【講座＋実習】

T-Kernel 2.0入門

T-Kernel 2.0の概要

背景

T-Kernelの実績と特長

- T-Kernelは2002年に発表
  - 業務用端末やカーナビ、プリンタなど多くの採用例
- 制御系も情報系も得意なOS
  - ITRONと同じリアルタイム性能
  - 標準化された基本ミドルウェア(Extension)によりプロセス、ファイルなどの情報処理機能を実現
- 最近の高機能な組込み機器のニーズに合致

マーケットからの要求

- T-Kernelの実績や採用製品は着実に増えている
- より高性能化するハードウェアを活かせるような機能追加の要求
  - もっと細かい時間粒度で制御したい
  - さらに大容量のストレージを扱えるようにしたい
  - キャッシュや省電力機能をよりきめ細かく制御をしたい
- さらに使いやすく
  - プロジェクトの立ち上げを早くするために、カーネルだけでなく、T-Monitorやデバイスドライバ、開発環境も含め、提供して欲しい
  - ユーザーにとってわかりやすいドキュメントが欲しい

T-Kernel 2.0の概要

方針

T-Kernel 2.0®

位置付け

- 上位互換
  - マーケットからの要求への対応は「機能の追加」で行う
    - T-Kernel 2.0 は従来の T-Kernel を包含
  - 従来のT-Kernel上で動作していたプログラムは、バイナリのまま、同じハードウェアのT-Kernel 2.0上でも動作
    - デバイスドライバ、ミドルウェア、アプリケーションいずれも、再コンパイル不要
    - 「100年ソフト」コンセプトを堅持

位置付け

- T-Kernel 2.0への移行を推奨する
  - あたらしくT-Kernelを実装する場合はT-Kernel 2.0を推奨
    - ただし強制はしない
  - 従来のT-KernelはT-Kernel 1.0と呼ぶ
    - T-Kernel 1.0を使った装置が継続的に動いていることは全く問題ない

提供方法

- リファレンス実装の提供
  - 実装依存部分まで含めた広範囲のソースコードを公開し、ユーザーにとって立ち上げ易くする
  - 「T-Engineリファレンスボード」上にT-Kernel 2.0をリファレンス実装して公開
    - T-Monitor、基本デバイスドライバもソース公開
    - リファレンス実装を元に自由に移植してよい

T-Kernel 2.0の概要

ライセンス

T-Kernel 2.0の配布ライセンスと トレーサビリティサービス

- 従来より使いやすいT-License 2.0
  - 改変版を含むソースの再配布を許可
- ucodeによる配布用ソースの履歴管理
  - ソースコードのトレーサビリティシステムを導入
  - 再配布の際にはディストリビューションucodeをつける
  - このucodeにより改変元や派生先などの履歴が分かる
- ダウンロードの手続きを簡素化

T-Kernelトレーサビリティサービス

- 来歴の確認
  - ディストリビューションucodeをT-Engineフォーラムに問い合わせると、オリジナルのリファレンス実装のどのディストリビューションからどのように派生したものかがわかる
- 修正項目の確認
  - ディストリビューションucodeから、どのような修正項目が入っているかがわかる
    - 問題のある修正があった場合、それがどのディストリビューションに含まれているかが確認できる

T-Kernelトレーサビリティサービス

T-Kernel 2.0の概要

T-Engineリファレンスボード

T-Engineリファレンスボード

- 従来の標準T-Engine/ μT-Engineボード
  - 小型の開発プラットフォームとして成功を収めたが、マイクロプロセッサがSoC化してきて、現在では規定の意味が希薄になった
  - 周辺や基板のサイズ統一の意味も重要ではなくなった

- T-Engineリファレンスボード
  - コアや周辺、基板サイズを規定するのではなく、 T-Kernel 2.0が稼働するプラットフォームと規定
  - 重要なことは、CPUやボードに関する詳細情報が開示されること

T-Engineリファレンスボードの要件

- CPU、メモリ構成
  - T-Kernel 2.0リファレンス実装が可能なこと
- I/O構成
  - デバッグ用ポート(シリアル　and/or USB)を持つこと
  - LANを持つこと
  - ファイル用のストレージを持つこと
- 寸法やコネクタ規定
  - なし

T-Engineリファレンスボードの要件

- オープン性
  - CPU(SoC)について、ユーザーズマニュアル等が公開されており、一般に入手可能であること（英文必須）
  - 回路図(部品表付き)および説明書が公開、あるいはボードを購入すれば入手可能であること
  - T-Monitor、T-Kernel 2.0、デバイスドライバのソースコードが移植され、それらが公開されていること

T-Engineリファレンスボード

- まず1機種をリリース
  - 今後、条件を満たすボードを認定して機種を増やす

型番:U00B0021-02

T-Kernel 2.0の追加機能

T-Kernel 1.0との関係

- 従来のT-Kernel 1.0の設計方針や特長は不変
- CPUの高性能化、デバイスの大容量化を活かす追加機能
  - 64ビットデータ、マイクロ秒単位の時間指定など
- T-Kernel 1.0に対しては上位互換
  - ソース互換およびバイナリ互換

T-Kernel 2.0の追加機能

- 64ビットデータの導入
- マイクロ秒単位の時間管理機能
- 物理タイマ機能
- 大容量デバイスへの対応
- その他の追加機能

T-Kernel 2.0の追加機能

データタイプ

64ビットデータの導入

- C言語の規格(C99)で64ビットのlong long型が正式に仕様化
- T-Kernelで使う大部分のコンパイラ(gccなど)でサポート済
- マイクロ秒単位の時間管理や64ビットデバイスに利用

```
typedef  signed long long          D;    /* 符号付き64 ビット整数 */
typedef  unsigned long long        UD;   /* 符号無し64 ビット整数 */
```

データタイプの定義

- 符号付き整数の宣言にsignedを追加
- W,UWをintからlongに変更
  - 32ビットであることを明確化
- MSECをINTからWに変更
  - 16ビットになると不都合なため

```
typedef     signed char             B;
typedef     signed short            H;
typedef     signed long             W;
typedef     signed long long        D;
typedef     signed int              INT;
typedef     W                       MSEC;
```

CONST修飾子の利用

- C言語のconst修飾子と同じ意味
  - チェックを無効化できるように、マクロによる別名を使用
- ポインタ参照先のデータを書き替えないことを明記

```
tk_cre_tsk ( CONST T_CTSK *pk_ctsk );     /* CONST有り */
  pk_ctsk の内容
        void*     exinf     拡張情報          /* T-Kernelへのパラメータ */
        ATR       tskatr    タスク属性        /* T-Kernelへのパラメータ */
        FP        task      タスク起動アドレス
                  ...

tk_ref_tsk ( ID tskid, T_RTSK *pk_rtsk );   /* CONST無し */
  pk_rtsk の内容
        void*     exinf     拡張情報          /* T-Kernelからの戻り値 */
        PRI       tskpr     現在の優先度      /* T-Kernelからの戻り値 */
                  ...
```

CONSTを使うシステムコール例

```
ID tskid = tk_cre_tsk ( CONST T_CTSK *pk_ctsk );
ID semid = tk_cre_sem ( CONST T_CSEM *pk_csem );
ER ercd = tk_set_tim ( CONST SYSTIM *pk_tim );
ID cycid = tk_cre_cyc ( CONST T_CCYC *pk_ccyc );
ER ercd = tk_def_int ( UINT dintno, CONST T_DINT *pk_dint );
ID dd = tk_opn_dev ( CONST UB *devnm, UINT omode );
ER ercd = LockSpace ( CONST void *addr, INT len );
```

CONSTの使い方

- 共通のインクルードファイルで以下を記述

```
# ifdef TKERNEL_CHECK_CONST
# define CONST const
# else
# define CONST
# endif
```

- プログラム中の関数定義はCONSTを用いて記述
- Makefile内の指定でconstを有効化(推奨)
  - CFLAGS += – DTKERNEL_CHECK_CONST

※ 上記を指定しない場合はconstのチェックをしない

VPは使用しない

- CONSTと組み合わせて書けない場合があるため
  - VPの代わりに ″void *″ を直接書く
  - 互換性維持のためにVP自体の定義は残す

- VPを使用しなくなった例

```
ID reqid = tk_rea_dev ( ID dd, W start, void *buf, W size, TMO tmout );
ER ercd = tk_get_mpl ( ID mplid, INT blksz, void **p_blk, TMO tmout );
ER ercd = ChkSpaceR ( void *addr, INT len );
ER ercd = MapMemory ( CONST void *paddr, INT len, UINT attr, void **laddr );
ID reqid = tk_wri_dev ( ID dd, W start, CONST void *buf, W size, TMO tmout );

void *exinf;    /* tk_cre_XXXのパケット内の拡張情報 */
```

T-Kernel 2.0の追加機能

マイクロ秒単位の時間管理機能

マイクロ秒単位の時間管理機能

- ITRONやT-Kernel 1.0の時間管理の指定はミリ秒単位
- より細かい時間分解能への要求
  - CPUの高性能化
    - システムコール実行時間が1マイクロ秒に迫る
    - タスクの開始から終了までミリ秒未満のケースも
  - FA用の制御装置、PLCなどでの要求

