(Get Time)

---

システムの現在時刻を参照します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_get_tim ( SYSTIM *pk_tim ) ;
```

```
typedef struct systim {
    W    hi;
    UW   lo;
} SYSTIM;
```

#### ■ パラメータ

##### ● 入力

pk\_tim      現在時刻を返すパケットアドレス  
(Packet of Current Time)

##### ● 出力

ercd          エラーコード (Error Code)  
                パケットに返すデータ  
hi             システムの現在時刻 ( 上位 32 ビット )  
lo             システムの現在時刻 ( 下位 32 ビット )

#### ■ エラーコード

E\_OK          0          正常終了

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

システム時刻の現在の値を読み出して復帰値 pk\_tim に返します。システム時刻は、1985 年 1 月 1 日 0 時 0 分 0 秒 (GMT) からの通算の ms とします。

pk\_tim が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.8.1.3 tk\_get\_otm(Get Operating Time)

システム稼働時間を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_get_otm ( SYSTIM *pk_tim ) ;
```

```
typedef struct systim {
    W    hi;
    UW   lo;
} SYSTIM;
```

#### ■ パラメータ

##### ● 入力

pk\_tim      稼働時間を返すパケットアドレス  
(Packet of Operating Time)

##### ● 出力

ercd          エラーコード (Error Code)  
                パケットに返すデータ  
hi              システムの現在時刻 (上位 32 ビット)  
lo              システムの現在時刻 (下位 32 ビット)

#### ■ エラーコード

E\_OK          0          正常終了

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

システム稼働時間を獲得します。システム稼働時間はシステム時刻 (時刻) と異なり、システム起動時からの単純増加する稼働時間を表し、単位は 1ms です。

システム稼働時間は tk\_set\_tim による時刻設定に影響しません。

pk\_tim が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.8.1.4 isig\_tim(Signal Time)

---

タイムティックを供給します。

---

タスク部	×	タスク独立部		ディスパッチ禁止状態	
------	---	--------	--	------------	--

#### ■ C 言語インタフェース

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

#### ■ パラメータ

- 入力

なし

- 出力

ercd                  エラーコード (Error Code)

#### ■ エラーコード

E\_OK                  0                  正常終了

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

システム時刻を更新します。本システムコールは  $\mu$ T-REALOS 独自の機能です。

本システムコールをユーザプログラムから、システム時刻の基準時間 ( = 1 ms ) ごとに呼び出すことによりシステム時刻を更新します。周期が 1ms のタイマ割込みを発生させ、その割込みハンドラから本システムコールを呼び出してください。

本システムコールをタスク部から呼び出すことはできません。タスク部から呼び出した場合にはその動作は保証されません。

## 3.8.2 周期ハンドラ機能のシステムコール

---

周期ハンドラ機能のシステムコールについて説明します。

---

### ■ 周期ハンドラ機能のシステムコール

周期ハンドラ機能は、以下の5種類のシステムコールで構成されます。

- tk\_cre\_cyc(Create Cyclic Handler)
- tk\_del\_cyc>Delete Cyclic Handler)
- tk\_sta\_cyc(Start Cyclic Handler)
- tk\_stp\_cyc(Stop Cyclic Handler)
- tk\_ref\_cyc(Refer Cyclic Handler Status)

### 3.8.2.1 tk\_cre\_cyc(Create Cyclic Handler)

周期ハンドラを生成します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ID cycid = tk_cre_cyc ( T_CCYC *pk_ccyc ) ;
```

```
typedef struct t_ccyc {
    VP  exinf;
    ATR cycatr;
    FP  cychdr;
    RELTIM cyctim;
    RELTIM cycphs;
} T_CCYC;
```

#### ■ パラメータ

##### ● 入力

**pk\_ccyc** 周期ハンドラ定義情報  
(Packet of Create Cyclic Handler)

パケットに設定するデータ

**exinf** 拡張情報 (Extended Information)

**cycatr** 周期ハンドラ属性 (Cyclic Handler Attribute)  
cycatr := (TA\_HLNG) | [TA\_STA] | [TA\_PHS]

属性	値	意味
TA_HLNG	0x00000001	対象周期ハンドラを C 言語で記述
TA_STA	0x00000002	周期ハンドラ起動
TA_PHS	0x00000004	周期ハンドラ起動位相を保存

**cyhdr** 周期ハンドラアドレス (Cyclic Handler Address)

**cyctim** 周期起動時間間隔 (Cycle Time)

**cycphs** 周期起動位相 (Cyclic Handler Phase)

##### ● 出力

**cycid** 周期ハンドラ ID (Cyclic Handler ID)  
またはエラーコード (Error Code)



## ■ エラーコード

E_LIMIT	-34	周期ハンドラの数システムの制限より大きい
E_RSATR	-11	予約属性 (cycatr に未定義の値が指定された)
E_PAR	-17	パラメータエラー (cychdr が NULL, cyctim が 0)

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

周期ハンドラを生成して周期ハンドラ ID を割り当てます。周期ハンドラは、指定された時間間隔で動くタスク独立部のハンドラです。

exinf は、対象となる周期ハンドラに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報は、周期ハンドラにパラメータとして渡すほか、tk\_ref\_cyc で取り出せます。

なお、ユーザの情報を入れるためにもっと大きな領域が必要な場合や、途中で内容を変更したい場合には、自分でそのためのメモリを確保し、そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容については関知しません。

cycatr は周期ハンドラの属性を指定します。以下の値は無視されます。また、未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

属性	値	意味
TA_ASM	0x00000000	対象周期ハンドラをアセンブラで記述

μT-Kernel仕様では、周期ハンドラをC言語で記述した場合にTA\_HLNG の指定を規定しています。ただし、μT-REALOS においては周期ハンドラの記述はC言語だけをサポートしているため、TA\_ASM が指定された場合 (TA\_HLNG の指定がない場合) でも周期ハンドラはC言語で記述されているものとして処理します。このため、TA\_HLNG の指定は必ずしも必要ではありませんが、μT-Kernel 仕様の互換性の観点から TA\_HLNG を必ず指定してください。

TA\_STA が指定された場合は、周期ハンドラの生成時から周期ハンドラは動作状態となり、前述の時間間隔で周期ハンドラを起動します。TA\_STA を指定しなかった場合には起動周期の計測は行いますが周期ハンドラは起動しません。

TA\_PHS が指定された場合は、tk\_sta\_cyc によって周期ハンドラが活性化されても起動周期はリセットされず、周期ハンドラの生成時から計測している周期を維持します。TA\_PHS が指定されていない場合は、tk\_sta\_cyc によって起動周期がリセットされ、tk\_sta\_cyc 呼出し時から cyctim 間隔で周期ハンドラを起動します。tk\_sta\_cyc によるリセットでは cycphs は適用しません。この場合、tk\_sta\_cyc から n 回目の周期ハンドラの起動は tk\_sta\_cyc 呼出し時から cyctim × n 以上の時間が経過した後となります。

cychdr は起動する周期ハンドラの実アドレスを指定します。cychdr が NULL の場合には E\_PAR のエラーで復帰します。また、cychdr が不正な場合でも、エラーチェックを行わず、動作は保証されません。周期ハンドラの記述形式については、「ユーザズガイド 4.6 周期ハンドラ」を参照してください。

### 第 3 章 システムコールインタフェース

cycphs は本システムコールによって周期ハンドラを生成してから最初の周期ハンドラの起動までの時間を表します。その後は cyctim 間隔で周期起動を繰り返します。この場合、周期ハンドラの  $n$  回目の起動は周期ハンドラを生成してから  $\text{cycphs} + \text{cyctim} \times (n - 1)$  以上の時間が経過した後になります。cycphs に 0 が指定された場合には周期ハンドラの生成直後に周期ハンドラを起動します。cyctim が 0 以下の場合には E\_PAR のエラーで復帰します。

周期ハンドラがシステムの上限值（コンフィギュレータで設定した最大周期ハンドラ数）まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。また、pk\_ccyc が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.8.2.2 tk\_del\_cyc(Delete Cyclic Handler)

周期ハンドラを削除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_del_cyc ( ID cycid ) ;
```

■ パラメータ

● 入力

cycid      周期ハンドラ ID  
            (Cyclic Handler ID)

● 出力

ercd      エラーコード (Error Code)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (cycid が 0 以下または最大周期ハンドラ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (cycid の周期ハンドラが存在しない)

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

周期ハンドラを削除します。具体的には、対象周期ハンドラを未生成状態にして ID 番号を解放します。

cycid で指定された周期ハンドラ ID が 0 以下または最大周期ハンドラ数 ( コンフィギュレータで設定した最大周期ハンドラ数 ) より大きい場合には E\_ID のエラーで復帰します。対象周期ハンドラが存在しない場合には E\_NOEXS のエラーで復帰します。

### 3.8.2.3 tk\_sta\_cyc(Start Cyclic Handler)

周期ハンドラの動作を開始します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_sta_cyc ( ID cycid ) ;
```

#### ■ パラメータ

##### ● 入力

cycid      周期ハンドラ ID  
(Cyclic Handler ID)

##### ● 出力

ercd      エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (cycid が 0 以下または最大周期ハンドラ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (cycid の周期ハンドラが存在しない)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

cycid で指定された周期ハンドラを動作状態にします。

cycid で指定された周期ハンドラ ID が 0 以下または最大周期ハンドラ数 ( コンフィギュレータで設定した最大周期ハンドラ数 ) より大きい場合には E\_ID のエラーで復帰します。対象周期ハンドラが存在しない場合には E\_NOEXS のエラーで復帰します。

TA\_PHS 属性を指定している場合は、周期ハンドラの起動周期はリセットされずに周期ハンドラを動作状態にします。既に動作状態の場合には動作状態を維持して本システムコールは正常終了します。

TA\_PHS 属性を指定していない場合は、起動周期をリセットして周期ハンドラを動作状態にします。既に動作状態の場合には起動周期のリセットをしたうえで動作状態を維持します。したがって、次に周期ハンドラを起動するのは cycetim 後となります。

### 3.8.2.4 tk\_stp\_cyc(Stop Cyclic Handler)

周期ハンドラの動作を停止します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_stp_cyc ( ID cycid ) ;
```

■ パラメータ

● 入力

cycid      周期ハンドラ ID  
            (Cyclic Handler ID)

● 出力

ercd      エラーコード (Error Code)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (cycid が 0 以下または最大周期ハンドラ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (cycid の周期ハンドラが存在しない)

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

cycid で指定された周期ハンドラを停止します。既に停止状態の場合にはその状態を維持したまま正常終了します。

cycid で指定された周期ハンドラ ID が 0 以下または最大周期ハンドラ数 ( コンフィギュレータで設定した最大周期ハンドラ数 ) より大きい場合には E\_ID のエラーで復帰します。対象周期ハンドラが存在しない場合には E\_NOEXS のエラーで復帰します。

■ 補足事項

tk\_cre\_cyc では回数の指定がないので、いったん定義された周期ハンドラは、tk\_stp\_cyc によって停止するか、周期起動ハンドラを削除するまで周期起動を繰り返します。

複数のタイムイベントハンドラが同時に動く場合、それらのハンドラはシリアルに起動します ( 1 つのハンドラの実行を終了してから別のハンドラの実行を始める )。また、タイムイベントハンドラはタスク独立部なので遅延ディスパッチの原則を適用します。

### 3.8.2.5 tk\_ref\_cyc(Refer Cyclic Handler Status)

周期ハンドラの状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_ref_cyc ( ID cycid, T_RCYC *pk_rcyc ) ;
```

```
typedef struct t_rcyc {
    VP      exinf;
    RELTIM  lfttim;
    UINT    cycstat;
} T_RCYC;
```

#### ■ パラメータ

##### ● 入力

cycid      周期ハンドラ ID (Cyclic Handler ID)

pk\_rcyc    周期ハンドラの状態を返すパケットの先頭アドレス  
(Packet of Refer Cyclic Handler)

##### ● 出力

ercd      エラーコード (Error Code)

          パケットに返すデータ

exinf      拡張情報 (Extended Information)

lfttim     次のハンドラ起動までの残り時間 (Left Time)

cycstat    周期ハンドラの状態 (Cyclic Handler Status)

cycstat:= (TCYC\_STP | TCYC\_STA)

状態	値	意味
TCYC_STP	0x00	周期ハンドラが動作していない
TCYC_STA	0x01	周期ハンドラが動作している

#### ■ エラーコード

E\_OK      0      正常終了

E\_ID      -18    不正 ID 番号  
(cycid が 0 以下または最大周期ハンドラ数より大きい)

E\_NOEXS   -42    オブジェクトが存在していない  
(cycid の周期ハンドラが存在しない)

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

cycid で示された周期ハンドラの状態を参照して周期ハンドラの状態 cycstat, 次のハンドラ起動までの残り時間 lfttim, 拡張情報 exinf を復帰値として返します。

cycid で指定された周期ハンドラ ID が 0 以下または最大周期ハンドラ数 ( コンフィギュレータで設定した最大周期ハンドラ数 ) より大きい場合には E\_ID のエラーで復帰します。対象周期ハンドラが存在しない場合には E\_NOEXS のエラーで復帰します。

pk\_rcyc が不正な場合でも, エラーチェックを行わず, 動作は保証されません。

### 3.8.3 アラームハンドラ機能のシステムコール

---

アラームハンドラ機能のシステムコールについて説明します。

---

#### ■ アラームハンドラ機能のシステムコール

アラームハンドラ機能は、以下の 5 種類のシステムコールで構成されます。

- tk\_cre\_alm(Create Alarm Handler)
- tk\_del\_alm>Delete Alarm Handler)
- tk\_sta\_alm(Start Alarm Handler)
- tk\_stp\_alm(Stop Alarm Handler)
- tk\_ref\_alm(Refer Alarm Handler Status)



3.8.3.1 tk\_cre\_alm(Create Alarm Handler)

アラームハンドラを生成します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ID almid = tk_cre_alm ( T_CALM *pk_calm ) ;

typedef struct t_calm {
    VP  exinf;
    ATR almatr;
    FP  almhdr;
} T_CALM;
```

■ パラメータ

● 入力

pk\_calm     アラームハンドラ定義情報  
              (Packet of Create Alarm Handler)

              パケットに設定するデータ

exinf        拡張情報 (Extended Information)

almatr       アラームハンドラ属性 (Alarm Handler Attribute)

              almatr := TA\_HLNG

属性	値	意味
TA_HLNG	0x00000001	対象アラームハンドラを C 言語で記述

almhdr       アラームハンドラアドレス (Alarm Handler Address)

● 出力

almid        アラームハンドラ ID (Alarm Handler ID)

              またはエラーコード (Error Code)

■ エラーコード

E\_LIMIT     -34     アラームハンドラの数システムの制限より大きい

E\_RSATR     -11     予約属性 (almatr に未定義の値が指定された)

E\_PAR       -17     パラメータエラー (almhdr が NULL)

■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

アラームハンドラを生成してアラームハンドラ ID を割り当てます。アラームハンドラ (指定時刻起動ハンドラ) は、指定された時刻に起動するタスク独立部のハンドラです。

exinf は、対象となるアラームハンドラに関する情報を入れておくためにユーザが自由に使用できます。ここで指定した情報は、アラームハンドラにパラメータとして渡すほか、tk\_ref\_alm で取り出せます。

なお、ユーザの情報を入れるためにもっと大きな領域が必要な場合や、途中で内容を変更したい場合には、自分でそのためのメモリを確保して、そのメモリパケットのアドレスを exinf に入れます。OS では exinf の内容については関知しません。

almatr はアラームハンドラの属性を指定します。以下の値は無視されます。また、未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

属性	値	意味
TA_ASM	0x00000000	対象アラームハンドラをアセンブラで記述

μT-Kernel仕様では、アラームハンドラをC言語で記述した場合にTA\_HLNGの指定を規定しています。ただし、μT-REALOSにおいてはアラームハンドラの記述はC言語だけをサポートしているため、TA\_HLNGの有無とは関係なく、アラームハンドラはC言語で記述されているものとして処理します。このため、TA\_HLNGの指定は必ずしも必要はありませんが、μT-Kernel仕様の互換性の観点からTA\_HLNGを必ず指定してください。

almhdr は起動するアラームハンドラの手元アドレスを指定します。almhdr が NULL の場合には E\_PAR のエラーで復帰します。また、almhdr が不正な場合でも、エラーチェックを行わず、動作は保証されません。アラームハンドラの記述形式については、「ユーザーズガイド 4.7 アラームハンドラ」を参照してください。

アラームハンドラがシステムの上限值 (コンフィギュレータで設定した最大アラームハンドラ数) まで作成されている状態で本システムコールを呼び出した場合には E\_LIMIT のエラーで復帰します。また、pk\_calm が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.8.3.2 tk\_del\_alm(Delete Alarm Handler)

アラームハンドラを削除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_del_alm ( ID almid ) ;
```

■ パラメータ

● 入力

almid            アラームハンドラ ID  
                  (Alarm Handler ID)

● 出力

ercd            エラーコード (Error Code)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (almid が 0 以下または最大アラームハンドラ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (almid のアラームハンドラが存在しない)

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

アラームハンドラを削除します。具体的には、対象アラームハンドラを未生成状態にして ID 番号を解放します。

almid で指定されたアラームハンドラ ID が 0 以下または最大アラームハンドラ数 (コンフィギュレータで設定した最大アラームハンドラ数) より大きい場合には E\_ID のエラーで復帰します。対象アラームハンドラが存在しない場合には E\_NOEXS のエラーで復帰します。

### 3.8.3.3 tk\_sta\_alm(Start Alarm Handler)

アラームハンドラの動作を開始します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_sta_alm ( ID almid, RELTIM almtim ) ;
```

#### ■ パラメータ

##### ● 入力

almid          アラームハンドラ ID  
                  (Alarm Handler ID)

almtim         アラームハンドラ起動時刻 (Alarm Handler Time)

##### ● 出力

ercd           エラーコード (Error Code)

#### ■ エラーコード

E\_OK           0          正常終了

E\_ID           -18        不正 ID 番号  
                  (almid が 0 以下または最大アラームハンドラ数より大きい)

E\_NOEXS        -42        オブジェクトが存在していない  
                  (almid のアラームハンドラが存在しない)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

almid で指定されたアラームハンドラの起動時刻を設定して動作状態にします。

almid が 0 以下または最大アラームハンドラ数 ( コンフィギュレータで設定した最大アラームハンドラ数 ) より大きい場合には E\_ID のエラーで復帰します。対象アラームハンドラが存在しない場合には E\_NOEXS のエラーで復帰します。

almtim は相対時刻で、本システムコールが呼び出された時刻から almtim で指定された時間が経過した後にアラームハンドラを起動します。既にアラームハンドラの起動時刻が設定されて動作状態であった場合には、その設定を解除した後に新たに起動時刻を設定して動作状態とします。almtim=0 の場合には起動時刻設定直後にアラームハンドラを起動します。起動されたアラームハンドラの処理終了後に停止状態となります。

### 3.8.3.4 tk\_stp\_alm(Stop Alarm Handler)

アラームハンドラの動作を停止します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_stp_alm ( ID almid ) ;
```

■ パラメータ

● 入力

almid          アラームハンドラ ID  
                  (Alarm Handler ID)

● 出力

ercd            エラーコード (Error Code)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (almid が 0 以下または最大アラームハンドラ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (almid のアラームハンドラが存在しない)

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

アラームハンドラを停止状態にします。既に停止状態の場合にはその状態を継続して正常終了します。

almid が 0 以下または最大アラームハンドラ数 ( コンフィギュレータで設定した最大アラームハンドラ数 ) より大きい場合には E\_ID のエラーで復帰します。対象アラームハンドラが存在しない場合には E\_NOEXS のエラーで復帰します。

### 3.8.3.5 tk\_ref\_alm(Refer Alarm Handler Status)

アラームハンドラの状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_ref_alm ( ID almid, T_RALM *pk_ralm ) ;
```

```
typedef struct t_ralm {
    VP      exinf;
    RELTIM  lfttim;
    UINT    almstat;
} T_RALM;
```

#### ■ パラメータ

##### ● 入力

almid        アラームハンドラ ID (Alarm Handler ID)

pk\_ralm     アラームハンドラの状態を返すパケットの先頭アドレス  
(Packet of Refer Alarm Handler)

