



【土曜講座】

組込みリアルタイム OS 入門

(μ T-Kernel 入門:協カルネサスエレクトロニクス)

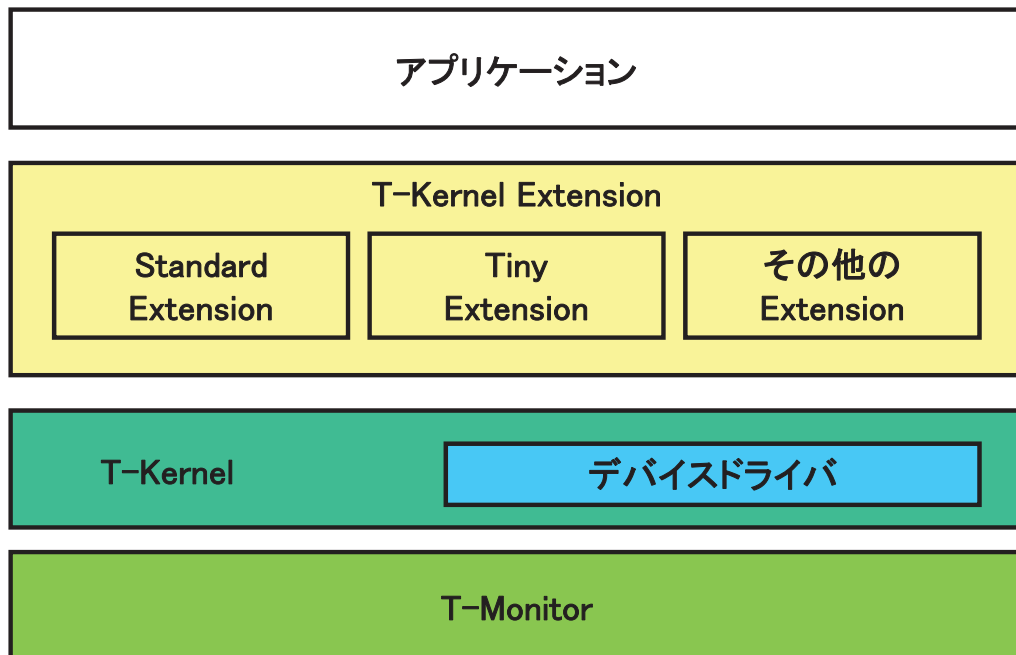
μ T-Kernel 入門



T-Engine/T-Kernelの概要



T-Engineのソフトウェア構成(その1)



T-Engineのソフトウェア構成(その2)

■ T-Monitor

ハードウェアの初期化やシステムの起動、例外・割込みのハンドリング、基本的なデバッグ機能を提供。(PC等のBIOSに相当する)

■ T-Kernel

μITRONの技術を継承したシステムの中心となるリアルタイムOS。

■ デバイスドライバ

標準化されたインタフェースによりハードウェアの制御を行うプログラム。

■ T-Kernel Extension

T-Kernelの機能を拡張し、より高度なOSの機能を実現するプログラム。

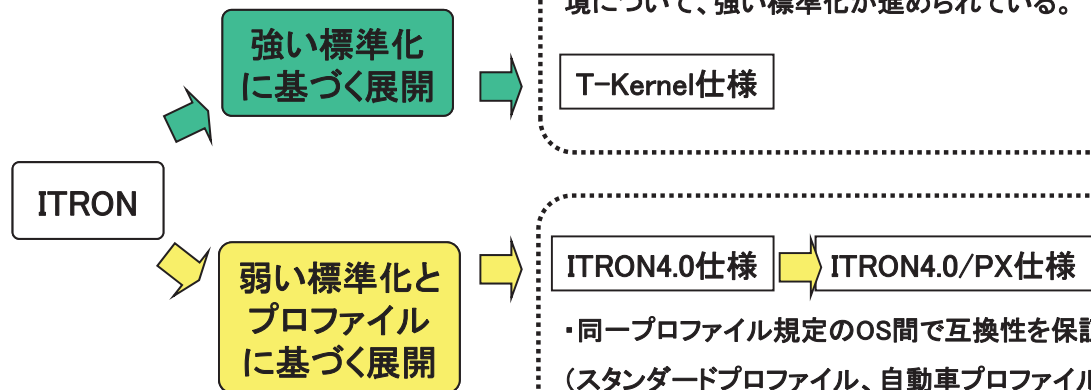
ファイルシステムやネットワークプロトコル等の機能を提供する。

標準仕様のT-Kernel Standard Extension、
比較的小規模な組み込み機器向けのT-Kernel Tiny Extension等が存在する。



T-Kernelの概要

ITRONの成果を生かしつつ、標準化の範囲を拡大し、高度な技術を取り入れることで、短期間で高度な組み込みシステムを作るためのソリューションの整備。



T-Engineプロジェクト

ユビキタスコンピューティング環境構築を目指して、オープンなリアルタイム標準開発環境を整備するプロジェクト。ハードウェア、OS、ミドルウェア、開発環境について、強い標準化が進められている。

T-Kernel仕様

ITRONの成果をベースとして、互換性や厳密性の向上を図り、生産性向上や再利用性、移植性を高めることを目的に仕様を整理。

ITRON4.0仕様

ITRON4.0/PX仕様

- ・同一プロファイル規定のOS間で互換性を保証
(スタンダードプロファイル、自動車プロファイル)
- ・保護機能拡張(ITRON4.0/PX仕様)によるMMUサポート(メモリ保護を主目的としている)



T-Kernelの特徴

■リアルタイム、マルチタスク、コンパクト

この点はITRONもT-Kernelも同じ。

各システムコールはITRON仕様と同等。

→ ITRONに慣れた技術者なら、直ぐにプログラミング可能!!