マイクロ秒単位の時間管理機能

- 指定可能な時間の範囲
  - 32ビットでミリ秒単位では約24日間
  - 32ビットマイクロ秒単位では、この1000分の1で35分間
  - 64ビットデータの採用により指定範囲を拡大
- マイクロ秒指定のシステムコールを追加
  - 互換性やマイクロ秒が不要な用途のため、ミリ秒単位のシステムコールも残る。混在利用も可能。

マイクロ秒単位の時間管理機能

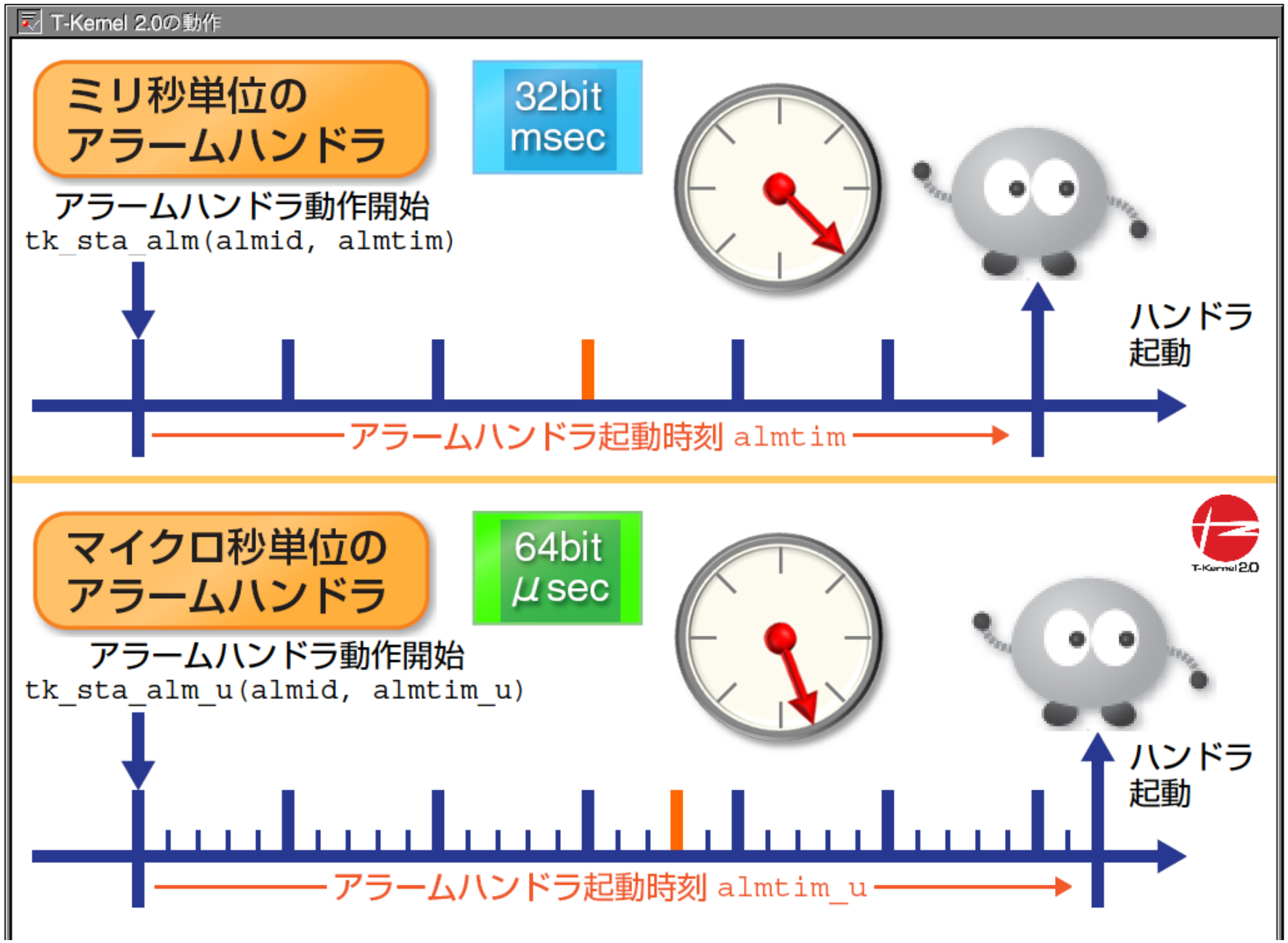
- 64 ビットマイクロ秒単位を扱うデータタイプは '_U' を付加

```
typedef         D       TMO_U;          /* タイムアウト(μsec) */
typedef         UD      RELTIM_U;       /* 相対時間(μsec) */
typedef         D       SYSTIM_U;       /* システム時刻(μsec) */
```

マイクロ秒単位の時間管理機能

- マイクロ秒単位のシステムコールは '_u' を付加
  例:「アラームハンドラの動作開始」を行うAPI
  - tk_sta_alm (ID almid, RELTIM almtim);
    起動時刻almtimは32ビットミリ秒単位で指定
  - tk_sta_alm_u (ID almid, RELTIM_U almtim_u);
    起動時刻almtim_uは64ビットマイクロ秒単位で指定
    ※ T-Kernel 2.0ではtk_sta_alm_u()が追加

マイクロ秒単位の時間管理機能



マイクロ秒単位の時間管理機能

- マイクロ秒単位の時間指定が可能なシステムコール例

tk_slp_tsk_u　　自タスクを起床待ち状態へ移行(マイクロ秒単位)

C言語インタフェース

```
ER ercd = tk_slp_tsk_u ( TMO_U tmout_u );
```

パラメータ

TMO_U	tmout_u	タイムアウト指定(マイクロ秒)

リターンパラメータ

ER	ercd	エラーコード
	...	

マイクロ秒単位の時間管理機能

- マイクロ秒単位の時間指定が可能なシステムコール例

tk_cre_cyc_u　　周期ハンドラの生成(マイクロ秒単位)

C言語インタフェース
ID cycid = tk_cre_cyc_u (CONST T_CCYC_U *pk_ccyc_u);

パラメータ

CONST T_CCYC_U*	pk_ccyc_u	周期ハンドラ定義情報
pk_ccyc_u の内容		
void*	exinf	拡張情報
ATR	cycatr	周期ハンドラ属性
FP	cychdr	周期ハンドラアドレス
RELTIM_U	cyctim_u	周期起動時間間隔(マイクロ秒)
RELTIM_U	cycphs_u	周期起動位相(マイクロ秒)

リターンパラメータ
ID　cycid　　周期ハンドラIDまたはエラーコード

マイクロ秒単位の時間管理機能

- マイクロ秒単位を扱う追加システムコール
  - 時刻の設定や取得

    tk_get_tim_u, tk_set_tim_uなど
  - 周期ハンドラやアラームハンドラの生成

    tk_cre_cyc_u, tk_sta_alm_u
  - 待ちに入るシステムコールのタイムアウト指定

    tk_slp_tsk_u, tk_wai_sem_u, tk_wai_flg_uなど
  - タスク遅延(tk_dly_tsk_u())での待ち時間指定
  - 時間情報を含むタスクやハンドラなどの状態参照

    tk_ref_tsk_u, tk_inf_tsk_u, tk_ref_cyc_uなど

タイマ割込み間隔

- 実際の時間分解能は「タイマ割込み間隔」に依存
  - 周期ハンドラの場合、実際にハンドラが起動されるのは、周期ハンドラを起動すべき時刻の直後のタイマ割込みの時点
  - タイマ割込み間隔を短くしないとマイクロ秒単位の動作をしない
- タイマ割込み間隔(タイムティック)は標準システム構成情報のTTimPeriodで指定

TTimPeriod　　〈msec〉〈μsec〉
　〈msec〉　　タイマ割込み間隔(ミリ秒単位)
　〈μsec〉　　タイマ割込み間隔(マイクロ秒単位)