##### ● 出力

ercd        エラーコード (Error Code)

            パケットに返すデータ

exinf       拡張情報 (Extended Information)

lfttim      次のハンドラ起動までの残り時間 (Left Time)

almstat     アラームハンドラの状態 (Alarm Handler Status)

almstat := (TALM\_STP | TALM\_STA)

状態	値	意味
ALM_STP	0x00	アラームハンドラが動作していない
ALM_STA	0x01	アラームハンドラが動作している

## ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (almid が 0 以下または最大アラームハンドラ数より大きい)
E_NOEXS	-42	オブジェクトが存在していない (almid のアラームハンドラが存在しない)

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

almid で指定されたアラームハンドラの状態を参照してハンドラ起動までの残り時間 lfttim、拡張情報 exinf を復帰値として返します。

almid が 0 以下または最大アラームハンドラ数 (コンフィギュレータで設定した最大アラームハンドラ数) より大きい場合には E\_ID のエラーで復帰します。対象アラームハンドラが存在しない場合には E\_NOEXS のエラーで復帰します。

アラームハンドラが動作している (TALM\_STA) 場合には lfttim には次にアラームハンドラを起動するまでの相対時間を返します。tk\_sta\_alm で指定された almtim - lfttim - 0 の範囲の値となります。lfttim はタイマ割込みごとに減算するため、次のタイマ割込みでアラームハンドラが起動する場合に lfttim は 0 となります。

アラームハンドラが動作していない (TALM\_STP) 場合には lfttim は不定です。

pk\_ralm が不正な場合でも、エラーチェックを行わず、動作は保証されません。

## 3.9 割込み管理機能のシステムコール

---

割込み管理機能のシステムコールについて説明します。

---

### ■ 割込み管理機能のシステムコール

割込み管理機能のシステムコールは、以下の 2 種類のシステムコールで構成されます。

- `tk_def_int`(Define Interrupt Handler)
- `tk_ret_int`(Return from Interrupt Handler)

また、割込み管理機能には、以下の 3 種類のマクロがあります。

- `DI`
- `EI`
- `isDI`



### 3.9.1 tk\_def\_int(Define Interrupt Handler)

割込みハンドラを定義または解除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_def_int ( UINT dintno, T_DINT *pk_dint ) ;

typedef struct t_dint {
    ATR intatr;
    FP  inthdr;
} T_DINT;
```

■ パラメータ

● 入力

- dintno      割込みベクタ番号
- pk\_dint     割込みハンドラ定義情報の先頭アドレス  
             (Packet of Define Interrupt Handler)
- パケットに設定するデータ
- intatr      割込みハンドラ属性 (Interrupt Handler Attribute)  
             intatr := (TA\_ASM    TA\_HLNG)

属性	値	意味
TA_ASM	0x00000000	対象ハンドラをアセンブラで記述
TA_HLNG	0x00000001	対象ハンドラを C 言語で記述

- inthdr      割込みハンドラアドレス  
             (Interrupt Handler Address)

● 出力

- ercd        エラーコード (Error Code)

■ エラーコード

- E\_OK        0        正常終了
- E\_RSATR     -11     予約属性 (intatr が不正あるいは使用できない)
- E\_PAR       -17     パラメータエラー (dintno が 256 以上)

■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

割り込みハンドラを定義して割り込みハンドラを使用可能にします。または、割り込みハンドラの定義を解除します。割り込みは、デバイスからの外部割り込み、CPU 例外による割り込みおよびソフトウェア割り込みのすべてを含みます。

dintno は、割り込みハンドラを定義する割り込みベクタ番号を指定します。0 から 255 までの値を指定可能です。dintno に 256 以上の値が指定された場合には E\_PAR のエラーで復帰します。既に定義済みの割り込み番号に割り込みハンドラを再定義することもできます。再定義の場合も、あらかじめその番号のハンドラの定義解除を行っておく必要はありません。

pk\_dint は、割り込みハンドラの定義情報を格納する T\_DINT 構造体のアドレスです。pk\_dint の値と pk\_def\_int の動作の関係を以下に示します。

pk_dint の値	tk_def_int の動作
正常アドレス (NULL 以外)	割り込みハンドラの定義
NULL	割り込みハンドラの定義解除
不正アドレス	動作は保証しない

intatr は、割り込みハンドラの記述言語を指定します。以下に TA\_HLNG 属性と TA\_ASM 属性でのハンドラの記述言語およびハンドラの最大登録数について示します。intatr に未定義の属性が指定された場合には E\_RSATR のエラーで復帰します。

属性	記述言語	最大登録数
TA_ASM	アセンブラ	256
TA_HLNG	C 言語	8

inthdr は、割り込みハンドラの手元アドレスを指定します。inthdr のアドレスが不当な場合は、対応する割り込みが発生したときに不当アドレスにジャンプするため、その後のシステムの動作は保証されません。

なお、割り込みハンドラとタスクの実行優先順位については、「ユーザズガイド 2.6 タスク/ハンドラの実行優先順位」を参照してください。また、割り込みハンドラの記述形式については、「ユーザズガイド 4.8 割り込みハンドラ」を参照してください。

(注意事項) SOFTUNE 言語ツールを使用する  $\mu$ T-REALOS では、割り込みハンドラを C 言語で記述していても "\_\_interrupt" を指定する場合には TA\_ASM 属性として定義してください。TA\_HLNG 属性で登録すると正しく動作しません。

### 3.9.2 tk\_ret\_int(Return from Interrupt Handler)

割込みハンドラから復帰します。

タスク部	×	タスク独立部		ディスパッチ禁止状態	
------	---	--------	--	------------	--

■ C 言語インタフェース

```
void tk_ret_int ( void ) ;
```

■ パラメータ

- 入力  
なし
- 出力  
なし

■ エラーコード

なし

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

本システムコールを呼び出しても何も処理されずに呼出し元に復帰します。

μT-Kernel 仕様では、「アセンブラ言語で記述された割込みハンドラの終了時に本システムコールを呼び出すことにより、遅延ディスパッチを発生させる」と規定されています。しかし、μT-REALOS では、ディスパッチ実装上の理由により、本システムコール呼出しの有無とは無関係に遅延ディスパッチが発生します。このため、アセンブラ言語で記述された割込みハンドラにおいても本システムコールを呼び出す必要はありません。また、ほかの μT-Kernel 仕様 OS との互換性を保つため、本システムコールを残しています。このため、C 言語で記述された割込みハンドラにおいても本システムコールを呼び出さないようにしてください。

### 3.9.3 DI

---

すべての外部割込みを禁止します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ インタフェース

```
void DI ( UINT intsts )
```

#### ■ パラメータ

##### ● 入力

なし

##### ● 出力

```
intsts    CPU 割込みの状態 (PS レジスタの値)
```

#### ■ エラーコード

なし

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

PS レジスタのCCR.I フラグを 0( 割込み禁止 ) に設定してすべての外部割込みを禁止します。割込みを禁止する前の PS レジスタの値を `intsts` に保存します。

使用例

```
void foo()
{
    UINT intsts;
    DI(intsts);
    if ( isDI(intsts) ) {
        /* この関数が呼び出された時点で既に割込み禁止であった */
    } else {
        /* この関数が呼び出された時点では割込み許可であった */
    }
    EI(intsts);
}
```

3.9.4 EI

すべての外部割込みを許可します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ インタフェース

void EI ( UINT intsts )

■ パラメータ

● 入力

intsts CPU 割込みの状態 (PS レジスタの値)

● 出力

なし

■ エラーコード

なし

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

すべての外部割込みを許可します。正確には , intsts で指定された PS レジスタの CCR.I フラグの値が 1( 割込み許可 ) の場合には PS レジスタの CCR.I フラグを "1" に設定して割込み許可の状態に戻します。

intsts で指定された PS レジスタの CCR.I フラグの値が "0" ( 割込み禁止 ) の場合には EI() を実行しても割込みを許可しません。ただし , intsts として "0" が指定された場合には必ず割込みを許可します。

intsts は , DI() で保存した値または "0" のいずれかです。それ以外の値を指定した場合には動作は保証されません。

使用例

```
void foo()
{
    UINT intsts;
    DI(intsts);
    if ( isDI(intsts) ) {
        /* この関数が呼び出された時点で既に割込み禁止であった */
    } else {
        /* この関数が呼び出された時点では割込み許可であった */
    }
    EI(intsts);
}
```

### 3.9.5 isDI

外部割込み禁止状態を調べます。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ インタフェース

```
BOOL stat = isDI ( UINT intsts )
```

#### ■ パラメータ

##### ● 入力

intsts      CPU 割込みの状態 (PS レジスタの値)

##### ● 出力

stat      TRUE("0" 以外の値): 割込み禁止状態  
FALSE: 割込み許可状態

#### ■ エラーコード

なし

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

intsts に保存されている PS レジスタの値を元に割込み許可 / 禁止状態を調べます。intsts に保存された PS レジスタの値で、CCR.I フラグが "0" の場合には割込み禁止状態と判断し、"1" の場合には割込み許可と判断します。ただし、本システムコールで調べることができるのは intsts を保存した DI() 発行前の割込み禁止状態だけです。現在の割込み禁止状態を確認する場合には tk\_ref\_sys を使用してください。

intsts は DI() で保存した値です。それ以外の値を指定した場合には動作は保証されません。

使用例

```
void foo()
{
    UINT intsts;
    DI(intsts);
    if ( isDI(intsts) ) {
        /* この関数が呼び出された時点で既に割込み禁止であった */
    } else {
        /* この関数が呼び出された時点では割込み許可であった */
    }
    EI(intsts);
}
```

## 3.10 システム状態管理機能のシステムコール

---

システム状態管理機能のシステムコールについて説明します。

---

### ■ システム状態管理機能のシステムコール

システム状態管理機能は、以下の6種類のシステムコールで構成されます。

- tk\_rot\_rdq(Rotate Ready Queue)
- tk\_get\_tid(Get Task Identifier)
- tk\_dis\_dsp(Disable Dispatch)
- tk\_ena\_dsp(Enable Dispatch)
- tk\_ref\_sys(Refer System Status)
- tk\_ref\_ver(Refer Version Information)

### 3.10.1 tk\_rot\_rdq(Rotate Ready Queue)

タスクの優先順位を回転します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_rot_rdq ( PRI tskpri ) ;
```

#### ■ パラメータ

##### ● 入力

tskpri      優先度 (Task Priority)

名前	値	意味
TPRI_RUN	0	現在実行中のタスクの優先度

##### ● 出力

ercd      エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_PAR	-17	パラメータエラー (tskpri が負またはシステムの最大優先度より大きい)

#### ■ ディスパッチ要因

本システムコールを呼び出したタスクと同じ優先度が指定された場合で、かつその優先度のタスクがほかにある場合に、次の優先順位のタスクにディスパッチします。

#### ■ 解説

tskpri で指定された優先度のタスクの優先順位を回転します。すなわち、対象優先度を持った実行できる状態のタスクの中で最も高い優先順位を持つタスクを同じ優先度を持つタスクの中で最低の優先順位とします。

tskpri が負またはシステムの最大優先度 (コンフィギュレータで設定した最大優先度) よりも大きい場合には E\_PAR のエラーで復帰します。

tskpri に TPRI\_RUN(=0) を指定すると、実行状態にあるタスクの優先度が tskpri に指定された場合と同じ動作をします。周期ハンドラなどのタスク独立部から tskpri に TPRI\_RUN を指定して本システムコールを呼び出すこともできます。



■ 補足事項

対象優先度を持った実行できる状態のタスクがない場合や、1つしかない場合には何もせずに正常終了します。

ディスパッチ禁止状態では、同じ優先度を持つタスクの中で最高の優先順位を持ったタスクが実行されているとは限らないため、この方法で自タスクの実行順位が同じ優先度を持つタスクの中で最低になるとは限りません。

tk\_rot\_rdq の実行例を図 3.10-1 と図 3.10-2 に示します。図 3.10-1 の状態で tskpri = 2 をパラメータとして本システムコールが呼び出されると、新しい優先順位は図 3.10-2 になり、次に実行するのはタスク C になります。

図 3.10-1 tk\_rot\_rdq 実行前の優先順位

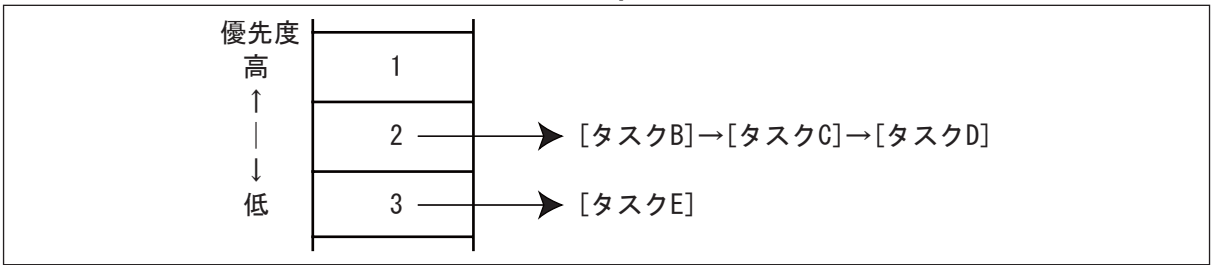
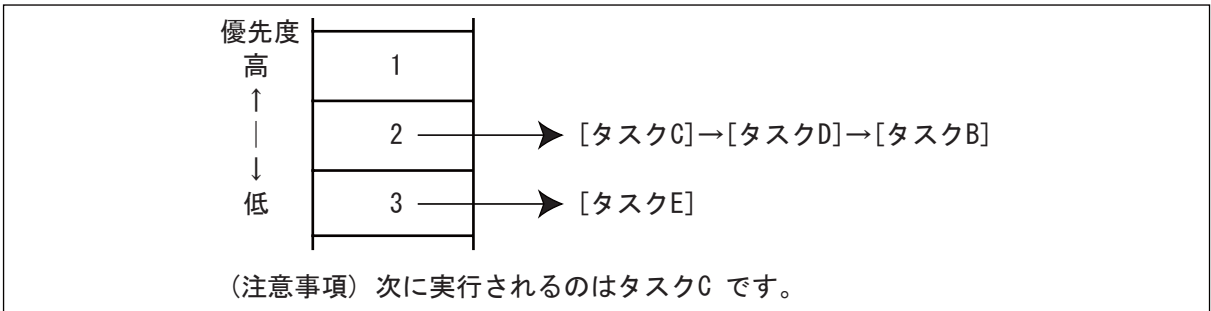


図 3.10-2 tk\_rot\_rdq(tskpri=2) 実行後の優先順位



## 3.10.2 tk\_get\_tid(Get Task Identifier)

---

実行状態タスクのタスク ID を参照します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

### ■ C 言語インタフェース

```
ID tskid = tk_get_tid ( void ) ;
```

### ■ パラメータ

#### ● 入力

なし

#### ● 出力

tskid          実行状態のタスク ID (Task ID)

### ■ エラーコード

なし

### ■ ディスパッチ要因

本システムコールではディスパッチしません。

### ■ 解説

現在実行状態にあるタスクの ID 番号を tskid に設定して復帰します。

タスク部から呼び出した場合には自タスクの ID が返ります。現在実行状態のタスクがない場合には "0" が返ります。

### ■ 補足事項

本システムコールで返すタスク ID は tk\_ref\_sys で返す runtskid と同一です。

### 3.10.3 tk\_dis\_dsp(Disable Dispatch)

ディスパッチを禁止します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

■ C 言語インタフェース

```
ER ercd = tk_dis_dsp ( void ) ;
```

■ パラメータ

● 入力

なし

● 出力

ercd            エラーコード (Error Code)

■ エラーコード

E_OK	0	正常終了
E_CTX	-25	コンテキストエラー (タスク独立部からの呼出し)

■ ディスパッチ要因

本システムコールではディスパッチしません。

■ 解説

タスクのディスパッチを禁止します。これ以後, tk\_ena\_dsp を実行するまでの間はディスパッチ禁止状態となり, 自タスクが実行状態から実行可能状態に移りません。また, 待ち状態に移ることもできなくなります。ただし, 外部割込みは禁止しませんので, ディスパッチ禁止状態の場合でも割込みハンドラは起動します。

ディスパッチ禁止状態においては, 実行中のタスクが割込みハンドラによってプリエンプト (CPU の実行権の横取り) される可能性があります, ほかのタスクによってプリエンプトされる可能性はありません。

ディスパッチ禁止状態の間は, 具体的には以下の動作をします。

- 割込みハンドラあるいは tk\_dis\_dsp を実行したタスクから呼び出されたシステムコールによって, tk\_dis\_dsp を実行したタスクより高い優先度のタスクが実行可能状態となっても, そのタスクにはディスパッチしません。優先度の高いタスクへのディスパッチはディスパッチ禁止状態が終了するまで遅延します。
- tk\_dis\_dsp を実行したタスクが, 自タスクを待ち状態に移す可能性があるシステムコール (tk\_slp\_tsk, tk\_wai\_sem など) を呼び出した場合には E\_CTX のエラーで復帰します。
- tk\_ref\_sys によってシステム状態を参照した場合には sysstat として TSS\_DDSP が返ります。

既にディスパッチ禁止状態にあるタスクが `tk_dis_dsp` を呼び出した場合にはディスパッチ禁止状態がそのまま継続するだけで正常終了します。ただし、`tk_dis_dsp` を何回か呼び出しても、その後に `tk_ena_dsp` を 1 回呼び出すだけでディスパッチ禁止状態を解除しますので注意してください。

#### ■ 補足事項

ディスパッチ禁止状態では実行状態のタスクを休止状態や未登録状態に移行できません。割込みおよびディスパッチ禁止状態で実行状態のタスクが `tk_ext_tsk` または `tk_exd_tsk` を呼び出した場合には `E_CTX` のエラーを検出します。ただし、`tk_ext_tsk` や `tk_exd_tsk` は元のコンテキストに復帰しないシステムコールなので、これらのシステムコールの復帰値としてエラーを通知できません。

### 3.10.4 tk\_ena\_dsp(Enable Dispatch)

ディスパッチを許可します。

タスク部		タスク独立部	×	ディスパッチ禁止状態	
------	--	--------	---	------------	--

■ C 言語インタフェース

```
ER ercd = tk_ena_dsp ( void ) ;
```

■ パラメータ

- 入力

なし

- 出力

ercd            エラーコード (Error Code)

■ エラーコード

E_OK	0	正常終了
E_CTX	-25	コンテキストエラー (タスク独立部からの呼出し)

■ ディスパッチ要因

遅延されているディスパッチがある場合にディスパッチします。

■ 解説

タスクのディスパッチを許可します。すなわち , tk\_dis\_dsp によって設定されていたディスパッチ禁止状態を解除します。

ディスパッチ禁止状態ではないタスクが本システムコールを呼び出した場合にはディスパッチを許可した状態がそのまま継続するだけで正常終了します。タスク独立部から呼び出した場合には E\_CTX のエラーで復帰します。

### 3.10.5 tk\_ref\_sys(Refer System Status)

システム状態を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_ref_sys ( T_RSYS *pk_rsys ) ;
```

```
typedef struct t_rsys {
    INT sysstat;
    ID runtskid;
    ID schedtskid;
} T_RSYS;
```

#### ■ パラメータ

##### ● 入力

pk\_rsys システム状態を返すパケットアドレス  
(Packet of Refer System)

##### ● 出力

ercd エラーコード (Error Code)

パケットに返すデータ

sysstat システム状態 (System State)