■ミドルウェア流通に向けた機能をサポート

オブジェクトIDの自動割り当て(動的なシステムの資源管理)

MMU対応(論理空間、メモリ保護)

デバイス管理等のAPIも標準化

※流通実現のため、他にも様々な取組みがなされている

■トロンフォーラムから無償ソース公開

オープンソースである(<http://www.tron.org/ja/>)

無償で最終製品に組込んで良い



T-Kernelの構成

■T-Kernel/OS (Operating System) ⇒ μ T-Kernelはこの部分を実装

ITRONの機能に相当

- タスク管理・付属同期機能、同期・通信機能、メモリ管理機能、
- 例外/割込み制御機能、時間管理機能、サブシステム管理機能

■T-Kernel/SM (System Manager)

- システムメモリ管理機能、アドレス空間管理機能、デバイス管理機能
- 割込管理機能、I/Oポートアクセスサポート機能、省電力機能
- システム構成管理機能

■T-Kernel/DS (Debugger Support)

- カーネル内部状態参照機能、実行トレース機能



T-Kernelのオブジェクト

特徴:カーネルオブジェクトは全てIDが動的に自動割当てされる(μ ITRON仕様との相違点)
メモリプール等のメモリ領域も全てカーネルによる自動割当てとなる

T-Kernelのカーネルオブジェクト一覧

分類	カーネルオブジェクト
タスク関連	タスク
同期・通信関連	セマフォ イベントフラグ メールボックス
拡張同期・通信関連	ミューテックス メッセージバッファ ランデブポート
メモリプール管理関連	固定長メモリプール 可変長メモリプール
時間管理関連	周期ハンドラ アラームハンドラ



T-Kernel/OSの特徴

■T-Kernel/OSとITRONの相違

- T-Kernel/OSは μ ITRON4.0仕様(3.0仕様)と**ほぼ同等の機能**を有している。

- ー ただし、システムコールレベルの細部仕様は異なる
- ー 互換性を保証している訳ではない

例:T-Kernel/OSの持つシステムコールは**先頭に「tk_」が付いた名称**となる

- ・ μ ITRON4.0仕様のsta_tskサービスコール
ER ercd = sta_tsk(ID tskid, VP_INT stacd);
- ・ T-Kernel/OSのsta_tskシステムコール
ER ercd = tk_sta_tsk(ID tskid, INT stacd);

- T-Kernel/OSにない μ ITRON4.0仕様の機能

- ー データキュー、オーバランハンドラ



T-Kernel/OSのデータ型

■システムコールで使用するデータ型

- T-Kernel/OSは μ ITRON仕様と同じく専用のデータ型を使用する。

- ー 内容的には μ ITRON仕様とほぼ同等
- ー ヘッダファイル「<tk/tkernel.h>」をインクルードすることで使用可能

例:

```
typedef char    B;      /* Signed 8 bit integer */
typedef short   H;      /* Signed 16 bit integer */
typedef int     W;      /* Signed 32 bit integer */

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

- その他に各CPU毎の共通基本データ型、T-Monitor用のヘッダファイルがある

- ー 各CPU毎の共通基本データ型のヘッダファイル → 「<basic.h>」
- ー T-Monitor用のヘッダファイル → 「<tk/tmonitor.h>」



オブジェクトIDは全て自動割り付け

■T-Kernel/OSではオブジェクトは全てカーネルによる自動割り付け

- μ ITRON4.0仕様における「**cre_*****」のサービスコールは廃止。
- T-Kernel/OSでは「**tk_cre_*****」が μ ITRON4.0仕様の「**acre_*****」に相当する。
 - tk_cre_tsk、tk_cre_sem、tk_cre_flg、tk_cre_mbx、tk_cre_mtx、tk_cre_mbf、tk_cre_por、tk_cre_mpf、tk_cre_mpl、tk_cre_cyc、tk_cre_alm、tk_cre_res
- T-Kernel/OSでは**静的APIはサポートしない**。

例:T-Kernel/OSではタスクの生成をtk_cre_tskシステムコールで行う

- ・ μ ITRON4.0仕様のcre_tsk、acre_tskサービスコール