　〈msec〉と〈μsec〉の合計が実際のタイマ割込み間隔となる
　〈μsec〉を省略した場合は0とみなし、T-Kernel 1.0との互換性を維持

　タイマ割込み間隔を1.5ミリ秒(1500マイクロ秒)とする例
　　TTimPeriod　1　500
または　TTimPeriod　0　1500

タイマ割込み間隔と周期ハンドラ動作例

- 以下のプログラムで周期ハンドラを生成

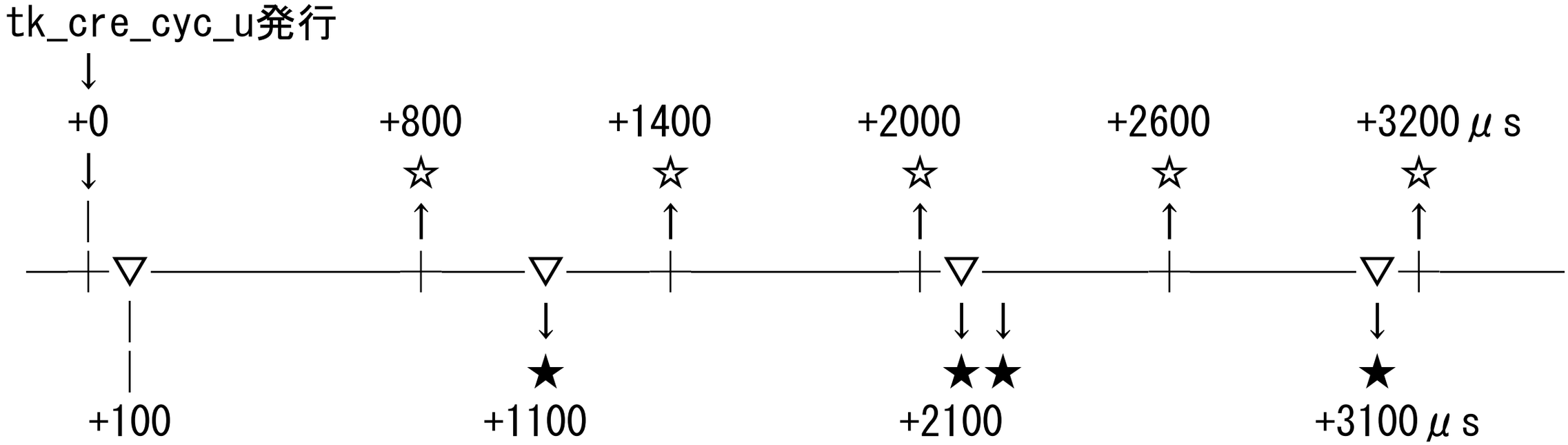
```
T_CCYC_U        ccyc_u;

  ccyc_u.cycatr = (TA_HLNG | TA_STA);       /* 周期ハンドラ属性 */
  ccyc_u.cychdr = cyclic_hander;            /* 周期ハンドラアドレス */
  ccyc_u.cyctim_u = 600;                    /* 周期起動時間間隔(マイクロ秒) */
  ccyc_u.cycphs_u= 800;                     /* 周期起動位相(マイクロ秒) */

  cycid = tk_cre_cyc_u (&ccyc_u);                 /* 周期ハンドラ生成 */
```