```
sysstat:= ( TSS_TSK | [TSS_DDSP] | [TSS_DINT] )
          ( TSS_QTSK | [TSS_DDSP] | [TSS_DINT] )
          ( TSS_INDP )
```

状態	値	意味
TSS_TSK	0	タスク部実行中
TSS_DDSP	1	ディスパッチ禁止中
TSS_DINT	2	割込み禁止中
TSS_INDP	4	タスク独立部実行中
TSS_QTSK	8	準タスク部実行中

runtskid 現在実行状態にあるタスクの ID (Running Task ID)

schedtskid 実行状態にすべきタスクの ID (Scheduled Task ID)

#### ■ エラーコード

E\_OK 0 正常終了

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

実行状態を参照してディスパッチ禁止中、タスク独立部実行中などといった情報を復帰値として返します。

runtskid には現在実行中のタスクの ID, schedtskid には実行状態にすべきタスクの ID を返します。通常は runtskid = schedtskid となりますが、ディスパッチ禁止中やタスク独立部実行中に、より優先度の高いタスクが存在する場合は runtskid < schedtskid となります。

なお、該当タスクがない場合には "0" を返します。

pk\_rsys が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.10.6 tk\_ref\_ver(Refer Version Information)

---

カーネルのバージョン情報を参照します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_ref_ver ( T_RVER *pk_rver ) ;
```

```
typedef struct t_rver {
    UH  maker;
    UH  prid;
    UH  spver;
    UH  prver;
    UH  prno[4];
} T_RVER;
```

#### ■ パラメータ

##### ● 入力

pk\_rver      バージョン情報を返すパケットの先頭アドレス  
(Packet of Version Information)

##### ● 出力

ercd          エラーコード (Error Code)  
                パケットに返すデータ

maker         カーネルのメーカコード (Maker)  
                =0x0009(メーカコード = 富士通)

prid          カーネルの識別番号 (Product ID)  
                =0x6911(μT-REALOS/FR)

spver         仕様書バージョン番号 (Specification Version)  
                =0x6100(μT-Kernel ver.1.00)

prver         カーネルのバージョン番号 (Product Version)  
                =0x0100(V01L00)

prno[4]       カーネル製品の管理情報 (Product Number)  
                prno[0] = 0x0000(R00)  
                prno[1] ~ [3] = 0x0000(未使用)

#### ■ エラーコード

E\_OK          0          正常終了

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。



## ■ 解説

使用しているカーネルのバージョン情報を参照して `pk_rver` で指定されるパケットに返します。

`pk_rver` が不正な場合でも、エラーチェックを行わず、動作は保証されません。

## 3.11 サブシステム機能のシステムコール

---

サブシステム機能のシステムコールについて説明します。

---

### ■ サブシステム機能のシステムコール

サブシステム機能は、以下の 2 種類のシステムコールで構成されます。

- tk\_def\_ssy(Define Subsystem)
- tk\_ref\_ssy(Refer Subsystem Status)

### 3.11.1 tk\_def\_ssy(Define Subsystem)

サブシステムを定義または定義を削除します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_def_ssy ( ID ssid, T_DSSY *pk_dssy ) ;

typedef struct t_dssy {
    ATR ssyatr;
    PRI ssypri;
    FP  svchdr;
    FP  breakfn;
    FP  startupfn;
    FP  cleanupfn;
    FP  eventfn;
    INT resblksz;
} T_DSSY;
```

#### ■ パラメータ

● 入力

ssid	サブシステム ID (Subsystem ID)
pk_dssy	サブシステム定義情報 (Packet of Define Subsystem) のパケット先頭 アドレス パケットに設定するデータ
ssyatr	サブシステム属性 (Subsystem Attribute)
ssypri	サブシステム優先度 (Subsystem Priority)
svchdr	拡張 SVC ハンドラアドレス (SVC Handler)
breakfn	ブレーク関数アドレス (Break Function)
startupfn	スタートアップ関数アドレス (Start-up Function)
cleanupfn	クリーンアップ関数アドレス (Clean-up Function)
eventfn	イベント処理関数アドレス (Event Function)
resblksz	リソース管理ブロックサイズ (バイト数) (Resource Block Size)

● 出力

ercd	エラーコード (Error Code)
------	---------------------

## ■ エラーコード

E_OK	0	正常終了
E_ID	-18	不正 ID 番号 (ssid が 10 より小さい, または最大サブシステム数より大きい)
E_OBJ	-41	ssid は既に定義済みである (pk_dssy = NULL の場合)
E_NOEXS	-42	ssid は定義されていない (pk_dssy = NULL の場合)
E_PAR	-17	ssypri が不正

## ■ ディスパッチ要因

本システムコールではディスパッチしません。

## ■ 解説

ssid のサブシステムを定義または定義を削除します。

ほかのサブシステムと重複しないように 1 つのサブシステムに 1 つのサブシステム ID を割り当ててください。OS に自動割当ての機能はありません。

サブシステム ID は 1 ~ 9 が  $\mu$ T-Kernel 用に予約されています。10 ~ 最大サブシステム数 (コンフィギュレータで設定した最大サブシステム数) が使用できる番号となります。

$\mu$ T-Kernel では, サブシステムの機能は, 拡張 SVC ハンドラの呼出し機能だけをサポートします。このため, t\_dssy のパケットのデータのうち, svchdr 以外は T-Kernel との互換のために用意されており, その値は無視されます。

svchdr は拡張 SVC ハンドラとして呼び出す関数の先頭アドレスを指定します。拡張 SVC ハンドラは, サブシステムの API となる部分で, システムコールと同様の方法で呼び出せます。

拡張 SVC ハンドラは C 言語だけで記述可能で, 以下の形式となります。

```
INT shdr( VP pk_para, FN fncd )
{
    /*
     * fncd により分岐して処理
     */
    return retcode; /* 拡張 SVC ハンドラの終了 */
}
```

fncd は機能コードです。機能コードの下位 8 ビットにはサブシステム ID が入ります。残りの上位ビットは, サブシステム側で任意に使用できます。通常は, サブシステム内の機能コードとして使用します。ただし, 機能コードは正の値でなければならないため, 最上位ビットは必ず "0" となります。

pk\_para は呼出し側から渡されたパラメータをパケット形式にしたものです。パケットの形式は, サブシステム側で任意に決められます。一般的には, C 言語で関数に引数を渡すときのスタックの形式と同じになります。これは, 多くの場合, C 言語の構造体の形式と同じです。

拡張 SVC ハンドラからの復帰値は, そのまま呼出し元へ関数の復帰値として戻します。

原則として、負の値をエラー値、"0" または正の値を正常時の復帰値とします。

なお、何らかの理由で拡張 SVC の呼出しに失敗した場合は、拡張 SVC ハンドラは呼び出されずに呼出し元には OS のエラーコード（負の値）を返すため、それと混同しないようにしてください。

拡張 SVC ハンドラはタスク部から呼び出された場合、準タスク部として実行します。タスク独立部からも呼出しが可能です。タスク独立部から呼び出された場合には拡張 SVC ハンドラもタスク独立部として実行します。

### 3.11.2 tk\_ref\_ssy(Refer Subsystem Status)

サブシステム定義情報を参照します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_ref_ssy ( ID ssid, T_RSSY *pk_rssy ) ;

typedef struct t_rssy {
    PRI ssypri;
    INT resblksz;
} T_RSSY;
```

#### ■ パラメータ

##### ● 入力

ssid           サブシステム ID (Subsystem ID)  
pk\_rssy       サブシステム定義情報 (Packet of Refer Subsystem)

##### ● 出力

ercd           エラーコード (Error Code)  
                パケットに返すデータ  
ssypri       サブシステム優先度 (Subsystem Priority)  
resblksz     リソース管理ブロックサイズ (バイト数)  
                (Resource Block Size)

#### ■ エラーコード

E\_OK           0       正常終了  
E\_ID           -18     不正 ID 番号 (ssid が 10 より小さい, または 255 より大きい)  
E\_NOEXS       -42     ssid は定義されていない。  
                (ssid のサブシステムが定義されていない)

#### ■ ディスパッチ要因

本システムコールではディスパッチしません。

#### ■ 解説

ssid で示された対象サブシステムの各種情報を参照します。

ssypri, resblksz は T-Kernel との互換性のために存在するメンバであり, μT-REALOS では使用しません。このため, これらの値は不定値となります。

ssid のサブシステムが定義されていない場合には E\_NOEXS のエラーで復帰します。

pk\_rssy が不正な場合でも, エラーチェックを行わず, 動作は保証されません。

## 3.12 デバイス管理機能のシステムコール

---

デバイス管理機能のシステムコールについて説明します。

---

### ■ デバイス管理機能のシステムコール

デバイス管理機能は、以下の 15 種類のシステムコールで構成されます。

- tk\_def\_dev(Define Device)
- tk\_ref\_idv(Refer Initial Device Information)
- tk\_opn\_dev(Open Device)
- tk\_cls\_dev(Close Device)
- tk\_rea\_dev(Read Device)
- tk\_srea\_dev(Synchronized Read Device)
- tk\_wri\_dev(Write Device)
- tk\_swri\_dev(Synchronized Write Device)
- tk\_wai\_dev(Wait Device)
- tk\_sus\_dev(Suspend Device)
- tk\_get\_dev(Get Device)
- tk\_ref\_dev(Refer Device)
- tk\_oref\_dev(Refer Device)
- tk\_lst\_dev(List Device)
- tk\_evt\_dev(Event Device)

なお本節の内容をより理解するために、「付録 C デバイスドライバインタフェース」も併せてお読みください。

### 3.12.1 tk\_def\_dev(Define Device)

---

デバイスを登録または登録を削除します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ID devid = tk_def_dev ( UB *devnm, T_DDEV *ddev, T_IDEV *idev ) ;
```

```
typedef struct t_ddev {  
    VP  exinf;  
    ATR drvatr;  
    ATR devatr;  
    INT nsub;  
    INT blksize;  
    FP  openfn;  
    FP  closefn;  
    FP  execfn;  
    FP  waitfn;  
    FP  abortfn;  
    FP  eventfn;  
} T_DDEV;
```

```
typedef struct t_idev {  
    ID  evtmbfid;  
} T_IDEV;
```



## ■ パラメータ

### ● 入力

devnm      物理デバイス名 (Device Name)  
 ddev      デバイス登録情報 (Packet of Define Device)

            パケットに設定するデータ

exinf      拡張情報  
 drvatr      ドライバ属性 (Driver Attribute)  
 drvatr := [ TDA\_OPENREQ ]

属性	値	意味
TDA_OPENREQ	0x0001	毎回オープン / クローズ

devatr      デバイス属性 (Device Attribute)  
 nsub      サブユニット数 ( Sub Unit No.)  
 blksize      デバイスのブロック数 (Block Size)  
 openfn      オープン関数エントリアドレス (Open Function)  
 closefn      クローズ関数エントリアドレス (Close Function)  
 execfn      処理開始関数エントリアドレス (Execute Function)  
 waitfn      完了待ち関数エントリアドレス (Wait Function)  
 abortfn      中止処理関数エントリアドレス (Abort Function)  
 eventfn      イベント関数エントリアドレス (Event Function)

### ● 出力

devid      デバイス ID (Device ID)  
      またはエラーコード (Error Code)  
 idev      デバイス初期情報 (Initial Device Information)  
      パケットに設定するデータ  
 evtmbfid      事象通知用メッセージバッファ ID

## ■ エラーコード

E\_LIMIT    -34      デバイスの登録数がシステムの上限より大きい  
 E\_PAR      -17      devnm の長さが 0 または 8 より大きい, nsub が負または 256 以上  
 E\_NOEXS    -42      devnm のデバイスは存在しない (ddev = NULL の場合 )

## ■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には,それが完了するまで待ち状態となり,ディスパッチが発生する場合があります。

### ■ 解説

devnm のデバイス名でデバイスを登録します。devnm のデバイスが既に登録されているときには新しい登録情報で更新します。更新の場合はデバイス ID を変更しません。ddev=NULL の場合には devnm のデバイス登録を削除します。すでに登録したデバイス数がシステムの上限值（コンフィギュレータで設定した最大デバイス登録数）に達している場合は、E\_LIMIT のエラーとなります。

T\_DDEV には以下の値を指定します。

- exinf  
デバイス管理機能の API からデバイス処理関数 (openfn ~ eventfn) を呼び出したときに、デバイス処理関数の引数としてデバイスドライバに渡します。デバイスドライバ側で任意に使用できます。
- drvatr  
ドライバ属性を指定します。
- devatr  
デバイス属性を指定します。デバイス属性の詳細は、「付録 C デバイスドライバインタフェース」を参照してください。
- nsub  
サブユニット数を指定します。サブユニットがないデバイスには "0" を指定します。
- blkksz  
デバイスのブロックサイズをバイト数で指定します。通常は、このサイズがデバイスの最小アクセスサイズとなります。
- openfn ~ eventfn  
デバイス処理関数のエントリアドレスを指定します。デバイス処理関数の詳細は、「付録 C デバイスドライバインタフェース」を参照してください。

idev にデバイス初期情報として evtmbfid にシステムデフォルトの事象通知用メッセージバッファ ID を返します。idev=NULL とした場合にはデバイス初期情報を格納しません。システムデフォルトの事象通知用メッセージバッファがない場合には evtmbfid に "0" を設定します。

事象通知用メッセージバッファは、デバイスで発生した様々な事象（例えば、メディアの装入、バッテリー異常など）を上位プログラムに通知するために使用するメッセージバッファの ID 番号です。デバイスの事象通知については、「付録 C デバイスドライバインタフェース」を参照してください。

ddev, idev が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.12.2 tk\_ref\_idv(Refer Initial Device Information)

デバイス初期情報を獲得します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_ref_idv ( T_IDEV *idev ) ;

typedef struct t_idev {
    ID evtmbfid;
} T_IDEV;
```

■ パラメータ

● 入力

idev            デバイス初期情報を格納する領域のアドレス  
                  (Initial Device Information)

● 出力

ercd            エラーコード (Error Code)  
                  パケットに返すデータ  
evtmbfid        事象通知用メッセージバッファ ID  
                  (Event Messagebuffer ID)

■ エラーコード

E\_OK            0            正常終了

■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了する  
まで待ち状態となり、ディスパッチが発生する場合があります。

■ 解説

デバイス初期情報を獲得します。tk\_def\_dev システムコールで得られるものと同じ内容  
です。  
idev が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.12.3 tk\_opn\_dev(Open Device)

デバイスをオープンします。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ID dd = tk_opn_dev ( UB *devnm, UINT omode ) ;
```

#### ■ パラメータ

##### ● 入力

devnm      デバイス名 (Device Name)  
 omode      オープンモード (Open Mode)  
 omode := (TD\_READ    TD\_WRITE    TD\_UPDATE) | [TD\_EXCL  
 TD\_WEXCL    TD\_REXCL]

モード	値	意味
TD_READ	0x0001	読出し専用
TD_WRITE	0x0002	書込み専用
TD_UPDATE	0x0003	読出しおよび書込み
TD_EXCL	0x0100	排他
TD_WEXCL	0x0200	排他書込み
TD_REXCL	0x0400	排他読出し

##### ● 出力

dd      デバイスディスクリプタ (Device Descriptor)  
 またはエラーコード (Error Code)

#### ■ エラーコード

E\_PAR      -17    TD\_READ, TD\_WRITE, TD\_UPDATE のいずれもが未設定または未定義のモード  
 E\_BUSY      -65    デバイスは使用中 (排他オープン中)  
 E\_NOEXS      -42    デバイスは存在しない  
 E\_LIMIT      -34    オープン可能な最大数より大きい  
 その他      デバイスドライバから返されたエラー

#### ■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了するまで待ち状態となり、ディスパッチが発生する場合があります。また、デバイスドライバにも依存します。

## ■ 解説

devnm で指定されたデバイスを omode で指定されたモードでオープンします。また、デバイスへのアクセスを準備します。復帰値にデバイスディスクリプタを返します。

omode は以下の値を指定します。

・TD\_READ, TD\_WRITE, TD\_UPDATE

アクセスモードを指定します。TD\_READ の場合には tk\_wri\_dev を使用できません。また、TD\_WRITE の場合には tk\_rea\_dev を使用できません。

・TD\_EXCL, TD\_WEXCL, TD\_REXCL

排他モードを指定します。TD\_EXCL は、同時オープンをすべて禁止します。TD\_WEXCL は、書き込みモード (TD\_WRITE または TD\_UPDATE) による同時オープンを禁止します。TD\_REXCL は、読出しモード (TD\_READ または TD\_UPDATE) による同時オープンを禁止します。

なお、物理デバイスをオープンした場合、その物理デバイスに属する論理デバイスをすべて同じモードでオープンしたのと同様に扱って排他オープンの処理を行います。

devnm が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.12.4 tk\_cls\_dev(Close Device)

デバイスをクローズします。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ER ercd = tk_cls_dev ( ID dd, UINT option ) ;
```

#### ■ パラメータ

##### ● 入力

dd            デバイスディスクリプタ (Device Descriptor)  
option        クローズオプション (Option)

オプション	値	意味
TD_EJECT	0x0001	クローズ時にメディア排出

##### ● 出力

ercd            エラーコード (Error Code)

#### ■ エラーコード

E_OK	0	正常終了
E_ID	-18	dd が不正 (dd が 0 以下または最大デバイスオープン数より大きい) またはオープンされていない
E_OACV	-27	オープンしたタスクとは別のタスクからの要求
その他		デバイスドライバから返されたエラー

#### ■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了するまで待ち状態となり、ディスパッチが発生する場合があります。また、デバイスドライバにも依存します。

#### ■ 解説

dd のデバイスディスクリプタをクローズします。処理中の要求があった場合には、その処理を中止させてクローズします。

option に TD\_EJECT が指定されていた場合、同一デバイスがほかのタスクからオープンされていないとメディアを排出します。ただし、メディアの排出ができないデバイスでは無視します。

### 3.12.5 tk\_rea\_dev(Read Device)

デバイスの読出しを開始します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ID reqid = tk_rea_dev ( ID dd, W start, VP buf, W size, TMO tmout ) ;
```

■ パラメータ

● 入力

- dd            デバイスディスクリプタ (Device Descriptor)
  - start        読出し開始位置 ( 0: 固有データ , < 0: 属性データ ) (Start)
  - buf           読み出したデータを格納するバッファ (Buffer)
  - size          読み出すサイズ (Size)
  - tmout        要求受け待ちタイムアウト時間 (ms) (Timeout)
- 0 から 0x7ffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

● 出力

- reqid        リクエスト ID (Request ID)  
              またはエラーコード (Error Code)

■ エラーコード

- E\_ID        -18    dd が不正 (dd が 0 以下または最大デバイスオープン数より大きい), またはオープンされていない
- E\_OACV     -27    オープンモードが不正 (TD\_WRITE でオープンされている)
- E\_LIMIT    -34    最大リクエスト数より大きい
- その他             デバイスドライバから返されたエラー

■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には, それが完了するまで待ち状態となり, ディスパッチが発生する場合があります。また, デバイスドライバにも依存します。

#### ■ 解説

デバイスから固有データまたは属性データの読出しを開始します。読出しを開始するだけで読出し完了を待たずに呼出し元へ復帰します。読出しが完了するまで `buf` を保持してください。 `tk_wai_dev` により読出し完了を待ちます。読出し開始のための処理にかかる時間はデバイスドライバにより異なります。必ずしも即座に復帰するとは限りません。固有データの場合には `start` および `size` の単位はデバイスごとに決められます。属性データの場合には `start` は属性データ番号、`size` はバイト数となり、`start` のデータ番号の属性データを読み出します。通常、`size` は読み出す属性データのサイズ以上です。複数の属性データを一度に読み出せません。`size=0` で本システムコール呼出し後、`tk_wai_dev` を呼び出すことにより、`tk_wai_dev` の復帰値、`asize` に読出し可能なサイズを設定します。

読出しまたは書込みの動作中である場合に新たな要求を受け付けられるか否かはデバイスドライバによります。新たな要求を受け付けられない状態の場合には要求受け付け待ちとなります。要求受け付け待ちのタイムアウト時間を `tmout` に指定します。

なお、タイムアウトするのは要求受け付けまでです。要求が受け付けられた後にはタイムアウトしません。



### 3.12.6 tk\_srea\_dev(Synchronous Read Device)

デバイスの同期読出しを行います。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_srea_dev ( ID dd, W start, VP buf, W size, W *asize ) ;
```

■ パラメータ

● 入力

dd	デバイスディスクリプタ (Device Descriptor)
start	読出し開始位置 ( 0: 固有データ , < 0: 属性データ ) (Start)
buf	読み出したデータを格納するバッファ (Buffer)
size	読み出すサイズ (Size)