```
ER ercd = cre_tsk( ID tskid, T_CTSK *pk_ctsk );
ER_ID tskid = acre_tsk( T_CTSK *pk_ctsk );
```
- ・ T-Kernel/OSのtk_cre_tskシステムコール


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



タイムアウト、ポーリングはWAITに集約

■T-Kernel/OSでは待ちを伴うシステムコールは一本化

- μ ITRON4.0仕様における「**t***_*****」のタイムアウト機能のサービスコールは廃止。
- μ ITRON4.0仕様における「**p***_*****」のポーリング機能のサービスコールも廃止。
- T-Kernel/OSでは**待ちを伴うシステムコールが上記の機能を兼ねる**。
 - tk_slp_tsk、tk_wai_tev、tk_wai_sem、tk_wai_flg、tk_rcv_mbx、tk_loc_mtx、tk_snd_mbf、tk_rcv_mbf、tk_act_por、tk_get_mpf、tk_get_mpl
- タイムアウト指定の**TMO_FEVR(永久待ち)**や**TMO_POL(ポーリング)**で待ち方を指定。

例:T-Kernel/OSではメールボックスからの受信は全てtk_rcv_mbxシステムコールで行う

- ・ μ ITRON4.0仕様のrcv_mbxサービスコール


```
ER recd = rcv_mbx( ID mbxid, T_MSG **ppk_msg );
ER recd = trcv_mbx( ID mbxid, T_MSG **ppk_msg, TMO tmout );
ER recd = prcv_mbx( ID mbxid, T_MSG **ppk_msg );
```
- ・ T-Kernel/OSのtk_rcv_mbxシステムコール

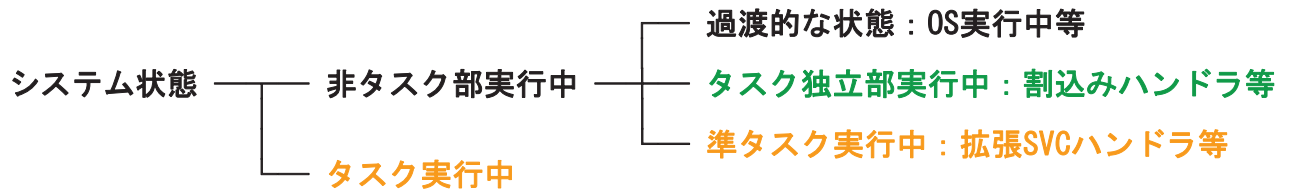