※ 最初のタイマ割込みがtk_cre_cyc_u発行の100マイクロ秒後とする。

タイマ割込み間隔と周期ハンドラ動作例

- タイマ割込み間隔が1ミリ秒(1000マイクロ秒)の場合の動作例

  TTimPeriod　　1



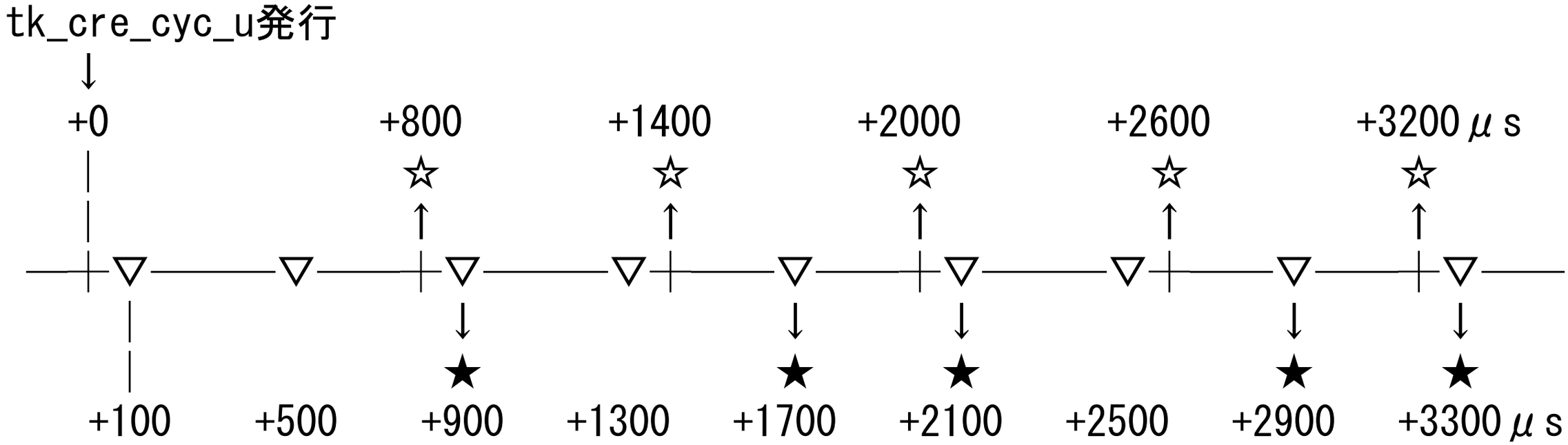
▽ タイマ割込み　　☆ 周期ハンドラを起動すべき時刻

★ 実際に周期ハンドラが起動される時刻

タイマ割込み間隔と周期ハンドラ動作例

- タイマ割込み間隔が400マイクロ秒の場合の動作例

  TTimPeriod　0　400



▽ タイマ割込み

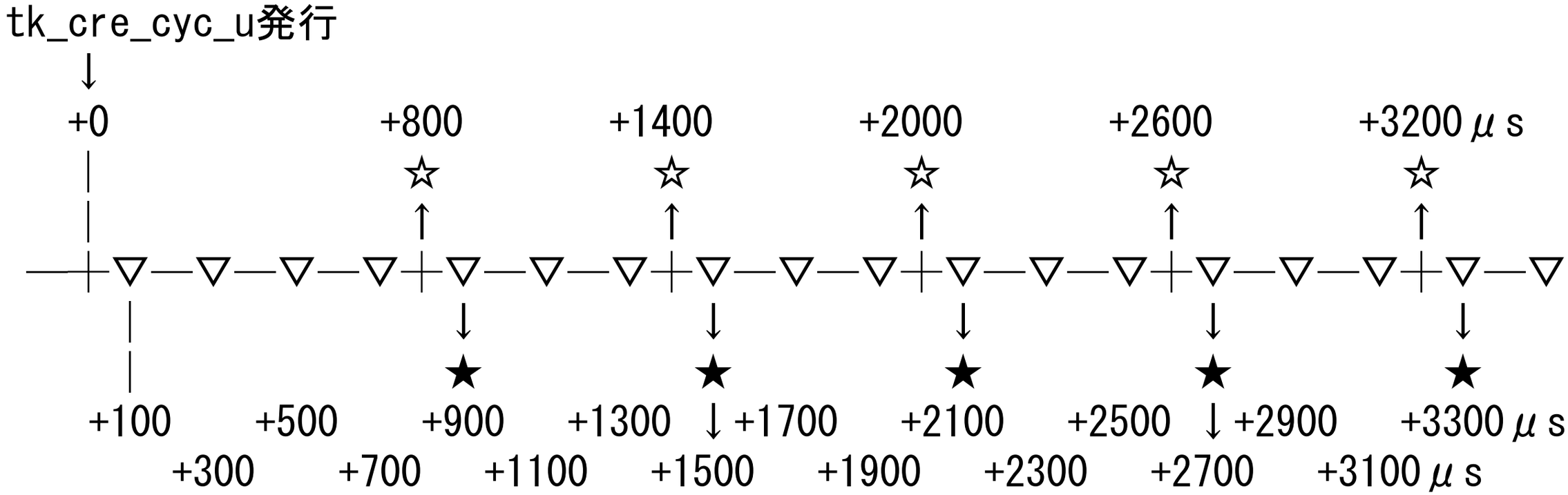
☆ 周期ハンドラを起動すべき時刻

★ 実際に周期ハンドラが起動される時刻

タイマ割込み間隔と周期ハンドラ動作例

- タイマ割込み間隔が200マイクロ秒の場合の動作例

TTimPeriod　0　200



▽ タイマ割込み　　☆ 周期ハンドラを起動すべき時刻

★ 実際に周期ハンドラが起動される時刻

T-Kernel 2.0の追加機能

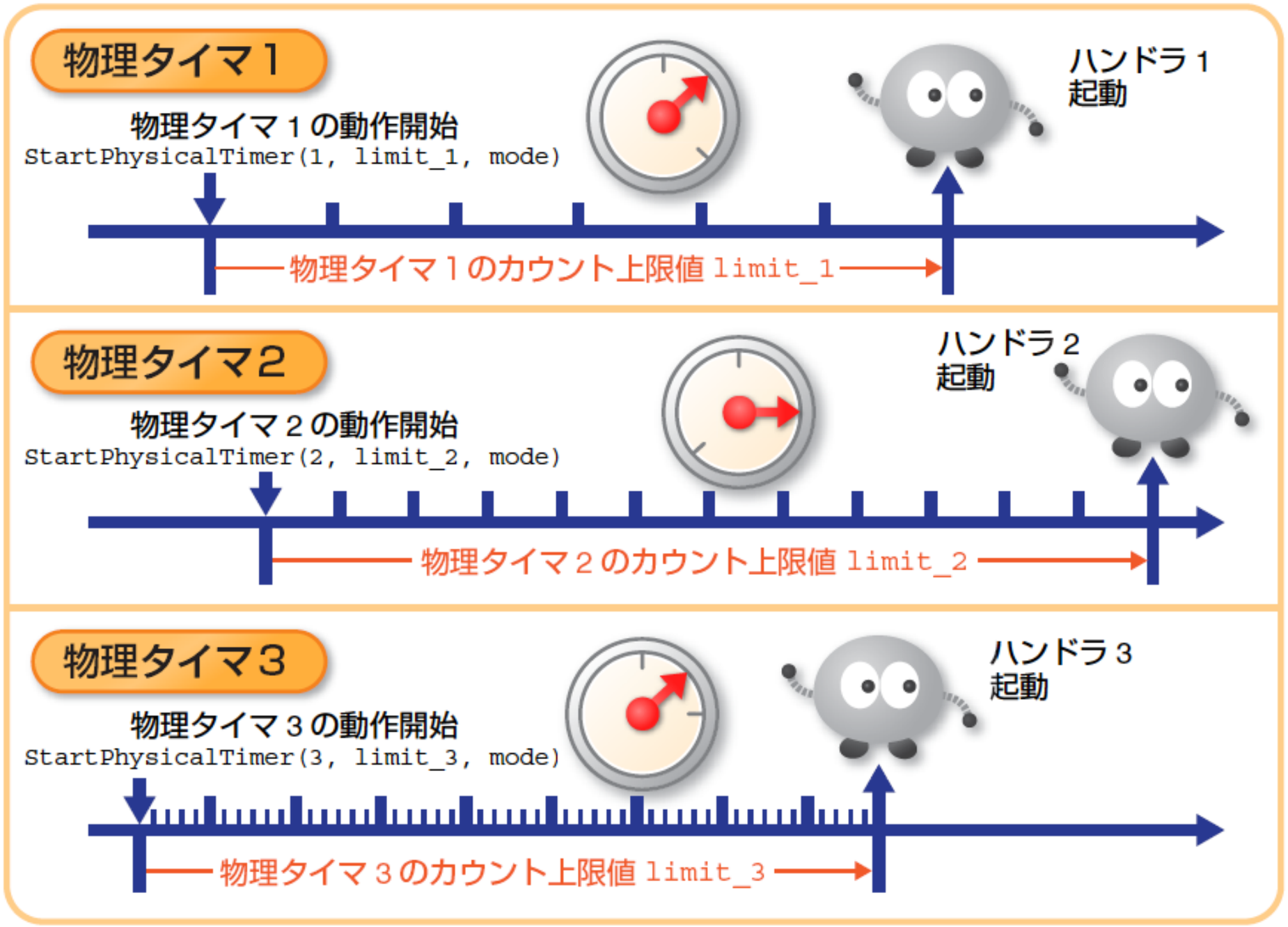
物理タイマ機能

T-Kernel 2.0®

物理タイマ機能

- SoC(System on a Chip)上の豊富なハードウェアタイマを活用
- 1つのハードウェアタイマが1つの物理タイマに対応
- 複数の物理タイマが独立して動作
- 機能はプリミティブ
  - 一定周期で単純にインクリメントするカウンタ
  - カウント値が上限値に達すると0に戻る
  - カウント値が0に戻る際にハンドラを実行
  - ワンショットの動作と周期的な動作を選択

物理タイマ機能



※物理タイマ１～物理タイマ３はすべて独立して動作する。

物理タイマの有効性が高い例

- 次の2つの周期的な処理を行う
  - 2500マイクロ秒毎に行う処理X
  - 1800マイクロ秒毎に行う処理Y

- 2つの方法を比較
  - 物理タイマ機能を使った実現方法
  - tk_cre_cyc_u()を使った実現方法

物理タイマの有効性が高い例

【物理タイマ機能を使った実現方法】

- 2つの物理タイマを使う
  - 物理タイマX: 2500マイクロ秒毎に起動、処理Xを行う
  - 物理タイマY: 1800マイクロ秒毎に起動、処理Yを行う
- 物理タイマのクロック周波数が10MHzの場合の設定例
  - 10MHzなので1クロックが0.1 マイクロ秒(＝100 ナノ秒)
  - 物理タイマXの上限値(limit)に24999(＝25000-1)を指定
    - ●カウント値が24999から0に戻る時にXのハンドラを起動
  - 物理タイマYの上限値(limit)に17999(＝18000-1)を指定
    - ●カウント値が17999から0に戻る時にYのハンドラを起動
- StartPhysicalTimer()のmodeはTA_CYC_PTMR(周期処理)
- タイマ割込み間隔(TTimPeriod)は無関係
  - デフォルト値(10ミリ秒)のままでもよい

物理タイマの有効性が高い例

【tk_cre_cyc_u()を使った実現方法】

- 2つの周期ハンドラを使う
  - 周期ハンドラX
    - 2500マイクロ秒毎に起動、処理Xを行う
  - 周期ハンドラY
    - 1800マイクロ秒毎に起動、処理Yを行う
- タイマ割込み間隔(TTimPeriod)は十分短くする
  - たとえば、2つの周期の公約数である100マイクロ秒とする

物理タイマの有効性が高い例

【物理タイマ機能とtk_cre_cyc_u()の比較】

- 10ミリ秒の間の時間関係の割込み回数
  - 物理タイマ機能を使う場合は合計10回程度
    - ●Xのハンドラ起動の割込み: 10ミリ秒÷2500マイクロ秒＝4回
    - ●Yのハンドラ起動の割込み: 10ミリ秒÷1800マイクロ秒＝5回
    - ●タイマ割込み: 10ミリ秒÷10ミリ秒＝1回
  - tk_cre_cyc_u()を使う場合はタイマ割込み間隔に依存
    - ●周期処理の時刻の正確さとのトレードオフになる
    - ●タイマ割込み: 10ミリ秒÷100マイクロ秒＝100回
    - （タイマ割込み間隔を100マイクロ秒とした場合）
- 一方、物理タイマの利用には十分な数のハードウェアタイマが必要

周期/アラームハンドラと物理タイマとの使い分け

- 周期/アラームハンドラの特長
  - 1つのシステムタイマとOSのソフトウェアで実現
  - メリット
    - 自由度が高い
    - 生成可能な個数がハードウェアに制約されない

- 物理タイマ機能の特長
  - ハードウェアタイマで実現
  - メリット
    - システムタイマ割込みが不要
    - 正確でオーバーヘッドのないタイマ機能を実現

物理タイマ機能のAPI(1/5)

StartPhysicalTimer　　物理タイマの動作開始

C言語インタフェース
ER ercd = StartPhysicalTimer (UINT ptmrno, UW limit, UINT mode);

パラメータ

UINT	ptmrno	物理タイマ番号
UW	limit	上限値
UINT	mode	動作モード

リターンパラメータ

ER	ercd	エラーコード

物理タイマ機能のAPI(2/5)

StopPhysicalTimer　　物理タイマの動作停止

C言語インタフェース
ER ercd = StopPhysicalTimer (UINT ptmrno);

パラメータ

UINT	ptmrno	物理タイマ番号

リターンパラメータ

ER	ercd	エラーコード

物理タイマ機能のAPI(3/5)

GetPhysicalTimerCount　　物理タイマのカウント値取得

C言語インタフェース
ER ercd = GetPhysicalTimerCount (UINT ptmrno, UW *p_count);

パラメータ

UINT	ptmrno	物理タイマ番号
UW*	p_count	物理タイマの現在カウント値を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
UW	count	現在カウント値

物理タイマ機能のAPI(4/5)

DefinePhysicalTimerHandler　　物理タイマハンドラ定義

C言語インタフェース
ER ercd = DefinePhysicalTimerHandler (UINT ptmrno, CONST T_DPTMR *pk_dptmr);

パラメータ

UINT	ptmrno	物理タイマ番号
CONST T_DPTMR*	pk_dptmr	物理タイマハンドラ定義情報

pk_dptmr の内容

void*	exinf	拡張情報
ATR	ptmratr	物理タイマハンドラ属性(TA_ASM || TA_HLNG)
FP	ptmrhdr	物理タイマハンドラアドレス

リターンパラメータ

ER	ercd	エラーコード

物理タイマ機能のAPI(5/5)

GetPhysicalTimerConfig　　物理タイマのコンフィグレーション情報取得

C言語インタフェース
ER ercd = GetPhysicalTimerConfig (UINT ptmrno, T_RPTMR *pk_rptmr);

パラメータ

UINT	ptmrno	物理タイマ番号
T_RPTMR*	pk_rptmr	物理タイマのコンフィグレーション情報を返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード

pk_rptmr の内容

UW	ptmrclk	物理タイマのクロック周波数
UW	maxcount	最大カウント値
BOOL	defhdr	物理タイマハンドラサポートの有無

T-Kernel 2.0の追加機能

大容量デバイスへの対応

大容量デバイスへの対応

- デバイス管理のAPIの開始位置(start)を64ビット対応に
- 指定可能な範囲
  - 512バイト/セクタのHDDでは、32ビット(約2G)で約1TB
  - 開始位置(start)を64ビットにしてこの制限を解消
- 64ビット指定のシステムコールは純粋に追加
  - 32ビット指定のAPIはそのまま残し、混在利用も可能