● 出力

ercd	エラーコード (Error Code)
asize	読み出したサイズ (Size)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	dd が不正 (dd が 0 以下または最大デバイスオープン数より大きい) またはオープンされていない
E_OACV	-27	オープンモードが不正 (TD_WRITE でオープンされている)
E_LIMIT	-34	最大リクエスト数より大きい
その他		デバイスドライバから返されたエラー

■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には , それが完了するまで待ち状態となり , ディスパッチが発生する場合があります。デバイスドライバにも依存します。

#### ■ 解説

デバイスデータの同期読出しを行います。本システムコールが正常に復帰した場合には、buf に指定されたアドレスを先頭として asize 分のデバイスデータを格納します。本システムコールは以下の処理と同等です。

```
ER tk_srea_dev( ID dd, W start, VP buf, W size, W *asize )
{
    ER er, err, ioer;
    err = tk_rea_dev(dd, start, buf, size, TMO_FEVR);
    if ( err > 0 ) {
        err = tk_wai_dev(dd, er, asize, &ioer, TMO_FEVR);
        if ( err > 0 ) err = ioerr;
    }
    return err;
}
```

asize が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.12.7 tk\_wri\_dev(Write Device)

デバイスの書き込みを開始します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ID reqid = tk_wri_dev ( ID dd, W start, VP buf, W size, TMO tmout ) ;
```

■ パラメータ

● 入力

- dd            デバイスディスクリプタ (Device Descriptor)
  - start        書き込み開始位置 ( 0: 固有データ , < 0: 属性データ ) (Start)
  - buf          書き込むデータを格納するバッファ (Buffer)
  - size        書き込むサイズ (Size)
  - tmout       要求受け待ちタイムアウト時間 (ms) (Timeout)
- 0 から 0x7ffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

● 出力

- reqid        リクエスト ID (Request ID)  
              またはエラーコード (Error Code)

■ エラーコード

- E\_ID        -18    dd が不正 (dd が 0 以下または最大デバイスオープン数より大きい) またはオープンされていない
- E\_OACV     -27    オープンモードが不正 (TD\_READ でオープンされている)
- E\_RDONLY   -67    書き込めないデバイス
- E\_LIMIT    -34    最大リクエスト数より大きい
- その他             デバイスドライバから返されたエラー

■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了するまで待ち状態となり、ディスパッチが発生する場合があります。また、デバイスドライバにも依存します。

#### ■ 解説

デバイスへ固有データまたは属性データの書き込みを開始します。書き込みを開始するだけで書き込み完了を待たずに呼出し元へ復帰します。書き込みが完了するまで `buf` を保持してください。 `tk_wai_dev` により書き込み完了を待ちます。書き込み開始のための処理にかかる時間はデバイスドライバにより異なります。必ずしも即座に復帰するとは限りません。

固有データの場合には `start` および `size` の単位はデバイスごとに決められます。属性データの場合には `start` は属性データ番号、`size` はバイト数となり、`start` のデータ番号の属性データに書き込みます。通常、`size` は書き込む属性データのサイズと同じです。複数の属性データを一度に書き込めません。`size=0` で本システムコール呼出し後、`tk_wai_dev` を呼び出すことにより、`tk_wai_dev` の復帰値、`asize` に書き込み可能なサイズを設定します。

読出しまたは書き込みの動作中である場合に新たな要求を受け付けられるか否かはデバイスドライバによります。新たな要求を受け付けられない状態の場合には要求受け付け待ちとなります。要求受け付け待ちのタイムアウト時間を `tmout` に指定します。`tmout` には `TMO_POL(=0)` または `TMO_FEVR(=-1)` も指定できます。

なお、タイムアウトするのは要求受け付けまでです。要求が受け付けられた後にはタイムアウトしません。

### 3.12.8 tk\_swri\_dev(Synchronous Write Device)

デバイスの同期書込みを行います。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ER ercd = tk_swri_dev ( ID dd, W start, VP buf, W size, W *asize ) ;
```

■ パラメータ

● 入力

dd	デバイスディスクリプタ (Device Descriptor)
start	書込み開始位置 ( 0: 固有データ , < 0: 属性データ ) (Start)
buf	書き込むデータを格納したバッファ (Buffer)
size	書き込むサイズ (Size)

● 出力

ercd	エラーコード (Error Code)
asize	書き込んだサイズ (Size)

■ エラーコード

E_OK	0	正常終了
E_ID	-18	dd が不正 (dd が 0 以下または最大デバイスオープン数より大きい) またはオープンされていない
E_OACV	-27	オープンモードが不正 (TD_READ でオープンされている)
E_RDONLY	-67	書き込めないデバイス
E_LIMIT	-34	最大リクエスト数より大きい
その他		デバイスドライバから返されたエラー

■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には , それが完了するまで待ち状態となり , ディスパッチが発生する場合があります。また , デバイスドライバにも依存します。

#### ■ 解説

デバイスの同期書込みを行います。以下の処理と同等です。

```
ER tk_swri_dev( ID dd, W start, VP buf, W size, W *asize )
{
    ER er, err, ioer;
    err = tk_wri_dev(dd, start, buf, size, TMO_FEVR);
    if ( err > 0 ) {
        err = tk_wai_dev(dd, er, asize, &ioer, TMO_FEVR);
        if ( err > 0 ) err = ioer;
    }
    return err;
}
```

asize が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.12.9 tk\_wai\_dev(Wait Device)

デバイスの要求完了を待ちます。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ID creqid = tk_wai_dev ( ID dd, ID reqid, W *asize, ER *ioer, TMO tmout ) ;
```

■ パラメータ

● 入力

- dd            デバイスディスクリプタ (Device Descriptor)
- reqid        リクエスト ID (Request ID)
- tmout        要求受け待ちタイムアウト時間 (ms) (Timeout)  
0 から 0xffffffff の数値以外に以下のマクロが指定可能

名前	値	意味
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング

● 出力

- creqid        完了したリクエスト ID (Request ID) またはエラーコード (Error Code)
- asize        読み出したサイズまたは書き込んだサイズ (Size)
- ioer        入出力エラー (IO Error)

■ エラーコード

- E\_ID        -18    dd が不正 (dd が 0 以下または最大デバイスオープン数より大きい) またはオープンされていない  
reqid が 1 より小さいかシステムの上限值より大きい, または dd に対する要求ではない
- E\_OACV     -27    オープンしたタスクとは別のタスクからの要求
- E\_OBJ      -41    reqid の要求はほかのタスクで完了待ちしている
- E\_NOEXS    -42    処理中の要求はない (reqid=0 の場合のみ)
- その他            デバイスドライバから返されたエラー

■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には,それが完了するまで待ち状態となり,ディスパッチが発生する場合があります。また,デバイスドライバにも依存します。

## ■ 解説

dd に対する reqid の要求の完了を待ちます。

reqid=0 の場合には dd に対する要求のうちのいずれかが完了するのを待ちます。この場合、tk\_wai\_dev 呼出し時点で処理中の要求だけが完了待ちの対象となります。本システムコールの呼出し後に要求された処理は完了待ちの対象外です。

複数の要求を同時に処理している場合には、その要求の完了順序は必ずしも要求した順序ではなくデバイスドライバに依存します。ただし、要求した順序で処理した場合と結果が矛盾しない順序で処理することを保証します。例えば、ディスクからの読出しの場合には以下の処理順序の変更が考えられます。

要求順ブロック番号 1 4 3 2 5

処理順ブロック番号 1 2 3 4 5

このように順序を入れ替えて処理することにより、シークや回転待ちを減らすことができ、より効率的にディスクアクセスができます。

tmout に完了待ちのタイムアウト時間を指定します。TMO\_POL(=0) または TMO\_FEVR (=1) を指定することもできます。タイムアウト (E\_TMOUT) した場合には要求された処理を継続中なので、再度本システムコールにより完了を待つ必要があります。reqid>0 かつ tmout=TMO\_FEVR の場合にはタイムアウトすることはなく、必ず処理が完了します。

要求された処理結果のエラー（入出力エラーなど）を復帰値ではなく ioer に格納します。復帰値は、要求の完了待ちが正しくできなかった場合にエラーで復帰します。復帰値にエラーが返された場合には ioer の内容は不定値です。

復帰値	ioer	意味
E_OK	E_OK	待ち対象の処理が正常に完了しました。
E_OK	エラー	待ち対象の処理がエラーで完了しました。処理エラー要因をエラーコードとして ioer に格納します。
エラー	不定	完了待ちが失敗しました。失敗した要因を復帰値にエラーコードとして格納します。

また、復帰値にエラーが返された場合は処理を継続中なので、再度本システムコールにより完了を待つ必要があります。

同じリクエスト ID に複数のタスクから同時に完了待ちできません。reqid=0 で待っているタスクがあれば、同じ dd に対するほかのタスクの完了待ちは、E\_OBJ のエラーとなります。

同様に、reqid>0 で待っているタスクがあれば、ほかのタスクで reqid=0 の完了待ちはできません。この場合も E\_OBJ のエラーとなります。

asize, ioer が不正な場合でも、エラーチェックを行わず、動作は保証されません。



3.12.10 tk\_sus\_dev(Suspend Device)

デバイスをサスペンドします。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
INT cnt = tk_sus_dev ( UINT mode ) ;
```

■ パラメータ

● 入力

mode                   モード (Mode)  
mode := ( (TD\_SUSPEND | [TD\_FORCE])   TD\_DISSUS  
          TD\_ENASUS   TD\_CHECK)

モード	値	意味
TD_SUSPEND	0x0001	サスペンド
TD_DISSUS	0x0002	サスペンドを禁止
TD_ENASUS	0x0003	サスペンドを許可
TD_CHECK	0x0004	サスペンド禁止要求カウント 獲得
TD_FORCE	0x8000	強制サスペンド指定

● 出力

cnt                   サスペンド禁止要求カウント数 (Count)  
                      またはエラーコード (Error Code)

■ エラーコード

E\_PAR               -17   モードが未定義  
E\_BUSY              -65   サスペンド禁止中  
E\_QOVR              -43   サスペンド禁止要求カウントオーバ

■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了する  
まで待ち状態となり、ディスパッチが発生する場合があります。また、デバイスドライバ  
にも依存します。

### ■ 解説

mode に従って処理を行って、その処理を行った後のサスペンド禁止要求カウンタ数を復帰値に返します。

- TD\_SUSPEND サスペンド  
サスペンド許可状態の場合にはサスペンドします。サスペンド禁止状態の場合には E\_BUSY を返します。
- TD\_SUSPEND/TD\_FORCE 強制サスペンド  
サスペンド禁止状態の場合にもサスペンドします。
- TD\_DISSUS サスペンド禁止  
サスペンドを禁止します。
- TD\_ENASUS サスペンド許可  
サスペンドを許可します。サスペンドを禁止した回数以上に許可した場合には何もありません。
- TD\_CHECK サスペンド禁止カウンタ獲得  
サスペンド禁止要求を行っている回数の獲得だけを行います。

### ■ 補足事項

サスペンドは、以下の手順によって行います。

1. 各ディスクデバイス以外のサスペンド処理
2. 各ディスクデバイスのサスペンド処理
3. サスペンド状態へ移行

リジューム ( サスペンドからの復帰 ) は、以下の手順によって行います。

1. サスペンド状態から復帰
2. 各ディスクデバイスのリジューム処理
3. 各ディスクデバイス以外のリジューム処理

サスペンド禁止は、その要求回数をカウンタします。同じ回数だけサスペンド許可を要求しないとサスペンドを許可しません。システム起動時はサスペンド許可 ( サスペンド禁止要求カウンタ = 0 ) です。サスペンド禁止要求カウンタはシステム全体で 1 つです。サスペンド禁止要求カウンタの上限は 2147483647(0x7fffffff) です。この上限より大きい場合には E\_QOVR のエラーで復帰します。

### 3.12.11 tk\_get\_dev(Get Device)

デバイスのデバイス名を獲得します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

■ C 言語インタフェース

```
ID phyid = tk_get_dev ( ID devid, UB *devnm ) ;
```

■ パラメータ

● 入力

devid          デバイス ID (Device ID)

● 出力

phyid          物理デバイスのデバイス ID (Physical Device ID) またはエラーコード (Error Code)

devnm          デバイス名 (Device Name)

■ エラーコード

E\_NOEXS      -42      devid のデバイスは存在しない

■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了するまで待ち状態となり、ディスパッチが発生する場合があります。

■ 解説

devid で示すデバイスのデバイス名を獲得して devnm に格納します。

devid は物理デバイスのデバイス ID または論理デバイスのデバイス ID です。devid が物理デバイスの場合には devnm には物理デバイス名が格納され、論理デバイスの場合には devnm には論理デバイス名を格納します。devnm には L\_DEVNM(8)+1 バイト以上の領域が必要です。

復帰値には、devid のデバイスが属する物理デバイスのデバイス ID を返します。

devnm が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.12.12 tk\_ref\_dev(Refer Device)

---

デバイスのデバイス情報を獲得します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ID devid = tk_ref_dev ( UB *devnm, T_RDEV *rdev ) ;
```

```
typedef struct t_rdev {
    ATR devatr;
    INT blksize;
    INT nsub;
    INT subno;
} T_RDEV;
```

#### ■ パラメータ

##### ● 入力

devnm	デバイス名 (Device Name)
rdev	デバイス情報を格納する領域のアドレス (Packet of Refer Device)

##### ● 出力

devid	デバイス ID (Device ID) またはエラーコード (Error Code)
パケットに返すデータ	
devatr	デバイス属性 (Device Attribute)
blksize	固有データのブロックサイズ (-1: 不明) (Block Size)
nsub	サブユニット数 (Number of Sub-unit)
subno	0: 物理デバイス, 1 ~ nsub: サブユニット番号 +1 (Sub-unit Number)

#### ■ エラーコード

E_NOEXS	-42	devnm のデバイスは存在しない
---------	-----	-------------------

#### ■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了するまで待ち状態となり、ディスパッチが発生する場合があります。

**■ 解説**

devnm で示すデバイスのデバイス情報を獲得して rdev に格納します。rdev=NULL とした場合にはデバイス情報を格納しません。

nsub は、devnm で示すデバイスが属する物理デバイスのサブユニット数です。

復帰値に devnm のデバイスのデバイス ID を返します。

devnm, rdev が不正な場合でも、エラーチェックを行わず、動作は保証されません。

デバイス属性については、「付録C デバイスドライバインタフェース」を参照してください。

### 3.12.13 tk\_oref\_dev(Refer Device)

---

デバイスのデバイス情報を獲得します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
ID devid = tk_oref_dev ( ID dd, T_RDEV *rdev ) ;
```

```
typedef struct t_rdev {
    ATR devatr;
    INT blkosz;
    INT nsub;
    INT subno;
} T_RDEV;
```

#### ■ パラメータ

##### ● 入力

dd	デバイスディスクリプタ (Device Descriptor)
rdev	デバイス情報を格納する領域のアドレス (Packet of Refer Device)

##### ● 出力

devid	デバイス ID (Device ID) またはエラーコード (Error Code)
パケットに返すデータ	
devatr	デバイス属性 (Device Attribute)
blkosz	固有データのブロックサイズ (-1: 不明) (Block Size)
nsub	サブユニット数 (Number of Sub-unit)
subno	0: 物理デバイス, 1 ~ nsub: サブユニット番号 +1 (Sub-unit Number)

#### ■ エラーコード

E_OACV	-27	オープンしたタスクとは別のタスクからの要求
--------	-----	-----------------------

#### ■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了するまで待ち状態となり、ディスパッチが発生する場合があります。

## ■ 解説

dd で示すデバイスのデバイス情報を獲得して rdev に格納します。rdev=NULL とした場合にはデバイス情報を格納しません。

nsub は、dd で示すデバイスが属する物理デバイスのサブユニット数です。

復帰値に dd で示すデバイスのデバイス ID を返します。

rdev が不正な場合でも、エラーチェックを行わず、動作は保証されません。

デバイス属性については、「付録 C デバイスドライバインタフェース」を参照してください。

### 3.12.14 tk\_lst\_dev(List Device)

---

登録済みデバイスの一覧を獲得します。

---

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
INT cnt = tk_lst_dev ( T_LDEV *ldev, INT start, INT ndev ) ;
```

```
typedef struct t_ldev {
    ATR devatr;
    INT blkosz;
    INT nsub;
    UB  devnm[L_DEVNM] ;
} T_LDEV;
```

#### ■ パラメータ

##### ● 入力

ldev	登録デバイス情報の格納領域 ( 配列 ) (Packet of List Device)
start	開始番号 (Start)
ndev	獲得数 (Number of Device)

##### ● 出力

cnt	残りのカウント数 (Count) またはエラーコード (Error Code)
パケットに返すデータ	
devatr	デバイス属性 (Device Attribute)
blkosz	固有データのブロックサイズ (-1: 不明) (Block Size)
nsub	サブユニット数 (Number of Sub-unit)
devnm[L_DEVNM]	物理デバイス名 (Device Name)

#### ■ エラーコード

E_NOEXS	-42	獲得すべき情報がない (start が登録済みのデバイスの個数以上)
E_PAR	-17	start または ndev が負

#### ■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了するまで待ち状態となり、ディスパッチが発生する場合があります。



## ■ 解説

登録済みのデバイスの情報を獲得します。

登録デバイスは物理デバイス単位で管理します。したがって、登録デバイス情報も物理デバイス単位で獲得します。

登録デバイスの数が  $N$  の場合には登録デバイスに  $0 \sim N-1$  の連番を振ります。この連番に従って、 $start$  番目から  $ndev$  個の登録情報を獲得して  $ldev$  に格納します。 $ldev$  には  $ndev$  個の情報を格納するのに十分な大きさが必要です。

復帰値には残りのカウンタ数 ( $N - start$ ) を返します。残りカウンタ数が  $ndev$  個に満たない場合には残りのすべてを格納します。復帰値  $ndev$  の場合には、すべての登録情報が獲得できたことを示します。

なお、この連番はデバイスの登録・抹消があると変化します。したがって、複数回に分けて獲得すると、正確な情報が得られない場合があります。

例えば、10 個のデバイスが登録されている状態で 3 個ずつ登録情報を獲得する場合には、 $start$  を 0 3 6 9 と変化させて本システムコールを 4 回呼び出すことにより、すべてのデバイスの登録情報を獲得できます。本システムコールの 1 回目の呼出しから 4 回目の呼出しの間にデバイスの登録、削除を行った場合には正確な情報が得られない可能性があります。この場合には再度  $start$  に "0" を指定して、最初からデバイス情報の獲得をやり直してください。

$ldev$  が不正な場合でも、エラーチェックを行わず、動作は保証されません。

### 3.12.15 tk\_evt\_dev(Event Device)

デバイスにドライバ要求イベントを送信します。

タスク部		タスク独立部		ディスパッチ禁止状態	
------	--	--------	--	------------	--

#### ■ C 言語インタフェース