```
ER recd = tk_rcv_mbx( ID mbxid, T_MSG **ppk_msg, TMO tmout );
```



システム状態

■T-Kernel/OSにおけるシステム状態

- T-Kernel/OSにおけるシステム状態は μ ITRON3.0仕様と同等。



- タスク独立部実行中は遅延ディスパッチ
待ち状態に入るシステムコールは発行できない(コンテキストエラー:E_CTX)
- 準タスク実行中、タスク実行中はタスクの優先度に基づきスケジューリング



T-Kernel/OSの機能

- タスク管理機能
- タスク付属同期機能
- タスク例外処理機能
- 同期・通信機能
 - セマフォ、イベントフラグ、メールボックス
- 拡張同期・通信機能
 - メッセージバッファ、ミューテックス、ランデブポート
- メモリプール機能
 - 固定長メモリプール、可変長メモリプール
- 時間管理機能
- 割込み管理機能
- システム状態管理機能
- サブシステム管理機能



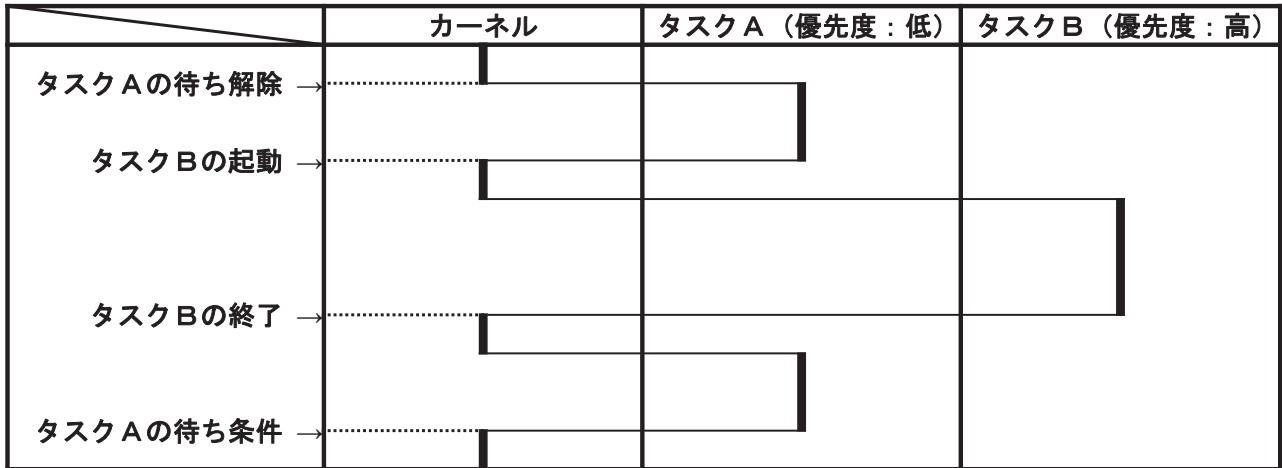
タスクのスケジューリング規則

タスク管理とタスク付属同期



タスクのスケジューリング規則

- T-Kernel仕様におけるタスクのスケジューリング規則は **μITRON仕様と同じ**。
- 基本は優先度方式、同一優先度はFCFS (First Come First Served) 方式。
優先度の高いタスクが先に実行されるのが特徴である。



タスクのスケジューリングを理解するためには
タスクの状態を先に理解する必要がある



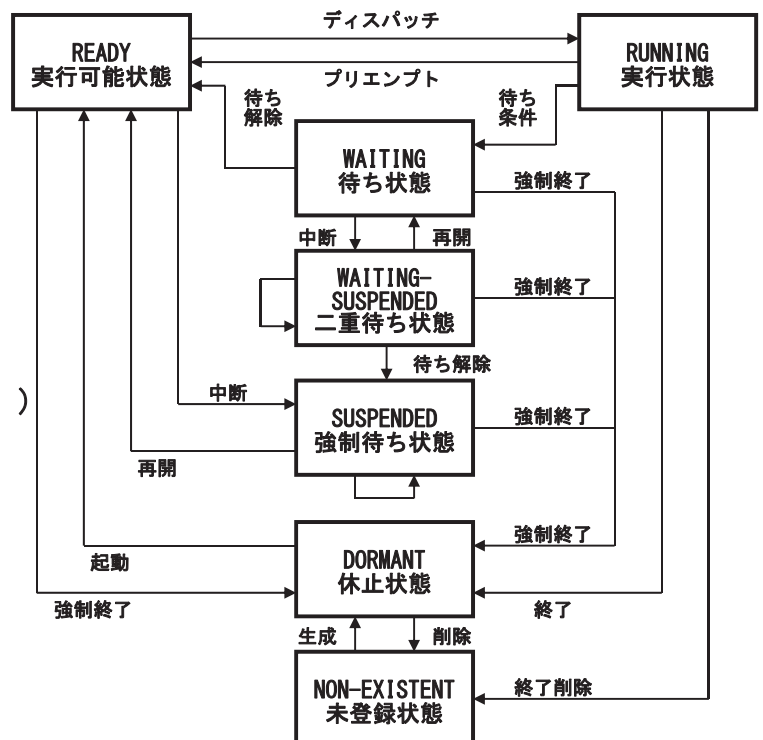
タスクの状態(その1)

■T-Kernel仕様におけるタスクの状態

- T-Kernel仕様におけるタスクの状態は **μITRON4.0仕様と同じ**。

- 実行状態 (RUNNING)
- 実行可能状態 (READY)
- 広義の待ち状態
 - 待ち状態 (WAITING)
 - 強制待ち状態 (SUSPENDED)
 - 二重待ち状態 (WAITING-SUSPENDED)
- 休止状態 (DORMANT)
- 未登録状態 (NON-EXISTENT)

全ての状態を覚える必要はない
先にタスクの基本状態を覚えよう！





タスクの状態(その2)

■ タスクの基本状態

● 実行状態 (RUNNING)

- 与えられた処理を実行している状態

● 実行可能状態 (READY)

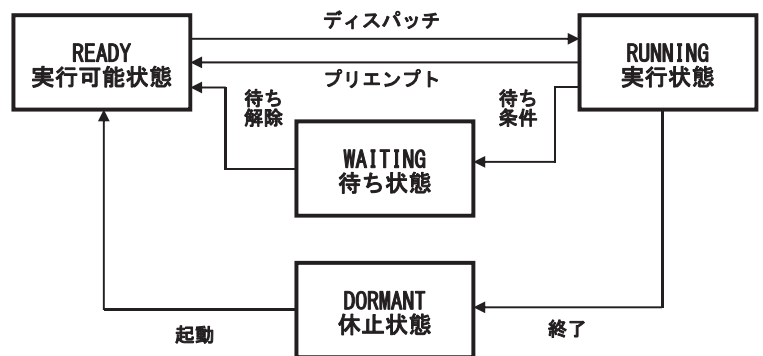
- 実行する準備は整っているものの、自身のタスクより優先度の高いタスクがRUNNING状態となっているため、実行されないでいる状態

● 待ち状態 (WAITING)

- 自身の動作すべき要因が発生するのを待っている状態