大容量デバイスへの対応

- 64ビットを扱うシステムコールは '_d' を付加

  例:「デバイスへの書き込み」を行うAPI

  - tk_wri_dev (ID dd, W start, CONST void *buf, W size, TMO tmout);

    書込み開始位置startは32ビットで指定

    タイムアウト時間tmoutは32ビットミリ秒単位で指定

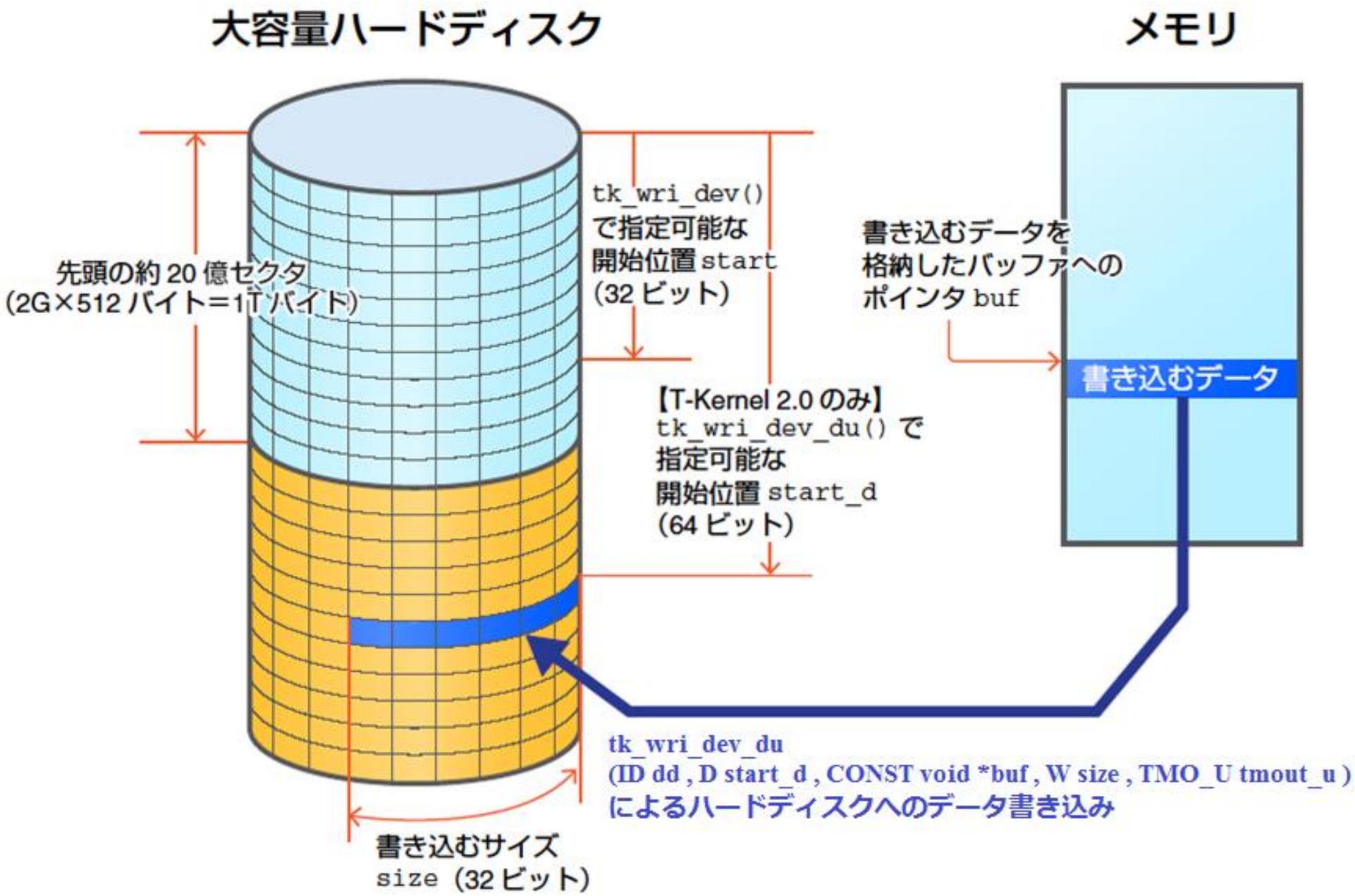
  - tk_wri_dev_du (ID dd, D start_d, CONST void *buf, W size, TMO_U tmout_u);

    書込み開始位置start_dは64ビットで指定

    タイムアウト時間tmout_uは64 ビットマイクロ秒で指定

    ※ T-Kernel 2.0ではtk_wri_dev_du()が追加

大容量デバイスへの対応



大容量デバイスへの対応

- T-Kernel/SMのデバイス管理機能の構成
  ※以下のそれぞれの箇所で64ビット対応が必要

[アプリケーション]
↓
アプリケーションインタフェース(tk_rea_devなど)
↓
[デバイス管理機能(T-Kernel/SM)]
↓
デバイスドライバインタフェース(openfn,execfnなど)
↓
[デバイスドライバ]
↓
[各デバイスのハードウェア]

アプリケーションインタフェースの64ビットAPI(1/4)

tk_rea_dev_du　　デバイスの読込み開始(64ビットマイクロ秒単位)

C言語インタフェース

```
ID reqid = tk_rea_dev_du ( ID dd, D start_d, void *buf, W size, TMO_U
tmout_u );
```

パラメータ

ID	dd	デバイスディスクリプタ
D	start_d	読込み開始位置(64ビット, ≧0:固有データ, ＜0:属性データ)
void*	buf	読み込んだデータを格納するバッファ
W	size	読み込むサイズ
TMO_U	tmout_u	要求受付待ちタイムアウト時間(マイクロ秒)

リターンパラメータ

ID	reqid	リクエストIDまたはエラーコード

アプリケーションインタフェースの64ビットAPI(2/4)

tk_srea_dev_d　　デバイスの同期読込み(64ビット)

C言語インタフェース
ER ercd = tk_srea_dev_d (ID dd, D start_d, void *buf, W size, W *asize);

パラメータ

ID	dd	デバイスディスクリプタ
D	start_d	読込み開始位置(64ビット, ≧0:固有データ, ＜0:属性データ)
void*	buf	読み込んだデータを格納するバッファ
W	size	読み込むサイズ
W*	asize	読込みサイズを返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
W	asize	実際の読込みサイズ

アプリケーションインタフェースの64ビットAPI(3/4)

tk_wri_dev_du　　デバイスの書込み開始(64ビットマイクロ秒単位)

C言語インタフェース

```
ID reqid = tk_wri_dev_du ( ID dd, D start_d, CONST void *buf, W size, TMO_U
tmout_u );
```

パラメータ

ID	dd	デバイスディスクリプタ
D	start_d	書込み開始位置(64ビット, ≧0:固有データ, ＜0:属性データ)
CONST void*	buf	書き込むデータを格納したバッファ
W	size	書き込むサイズ
TMO_U	tmout_u	要求受付待ちタイムアウト時間(マイクロ秒)

リターンパラメータ

ID	reqid	リクエストIDまたはエラーコード

アプリケーションインタフェースの64ビットAPI(4/4)

tk_swri_dev_d　　デバイスの同期書込み(64ビット)

C言語インタフェース

ER ercd = tk_swri_dev_d (ID dd, D start_d, CONST void *buf, W size, W *asize);

パラメータ

ID	dd	デバイスディスクリプタ
D	start_d	書込み開始位置(64ビット, ≧0:固有データ, ＜0:属性データ)
CONST void*	buf	書き込むデータを格納したバッファ
W	size	書き込むサイズ
W*	asize	書込みサイズを返す領域へのポインタ

リターンパラメータ

ER	ercd	エラーコード
W	asize	実際の書込みサイズ

デバイス情報取得機能の64ビット対応

- 属性データによるデバイス情報取得機能の64ビット対応版
  - TDN_DISKINFO=(-2)の64ビット版としてTDN_DISKINFO_D=(-5)が追加
- ドライバはTDN_DISKINFO、TDN_DISKINFO_Dの少なくとも一方に対応
  - T-Kernel 1.0との互換性のため、可能な限りTDN_DISKINFOにも対応
  - DiskInfoのblockcontが32ビットに収まらない場合はエラー(E_PAR)
- デバイスドライバ属性のTDA_DEV_Dとは別(直接の依存関係はない)

```
#define TDN_EVENT           (-1)    /* RW:事象通知用メッセージバッファID */
#define TDN_DISKINFO        (-2)    /* R-:ディスク情報 */
#define TDN_DISPSPEC        (-3)    /* R-:表示デバイス仕様 */
#define TDN_PCMCIAINFO      (-4)    /* R-:PCカード情報 */
#define TDN_DISKINFO_D      (-5)    /* R-:ディスク情報(64ビットデバイス) */
```