```
INT retval = tk_evt_dev ( ID devid, INT evttyp, VP evtinf ) ;
```

#### ■ パラメータ

##### ● 入力

devid イベント送信先デバイス ID (Device ID)  
 evttyp ドライバ要求イベントタイプ (Event Type)

イベント	値	意味
TDV_CARDEVT	1	PC カードイベント
TDV_USBEVT	2	USB イベント

evtinf イベントタイプ別の情報 (Event Information)

##### ● 出力

retval デバイスドライバからの復帰値 (Return Value)  
 またはエラーコード (Error Code)

#### ■ エラーコード

E\_NOEXS -42 devid のデバイスは存在しない  
 E\_PAR -17 デバイス管理内部イベント (evttyp < 0) を指定できない

#### ■ ディスパッチ要因

他タスクがデバイス管理機能の API を呼び出して待ち状態の場合には、それが完了するまで待ち状態となり、ディスパッチが発生する場合があります。

#### ■ 解説

devid のデバイス ( デバイスドライバ ) にドライバ要求イベントを送信します。  
 各イベントタイプの詳細は、「T-Kernel 標準デバイスドライバ仕様書」を参照してください ( 本仕様書は T-Engine フォーラムのホームページから無料でダウンロードできます )。

エラーコード，定数マクロ，デバイスドライバインタフェース， $\mu$ ITRON 仕様 OS からの移行時の注意事項について説明します。また，アルファベット順のシステムコール索引を掲載します。

付録 A エラーコード一覧

付録 B 定数マクロ一覧

付録 C デバイスドライバインタフェース

付録 D  $\mu$ ITRON 仕様 OS からの移行時の注意点

付録 E システムコール索引

## 付録 A エラーコード一覧

エラーコードの一覧を示します。

### ■ エラーコード一覧

表 A-1 エラーコード一覧

ラベル	エラーコード *1			意味
E_OK	0	H'00	H'00000000	正常終了
E_SYS*2	- 5	-H'05	H'fffffffb	システムエラー
E_NOCOP*2	- 6	-H'06	H'fffffffa	コプロセッサ使用不可
E_NOSPT*2	- 9	-H'09	H'ffffff7	未サポート機能
E_RSFN*2	-10	-H'0a	H'ffffff6	予約機能コード番号
E_RSATR	-11	-H'0b	H'ffffff5	予約属性
E_PAR	-17	-H'11	H'ffffffef	パラメータエラー
E_ID	-18	-H'12	H'ffffffee	不正 ID 番号
E_CTX	-25	-H'19	H'ffffffe7	コンテキストエラー
E_MACV*2	-26	-H'1a	H'ffffffe6	メモリアクセス不能, メモリアクセス権違反
E_OACV	-27	-H'1b	H'ffffffe5	オブジェクトアクセス権違反
E_ILUSE	-28	-H'1c	H'ffffffe4	システムコール不正使用
E_NOMEM	-33	-H'21	H'fffffdff	メモリ不足
E_LIMIT	-34	-H'22	H'fffffdfe	システムの制限を超過
E_OBJ	-41	-H'29	H'fffffd7	オブジェクトの状態が不正
E_NOEXS	-42	-H'2a	H'fffffd6	オブジェクトが存在していない
E_QOVR	-43	-H'2b	H'fffffd5	キューイングまたはネストのオーバフロー
E_RLWAI	-49	-H'31	H'fffffc7f	待ち状態強制解除
E_TMOUT	-50	-H'32	H'fffffc7e	ポーリング失敗またはタイムアウト
E_DLT	-51	-H'33	H'fffffc7d	待ちオブジェクトが削除された
E_DISWAI*2	-52	-H'34	H'fffffc7c	待ち禁止による待ち解除
E_IO	-57	-H'39	H'fffffc77	入出力エラー
E_NOMDA*2	-58	-H'3a	H'fffffc76	メディアがない
E_BUSY	-65	-H'41	H'fffffb7e	ビジー状態
E_ABORT*2	-66	-H'42	H'fffffb7d	中止した
E_RONLY	-67	-H'43	H'fffffb7c	書き込み禁止

\*1: エラーコードは, 左から符号付き 10 進, 符号付き 16 進, 16 進絶対値の 3 種類の表記で記載しています。

\*2:  $\mu$ T-Kernel では使用しません。T-Kernel との互換性のために規定されています。

## 付録 B 定数マクロ一覧

定数マクロの一覧を示します。

### ■ 定数マクロ一覧

表 B-1 定数マクロ一覧 (1 / 4)

種別	マクロ名	値	意味
タスク	TSK_SELF	0	自タスク
	TPRI_INI	0	起動時優先度
	TPRI_RUN	0	実行中タスクの優先度
オブジェクト生成	TA_ASM	0	アセンブラ言語で記述
	TA_HLNG	1	C 言語で記述
	TA_USERBUF	0x20	ユーザバッファを指定
	TA_DSNAME	0x40	DS オブジェクト名称を指定
	TA_RNG0	0	メモリ保護レベル 0
	TA_RNG1	0x100	メモリ保護レベル 1
	TA_RNG2	0x200	メモリ保護レベル 2
	TA_RNG3	0x300	メモリ保護レベル 3
タスクの状態	TTS_RUN	0x1	実行状態
	TTS_RDY	0x2	実行可能状態
	TTS_WAI	0x4	待ち状態
	TTS_SUS	0x8	強制待ち状態
	TTS_WAS	0xc	2重待ち状態 (WAITING + SUSPENDED)
	TTS_DMT	0x10	休止状態
タスクの待ち状態	TTW_SLP	0x1	起床待ち
	TTW_DLY	0x2	遅延待ち
	TTW_SEM	0x4	セマフォ待ち
	TTW_FLG	0x8	イベントフラグ待ち
	TTW_MBX	0x40	メールボックス受信待ち
	TTW_MTX	0x80	ミューテックス待ち
	TTW_SMBF	0x100	メッセージバッファ送信待ち
	TTW_RMBF	0x200	メッセージバッファ受信待ち
	TTW_CAL	0x400	ランデブ呼出し待ち

表 B-1 定数マクロ一覧 (2 / 4)

種別	マクロ名	値	意味
タスクの待ち状態	TTW_ACP	0x800	ランデブ受付け待ち
	TTW_RDV	0x1000	ランデブ終了待ち
	TTW_MPF	0x2000	固定長メモリプール獲得待ち
	TTW_MPL	0x4000	可変長メモリプール獲得待ち
待ち行列	TA_TFIFO	0	待ち行列が FIFO 順
	TA_TPRI	0x1	待ち行列がタスク優先度順
	TA_FIRST	0	待ち行列先頭タスクを優先
	TA_CNT	0x2	要求数の少ないタスクを優先
ミューテックス属性	TA_INHERIT	0x2	優先度継承プロトコル
	TA_CEILING	0x3	優先度上限プロトコル
イベントフラグ属性	TA_WSGL	0	複数タスク待ちを禁止
	TA_WMUL	0x8	複数タスク待ちを許可
イベントフラグ待ちモード	TWF_ANDW	0	イベントフラグの AND 待ち
	TWF_ORW	0x1	イベントフラグの OR 待ち
	TWF_CLR	0x10	すべてのイベントフラグをクリア
	TWF_BITCLR	0x20	特定のイベントフラグをクリア
メールボックス属性	TA_MFIFO	0	メッセージを FIFO で管理
	TA_MPRI	0x2	メッセージを優先度順で管理
周期ハンドラ属性	TA_STA	0x2	周期ハンドラの起動
	TA_PHS	0x4	周期ハンドラの位相を保存
周期ハンドラの動作	TCYC_STP	0	周期ハンドラを停止
	TCYC_STA	0x1	周期ハンドラの動作開始
アラームハンドラの動作	TALM_STP	0	アラームハンドラを停止
	TALM_STA	0x1	アラームハンドラの動作開始
システムの状態	TSS_TSK	0	タスク実行中
	TSS_DDSP	0x1	ディスパッチ禁止状態
	TSS_DINT	0x2	割り込み禁止状態
	TSS_INDP	0x4	タスク独立部実行中
	TSS_QTSK	0x8	準タスク実行中
デバイス共通属性	TD_PROTECT	0x8000	書込み禁止
	TD_REMOVABLE	0x4000	メディアの取外し可能

表 B-1 定数マクロ一覧 (3 / 4)

種別	マクロ名	値	意味
デバイスタイプ	TD_DEVKIND	0xff	デバイスタイプ / メディアタイプマスク
	TD_DEVTYPE	0xf0	デバイスタイプマスク
	TDK_UNDEF	0	未定義デバイス
ディスクデバイス	TDK_DISK	0x10	ディスクデバイス
	TDK_DISK_UNDEF	0x10	未定義のディスクデバイス
	TDK_DISK_RAM	0x11	RAM ディスク
	TDK_DISK_ROM	0x12	ROM ディスク
	TDK_DISK_FLA	0x13	フラッシュメモリディスク
	TDK_DISK_FD	0x14	フロッピーディスク
	TDK_DISK_HD	0x15	ハードディスク
	TDK_DISK_CDROM	0x16	CD-ROM
デバイスオープンモード	TD_READ	0x1	読み出し専用
	TD_WRITE	0x2	書き込み専用
	TD_UPDATE	0x3	読み出しおよび書き込み
	TD_EXCL	0x100	オープンの排他
	TD_WEXCL	0x200	書き込み専用オープンの排他
	TD_REXCL	0x400	読み出し専用オープンの排他
デバイスクローズオプション	TD_EJECT	0x1	クローズ時にメディアを排出
デバイスサスペンドモード	TD_SUSPEND	0x1	サスペンド
	TD_DISSUS	0x2	サスペンド禁止
	TD_ENASUS	0x3	サスペンド許可
	TD_CHECK	0x4	サスペンド許可 / 禁止状態のチェック
	TD_FORCE	0x8000	強制サスペンド
事象通知用メッセージバッファ	TDN_EVENT	-1	事象通知用メッセージバッファ ID
	TDN_DISKINFO	-2	ディスク情報
	TDN_DISPSPEC	-3	ディスプレイデバイス情報
	TDN_PCMCIAINFO	-4	PC カード情報
デバイス登録	TDA_OPENREQ	0x1	毎回オープン / クローズ
デバイス要求	TDC_READ	0x1	読み出し要求
	TDC_WRITE	0x2	書き込み要求

表 B-1 定数マクロ一覧 (4 / 4)

種別	マクロ名	値	意味
ドライバイベント	TDV_SUSPEND	-1	サスペンド
	TDV_RESUME	-2	リジューム
	TDV_CARDEVT	0x1	PC カードイベント
	TDV_USBEVT	0x2	USB イベント
NULL	NULL	0	NULL
	TA_NULL	0	特別な属性を指定しない
タイムアウト値	TMO_FEVR	-1	永久待ち
	TMO_POL	0	ポーリング



## 付録 C デバイスドライバインタフェース

カーネルとデバイスドライバ間のインタフェースについて説明します。

### ■ デバイスドライバインタフェース

デバイスドライバインタフェースは、 $\mu$ T-REALOS のデバイス管理機能とユーザプログラムのデバイスドライバ間のインタフェースです。デバイスドライバインタフェースは以下の内容で構成されます。

- デバイス名の命名規則
- デバイス ID
- デバイス属性
- デバイスディスクリプタ
- リクエスト ID
- データ番号
- デバイスへの入出力処理要求のデータ形式
- デバイス処理関数
- デバイスの属性データ
- デバイス事象通知
- サスペンド / リジューム処理

### ■ デバイス名の命名規則

デバイス名は最大 8 文字の文字列で、以下の要素により構成されます。

- 種別               : デバイスの種別を示す名前です。使用可能な文字は a ~ z, A ~ Z です。
- ユニット           : 物理的なデバイスを示す番号です。使用可能な文字は a ~ z です。1 文字で指定します。ユニットごとに a から順に割り当てます。
- サブユニット      : 論理的なデバイスを示す番号です。使用可能な文字は 0 ~ 254 の数字で最大 3 文字です。サブユニットごとに 0 から順に割り当てます。

デバイス名は、種別 + ユニット + サブユニットの形式で表しますが、ユニット、サブユニットはデバイスによっては存在しない場合もあります。その場合にはそれぞれのフィールドは存在しません。

特に、種別 + ユニットの形式を物理デバイス名とよびます。また、種別 + ユニット + サブユニットを論理デバイス名とよび、物理デバイス名と区別します。サブユニットがない場合には物理デバイス名と論理デバイス名は同じになります。単にデバイス名といったときは論理デバイス名を指します。

サブユニットは、一般的にはハードディスクの区画 (パーティション) を表しますが、それ以外でも論理的なデバイスを表すために使用できます。

(例)

had	ハードディスク (ディスク全体)
hda0	ハードディスク (先頭の区画)
fda	フロッピーディスク

```

rsa      シリアルポート
kbpd     キーボード / ポインティングデバイス

```

## ■ デバイス ID

デバイス ( デバイスドライバ ) を  $\mu$ T-Kernel に登録することにより, デバイス ( 物理デバイス名 ) にデバイス ID (>0) が割り当てられます。デバイス ID は物理デバイスごとに割り当てられて, 論理デバイスのデバイス ID は物理デバイスに割り当てられたデバイス ID にサブユニット番号 +1(1 ~ 255) を加えたものとなります。

devid: 登録時に割り当てられたデバイス ID

```

devid      物理デバイス
devid+n+1  n 番目のサブユニット ( 論理デバイス )

```

( 例 )

```

hda      devid ハードディスク全体
hda0     devid + 1 ハードディスクの先頭の区画
hda1     devid + 2 ハードディスクの 2 番目の区画

```

## ■ デバイス属性

各デバイスの特長を表してデバイスの種類分けを行うため, デバイス属性を以下のように定義します。

IIII IIII IIII IIII PRxx xxxx KKKK KKKK

上位 16 ビットは, デバイス依存属性でデバイスごとに定義します。下位 16 ビットは標準属性です。

標準属性は, 以下の情報を持ちます。

- 共通属性 ( ビット 15 ~ 8 )

名前	値	意味
TD_PROTECT	0x8000	P: 書込み禁止
TD_REMOVABLE	0x4000	R: メディアの取外し可能

- デバイスタイプ ( ビット 7 ~ 4 ) およびメディア種別 ( ビット 3 ~ 0 )

名前	値	意味
TD_DEVKIND	0x00ff	K: デバイス / メディア種別
TD_DEVTYPE	0xx00f0	デバイスタイプ
TDK_UNDEF	0x0000	未定義 / 不明
TDK_DISK	0x0010	ディスクデバイス
TDK_DISK_UNDEF	0x0010	その他のディスク
TDK_DISK_RAM	0x0011	RAM ディスク ( 主メモリ使用 )
TDK_DISK_ROM	0x0012	ROM ディスク ( 主メモリ使用 )
TDK_DISK_FLA	0x0013	Flash ROM, その他のシリコンディスク

名前	値	意味
TDK_DISK_FD	0x0014	フロッピーディスク
TDK_DISK_HD	0x0015	ハードディスク
TDK_DISK_CDROM	0x0016	CD-ROM

上記以外のデバイスタイプは、現在未定義ですが、将来的には新たなデバイスが追加で定義される予定です。未定義のデバイスは未定義 TDK\_UNDEF とします。

なお、デバイスタイプは、アプリケーションがデバイス/メディアの種類によって処理内容を変える必要がある場合など、デバイスを使用する側で区別が必要となる場合に定義します。特に明確な区別をする必要がないデバイスに関しては、デバイスタイプを割り当てて必要はありません。

## ■ デバイスディスクリプタ

デバイスをアクセスするための識別子で、デバイスをオープンしたときに  $\mu$ T-Kernel によりデバイスディスクリプタ (>0) が割り当てられます。

## ■ リクエスト ID

デバイスに入出力を要求したときに、その要求の識別子としてリクエスト ID(>0) が割り当てられます。このリクエスト ID により入出力の完了を待てます。

## ■ データ番号

デバイスのデータはデータ番号により指定します。データ番号は以下の 2 種類に分類されます。

- 固有データ：データ番号 0

デバイス固有のデータで、データ番号はデバイスごとに定義します。

(例) ディスクデータ番号 = 物理ブロック番号

シリアル回線データ番号は "0" だけを使用

- 属性データ：データ番号 < 0

ドライバやデバイスの状態の獲得や設定、特殊機能などを指定します。

データ番号のいくつかは共通で定義されていますが、デバイス独自にも定義できます。

## ■ 属性データ

属性データは、以下の 3 種類に分類されます。

- 共通属性

すべてのデバイス (デバイスドライバ) に共通に定義する属性。

- デバイス種別属性

同じ種類に分類されるデバイス (デバイスドライバ) に共通に定義する属性。

- デバイス個別属性

各デバイス (デバイスドライバ) ごとに独自に定義した属性。

デバイス種別属性およびデバイス個別属性については、デバイスごとに定義します。ここでは、共通属性だけを定義します。

共通属性の属性データ番号は -1 ~ -99 の範囲となります。共通属性のデータ番号はすべてのデバイスで共通となりますが、すべてのデバイスが必ずしもすべての共通属性に対応しているとは限りません。対応していないデータ番号が指定されたときには E\_PAR のエラーで復帰します。

共通属性	属性データ番号	意味
TDN_EVENT	(-1)	RW: 事象通知用メッセージバッファ ID
TDN_DISKINFO	(-2)	R-: ディスク情報
TDN_DISPSPEC	(-3)	R-: 表示デバイス仕様

RW: 読出し (tk\_rea\_dev)/ 書込み (tk\_wri\_dev) 可能

R- : 読出し (tk\_rea\_dev) のみ可能

#### ● TDN\_EVENT : 事象通知用メッセージバッファ ID

データ形式 :ID

デバイス事象通知用のメッセージバッファの ID です。デバイス登録時にシステムデフォルトのメッセージバッファ ID を渡すので、ドライバ起動時の初期設定としてその ID を設定します。

"0" が設定されている場合にはデバイス事象通知を行いません。

#### ● TDN\_DISKINFO : ディスク情報

データ形式 DiskInfo

```
typedef enum {
    DiskFmt_STD = 0,          /* 標準 (HD など) */
    DiskFmt_2DD = 1,         /* 2DD 720KB */
    DiskFmt_2HD = 2,         /* 2HD 1.44MB */
    DiskFmt_CDROM = 4,       /* CD-ROM 640MB */
} DiskFormat;

typedef struct {
    DiskFormat format;        /* フォーマット形式 */
    UW protect:1;            /* プロテクト状態 */
    UW removable:1;          /* 取外し可否 */
    UW rsv:30;               /* 予約 (常に "0") */
    W blocksize;             /* ブロックバイト数 */
    W blockcount;            /* 総ブロック数 */
} DiskInfo;
```

## ● TDN\_DISPSPEC : 表示デバイス仕様

データ形式 DEV\_SPEC

```
typedef struct {
    H attr;                /* デバイス属性 */
    H planes;              /* プレーン数 */
    H pixbits;              /* ピクセルビット数 (境界 / 有効) */
    H hpixels;              /* 横のピクセル数 */
    H vpixels;              /* 縦のピクセル数 */
    H hres;                 /* 横の解像度 */
    H vres;                 /* 縦の解像度 */
    H color[4];             /* カラー情報 */
    H resv[6];              /* 予約 */
} DEV_SPEC;
```

## ■ 入出力要求のデータ形式

デバイスドライバへの入出力要求は、リクエスト ID に対応した以下の要求パケットにより行います。

```
typedef struct t_devreq {
    struct t_devreq *next; /* I: 要求パケットのリンク (NULL: 終端) */
    VP exinf;              /* X: 拡張情報 */
    ID devid;              /* I: 対象デバイス ID */
    INT cmd:4;             /* I: 要求コマンド */
    BOOL abort:1;          /* I: 中止要求を行ったとき TRUE */
    INT start;             /* I: 開始データ番号 */
    INT size;              /* I: 要求サイズ */
    VP buf;                /* I: 入出力バッファアドレス */
    INT asize;             /* O: 結果サイズ */
    ER error;              /* O: 結果エラー */
} T_DEVREQ;
```

コメント内の I は入力パラメータ、O は出力パラメータ、X はデバイス管理機能で未参照のパラメータで、入力パラメータはデバイスドライバでは変更できません。入力パラメータ (I) 以外は、デバイス管理機能が最初に "0" にクリアします。その後、デバイス管理機能は変更しません。

next は、要求パケットをリンクするために使用します。デバイス管理機能内の要求パケットの管理に使用するほか、完了待ち関数 (waitfn)、中止処理関数 (abortfn) でも使用します。

exinf はデバイスドライバ側で任意に使用できます。デバイス管理機能では内容には関知しません。

devid は要求対象のデバイス ID を指定します。

cmd は要求コマンドを指定します。

cmd := (TDC\_READ || TDC\_WRITE)

cmd	値	意味
TDC_READ	1	読出し要求
TDC_WRITE	2	書込み要求

abort に TRUE が設定されている場合、中止処理が要求されたことを示します。TRUE が設定される条件は以下のとおりです。

- デバイスがクローズされた場合

start, size は、tk\_rea\_dev, tk\_wri\_dev で指定された start, size をそのまま設定します。

buf は、tk\_rea\_dev, tk\_wri\_dev で指定された buf をそのまま設定します。

asize は、tk\_wai\_dev の asize に返す値をデバイスドライバが設定します。

error は、tk\_wai\_dev の復帰値として返すエラーコードをデバイスドライバが設定します。正常な場合には E\_OK を設定します。

## ■ デバイス処理関数

これらの処理関数をデバイス管理機能によって呼び出します。これらの処理関数は、再入可能 (reentrant) な構造で作成してください。また、処理関数を排他的に呼び出すことは保証しません。例えば、複数のタスクから同じデバイスに同時に要求があった場合には、それぞれのタスクが同時に処理関数を呼び出す場合があります。このため、デバイスドライバ側は、必要に応じて排他制御を行う必要があります。

### ● オープン関数 (openfn)

<呼出しインタフェース>