- 途中で中断している状態

● 休止状態 (DORMANT)

- 自身が起動されるのを待っている状態
- 処理が終了している状態



この4つの状態を覚えれば
殆どのシステムは作成できる！！



スケジューリングとの対応

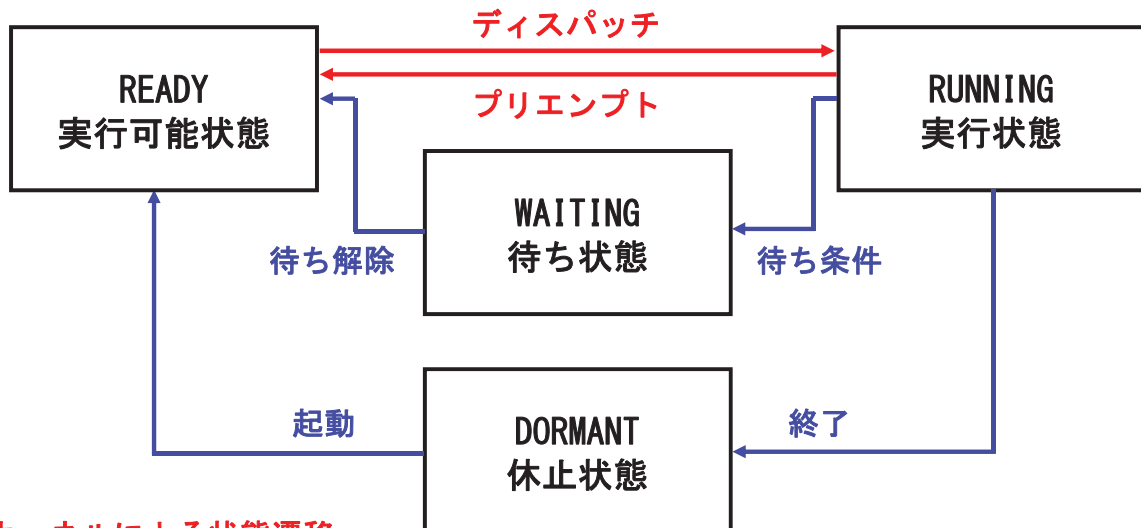


タスクの状態を変化させるのが
システムコール (サービスコール) である
T-Kernel 仕様 μ ITRON4.0 仕様

タスクは4つの状態を遷移しながら動作する



状態遷移の要因



■ カーネルによる状態遷移

- READY状態とRUNNING状態間の状態遷移
- ディスパッチとプリエンプトはタスクのスケジューリング規則に従いカーネルが実施

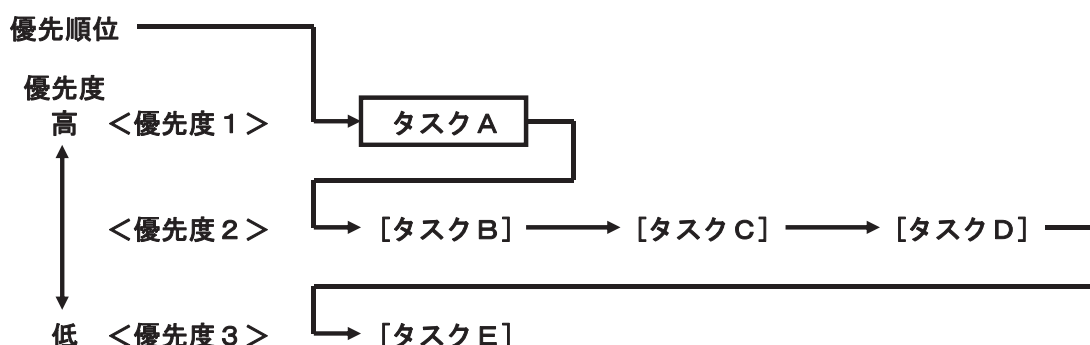
■ プログラマの意思による状態遷移

- WAITING状態やDORMANT状態からREADY状態への遷移
- RUNNING状態からWAITING状態やDORMANT状態への遷移
- システムコールの発行により、プログラマが制御



レディキュー

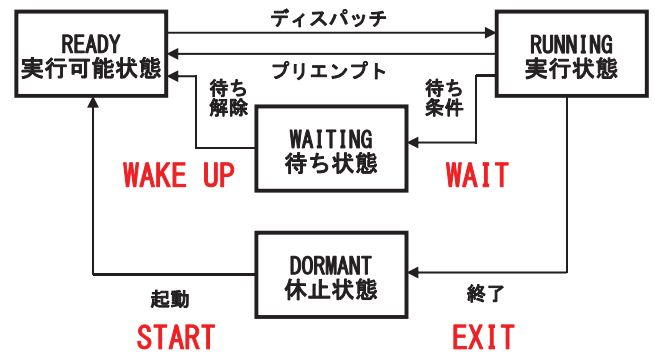
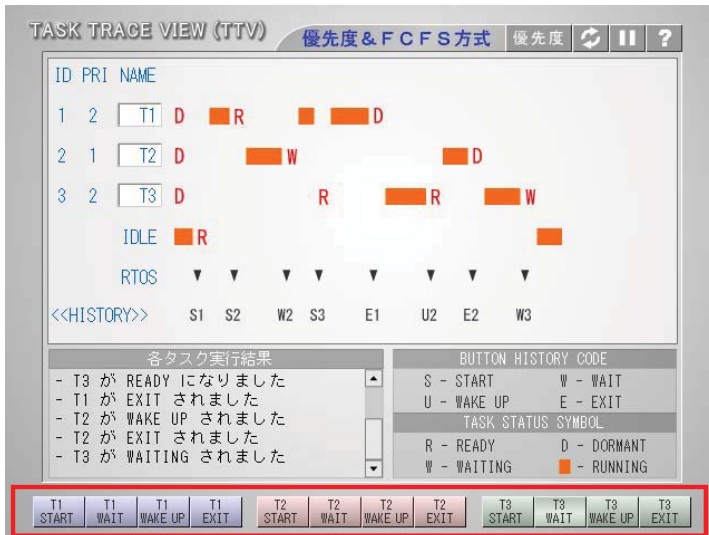
- カーネルがREADY状態の中からRUNNING状態とするタスクを検索するためのキュー。
- WAITING状態やDORMANT状態からREADY状態になるとレディキューに登録される。
RUNNING状態からWAITING状態やDORMANT状態になるとレディキューから外される。
- 優先度が一番高く、その中で最も先にREADY状態となったタスクがRUNNING状態となる。
これが優先度方式とFCFS方式の混在型であるタスクのスケジューリング方式。



TTV (Task Trace View) でタスクのスケジューリング規則を確認しよう！
TTVはタスクのスケジューリングをシミュレーションするツール



TTV(Task Trace View)



これらのボタンをシステムコールと考えて、タスクの状態を変更させる



TTV(Task Trace View)



スケジューリング方式は**優先度&FCFS方式**を選択

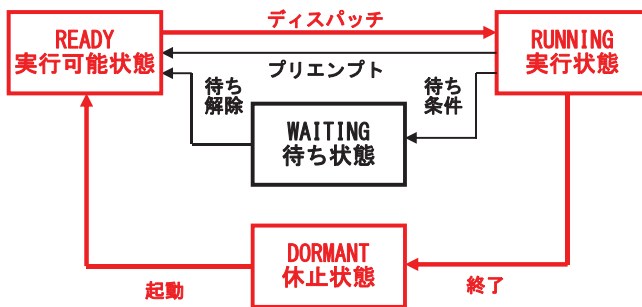


各タスクの**優先度**は任意に決定して良い。
ただし、**値の小さい方**が高い**優先度**となる。



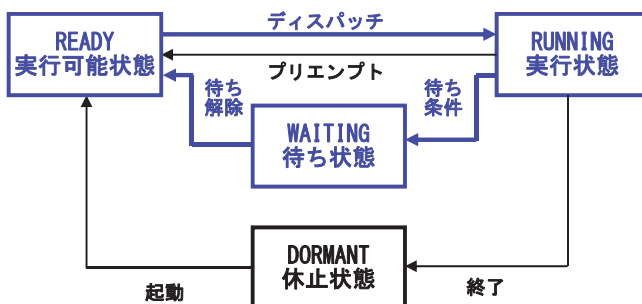
タスクの基本構造

- タスクは動作する必要がなければレディキューから外れる必要がある。
優先度の高いタスクがRUNNING状態である限り、優先度の低いタスクは動作しない。



- DORMANT状態を利用したタスクの構造