デバイス情報取得機能の64ビット対応

- 属性データによるデバイス情報取得機能の64ビット版 TDN_DISKINFO_D
  - 32ビット用のTDN_DISKINFOでは、以下の “D　blockcont_d;” の代わりに、 ″W　blockcont;″

◆TDN_DISKINFO_Dを指定した場合のデータ形式(64ビット)

```
typedef struct diskinfo_d {
    DiskFormat format;      /* フォーマット形式 */
    BOOL    protect:1;      /* プロテクト状態 */
    BOOL    removable:1;    /* 取り外し可否 */
    UW      rsv:30;         /* 予約 (0) */
    W       blocksize;      /* ブロックバイト数 */
    D       blockcont_d;    /* 64ビットの総ブロック数 */
} DiskInfo_D;
```

デバイスドライバインタフェースの64ビット対応

- デバイス属性に「マイクロ秒単位」「64ビットデバイス」を追加
  - デバイス属性はtk_def_devのパラメータddev->devatrで指定される

TDA_TMO_U: マイクロ秒単位タイムアウト時間を使用
ドライバ処理関数のタイムアウト指定がマイクロ秒単位になる
具体的には、execfn,waitfnのtmoutがTMO_U型

TDA_DEV_D: 64ビットデバイスを使用
ドライバ処理関数への要求パケットが64ビットになる
具体的には、execfn,waitfn,abortfnのdevreqがT_DEVREQ_D型

デバイスドライバインタフェースの64ビット対応

- execfn() での例

◆TDA_TMO_U指定なし、TDA_DEV_D指定なし

```
/* 処理開始関数(32ビット要求パケット、ミリ秒タイムアウト) */
ER ercd = execfn ( T_DEVREQ *devreq, TMO tmout, void *exinf );
```

◆TDA_TMO_U指定なし、TDA_DEV_D指定あり

```
/* 処理開始関数(64ビット要求パケット、ミリ秒タイムアウト) */
ER ercd = execfn ( T_DEVREQ_D *devreq_d, TMO tmout, void *exinf );
```

◆TDA_TMO_U指定あり、TDA_DEV_D指定なし

```
/* 処理開始関数(32ビット要求パケット、マイクロ秒タイムアウト) */
ER ercd = execfn ( T_DEVREQ *devreq, TMO_U tmout_u, void *exinf );
```

◆TDA_TMO_U指定あり、TDA_DEV_D指定あり

```
/* 処理開始関数(64ビット要求パケット、マイクロ秒タイムアウト) */
ER ercd = execfn ( T_DEVREQ_D *devreq_d, TMO_U tmout_u, void *exinf );
```

デバイスドライバインタフェースの64ビット対応

- 64ビットのデバイス要求パケットT_DEVREQ_D
  - 32ビット用のT_DEVREQでは、以下の "D start_d;" の代わりに "W start;"

```
typedef struct t_devreq_d {
    struct t_devreq_d *next;  /* In:要求パケットのリンク (NULL:終端) */
    void     *exinf;          /* X:拡張情報 */
    ID       devid;           /* In:対象デバイスID */
    INT      cmd:4;           /* In:要求コマンド */
    BOOL     abort:1;         /* In:中止要求を行った時 TRUE */
    BOOL     nolock:1;        /* In:ロック(常駐化)不要のとき TRUE */
    INT      rsv:26;          /* In:予約(常に0) */
    T_TSKSPC tskspc;          /* In:要求タスクのタスク固有空間 */
    D        start_d;         /* In:開始データ番号, 64ビット */
    W        size;            /* In:要求サイズ */
    void     *buf;            /* In:入出力バッファアドレス */
    W        asize;           /* Out:結果サイズ */
    ER       error;           /* Out:結果エラー */
} T_DEVREQ_D;
```

T-Kernel 2.0の追加機能

その他の追加機能

その他の追加機能

- システム管理プログラムなどで使うAPI

アドレス空間の各種情報の取得	GetSpaceInfo
メモリアクセス権の設定	SetMemoryAccess
キャッシュモードの設定	SetCacheMode
キャッシュの制御	ControlCache

- 高速ロック・マルチロック機能
  - ライブラリで実現されていた機能をT-Kernelに追加
  - デバイスドライバなどで既に利用

高速ロック・マルチロック

- 複数タスク間の排他制御をより高速に行う
  - T-Kernel/SMのライブラリ関数として実装
  - 待ちに入らない場合のロック獲得の操作を高速化
- 高速ロックの実装方法
  - 「待ちに入らない場合」はカウンタ操作のみ
    - システムコールを呼ばないため高速
  - 「待ちに入る場合」は通常のセマフォの機能を使う
    - 待ち行列の管理、待ち状態からの解放もセマフォで処理
    - 特に高速ではない

→ 待ちに入る可能性が低い場合に有効な機能

高速ロック･マルチロック

- 高速ロック
  - 排他制御用バイナリセマフォの高速版
  - ロック状態を示すカウンタとセマフォにより実装

- 高速マルチロック
  - 排他制御用バイナリセマフォ32個をグループ化
  - 0～31番のロック番号で区別
  - 複数個の高速ロックを使うよりもリソースを節約
  - ロック状態を示すカウンタとイベントフラグにより実装

高速ロック・マルチロック

- 高速ロックの処理(Lock)

```
EXPORT void Lock( FastLock *lock ) {
        if ( Inc(lock) > 0 ) {
          tk_wai_sem(lock->id, 1, TMO_FEVR);
        }
    }

Inline INT Inc( FastLock *lock ) {
    UINT     imask; INT    c;
    DI(imask);
    c = ++lock->cnt;
    EI(imask);
    return c;
}
```

高速ロック・マルチロック

- 高速ロックの処理(Unlock)

```
EXPORT void Unlock( FastLock *lock ) {
        if ( Dec(lock) > 0 ) {
          tk_sig_sem(lock->id, 1);
        }
    }

Inline INT Dec( FastLock *lock ) {
    UINT     imask; INT    c;
    DI(imask);
    c = lock->cnt--;
    EI(imask);
    return c;
}
```

高速ロック･マルチロックのAPI(1/10)

CreateLock　　高速ロックの生成

C言語インタフェース
```
ER ercd = CreateLock ( FastLock *lock, CONST UB *name );
```

パラメータ

FastLock*	lock	高速ロックの管理ブロック
CONST UB*	name	高速ロックの名前

リターンパラメータ

ER	ercd	エラーコード

高速ロック・マルチロックのAPI(2/10)

DeleteLock　　高速ロックの削除

C言語インタフェース
void DeleteLock (FastLock *lock);

パラメータ
FastLock*　　lock　　高速ロックの管理ブロック

リターンパラメータ
なし

高速ロック・マルチロックのAPI(3/10)

Lock　　高速ロックのロック操作

C言語インタフェース
void Lock (FastLock *lock);

パラメータ
FastLock*　　lock　　高速ロックの管理ブロック

リターンパラメータ
なし

高速ロック・マルチロックのAPI(4/10)

Unlock　　高速ロックのロック解除操作

C言語インタフェース
void Unlock (FastLock *lock);

パラメータ
FastLock*　　lock　　高速ロックの管理ブロック

リターンパラメータ
なし

高速ロック・マルチロックのAPI(5/10)

CreateMLock　　高速マルチロックの生成

C言語インタフェース
ER ercd = CreateMLock (FastMLock *lock, CONST UB *name);

パラメータ

FastMLock*	lock	高速マルチロックの管理ブロック
CONST UB*	name	高速マルチロックの名前

リターンパラメータ

ER	ercd	エラーコード

高速ロック・マルチロックのAPI(6/10)

DeleteMLock　　高速マルチロックの削除

C言語インタフェース
ER ercd = DeleteMLock (FastMLock *lock);

パラメータ

FastMLock*	lock	高速マルチロックの管理ブロック

リターンパラメータ

ER	ercd	エラーコード

高速ロック・マルチロックのAPI(7/10)

MLock　　高速マルチロックのロック操作

C言語インタフェース
ER ercd = MLock (FastMLock *lock, INT no);

パラメータ

FastMLock*	lock	高速マルチロックの管理ブロック
INT	no	ロック番号

リターンパラメータ

ER	ercd	エラーコード

高速ロック・マルチロックのAPI(8/10)

MLockTmo　高速マルチロックのロック操作(タイムアウト指定付き)

C言語インタフェース
ER ercd = MLockTmo (FastMLock *lock, INT no, TMO tmout);

パラメータ

FastMLock*	lock	高速マルチロックの管理ブロック
INT	no	ロック番号
TMO	tmout	タイムアウト指定(ミリ秒)

リターンパラメータ

ER	ercd	エラーコード

高速ロック・マルチロックのAPI(9/10)

MLockTmo_u　　高速マルチロックのロック操作(タイムアウト指定付き、マイクロ秒単位)

C言語インタフェース

```
ER ercd = MLockTmo_u ( FastMLock *lock, INT no, TMO_U tmout_u );
```