```
ER ercd = openfn( ID devid, UINT omode, VP exinf )
```

<パラメータ>

#### 【入力】

devid	オープンするデバイスのデバイス ID
omode	オープンモード (tk_opn_dev と同じ)
exinf	デバイス登録時に指定された拡張情報

#### 【出力】

ercd	エラーコード
------	--------

## &lt; 解説 &gt;

tk\_opn\_dev が呼び出されたときに openfn を呼び出します。

openfn では、デバイスの使用開始のための処理を行います。処理内容はデバイスに依存し、デバイスに合わせて自由に処理できます。必要な処理がない場合には何も処理する必要はありません。また、オープンされているか否かをデバイスドライバで記憶する必要もなく、オープンされていない (openfn が呼び出されていない) 理由だけで、ほかの処理関数が呼び出されたときエラーにする必要もありません。オープンされていない状態でほかの処理関数が呼び出された場合、デバイスドライバの動作に問題ない場合には要求を処理できます。

openfn でデバイスの初期化などを行う場合でも、原則として待ちを伴う処理は行いません。できる限り速やかに処理を行って openfn から復帰してください。例えば、シリアル回線のように通信モードを設定する必要があるデバイスでは、tk\_wri\_devにより通信モードが設定されたときにデバイスを初期化できます。このため、openfn ではデバイスの初期化を行う必要はありません。

同じデバイスが多重オープンされた場合、通常は最初のオープン時だけを呼び出しますが、デバイス登録時にドライバ属性として TDA\_OPENREQ が指定された場合にはすべてのオープン時に呼び出します。多重オープンやオープンモードに関する処理はデバイス管理機能で行うため、openfn ではそれらに関する処理は特に必要ありません。omode も参考情報として渡すだけで、omode に関する処理を行う必要はありません。

## ● クローズ関数 (closefn)

## &lt; 呼出しインタフェース &gt;

```
ER ercd = closefn( ID devid, UINT option, VP exinf )
```

## &lt; パラメータ &gt;

## 【入力】

devid	クローズするデバイスのデバイス ID
option	クローズオプション (tk_cls_dev と同じ)
exinf	デバイス登録時に指定された拡張情報

## 【出力】

ercd	エラーコード
------	--------

## &lt; 解説 &gt;

tk\_cls\_dev が呼び出されたときに closefn を呼び出します。

closefn では、デバイスの使用終了のための処理を行います。処理内容はデバイスに依存し、デバイスに合わせて自由に処理できます。必要な処理がない場合には何も処理する必要はありません。

メディアの取外しが可能なデバイスの場合、option に TD\_EJECT が指定されている場合にはメディアの排出を行います。

closefn でデバイスの終了処理やメディアの排出を行う場合でも、待ちを伴う処理は原則として行いません。できる限り速やかに処理を行って closefn から復帰してください。メディアの排出に時間がかかる場合には、排出の完了を待たずに closefn から復帰しても構いません。

同じデバイスが多重オープンされていた場合、通常は最後のクローズ時だけ呼び出しますが、デバイス登録時にドライバ属性として TDA\_OPENREQ が指定された場合にはすべてのクローズ時に呼び出します。ただし、この場合も最後のクローズにしか option に TDA\_EJECT を指定することはありません。多重オープンやオープンモードに関する処理はデバイス管理機能で行うため、closefn ではそれらに関する処理は特に必要ありません。

## ● 処理開始関数 (execfn)

< 呼出しインタフェース >

```
ER ercd = execfn( T_DEVREQ *devreq, TMO tmout, VP exinf )
```

< パラメータ >

### 【入力】

devreq	要求パケット
tmout	要求受け付け待ちタイムアウト時間 (ms)
exinf	デバイス登録時に指定された拡張情報

### 【出力】

ercd	エラーコード
------	--------

< 解説 >

tk\_rea\_dev, tk\_srea\_dev, tk\_wri\_dev または tk\_swri\_dev が呼び出されたときに execfn を呼び出します。

execfn では、devreq の要求の処理を開始します。処理を開始するだけで完了を待たずに呼出し元へ復帰します。処理開始のためにかかる時間はデバイスドライバに依存し、必ずしも即座に完了するとは限りません。

新たな要求を受け付けられない状態の場合には要求受け待ちとなります。tmout に指定された時間以内に新たな要求を受け付けられないとタイムアウトの処理を行います。tmout には、TMO\_POL(= 0) または TMO\_FEVR(= -1) を指定することもできます。タイムアウトした場合には execfn の復帰値に E\_TMOUT を返します。要求パケットの error は変更しません。タイムアウトの処理を行うのは要求を受け付けるまでで、要求を受け付けた後にタイムアウトは発生しません。

execfn の復帰値にエラーを返した場合には要求が受け付けられなかったものとして要求パケットは消滅します。デバイスドライバ側の理由で本関数の処理を中止する場合には、その要求の受け付け前（処理開始前）であれば execfn の復帰値に E\_ABORT を返します。この場合、要求パケットは消滅します。要求受け付け後（処理開始後）の場合には E\_OK を返します。この場合、要求パケットは waitfn を実行して処理完了を確認するまで消滅しません。



## ● 完了待ち関数 (waitfn)

## &lt; 呼出しインタフェース &gt;

```
INT pktno = waitfn( T_DEVREQ *devreq, INT nreq, TMO tmout, VP exinf)
```

## &lt; パラメータ &gt;

## 【入力】

devreq	要求パケットのリスト
nreq	要求パケットの数
tmout	タイムアウト時間 (ms)
exinf	デバイス登録時に指定された拡張情報

## 【出力】

pktno	完了した要求パケットの番号またはエラーコード
-------	------------------------

## &lt; 解説 &gt;

tk\_srea\_dev, tk\_swri\_dev または tk\_wai\_dev が呼び出されたときに waitfn を呼び出します。

devreq は devreq->next で接続された要求パケットのリストで、devreq から nreq 個分の要求パケットについて、そのうちのいずれかが完了するのを待ちます。リストの最後の next は必ずしも NULL になっているとは限らないため、必ず nreq の指定に従います。復帰値に完了した要求パケットの番号 (devreq から何番目か) を返します。最初が 0 番目で最後が nreq-1 番目となります。

なお、完了とは、正常終了 / 異常 (エラー) 終了 / 中止のいずれかです。

tmout に完了待ちのタイムアウト時間を指定します。TMO\_POL(=0) または TMO\_FEVR (= -1) を指定することもできます。タイムアウトしても要求された処理は継続中です。タイムアウトの場合には waitfn の復帰値に E\_TMOUT を返します。要求パケットの error は変更しません。

なお、要求された処理を継続中で waitfn から復帰するときには、waitfn の復帰値に必ずエラーを返します。復帰値にエラーを返した場合は、必ず処理が継続していなければならず、復帰値にパケット番号を返した場合には必ず処理を完了してください。waitfn の復帰値にエラーが返されている限り、その要求は処理中として要求パケットは消滅しません。waitfn の復帰値に処理を完了した要求パケットの番号が返されたときに、その要求の処理は完了したもとして要求パケットは消滅します。

入出力エラーなどデバイスに関するエラーを要求パケットの error に格納します。waitfn の復帰値には完了待ちが正しくできなかった場合のエラーを返します。waitfn の復帰値が tk\_wai\_dev の復帰値となり、要求パケットの error を ioer に戻します。

デバイスドライバ側の理由や中止要求 (要求パケットの abort フラグ = TRUE) で完了待ちの処理を中止する場合、単一要求の完了待ち (nreq=1) の場合と複数要求の完了待ち (nreq>1) の場合では中止の処理を変えてください。単一要求の完了待ちの場合には処理中の要求を中止します。複数要求の完了待ちの場合には要求の処理自体は中止せずに完了待ちだけを中止 (待ち解除) します。複数要求待ちの中止 (待ち解除) の場合には waitfn の復帰値に E\_ABORT を返します。

また、複数要求の場合は中止処理を無視することもできます。ただし、中止要求に対しては、できる限り中止処理を行うことを推奨します。

なお、中止処理では waitfn からできる限り速やかに復帰することが重要であり、処理を中止しなくてもすぐに処理が終了する場合には中止する必要はありません。処理が中止された場合は、要求パケットの error に E\_ABORT を返すことを原則としますが、そのデバイスの特性に合わせて E\_ABORT 以外のエラーを返しても構いません。また、中止する直前までの処理を有効として E\_OK を返しても構いません。

なお、中止要求があっても正常に最後まで処理した場合には E\_OK を返します。

## ● 中止処理関数 (abortfn)

### < 呼出しインタフェース >

```
ER ercd = abortfn( ID tskid, T_DEVREQ *devreq, INT nreq, VP exinf )
```

### < パラメータ >

#### 【入力】

tskid	execfn, waitfn を実行しているタスクのタスク ID
devreq	要求パケットのリスト
nreq	要求パケットの数
exinf	デバイス登録時に指定された拡張情報

#### 【出力】

ercd	エラーコード
------	--------

### < 解説 >

tk\_cls\_dev が呼び出されたときに abortfn を呼び出します。

abortfn は、指定された要求を実行中の execfn, waitfn を中止させます。通常は、処理中の要求を中止させます。ただし、中止しなくてもすぐに処理が終了する場合には必ずしも中止しなくても構いません。重要なのは、できる限り速やかに execfn, waitfn から復帰することです。

tskid は、devreq で指定された要求を実行中のタスクです。つまり、execfn, waitfn を実行しているタスクです。devreq, nreq は、execfn, waitfn の入力パラメータとして指定されたものと同じです。ただし、execfn の場合は常に nreq=1 です。

abortfn は、execfn, waitfn を実行しているタスクとは別のタスクから呼び出されます。両者は並行して実行するため、必要に応じて排他制御などを行う必要があります。また、execfn, waitfn の呼出し直前や execfn, waitfn から復帰する途中に abortfn を呼び出す可能性もあります。この場合においても正しく動作するように配慮する必要があります。

abortfn を呼び出す前に、中止処理対象の要求パケットの abort フラグに TRUE を設定します。execfn, waitfn はこれにより中止要求の有無を知ることができます。

複数要求待ち (nreq>1) の waitfn 実行中の場合は、特別な扱いとなりほかの場合とは以下の点で異なります。

- ・要求の処理を中止せずに完了待ちのみを中止（待ちを解除）します。
- ・要求パケットの abort フラグはセットしません (abort=FALSE のまま)。

なお、`execfn`、`waitfn` の実行中でないときに要求を中止させる場合には、`abortfn` を呼び出すことなく、要求パケットの `abort` フラグをセットします。`abort` フラグがセットされた状態で `execfn` が呼び出されたときには要求を受け付けません。`waitfn` が呼び出されたときには `abortfn` が呼び出された場合と同様の中止処理を行います。

`execfn` により処理を開始した要求が `waitfn` による完了待ちでない状態で中止された場合には、後で `waitfn` が呼び出されたときに中止されて処理が完了したことを知らせます。処理が中止されても `waitfn` により完了を確認するまでは要求自体は消滅しません。`abortfn` は中止処理を開始するだけで中止が完了するのを待たずに速やかに復帰します。

`abortfn` を以下の場合に呼び出します。

- `tk_cls_dev` およびサブシステムのクリーンアップ処理によるデバイスクローズ時に、クローズするデバイスディスクリプタによる処理中の要求があった場合、そのデバイスディスクリプタによる処理中の要求を中止します。

## ● イベント関数 (`eventfn`)

< 呼出しインタフェース >

```
INT rtncd = eventfn( INT evttyp, VP evtinf, VP exinf )
```

< パラメータ >

### 【入力】

<code>evttyp</code>	ドライバ要求イベントタイプ
<code>evtinf</code>	イベントタイプ別の情報
<code>exinf</code>	デバイス登録時に指定された拡張情報

### 【出力】

<code>rtncd</code>	イベントタイプ別に定義された復帰値またはエラー
--------------------	-------------------------

< 解説 >

ドライバ要求イベントタイプには以下のものがあり、正の値が `tk_evt_dev` の呼出しによるもの、負の値がデバイス管理機能内部からの呼出しによるものです。

イベント	値	意味
TDV_SUSPEND	(-1)	サスペンド
TDV_RESUME	(-2)	リジューム
TDV_CARDEVT	1	PC カードイベント
TDV_USBEVT	2	USB イベント

イベント関数で行う処理をイベントタイプごとに定義します。`tk_evt_dev` による呼出しの場合には `eventfn` の復帰値はそのまま `tk_evt_dev` の復帰値となります。

イベント関数への要求は、ほかの要求の処理中であっても受け付けられます。よって、できる限り速やかに処理してください。

## ■ デバイス事象通知

デバイスドライバは、デバイスで発生した事象を事象通知用メッセージバッファ (TDN\_EVENT) にデバイス事象通知として送信します。

事象タイプには以下のものがあります。

```
typedef enum tdevttyp {
    TDE_unknown = 0,          /* 未定義 */
    TDE_MOUNT = 0x01,         /* メディア挿入 */
    TDE_EJECT = 0x02,         /* メディア排出 */
    TDE_ILLMOUNT = 0x03,      /* メディア不正挿入 */
    TDE_ILLEJECT = 0x04,      /* メディア不正排出 */
    TDE_REMOUNT = 0x05,       /* メディア再挿入 */
    TDE_CARDBATLOW = 0x06,     /* カードバッテリー残量警告 */
    TDE_CARDBATFAIL = 0x07,   /* カードバッテリー異常 */
    TDE_REQEJECT = 0x08,      /* メディア排出要求 */
    TDE_PDBUT = 0x11,         /* PD ボタン状態の変化 */
    TDE_PDMOVE = 0x12,        /* PD 位置移動 */
    TDE_PDSTATE = 0x13,       /* PD の状態変化 */
    TDE_PDEXT = 0x14,         /* PD 拡張事象 */
    TDE_KEYDOWN = 0x21,       /* キーダウン */
    TDE_KEYUP = 0x22,         /* キーアップ */
    TDE_KEYMETA = 0x23,       /* メタキー状態の変化 */
    TDE_POWEROFF = 0x31,      /* 電源スイッチオフ */
    TDE_POWERLOW = 0x32,      /* 電源残量警告 */
    TDE_POWERFAIL = 0x33,     /* 電源異常 */
    TDE_POWERUSUS = 0x34,     /* 自動サスペンド */
    TDE_POWERUPTM = 0x35,     /* 時計更新 */
    TDE_CKPWON = 0x41         /* 自動電源オン通知 */
} TDEvtTyp;
```

デバイス事象通知の形式は以下のようになります。事象通知の内容は事象タイプごとに異なり、サイズも異なります。

```
typedef struct t_devevt {
    TDEvtTyp evttyp;          /* 事象タイプ */
    /* 以下に事象タイプ別の情報を付加する */
} T_DEVEVT;
```

デバイス ID 付きのデバイス事象通知の形式は以下のようになります。