```
void task(INT stacd, VP exinf)
{
    // タスクに与えられた処理
    tk_ext_tsk();
}
```



- WAITING状態を利用したタスクの構造

```
void task(INT stacd, VP exinf)
{
    while(1) {
        待ちを伴うシステムコール
        // タスクに与えられた処理
    }
}
```



タスクの生成例

```
typedef enum {
    TSK_A, TSK_KIND_NUM
} T_TSK_KIND;
ID gTskid[TSK_KIND_NUM];
// タスクID番号格納用変数
```

```
void create(void)
{
    T_GTSK ctsk = { 0 };
    ID tskid;

    ctsk.exinf = 0;
    ctsk.tskatr = TA_HLNG | TA_RNGO;
    ctsk.task = (FP) tsk_a;
    ctsk.itstkpri = 10;
    ctsk.stksz = 1024;

    tskid = tk_cre_tsk(&ctsk);
    if( tskid > E_OK )
        gTskid[TSK_A] = tskid;
}
```

- 生成するタスクの記述例

```
void tsk_a(INT stacd, VP exinf)
{
    // タスクに与えられた処理
    tk_ext_tsk(); // または tk_exd_tsk();
}
```

// タスク生成用のパラメータパケット
// タスクID用の変数

// 拡張情報（タスクの第2引数）の指定は自由
// 高級言語、保護レベル0を指定
// タスクの起動アドレス
// タスクの優先度
// ユーザスタックサイズの指定

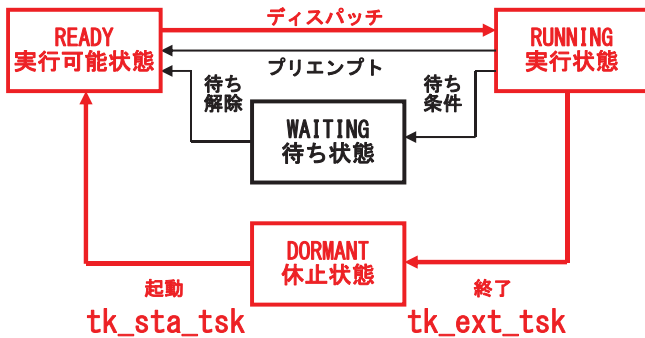
// tk_cre_tskシステムコールにより生成
// エラーコードのチェック
// 正常終了なら、ID番号格納用変数に代入



タスク管理のシステムコール

タスク生成	ID tskid = tk_cre_tsk(T_CTSK *pk_ctsk);
タスク削除	ER ercd = tk_del_tsk(ID tskid);
タスク起動	ER ercd = tk_sta_tsk(ID tskid, INT stacd);
自タスク終了	void tk_ext_tsk();
自タスクの終了と削除	void tk_exd_tsk();
他タスク強制終了	ER ercd = tk_ter_tsk(ID tskid);

■ DORMANT状態を利用したスケジューリングを実現する