パラメータ

FastMLock*	lock	高速マルチロックの管理ブロック
INT	no	ロック番号
TMO_U	tmout_u	タイムアウト指定(マイクロ秒)

リターンパラメータ

ER	ercd	エラーコード

高速ロック・マルチロックのAPI(10/10)

MUnlock　　高速マルチロックのロック解除操作

C言語インタフェース
ER ercd = MUnlock (FastMLock *lock, INT no);

パラメータ

FastMLock*	lock	高速マルチロックの管理ブロック
INT	no	ロック番号

リターンパラメータ

ER	ercd	エラーコード

T-Kernel 2.0

仕様書の改善

仕様書の改善

- 電子化に配慮した仕様書
  - 元の仕様書はXMLで記述
  - 通常はHTMLやPDFに変換したものを利用
  - 開発環境のヘルプ情報の追加に利用するなど、高度な応用が可能
- T-Kernel/SMのAPIの書式を他の部分と統一
- 「利用可能なコンテキスト」の明記

タスク部	準タスク部	タスク独立部
○	○	×

- 説明が分かりにくかった部分の補足や修正
  - 特にサブシステム管理機能やT-Kernel/SM
- ページ数が大幅に増加
  - T-Kernel 1.0仕様書 264ページ
→T-Kernel 2.0仕様書 515ページ

HTML版とXML版の仕様書


```
<refentry id="tk_slp_tsk_u">
  <refentryinfo>
    <date>2010-07-12</date>
  </refentryinfo>
  <refmeta>
    <refentrytitle>tk_slp_tsk_u</refentrytitle>
    <manvolnum>2</manvolnum>
  </refmeta>
  <refnamediv>
    <refname>tk_slp_tsk_u</refname>
    <refpurpose>自タスクを起床待ち状態へ移行(マイクロ秒単位)</refpurpose>
  </refnamediv>
  <refsect1>
    <title>C言語インタフェース</title>
    <funcsynopsis>
      <funcsynopsisinfo>#include &lt;tk/tkernel.h&gt;</funcsynopsisinfo>
      <funcprototype>
        <funcdef>ER ercd = <function>tk_slp_tsk_u</function>
        </funcdef>
        <paramdef>TMO_U <parameter>tmout_u</parameter>
        </paramdef>
      </funcprototype>
    </funcsynopsis>
  </refsect1>
  <refsect1>
    <title>パラメータ</title>
    <informaltable frame="none" pgwide="1">
      <tgroup cols="4" align="left" colsep="0" rowsep="0">
```

T-Kernel 2.0

ワンストップサービス

ワンストップサービス

- T-Kernel 2.0もオープンソース
  - T-Engineリファレンスボードで動作するソースを公開
  - 組込み向けに利用しやすいT-License 2.0
- 公開範囲の拡大とワンパッケージ化
  - T-Monitor、一部のデバイスドライバ、開発環境、PC上のシミュレータも含めて一括公開
- ucodeを用いたトレーサビリティシステム
- 2011年5月17日から公開開始

T-Engine Forum Download Center

提供されるソフトウェア(1/3)

- T-Kernel 2.0
  - 従来の T-Kernel 1.0 を包含
  - T-Kernel 2.0 の追加機能(マイクロ秒単位の時間管理機能など)
  - 動作対象機種の tef_em1d に合わせてARM11コア依存部を追加

- T-Monitor
  - ハードウェアの初期設定
  - T-Kernelのブート処理
  - 割込みや例外のハンドリング
  - ハードウェア階層の対話型デバッグ機能
    - メモリやレジスタの参照

提供されるソフトウェア(2/3)

- デバイスドライバ
  - 時計(RTC)
  - シリアルコンソール
  - タッチパネル
  - スクリーン(LCD)
  - システムディスク

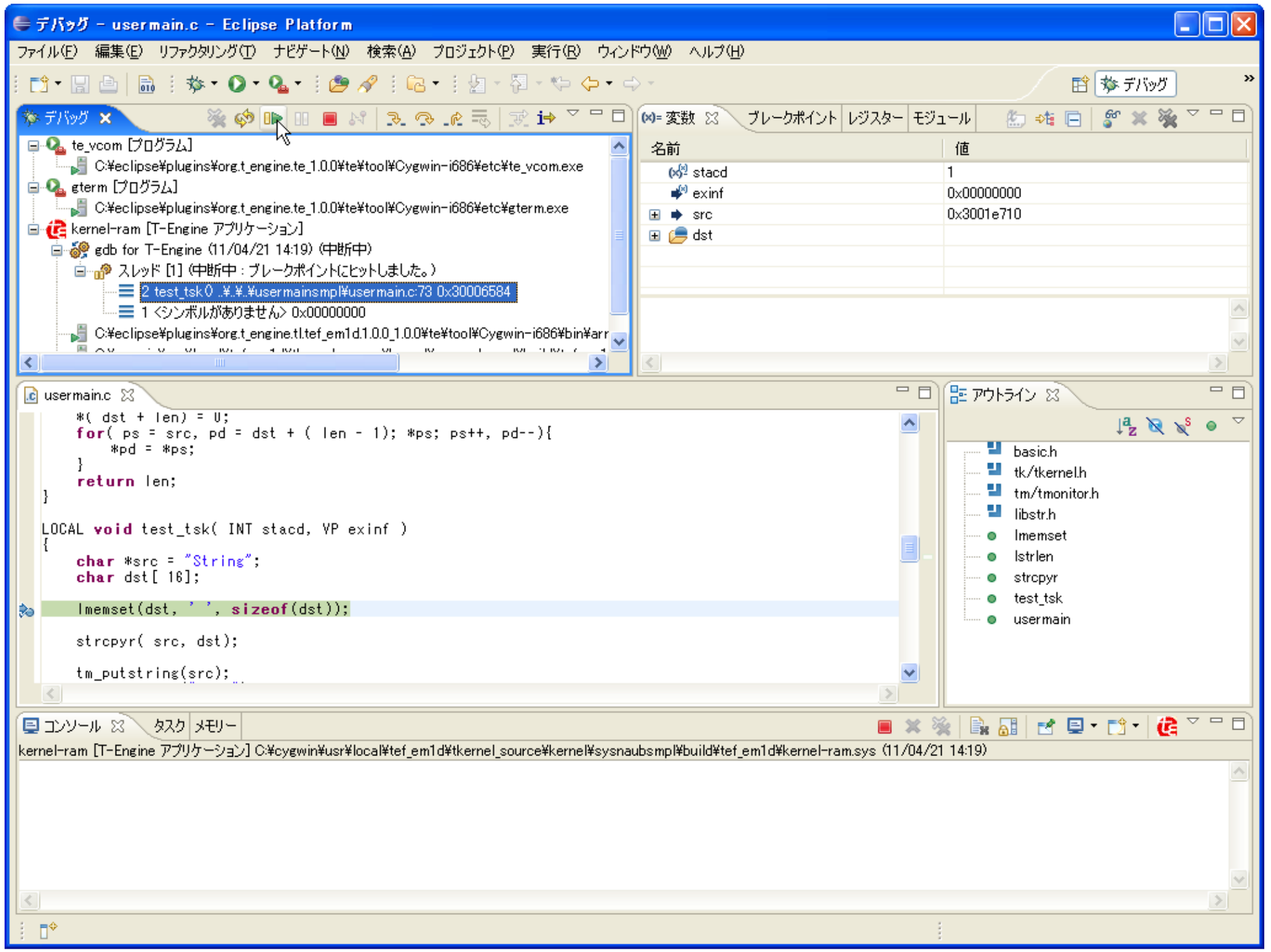
提供されるソフトウェア(3/3)

- Eclipse開発環境
  - Windows PC上で動作
  - コンパイルやビルド
  - 実機へのプログラム転送と実行
  - デバッグ機能
    - ブレークポイントの設定
    - 変数値の参照や変更など

  ※ このほか、Linuxのコマンドベース(非GUI)による開発も可能

- QEMUによるエミュレータ
  - Windows PC上で動作
    - ハードウェア(実機)がなくても開発できる
  - CPUおよびボード搭載の各デバイスに対応
    - タイマ、microSDカード、I2C、UART(シリアル)、USB、RTC、LCD画面、タッチパネル、LANなど

Eclipse開発環境



- ソースパッケージの構成

```
tkernel_source
    |----config       設定情報
    |----drv          デバイスドライバ
    |----include      インクルードファイル
    |----kernel       T-Kernel 2.0本体
    |----lib          ライブラリ
    |----monitor      T-Monitor
```

- T-Kernel 2.0本体のソース

```
tkernel_source
    |-kernel
        |--sysdepend              ハードウェア依存部
        |    |--cpu
        |    |    |--em1d         CPU依存部
        |    |--device
        |         |--tef_em1d     デバイス依存部
        |--sysinit                初期化
        |--sysmain                システムメイン
        |--sysmgr                 T-Kernel/SM
        |--tkernel                T-Kernel/OS,/DS
        |    |--build             ビルド(make)用
        |    |    |--tef_em1d     tef_em1dでのビルド(make)用
        |    |--src               ソース
        |--usermain               アプリ利用時のメイン
        |--usermain_drv           ドライバ利用時のメイン
```