```
typedef struct t_devevt_id {
    TDEvtTyp evttyp;    /* 事象タイプ */
    ID devid;           /* デバイス ID */
    /* 以下に事象タイプ別の情報を付加する */
} T_DEVEVT_ID;
```

事象通知用メッセージバッファが一杯で事象通知を送信できない場合には、その事象が通知されないことで、事象通知の受信側の動作に悪影響が出ないようにしてください。メッセージバッファが空くまで待ってから事象通知を行ってもかまいませんが、その場合も原則として事象通知以外のデバイスドライバの処理が滞らないようにしてください。

なお、事象通知の受信側は、できる限りメッセージバッファが溢れないように処理してください。

## ■ サスペンド/リジューム処理

イベント関数 (eventfn) へのサスペンド / リジューム (TDV\_SUSPEND/TDV\_RESUME) イベントの呼出しにより、各デバイスドライバはデバイスのサスペンド/リジューム処理を行います。サスペンド/リジュームイベントは物理デバイスだけを呼び出します。

### ● サスペンド (TDV\_SUSPEND)

evttyp = TDV\_SUSPEND

evtinf = NULL (なし)

以下の手順でサスペンド処理を行います。

1. 現在処理中の要求がある場合には完了するまで待つか、中断または中止します。どの方法を選択するかはデバイスドライバの実装に依存します。ただし、できるだけ速やかにサスペンドする必要があるため、完了までに時間がかかる場合には中断または中止してください。

サスペンドイベントは物理デバイスだけに発行しますが、そのデバイスに含まれるすべての論理デバイスも同様に処理します。

中断：処理を一時的に中断してリジューム後に続きを行います。

中止：中止処理関数 (abortfn) による中止と同様に処理を中止します。リジューム後も再開しません。

2. リジュームイベント以外の新たな要求を受け付けないようにします。
3. デバイスの電源を切るなどのサスペンド処理を行います。

中止はアプリケーションへの影響が大きいと考えるため、極力避けてください。シリアル回線からの長期の入力待ちなどで、かつ中断とするのが難しい場合以外は中止にせず終了まで待つか、可能な場合には中断します。

サスペンド期間中のデバイスドライバへの要求は、リジュームまで待たせてリジューム後に受付け処理します。ただし、デバイスへのアクセスを伴わない処理などはサスペンド中でも受付け可能です。

● リジューム (TDV\_RESUME)

evttyp = TDV\_RESUME

evtinf = NULL ( なし )

以下の手順でリジューム処理を行います。

1. デバイスの電源を入れてデバイスの状態を復帰させるなどのリジューム処理を行います。
2. 中断していた処理がある場合には再開します。
3. 要求受付けを再開します。

## 付録 D μITRON 仕様 OS からの移行時の注意点

μITRON 仕様 OS と μT-REALOS の相違点について説明します。

### ■ μITRON 仕様 OS との相違点について

μITRON4.0 仕様準拠のリアルタイム OS である SOFTUNE REALOS/FR Spec.4(以降, REALOS/FR Spec.4 とよびます)と μT-REALOS の機能的な違いについて, 表 D-1 に示します。REALOS/FR Spec.4 のユーザプログラムを μT-REALOS に移植する場合には, これらの点について注意してください。

表 D-1 μT-REALOS と REALOS/FR Spec.4 の機能相違点一覧

機能	μT-REALOS	REALOS/FR Spec.4
オブジェクトの静的生成	×	
オブジェクト生成時の ID 番号指定	×	
サブシステム機能		×
デバイス管理機能		×
カーネルでの領域確保 (タスクスタック, メモリプール領域)		×
タスク例外機能	×	
セマフォ資源の複数獲得 / 解放		×
データキュー機能	×	
ランデブポート機能		×
オブジェクト待ちシステムコール* (永久待ち, ポーリング, タイムアウト)	同一システムコール	別システムコール
割込みレベルの変更	×	
イベントフラグのビットクリア	ビット単位および 全ビット	全ビット
割込みハンドラの動的登録		×

○ : 該当機能をサポート    × : 該当機能を未サポート

\*: セマフォ待ちのシステムコールには以下の違いがあります。tk\_wai\_sem では, タイムアウト時間 (tmout) の引数を持ち, tmout の値により待ちの種類を制御します。

表 D-2 セマフォ待ちシステムコールの相違点

	REALOS/FR Spec.4	μT-REALOS
永久待ち	wai_sem()	tk_wai_sem()
ポーリング	pol_sem()	
タイムアウト	twai_sem()	

## 付録 E システムコール索引

システムコールのアルファベット順一覧とその説明の掲載ページを以下に示します。

### ■ システムコール索引

表 E-1 システムコール索引 (1 / 4)

システムコール名	意味	項
DI	外部割込み禁止	182
EI	外部割込み許可	183
isDI	外部割込み禁止状態の獲得	184
isig_tim(Signal Time)	システム時刻の更新	160
tk_acp_por(Accept Port for Rendezvous)	ランデブポートに対するランデブ受付け	122
tk_cal_por(Call Port for Rendezvous)	ランデブポートに対するランデブ呼出し	119
tk_can_wup(Cancel Wakeup Task)	タスク起床要求を無効化	50
tk_chg_pri(Change Task Priority)	タスク優先度変更	34
tk_clr_flg(Clear Event Flag)	イベントフラグのクリア	77
tk_cls_dev(Close Device)	デバイスのクローズ	208
tk_cre_alm(Create Alarm Handler)	アラームハンドラの生成	171
tk_cre_cyc(Create Cyclic Handler)	周期ハンドラの生成	162
tk_cre_flg(Create Event Flag)	イベントフラグの生成	73
tk_cre_mbf(Create Message Buffer)	メッセージバッファの生成	105
tk_cre_mbx(Create Mailbox)	メールボックスの生成	84
tk_cre_mpf(Create Fixed-size Memory Pool)	固定長メモリプールの生成	134
tk_cre_mpl(Create Variable-size Memory Pool)	可変長メモリプールの生成	145
tk_cre_mtx(Create Mutex)	ミューテックスの生成	95
tk_cre_por(Create Port for Rendezvous)	ランデブポートの生成	116
tk_cre_sem(Create Semaphore)	セマフォの生成	63
tk_cre_tsk(Create Task)	タスクの生成	25
tk_def_dev(Define Device)	デバイスの登録	202
tk_def_int(Define Interrupt Handler)	割込みハンドラ定義	179
tk_def_ssy(Define Subsystem)	サブシステム定義	197
tk_del_alm>Delete Alarm Handler)	アラームハンドラの削除	173
tk_del_cyc>Delete Cyclic Handler)	周期ハンドラの削除	165
tk_del_flg>Delete Event Flag)	イベントフラグの削除	75
tk_del_mbf>Delete Message Buffer)	メッセージバッファの削除	108



表 E-1 システムコール索引 (2 / 4)

システムコール名	意味	項
tk_del_mbx(Delete Mailbox)	メールボックスの削除	86
tk_del_mpf(Delete Fixed-size Memory Pool)	固定長メモリプールの削除	137
tk_del_mpl(Delete Variable-size Memory Pool)	可変長メモリプールの削除	148
tk_del_mtx(Delete Mutex)	ミューテックスの削除	97
tk_del_por(Delete Port for Rendezvous)	ランデブポートの削除	118
tk_del_sem(Delete Semaphore)	セマフォの削除	65
tk_del_tsk(Delete Task)	タスクの削除	28
tk_dis_dsp(Disable Dispatch)	ディスパッチ禁止	189
tk_dly_tsk(Delay Task)	タスク遅延	60
tk_ena_dsp(Enable Dispatch)	ディスパッチ許可	191
tk_evt_dev(Event Device)	デバイスにドライバ要求イベントを送信	228
tk_exd_tsk(Exit and Delete Task)	自タスクの終了と削除	31
tk_ext_tsk(Exit Task)	自タスクの終了	30
tk_frsm_tsk(Force Resume Task)	強制待ち状態のタスクを強制再開	58
tk_fwd_por(Forward Rendezvous to Another Port)	ランデブポートに対するランデブ	125
tk_get_dev(Get Device)	デバイス名獲得	221
tk_get_mpf(Get Fixed-size Memory Block)	固定長メモリブロック獲得	138
tk_get_mpl(Get Variable-size Memory Block)	可変長メモリブロック獲得	149
tk_get_otm(Get Operating Time)	システム稼働時間参照	159
tk_get_reg(Get Task Registers)	タスクレジスタの獲得	37
tk_get_tid(Get Task Identifier)	実行状態タスクのタスク ID 参照	188
tk_get_tim(Get Time)	システム時刻参照	158
tk_loc_mtx(Lock Mutex)	ミューテックスのロック	98
tk_lst_dev(List Device)	登録済みデバイス一覧の獲得	226
tk_opn_dev(Open Device)	デバイスのオープン	206
tk_oref_dev(Refer Device)	デバイス情報獲得	224
tk_rcv_mbf(Receive Message from Message Buffer)	メッセージバッファから受信	111
tk_rcv_mbx(Receive Message from Mailbox)	メールボックスから受信	89
tk_rea_dev(Read Device)	デバイスからの読出し開始	209
tk_ref_alm(Refer Alarm Handler Status)	アラームハンドラの状態を参照	176
tk_ref_cyc(Refer Cyclic Handler Status)	周期ハンドラの状態を参照	168
tk_ref_dev(Refer Device)	デバイス情報獲得	222
tk_ref_flg(Refer Event Flag Status)	イベントフラグの状態を参照	81
tk_ref_idv(Refer Initial Device Information)	デバイス初期情報の獲得	205

表 E-1 システムコール索引 (3 / 4)

システムコール名	意味	項
tk_ref_mbf(Refer Message Buffer Status)	メッセージバッファの状態を参照	113
tk_ref_mbx(Refer Mailbox Status)	メールボックスの状態を参照	91
tk_ref_mpf(Refer Fixed-size Memory Pool Status)	固定長メモリプールの状態を参照	142
tk_ref_mpl(Refer Variable-size Memory Pool Status)	可変長メモリプールの状態を参照	153
tk_ref_mtx(Refer Mutex Status)	ミューテックスの状態を参照	102
tk_ref_por(Refer Port Status)	ランデブポートの状態を参照	130
tk_ref_sem(Refer Semaphore Status)	セマフォの状態を参照	70
tk_ref_ssy(Refer Subsystem Status)	サブシステム定義情報を参照	200
tk_ref_sys(Refer System Status)	システムの状態を参照	192
tk_ref_tsk(Refer Task Status)	タスクの状態を参照	41
tk_ref_ver(Refer Version Information)	バージョン情報を参照	194
tk_rel_mpf(Release Fixed-size Memory Block)	固定長メモリブロックを返却	140
tk_rel_mpl(Release Variable-size Memory Block)	可変長メモリブロックを返却	151
tk_rel_wai(Release Wait)	他タスクの待ち状態解除	52
tk_ret_int(Return from Interrupt Handler)	割込みハンドラからの復帰	181
tk_rot_rdq(Rotate Ready Queue)	タスク優先順位の回転	186
tk_rpl_rdv(Reply Rendezvous)	ランデブ返答	128
tk_rsm_tsk(Resume Task)	強制待ち状態のタスクを再開	56
tk_set_flg(Set Event Flag)	イベントフラグのセット	76
tk_set_reg(Set Task Registers)	タスクレジスタの設定	39
tk_set_tim(Set Time)	システム時刻の設定	157
tk_sig_sem(Signal Semaphore)	セマフォ資源の返却	66
tk_slp_tsk(Sleep Task)	自タスクを起床待ち状態へ移行	46
tk_snd_mbf(Send Message to Message Buffer)	メッセージバッファへ送信	109
tk_snd_mbx(Send Message to Mailbox)	メールボックスへ送信	87
tk_srea_dev(Synchronous Read Device)	デバイスの同期読出し	211
tk_sta_alm(Start Alarm Handler)	アラームハンドラの動作開始	174
tk_sta_cyc(Start Cyclic Handler)	周期ハンドラの動作開始	166
tk_sta_tsk (Start Task)	タスクの起動	29
tk_stp_alm(Stop Alarm Handler)	アラームハンドラの動作停止	175
tk_stp_cyc(Stop Cyclic Handler)	周期ハンドラの動作停止	167
tk_sus_dev(Suspend Device)	デバイスのサスペンド	219
tk_sus_tsk(Suspend Task)	他タスクを強制待ち状態へ移行	54
tk_swri_dev(Synchronous Write Device)	デバイスの同期書込み	215

表 E-1 システムコール索引 (4 / 4)

システムコール名	意味	項
tk_ter_tsk(Terminate Task)	他タスクを強制終了	32
tk_unl_mtx(Unlock Mutex)	ミューテックスのアンロック	100
tk_wai_dev(Wait Device)	デバイスの要求完了待ち	217
tk_wai_flg(Wait Event Flag)	イベントフラグ待ち	78
tk_wai_sem(Wait on Semaphore)	セマフォ資源の獲得待ち	68
tk_wri_dev(Write Device)	デバイスの書き込み開始	213
tk_wup_tsk(Wakeup Task)	他タスクの起床	48



## 索引

## A

## Accept

tk\_acp\_por(Accept Port for Rendezvous) ..... 122

## Alarm Handler

tk\_cre\_alm(Create Alarm Handler) ..... 171  
 tk\_del\_alm(Delete Alarm Handler) ..... 173  
 tk\_ref\_alm(Refer Alarm Handler Status) ..... 176  
 tk\_sta\_alm(Start Alarm Handler) ..... 174  
 tk\_stp\_alm(Stop Alarm Handler) ..... 175

## B

## Buffer

tk\_cre\_mbf(Create Message Buffer) ..... 105  
 tk\_del\_mbf(Delete Message Buffer) ..... 108  
 tk\_ref\_mbf(Refer Message Buffer Status) ..... 113  
 tk\_snd\_mbf(Send Message to Message Buffer)  
 ..... 109

## C

## Call

tk\_cal\_por(Call Port for Rendezvous) ..... 119

## Cancel

tk\_can\_wup(Cancel Wakeup Task) ..... 50

## Change

tk\_chg\_pri(Change Task Priority) ..... 34

## Clear

tk\_clr\_flg(Clear Event Flag) ..... 77

## Close

tk\_cls\_dev(Close Device) ..... 208

## Create

tk\_cre\_alm(Create Alarm Handler) ..... 171  
 tk\_cre\_cyc(Create Cyclic Handler) ..... 162  
 tk\_cre\_flg(Create Event Flag) ..... 73  
 tk\_cre\_mbf(Create Message Buffer) ..... 105  
 tk\_cre\_mbx(Create Mailbox) ..... 84  
 tk\_cre\_mpf(Create Fixed-size Memory Pool)  
 ..... 134  
 tk\_cre\_mpl(Create Variable-size Memory Pool)  
 ..... 145  
 tk\_cre\_mtx(Create Mutex) ..... 95  
 tk\_cre\_por(Create Port for Rendezvous) ..... 116  
 tk\_cre\_sem(Create Semaphore) ..... 63  
 tk\_cre\_tsk(Create Task) ..... 25

## Cyclic Handler

tk\_cre\_cyc(Create Cyclic Handler) ..... 162  
 tk\_del\_cyc(Delete Cyclic Handler) ..... 165  
 tk\_ref\_cyc(Refer Cyclic Handler Status) ..... 168  
 tk\_sta\_cyc(Start Cyclic Handler) ..... 166  
 tk\_stp\_cyc(Stop Cyclic Handler) ..... 167

## D

## Define

tk\_def\_dev(Define Device) ..... 202  
 tk\_def\_int(Define Interrupt Handler) ..... 179  
 tk\_def\_ssy(Define Subsystem) ..... 197

## Delay

tk\_dly\_tsk(Delay Task) ..... 60

## Delete

tk\_del\_alm(Delete Alarm Handler) ..... 173  
 tk\_del\_cyc(Delete Cyclic Handler) ..... 165  
 tk\_del\_flg(Delete Event Flag) ..... 75  
 tk\_del\_mbf(Delete Message Buffer) ..... 108  
 tk\_del\_mbx(Delete Mailbox) ..... 86  
 tk\_del\_mpf(Delete Fixed-size Memory Pool)  
 ..... 137  
 tk\_del\_mpl(Delete Variable-size Memory Pool)  
 ..... 148  
 tk\_del\_mtx(Delete Mutex) ..... 97  
 tk\_del\_por(Delete Port for Rendezvous) ..... 118  
 tk\_del\_sem(Delete Semaphore) ..... 65  
 tk\_del\_tsk(Delete Task) ..... 28  
 tk\_exd\_tsk(Exit and Delete Task) ..... 31

## Device

tk\_cls\_dev(Close Device) ..... 208  
 tk\_def\_dev(Define Device) ..... 202  
 tk\_evt\_dev(Event Device) ..... 228  
 tk\_get\_dev(Get Device) ..... 221  
 tk\_lst\_dev(List Device) ..... 226  
 tk\_opn\_dev(Open Device) ..... 206  
 tk\_oref\_dev(Refer Device) ..... 224  
 tk\_rea\_dev(Read Device) ..... 209  
 tk\_ref\_dev(Refer Device) ..... 222  
 tk\_ref\_idv(Refer Initial Device Information) ..... 205  
 tk\_srea\_dev(Synchronous Read Device) ..... 211  
 tk\_sus\_dev(Suspend Device) ..... 219  
 tk\_swri\_dev(Synchronous Write Device) ..... 215  
 tk\_wai\_dev(Wait Device) ..... 217  
 tk\_wri\_dev(Write Device) ..... 213

## DI

DI ..... 182

## Disable

tk\_dis\_dsp(Disable Dispatch) ..... 189

## Dispatch

tk\_dis\_dsp(Disable Dispatch) ..... 189  
 tk\_ena\_dsp(Enable Dispatch) ..... 191

## E

## EI

EI ..... 183

## Enable

tk\_ena\_dsp(Enable Dispatch) ..... 191

## Event

tk\_clr\_flg(Clear Event Flag) ..... 77  
 tk\_cre\_flg(Create Event Flag) ..... 73

tk_del_flg(Delete Event Flag) .....	75
tk_evt_dev(Event Device) .....	228
tk_ref_flg(Refer Event Flag Status) .....	81
tk_set_flg(Set Event Flag) .....	76
tk_wai_flg(Wait Event Flag) .....	78

## Exit

tk_ext_tsk(Exit and Delete Task) .....	31
tk_ext_tsk(Exit Task) .....	30

## F

## Fixed-size

tk_cre_mpf(Create Fixed-size Memory Pool) .....	134
tk_del_mpf(Delete Fixed-size Memory Pool) .....	137
tk_get_mpf(Get Fixed-size Memory Block) .....	138
tk_ref_mpf(Refer Fixed-size Memory Pool Status) .....	142
tk_rel_mpf(Release Fixed-size Memory Block) .....	140

## Flag

tk_clr_flg(Clear Event Flag) .....	77
tk_cre_flg(Create Event Flag) .....	73
tk_del_flg(Delete Event Flag) .....	75
tk_ref_flg(Refer Event Flag Status) .....	81
tk_set_flg(Set Event Flag) .....	76
tk_wai_flg(Wait Event Flag) .....	78

## Force

tk_frm_tsk(Force Resume Task) .....	58
-------------------------------------	----

## Forward

tk_fwd_por(Forward Rendezvous to Another Port) .....	125
---	-----

## G

## Get

tk_get_dev(Get Device) .....	221
tk_get_mpf(Get Fixed-size Memory Block) .....	138
tk_get_mpl(Get Variable-size Memory Block) .....	149
tk_get_otm(Get Operating Time) .....	159
tk_get_reg(Get Task Registers) .....	37
tk_get_tid(Get Task Identifier) .....	188
tk_get_tim(Get Time) .....	158

## H

## Handler

tk_cre_alm(Create Alarm Handler) .....	171
tk_cre_cyc(Create Cyclic Handler) .....	162
tk_def_int(Define Interrupt Handler) .....	179
tk_del_alm(Delete Alarm Handler) .....	173
tk_del_cyc(Delete Cyclic Handler) .....	165
tk_ref_alm(Refer Alarm Handler Status) .....	176
tk_ref_cyc(Refer Cyclic Handler Status) .....	168
tk_ret_int(Return from Interrupt Handler) .....	181
tk_sta_alm(Start Alarm Handler) .....	174
tk_sta_cyc(Start Cyclic Handler) .....	166
tk_stp_alm(Stop Alarm Handler) .....	175
tk_stp_cyc(Stop Cyclic Handler) .....	167

## I

## Identifier

tk_get_tid(Get Task Identifier) .....	188
---------------------------------------	-----

## Information

tk_ref_idv(Refer Initial Device Information) .....	205
tk_ref_ver(Refer Version Information) .....	194

## Initial

tk_ref_idv(Refer Initial Device Information) .....	205
--	-----

## Interrupt

tk_def_int(Define Interrupt Handler) .....	179
tk_ret_int(Return from Interrupt Handler) .....	181

## isDI

isDI .....	184
------------	-----

## isig\_tim

isig_tim(Signal Time) .....	160
-----------------------------	-----

## K

## Kernel

μT-Kernel 固有の意味を持つデータ型の一覧 .....	12
------------------------------------	----

## L

## List

tk_lst_dev(List Device) .....	226
-------------------------------	-----

## Lock

tk_loc_mtx(Lock Mutex) .....	98
------------------------------	----

## M

## Mailbox

tk_cre_mbx(Create Mailbox) .....	84
tk_del_mbx(Delete Mailbox) .....	86
tk_rcv_mbx(Receive Message from Mailbox) .....	89
tk_ref_mbx(Refer Mailbox Status) .....	91
tk_snd_mbx(Send Message to Mailbox) .....	87

## Memory

tk_cre_mpf(Create Fixed-size Memory Pool) .....	134
tk_cre_mpl(Create Variable-size Memory Pool) .....	145
tk_del_mpf(Delete Fixed-size Memory Pool) .....	137
tk_del_mpl(Delete Variable-size Memory Pool) .....	148
tk_get_mpf(Get Fixed-size Memory Block) .....	138
tk_get_mpl(Get Variable-size Memory Block) .....	149
tk_ref_mpf(Refer Fixed-size Memory Pool Status) .....	142
tk_ref_mpl(Refer Variable-size Memory Pool Status) .....	153
tk_rel_mpf(Release Fixed-size Memory Block) .....	140
tk_rel_mpl(Release Variable-size Memory Block) .....	151

## Message

tk_cre_mbf(Create Message Buffer) .....	105
tk_del_mbf(Delete Message Buffer) .....	108
tk_rcv_mbf(Receive Message from Message Buffer) .....	111

tk_rcv_mbx(Receive Message from Mailbox) .....	89
tk_ref_mbf(Refer Message Buffer Status) .....	113
tk_snd_mbf(Send Message to Message Buffer) .....	109
tk_snd_mbx(Send Message to Mailbox) .....	87
<b>Message Buffer</b>	
tk_cre_mbf(Create Message Buffer) .....	105
tk_del_mbf>Delete Message Buffer) .....	108
tk_ref_mbf(Refer Message Buffer Status) .....	113
tk_snd_mbf(Send Message to Message Buffer) .....	109
<b>μITRON 仕様 OS</b>	
μITRON 仕様 OS との相違点について .....	249
<b>μT-Kernel</b>	
μT-Kernel 固有の意味を持つデータ型の一覧 .....	12
<b>μT-REALOS</b>	