```
void tsk_a(INT stacd, VP exinf)
{
    tk_sta_tsk( gTskid[TSK_B], 0 );
    // タスクに与えられた処理
    tk_ext_tsk( );
}
```

```
void tsk_b(INT stacd, VP exinf)
{
    // タスクに与えられた処理
    tk_ext_tsk( );
}
```

● システムコールのパラメータ

ID tskid → タスクID
INT stacd → 起動コード



タスク管理のスケジューリング

■ 優先度：タスクA＞タスクB

	カーネル	タスクA（優先度：高）	タスクB（優先度：低）
		RUNNING	DORMANT
タスクAの tk_sta_tsk →			
タスクAの tk_ext_tsk →		タスクの処理	READY
		DORMANT	RUNNING
タスクBの tk_ext_tsk →			タスクの処理
			DORMANT

■ 優先度：タスクA＜タスクB

	カーネル	タスクA（優先度：低）	タスクB（優先度：高）
		RUNNING	DORMANT
タスクAの tk_sta_tsk →			
タスクBの tk_ext_tsk →		READY	RUNNING
			タスクの処理
タスクAの tk_ext_tsk →		RUNNING	DORMANT
		タスクの処理	
		DORMANT	

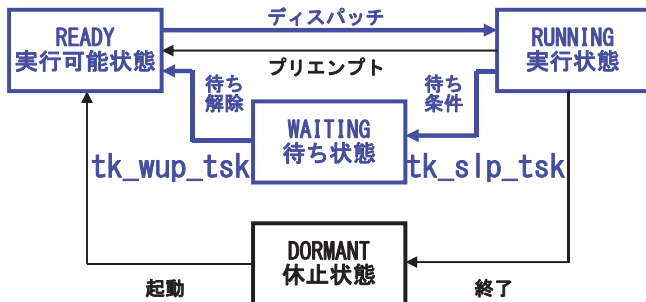


タスク付属同期のシステムコール

自タスクを起床待ち状態へ移行
他タスクの起床
タスクの起床要求を無効化
他タスクの待ち状態解除
タスク遅延

```
ER ercd = tk_slp_tsk( TMO tmout );
ER ercd = tk_wup_tsk( ID tskid );
INT wupcnt = tk_can_wup( ID tskid );
ER ercd = tk_rel_wai( ID tskid );
ER ercd = tk_dly_tsk( RELTIM dlytim );
```

■ WAITING状態を利用したスケジューリングを実現する



```
void tsk_a(INT stacd, VP exinf)
{
    tk_wup_tsk( gTskid[TSK_B] );
    // タスクに与えられた処理
    tk_ext_tsk( );
}
```

```
void tsk_b(INT stacd, VP exinf)
{
    while( 1 ) {
        tk_slp_tsk( TMO_FEVR );
        // タスクに与えられた処理
    }
}
```

● システムコールのパラメータ

ID tskid → タスクID
TMO tmout → タイムアウト：TMO_FEVRで永久待ち



タスク付属同期のスケジューリング

■ 優先度：タスクA ≥ タスクB

	カーネル	タスクA (優先度：高)	タスクB (優先度：低)
		RUNNING	WAITING
タスクAの tk_wup_tsk →			
		タスクの処理	READY
タスクAの tk_ext_tsk →		DORMANT	
			RUNNING
タスクBの tk_slp_tsk →			タスクの処理
			WAITING

■ 優先度：タスクA < タスクB

	カーネル	タスクA (優先度：低)	タスクB (優先度：高)
		RUNNING	WAITING
タスクAの tk_wup_tsk →			
		READY	
タスクBの tk_slp_tsk →			RUNNING
			タスクの処理
タスクBの tk_slp_tsk →			WAITING
		RUNNING	
タスクAの tk_ext_tsk →		タスクの処理	
		DORMANT	



タスク管理とタスク付属同期の違い

```
void tsk_a(INT stacd, VP exinf)
{
  ER ercd;
  ercd = tk_sta_tsk( gTskid[TSK_B], 0 );
  ercd = tk_sta_tsk( gTskid[TSK_B], 0 );
  tk_ext_tsk();
}
```

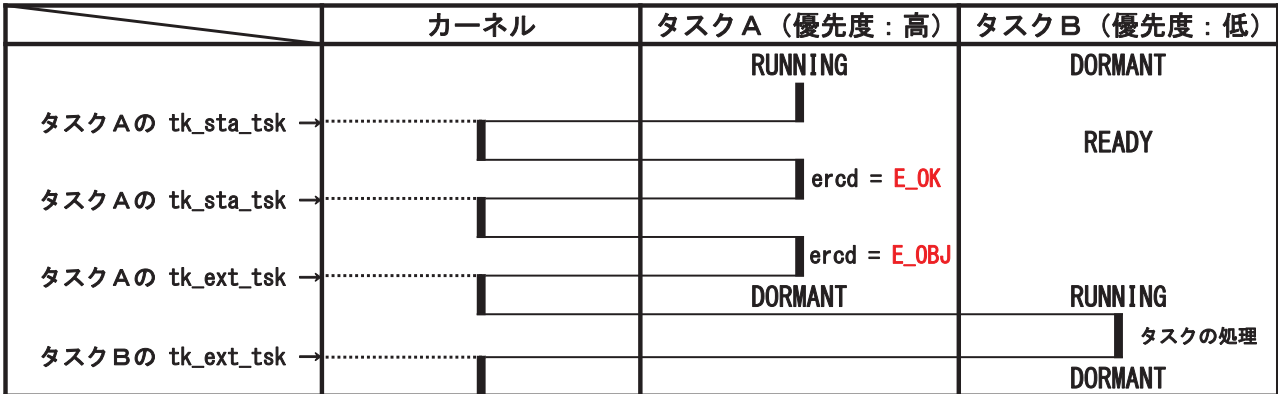
```
void tsk_b(INT stacd, VP exinf)
{
  // タスクに与えられた処理
  tk_ext_tsk();
}
```

● システムコールのエラーコード

E_OK → 正常終了

E_OBJ → オブジェクト状態不正

■ 優先度：タスク A ≥ タスク B



tk_sta_tsk と tk_ext_tsk の組み合わせは重複した起動には耐えられない！

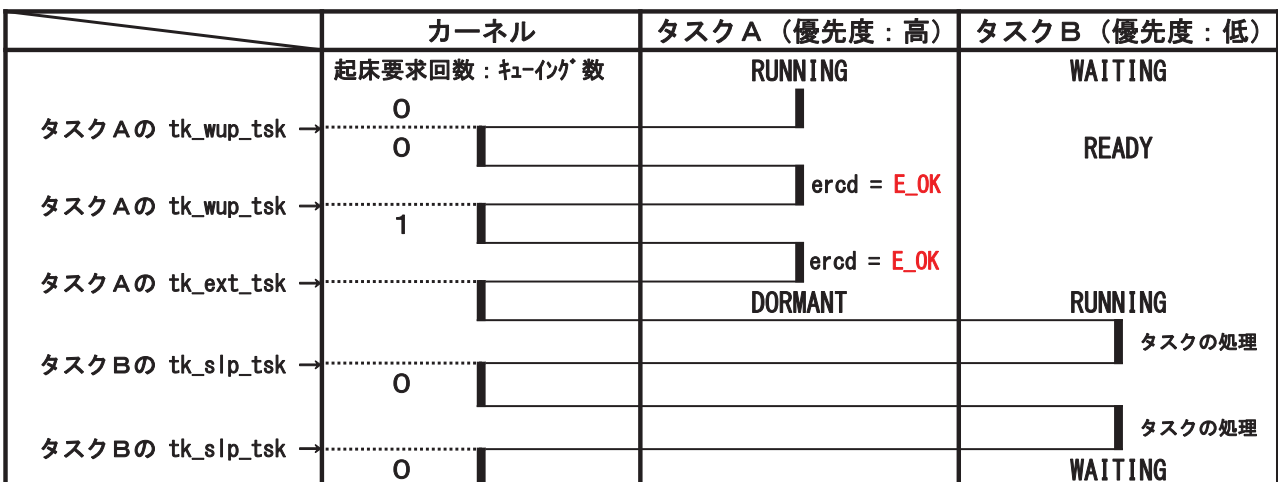


タスク管理とタスク付属同期の違い

```
void tsk_a(INT stacd, VP exinf)
{
  ER ercd;
  ercd = tk_wup_tsk( gTskid[TSK_B] );
  ercd = tk_wup_tsk( gTskid[TSK_B] );
  tk_ext_tsk();
}
```

```
void tsk_b(INT stacd, VP exinf)
{
  while( 1 ) {
    tk_slp_tsk( TMO_FEVR );
    // タスクに与えられた処理
  }
}
```

■ 優先度：タスク A ≥ タスク B



tk_wup_tsk と tk_slp_tsk の組み合わせは重複した起床に耐えられる！

トロンフォーラム
【土曜講座】組込みリアルタイム OS 入門
(μ T-Kernel 入門:協カルネサスエレクトロニクス)
 μ T-Kernel 入門

2016 年 12 月 3 日発行

発行所
トロンフォーラム
(YRP ユビキタス・ネットワーキング研究所内)
〒141-0031 東京都品川区西五反田 2-12-3 第一誠実ビル 9F
URL: <http://www.tron.org/ja/>
TEL:03-5437-0572(代表) FAX:03-5437-2399(代表)

本テキストは、クリエイティブ・コモンズ 表示 - 継承 4.0 国際 ライセンスの下に提供されています。



<https://creativecommons.org/licenses/by-sa/4.0/deed.ja>

Copyright ©2016 TRON Forum

【ご注意およびお願い】

- 1.本テキストの中で第三者が著作権等の権利を有している箇所については、利用者の方が当該第三者から利用許諾を得てください。
 - 2.本テキストの内容については、その正確性、網羅性、特定目的への適合性等、一切の保証をしないほか、本テキストを利用したことにより損害が生じても著者は責任を負いません。
 - 3.本テキストをご利用いただく際、可能であれば office@tron.org までご利用者のお名前、ご所属、ご連絡先メールアドレスをご連絡いただければ幸いです。
-

トロンフォーラム©2016

Printed in Japan