μT-REALOS の基本的な用語 .....	2
<b>Mutex</b>	
tk_cre_mtx(Create Mutex) .....	95
tk_del_mtx>Delete Mutex) .....	97
tk_loc_mtx(Lock Mutex) .....	98
tk_ref_mtx(Refer Mutex Status) .....	102
tk_unl_mtx(Unlock Mutex) .....	100
<b>O</b>	
<b>Open</b>	
tk_opn_dev(Open Device) .....	206
<b>Operating</b>	
tk_get_otm(Get Operating Time) .....	159
<b>OS</b>	
μITRON 仕様 OS との相違点について .....	249
<b>P</b>	
<b>Port</b>	
tk_acp_por(Accept Port for Rendezvous) .....	122
tk_cal_por(Call Port for Rendezvous) .....	119
tk_cre_por(Create Port for Rendezvous) .....	116
tk_del_por>Delete Port for Rendezvous) .....	118
tk_ref_por(Refer Port Status) .....	130
<b>Priority</b>	
tk_chg_pri(Change Task Priority) .....	34
<b>Q</b>	
<b>Queue</b>	
tk_rot_rdq(Rotate Ready Queue) .....	186
<b>R</b>	
<b>Read</b>	
tk_rea_dev(Read Device) .....	209
tk_srea_dev(Synchronous Read Device) .....	211
<b>Ready Queue</b>	
tk_rot_rdq(Rotate Ready Queue) .....	186
<b>REALOS</b>	
μT-REALOS の基本的な用語 .....	2

<b>Receive</b>	
tk_rcv_mbf(Receive Message from Message Buffer) .....	111
<b>Receive Message</b>	
tk_rcv_mbx(Receive Message from Mailbox) .....	89
<b>Refer</b>	
tk_oref_dev(Refer Device) .....	224
tk_ref_alm(Refer Alarm Handler Status) .....	176
tk_ref_cyc(Refer Cyclic Handler Status) .....	168
tk_ref_dev(Refer Device) .....	222
tk_ref_flg(Refer Event Flag Status) .....	81
tk_ref_idv(Refer Initial Device Information) .....	205
tk_ref_mbf(Refer Message Buffer Status) .....	113
tk_ref_mbx(Refer Mailbox Status) .....	91
tk_ref_mpf(Refer Fixed-size Memory Pool Status) .....	142
tk_ref_mpl(Refer Variable-size Memory Pool Status) .....	153
tk_ref_mtx(Refer Mutex Status) .....	102
tk_ref_por(Refer Port Status) .....	130
tk_ref_sem(Refer Semaphore Status) .....	70
tk_ref_ssy(Refer Subsystem Status) .....	200
tk_ref_sys(Refer System Status) .....	192
tk_ref_tsk(Refer Task Status) .....	41
tk_ref_ver(Refer Version Information) .....	194
<b>Registers</b>	
tk_get_reg(Get Task Registers) .....	37
tk_set_reg(Set Task Registers) .....	39
<b>Release</b>	
tk_rel_mpf(Release Fixed-size Memory Block) .....	140
tk_rel_mpl(Release Variable-size Memory Block) .....	151
tk_rel_wai(Release Wait) .....	52
<b>Rendezvous</b>	
tk_acp_por(Accept Port for Rendezvous) .....	122
tk_cal_por(Call Port for Rendezvous) .....	119
tk_cre_por(Create Port for Rendezvous) .....	116
tk_del_por>Delete Port for Rendezvous) .....	118
tk_fwd_por(Forward Rendezvous to Another Port) .....	125
tk_rpl_rdv(Reply Rendezvous) .....	128
<b>Reply</b>	
tk_rpl_rdv(Reply Rendezvous) .....	128
<b>Resume</b>	
tk_frm_tsk(Force Resume Task) .....	58
tk_rsm_tsk(Resume Task) .....	56
<b>Return</b>	
tk_ret_int(Return from Interrupt Handler) .....	181
<b>Rotate</b>	
tk_rot_rdq(Rotate Ready Queue) .....	186
<b>S</b>	
<b>Semaphore</b>	
tk_cre_sem(Create Semaphore) .....	63
tk_del_sem>Delete Semaphore) .....	65
tk_ref_sem(Refer Semaphore Status) .....	70
tk_sig_sem(Signal Semaphore) .....	66
tk_wai_sem(Wait on Semaphore) .....	68

## 索引

### Send

tk_snd_mbf(Send Message to Message Buffer)	109
tk_snd_mbx(Send Message to Mailbox)	87

### Set

tk_set_flg(Set Event Flag)	76
tk_set_reg(Set Task Registers)	39
tk_set_tim(Set Time)	157

### Signal

isig_tim(Signal Time)	160
tk_sig_sem(Signal Semaphore)	66

### Sleep

tk_slp_tsk(Sleep Task)	46
------------------------	----

### Start

tk_sta_alm(Start Alarm Handler)	174
tk_sta_cyc(Start Cyclic Handler)	166
tk_sta_tsk(Start Task)	29

### Status

tk_ref_alm(Refer Alarm Handler Status)	176
tk_ref_cyc(Refer Cyclic Handler Status)	168
tk_ref_flg(Refer Event Flag Status)	81
tk_ref_mbf(Refer Message Buffer Status)	113
tk_ref_mbx(Refer Mailbox Status)	91
tk_ref_mtx(Refer Mutex Status)	102
tk_ref_por(Refer Port Status)	130
tk_ref_sem(Refer Semaphore Status)	70
tk_ref_ssy(Refer Subsystem Status)	200
tk_ref_sys(Refer System Status)	192
tk_ref_tsk(Refer Task Status)	41

### Stop

tk_stp_alm(Stop Alarm Handler)	175
tk_stp_cyc(Stop Cyclic Handler)	167

### Subsystem

tk_def_ssy(Define Subsystem)	197
tk_ref_ssy(Refer Subsystem Status)	200

### Suspend

tk_sus_dev(Suspend Device)	219
tk_sus_tsk(Suspend Task)	54

### Synchronous

tk_srea_dev(Synchronous Read Device)	211
tk_swri_dev(Synchronous Write Device)	215

### System

tk_ref_sys(Refer System Status)	192
---------------------------------	-----

## T

### Task

tk_can_wup(Cancel Wakeup Task)	50
tk_chg_pri(Change Task Priority)	34
tk_cre_tsk(Create Task)	25
tk_del_tsk(Delete Task)	28
tk_dly_tsk(Delay Task)	60
tk_exd_tsk(Exit and Delete Task)	31
tk_ext_tsk(Exit Task)	30
tk_frm_tsk(Force Resume Task)	58
tk_get_reg(Get Task Registers)	37
tk_get_tid(Get Task Identifier)	188
tk_ref_tsk(Refer Task Status)	41
tk_rsm_tsk(Resume Task)	56
tk_set_reg(Set Task Registers)	39
tk_slp_tsk(Sleep Task)	46
tk_sta_tsk(Start Task)	29

tk_sus_tsk(Suspend Task)	54
tk_ter_tsk(Terminate Task)	32
tk_wup_tsk(Wakeup Task)	48

### Terminate

tk_ter_tsk(Terminate Task)	32
----------------------------	----

### Time

isig_tim(Signal Time)	160
tk_get_otm(Get Operating Time)	159
tk_get_tim(Get Time)	158
tk_set_tim(Set Time)	157

### tk\_acp\_por

tk_acp_por(Accept Port for Rendezvous)	122
--	-----

### tk\_cal\_por

tk_cal_por(Call Port for Rendezvous)	119
--------------------------------------	-----

### tk\_can\_wup

tk_can_wup(Cancel Wakeup Task)	50
--------------------------------	----

### tk\_chg\_pri

tk_chg_pri(Change Task Priority)	34
----------------------------------	----

### tk\_clr\_flg

tk_clr_flg(Clear Event Flag)	77
------------------------------	----

### tk\_cls\_dev

tk_cls_dev(Close Device)	208
--------------------------	-----

### tk\_cre\_alm

tk_cre_alm(Create Alarm Handler)	171
----------------------------------	-----

### tk\_cre\_cyc

tk_cre_cyc(Create Cyclic Handler)	162
-----------------------------------	-----

### tk\_cre\_flg

tk_cre_flg(Create Event Flag)	73
-------------------------------	----

### tk\_cre\_mbf

tk_cre_mbf(Create Message Buffer)	105
-----------------------------------	-----

### tk\_cre\_mbx

tk_cre_mbx(Create Mailbox)	84
----------------------------	----

### tk\_cre\_mpf

tk_cre_mpf(Create Fixed-size Memory Pool)	134
---	-----

### tk\_cre\_mpl

tk_cre_mpl(Create Variable-size Memory Pool)	145
--	-----

### tk\_cre\_mtx

tk_cre_mtx(Create Mutex)	95
--------------------------	----

### tk\_cre\_por

tk_cre_por(Create Port for Rendezvous)	116
--	-----

### tk\_cre\_sem

tk_cre_sem(Create Semaphore)	63
------------------------------	----

### tk\_cre\_tsk

tk_cre_tsk(Create Task)	25
-------------------------	----

### tk\_def\_dev

tk_def_dev(Define Device)	202
---------------------------	-----

### tk\_def\_int

tk_def_int(Define Interrupt Handler)	179
--------------------------------------	-----

### tk\_def\_ssy

tk_def_ssy(Define Subsystem)	197
------------------------------	-----

### tk\_del\_alm

tk_del_alm(Delete Alarm Handler)	173
----------------------------------	-----

### tk\_del\_cyc

tk_del_cyc(Delete Cyclic Handler)	165
-----------------------------------	-----

### tk\_del\_flg

tk_del_flg(Delete Event Flag)	75
-------------------------------	----

### tk\_del\_mbf

tk_del_mbf(Delete Message Buffer)	108
-----------------------------------	-----

### tk\_del\_mbx

tk_del_mbx(Delete Mailbox)	86
----------------------------	----



tk_del_mpf		
tk_del_mpf(Delete Fixed-size Memory Pool)	137	
tk_del_mpl		
tk_del_mpl(Delete Variable-size Memory Pool)	148	
tk_del_mtx		
tk_del_mtx(Delete Mutex)	97	
tk_del_por		
tk_del_por(Delete Port for Rendezvous)	118	
tk_del_sem		
tk_del_sem(Delete Semaphore)	65	
tk_del_tsk		
tk_del_tsk(Delete Task)	28	
tk_dis_dsp		
tk_dis_dsp(Disable Dispatch)	189	
tk_dly_tsk		
tk_dly_tsk(Delay Task)	60	
tk_ena_dsp		
tk_ena_dsp(Enable Dispatch)	191	
tk_evt_dev		
tk_evt_dev(Event Device)	228	
tk_exd_tsk		
tk_exd_tsk(Exit and Delete Task)	31	
tk_ext_tsk		
tk_ext_tsk(Exit Task)	30	
tk_frsm_tsk		
tk_frsm_tsk(Force Resume Task)	58	
tk_fwd_por		
tk_fwd_por(Forward Rendezvous to Another Port)	125	
tk_get_dev		
tk_get_dev(Get Device)	221	
tk_get_mpf		
tk_get_mpf(Get Fixed-size Memory Block)	138	
tk_get_mpl		
tk_get_mpl(Get Variable-size Memory Block)	149	
tk_get_otm		
tk_get_otm(Get Operating Time)	159	
tk_get_reg		
tk_get_reg(Get Task Registers)	37	
tk_get_tid		
tk_get_tid(Get Task Identifier)	188	
tk_get_tim		
tk_get_tim(Get Time)	158	
tk_loc_mtx		
tk_loc_mtx(Lock Mutex)	98	
tk_lst_dev		
tk_lst_dev(List Device)	226	
tk_opn_dev		
tk_opn_dev(Open Device)	206	
tk_oref_dev		
tk_oref_dev(Refer Device)	224	
tk_rcv_mbf		
tk_rcv_mbf(Receive Message from Message Buffer)	111	
tk_rcv_mbx		
tk_rcv_mbx(Receive Message from Mailbox)	89	
tk_rea_dev		
tk_rea_dev(Read Device)	209	
tk_ref_alm		
tk_ref_alm(Refer Alarm Handler Status)	176	
tk_ref_cyc		
tk_ref_cyc(Refer Cyclic Handler Status)	168	
tk_ref_dev		
tk_ref_dev(Refer Device)	222	
tk_ref_flg		
tk_ref_flg(Refer Event Flag Status)	81	
tk_ref_idv		
tk_ref_idv(Refer Initial Device Information)	205	
tk_ref_mbf		
tk_ref_mbf(Refer Message Buffer Status)	113	
tk_ref_mbx		
tk_ref_mbx(Refer Mailbox Status)	91	
tk_ref_mpf		
tk_ref_mpf(Refer Fixed-size Memory Pool Status)	142	
tk_ref_mpl		
tk_ref_mpl(Refer Variable-size Memory Pool Status)	153	
tk_ref_mtx		
tk_ref_mtx(Refer Mutex Status)	102	
tk_ref_por		
tk_ref_por(Refer Port Status)	130	
tk_ref_sem		
tk_ref_sem(Refer Semaphore Status)	70	
tk_ref_ssy		
tk_ref_ssy(Refer Subsystem Status)	200	
tk_ref_sys		
tk_ref_sys(Refer System Status)	192	
tk_ref_tsk		
tk_ref_tsk(Refer Task Status)	41	
tk_ref_ver		
tk_ref_ver(Refer Version Information)	194	
tk_rel_mpf		
tk_rel_mpf(Release Fixed-size Memory Block)	140	
tk_rel_mpl		
tk_rel_mpl(Release Variable-size Memory Block)	151	
tk_rel_wai		
tk_rel_wai(Release Wait)	52	
tk_ret_int		
tk_ret_int(Return from Interrupt Handler)	181	
tk_rot_rdq		
tk_rot_rdq(Rotate Ready Queue)	186	
tk_rpl_rdv		
tk_rpl_rdv(Reply Rendezvous)	128	
tk_rsm_tsk		
tk_rsm_tsk(Resume Task)	56	
tk_set_flg		
tk_set_flg(Set Event Flag)	76	
tk_set_reg		
tk_set_reg(Set Task Registers)	39	
tk_set_tim		
tk_set_tim(Set Time)	157	
tk_sig_sem		
tk_sig_sem(Signal Semaphore)	66	
tk_slp_tsk		
tk_slp_tsk(Sleep Task)	46	
tk_snd_mbf		
tk_snd_mbf(Send Message to Message Buffer)	109	
tk_snd_mbx		
tk_snd_mbx(Send Message to Mailbox)	87	

## 索引

tk_srea_dev	
tk_srea_dev(Synchronous Read Device) .....	211
tk_sta_alm	
tk_sta_alm(Start Alarm Handler) .....	174
tk_sta_cyc	
tk_sta_cyc(Start Cyclic Handler) .....	166
tk_sta_tsk	
tk_sta_tsk(Start Task) .....	29
tk_stp_alm	
tk_stp_alm(Stop Alarm Handler) .....	175
tk_stp_cyc	
tk_stp_cyc(Stop Cyclic Handler) .....	167
tk_sus_dev	
tk_sus_dev(Suspend Device) .....	219
tk_sus_tsk	
tk_sus_tsk(Suspend Task) .....	54
tk_swri_dev	
tk_swri_dev(Synchronous Write Device) .....	215
tk_ter_tsk	
tk_ter_tsk(Terminate Task) .....	32
tk_unl_mtx	
tk_unl_mtx(Unlock Mutex) .....	100
tk_wai_dev	
tk_wai_dev(Wait Device) .....	217
tk_wai_flg	
tk_wai_flg(Wait Event Flag) .....	78
tk_wai_sem	
tk_wai_sem(Wait on Semaphore) .....	68
tk_wri_dev	
tk_wri_dev(Write Device) .....	213
tk_wup_tsk	
tk_wup_tsk(Wakeup Task) .....	48

## U

### Unlock

tk_unl_mtx(Unlock Mutex) .....	100
--------------------------------	-----

## V

### Variable

tk_cre_mpl(Create Variable-size Memory Pool)	
.....	145
tk_del_mpl>Delete Variable-size Memory Pool)	
.....	148
tk_get_mpl(Get Variable-size Memory Block)	
.....	149
tk_ref_mpl(Refer Variable-size Memory Pool Status)	
.....	153
tk_rel_mpl(Release Variable-size Memory Block)	
.....	151

### Variable-size

tk_cre_mpl(Create Variable-size Memory Pool)	
.....	145
tk_del_mpl>Delete Variable-size Memory Pool)	
.....	148
tk_get_mpl(Get Variable-size Memory Block)	
.....	149
tk_ref_mpl(Refer Variable-size Memory Pool Status)	
.....	153
tk_rel_mpl(Release Variable-size Memory Block)	
.....	151

### Version

tk_ref_ver(Refer Version Information) .....	194
---	-----

## W

### Wait

tk_rel_wai(Release Wait) .....	52
tk_wai_dev(Wait Device) .....	217
tk_wai_flg(Wait Event Flag) .....	78
tk_wai_sem(Wait on Semaphore) .....	68

### Wakeup

tk_can_wup(Cancel Wakeup Task) .....	50
tk_wup_tsk(Wakeup Task) .....	48

### Write

tk_swri_dev(Synchronous Write Device) .....	215
tk_wri_dev(Write Device) .....	213

## あ

アラームハンドラ機能	
アラームハンドラ機能のシステムコール	170

## い

イベントフラグ機能	
イベントフラグ機能のシステムコール	72
インタフェース	
デバイスドライバインタフェース	235

## え

エラーコード	
エラーコード	4
エラーコード一覧	230

## か

拡張仕様	
拡張仕様	8
拡張同期	
拡張同期・通信機能のシステムコール	93
可変長メモリプール機能	
可変長メモリプール機能のシステムコール	144
関数	
デバイス処理関数	240

## き

基本的な用語	
μT-REALOS の基本的な用語	2

## こ

固定長メモリプール機能	
固定長メモリプール機能のシステムコール	133

## さ

索引	
システムコール索引	250
サスペンド	
サスペンド / リジューム処理	247
サブシステム機能	
サブシステム機能のシステムコール	196

## し

時間管理機能	
時間管理機能のシステムコール	155
事象通知	
デバイス事象通知	246
システムコール	
アラームハンドラ機能のシステムコール	170

イベントフラグ機能のシステムコール	72
拡張同期・通信機能のシステムコール	93
可変長メモリプール機能のシステムコール	144

固定長メモリプール機能のシステムコール	133
---------------------	-----

サブシステム機能のシステムコール	196
時間管理機能のシステムコール	155
システムコール	4
システムコール一覧	16
システムコール索引	250
システムコール説明内容	22
システム時刻管理機能のシステムコール	156

システム状態管理機能のシステムコール	185
--------------------	-----

周期ハンドラ機能のシステムコール	161
セマフォ機能のシステムコール	62
タスク管理機能のシステムコール	24
タスク付属同期機能のシステムコール	45
デバイス管理機能のシステムコール	201
同期・通信機能のシステムコール	61
ミュutex機能のシステムコール	94
メールボックス機能のシステムコール	83
メッセージバッファ機能のシステムコール	104

メモリプール管理機能のシステムコール	132
ランデブポート機能のシステムコール	115
割込み管理機能のシステムコール	178

システム時刻管理機能	
システム時刻管理機能のシステムコール	156

システム状態管理機能	
システム状態管理機能のシステムコール	185

実装依存	
実装依存	7

実装定義	
実装定義	5

周期ハンドラ機能	
周期ハンドラ機能のシステムコール	161

## せ

セマフォ機能	
セマフォ機能のシステムコール	62

## そ

属性	
属性データ	237

## た

タスク管理機能	
タスク管理機能のシステムコール	24
タスク付属同期機能	
タスク付属同期機能のシステムコール	45

## つ

通信	
同期・通信機能のシステムコール	61

## 索引

### 通信機能

拡張同期・通信機能のシステムコール .....93

## て

### 定数マクロ

定数マクロ一覧 .....231

### データ型

μT-Kernel 固有の意味を持つデータ型の一覧  
.....12

データ型 .....4

汎用的なデータ型 .....10

### データ形式

入出力要求のデータ形式 .....239

### データ番号

データ番号 .....237

### デバイス

デバイス ID .....236

デバイス事象通知 .....246

デバイス処理関数 .....240

デバイス属性 .....236

デバイス名の命名規則 .....235

### デバイス ID

デバイス ID .....236

### デバイス管理機能

デバイス管理機能のシステムコール .....201

### デバイス処理関数

デバイス処理関数 .....240

### デバイス属性

デバイス属性 .....236

### デバイスディスクリプタ

デバイスディスクリプタ .....237

### デバイスドライバインタフェース

デバイスドライバインタフェース .....235

### デバイス名

デバイス名の命名規則 .....235

## と

### 同期

同期・通信機能のシステムコール .....61

## に

### 入出力要求

入出力要求のデータ形式 .....239

## は

### 汎用的なデータ型

汎用的なデータ型 .....10

## ま

### マクロ

定数マクロ一覧 .....231

## み

### ミューテックス機能

ミューテックス機能のシステムコール .....94

## め

### 命名規則

デバイス名の命名規則 .....235

### メールボックス機能

メールボックス機能のシステムコール .....83

### メッセージバッファ機能

メッセージバッファ機能のシステムコール  
.....104

### メモリプール管理機能

メモリプール管理機能のシステムコール .....132

## ら

### ランデブポート機能

ランデブポート機能のシステムコール .....115

## り

### リクエスト

リクエスト ID .....237

### リクエスト ID

リクエスト ID .....237

### リジューム

サスペンド / リジューム処理 .....247

## わ

### 割込み管理機能

割込み管理機能のシステムコール .....178

CM81-00321-1

---

**富士通マイクロエレクトロニクス・CONTROLLER MANUAL**

FR ファミリ

μT-Kernel 仕様準拠

SOFTUNE™ μT-REALOS/FR

API リファレンス

---

2008 年 6 月 初版発行

---

発行	<b>富士通マイクロエレクトロニクス株式会社</b>
編集	マーケティング統括部    ビジネス推進部

---