- T-Monitorのソース

```
tkernel_source
    |--monitor
        |--cmdsvc                コマンド,SVC処理
        |    |--src
        |         |--armv6       ARMv6依存部
        |--driver                T-Monitor用ドライバ
        |    |--flash            Flash ROM
        |    |--memdisk          メモリディスク
        |    |--sio              シリアルI/O
        |--hwdepend              ハードウェア依存部
        |    |--tef_em1d         tef_em1d依存部
        |--tmmain                T-Monitorメイン
            |--build             ビルド(make)用
            |--src               ソース
```

- デバイスドライバのソース

```
tkernel_source
    |--drv
        |--tef_em1d                 機種名を表わすディレクトリ
            |--clk                  時計(RTC)
            |--console              シリアルコンソール
            |--kbpd                 KB/PD(タッチパネル)
            |--lowkbpd              KB/PD実IO
            |--screen               スクリーン(LCD)
            |--sysdsk               システムディスク
                |--build            ビルド(make)用
                |--src              ソース
```

付録A

T-Kernel/OSのシステムコール

T-Kernel/OSの機能

[1] タスク管理機能
[2] タスク付属同期機能
[3] タスク例外処理機能
[4] 同期・通信機能
[5] 拡張同期・通信機能
[6] メモリプール管理機能
[7] 時間管理機能
[8] 割込み管理機能
[9] システム状態管理機能
[10] サブシステム管理機能

[1] タスク管理機能

- tk_cre_tsk　　タスク生成
- tk_del_tsk　　タスク削除
- tk_sta_tsk　　タスク起動
- tk_ext_tsk　　自タスク終了
- tk_exd_tsk　　自タスクの終了と削除
- tk_ter_tsk　　他タスク強制終了
- tk_chg_pri　　タスク優先度変更
- tk_chg_slt　　タスクスライスタイム変更
- tk_chg_slt_u　　タスクスライスタイム変更(マイクロ秒単位)

[1] タスク管理機能

- tk_get_tsp　　タスク固有空間の参照
- tk_set_tsp　　タスク固有空間の設定
- tk_get_rid　　タスクの所属リソースグループの参照
- tk_set_rid　　タスクの所属リソースグループの設定
- tk_get_reg　　タスクレジスタの取得
- tk_set_reg　　タスクレジスタの設定
- tk_get_cpr　　コプロセッサのレジスタの取得
- tk_set_cpr　　コプロセッサのレジスタの設定
- tk_inf_tsk　　タスク統計情報参照
- tk_inf_tsk_u　　タスク統計情報参照(マイクロ秒単位)
- tk_ref_tsk　　タスク状態参照
- tk_ref_tsk_u　　タスク状態参照(マイクロ秒単位)

[2] 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
- tk_slp_tsk_u　　自タスクを起床待ち状態へ移行(マイクロ秒単位)
- tk_wup_tsk　　他タスクの起床
- tk_can_wup　　タスクの起床要求を無効化
- tk_rel_wai　　他タスクの待ち状態解除
- tk_sus_tsk　　他タスクを強制待ち状態へ移行
- tk_rsm_tsk　　強制待ち状態のタスクを再開
- tk_frsm_tsk　　強制待ち状態のタスクを強制再開
- tk_dly_tsk　　タスク遅延
- tk_dly_tsk_u　　タスク遅延(マイクロ秒単位)

[2] タスク付属同期機能

- tk_sig_tev　　タスクイベントの送信
- tk_wai_tev　　タスクイベント待ち
- tk_wai_tev_u　　タスクイベント待ち(マイクロ秒単位)
- tk_dis_wai　　タスク待ち状態の禁止
- tk_ena_wai　　タスク待ち禁止の解除

[3] タスク例外処理機能

- tk_def_tex　　タスク例外ハンドラの定義
- tk_ena_tex　　タスク例外の許可
- tk_dis_tex　　タスク例外の禁止
- tk_ras_tex　　タスク例外を発生
- tk_end_tex　　タスク例外ハンドラの終了
- tk_ref_tex　　タスク例外の状態参照

[4] 同期・通信機能（セマフォ）

- tk_cre_sem　　セマフォ生成
- tk_del_sem　　セマフォ削除
- tk_sig_sem　　セマフォ資源返却
- tk_wai_sem　　セマフォ資源獲得
- tk_wai_sem_u　セマフォ資源獲得(マイクロ秒単位)
- tk_ref_sem　　セマフォ状態参照

[4] 同期・通信機能（イベントフラグ）

- tk_cre_flg　　イベントフラグ生成
- tk_del_flg　　イベントフラグ削除
- tk_set_flg　　イベントフラグのセット
- tk_clr_flg　　イベントフラグのクリア
- tk_wai_flg　　イベントフラグ待ち
- tk_wai_flg_u　　イベントフラグ待ち(マイクロ秒単位)
- tk_ref_flg　　イベントフラグ状態参照

[4] 同期・通信機能（メールボックス）

- tk_cre_mbx　　メールボックス生成
- tk_del_mbx　　メールボックス削除
- tk_snd_mbx　　メールボックスへ送信
- tk_rcv_mbx　　メールボックスから受信
- tk_rcv_mbx_u　メールボックスから受信(マイクロ秒単位)
- tk_ref_mbx　　メールボックス状態参照

[5] 拡張同期・通信機能（ミューテックス）

- tk_cre_mtx　　ミューテックス生成
- tk_del_mtx　　ミューテックス削除
- tk_loc_mtx　　ミューテックスのロック
- tk_loc_mtx_u　　ミューテックスのロック(マイクロ秒単位)
- tk_unl_mtx　　ミューテックスのアンロック
- tk_ref_mtx　　ミューテックス状態参照

[5] 拡張同期・通信機能（メッセージバッファ）

- tk_cre_mbf　　メッセージバッファ生成
- tk_del_mbf　　メッセージバッファ削除
- tk_snd_mbf　　メッセージバッファへ送信
- tk_snd_mbf_u　　メッセージバッファへ送信(マイクロ秒単位)
- tk_rcv_mbf　　メッセージバッファから受信
- tk_rcv_mbf_u　　メッセージバッファから受信(マイクロ秒単位)
- tk_ref_mbf　　メッセージバッファ状態参照

[5] 拡張同期・通信機能（ランデブ）

- tk_cre_por　　ランデブポート生成
- tk_del_por　　ランデブポート削除
- tk_cal_por　　ランデブポートに対するランデブの呼出
- tk_cal_por_u　　ランデブポートに対するランデブの呼出（マイクロ秒単位）
- tk_acp_por　　ランデブポートに対するランデブ受付
- tk_acp_por_u　　ランデブポートに対するランデブ受付（マイクロ秒単位）
- tk_fwd_por　　ランデブポートに対するランデブ回送
- tk_rpl_rdv　　ランデブ返答
- tk_ref_por　　ランデブポート状態参照

[6] メモリプール管理機能(固定長メモリプール)

- tk_cre_mpf　　固定長メモリプール生成
- tk_del_mpf　　固定長メモリプール削除
- tk_get_mpf　　固定長メモリブロック獲得
- tk_get_mpf_u　　固定長メモリブロック獲得(マイクロ秒単位)
- tk_rel_mpf　　固定長メモリブロック返却
- tk_ref_mpf　　固定長メモリプール状態参照

[6] メモリプール管理機能(可変長メモリプール)

- tk_cre_mpl　　可変長メモリプール生成
- tk_del_mpl　　可変長メモリプール削除
- tk_get_mpl　　可変長メモリブロック獲得
- tk_get_mpl_u　　可変長メモリブロック獲得(マイクロ秒単位)
- tk_rel_mpl　　可変長メモリブロック返却
- tk_ref_mpl　　可変長メモリプール状態参照

[7] 時間管理機能(システム時刻管理)

- tk_set_tim　　システム時刻設定
- tk_set_tim_u　　システム時刻設定(マイクロ秒単位)
- tk_get_tim　　システム時刻参照
- tk_get_tim_u　　システム時刻参照(マイクロ秒単位)
- tk_get_otm　　システム稼働時間参照
- tk_get_otm_u　　システム稼働時間参照(マイクロ秒単位)

[7] 時間管理機能(周期ハンドラ)

- tk_cre_cyc　　周期ハンドラの生成
- tk_cre_cyc_u　　周期ハンドラの生成(マイクロ秒単位)
- tk_del_cyc　　周期ハンドラの削除
- tk_sta_cyc　　周期ハンドラの動作開始
- tk_stp_cyc　　周期ハンドラの動作停止
- tk_ref_cyc　　周期ハンドラ状態参照
- tk_ref_cyc_u　　周期ハンドラ状態参照(マイクロ秒単位)

[7] 時間管理機能(アラームハンドラ)

- tk_cre_alm　　アラームハンドラの生成
- tk_del_alm　　アラームハンドラの削除
- tk_sta_alm　　アラームハンドラの動作開始
- tk_sta_alm_u　　アラームハンドラの動作開始(マイクロ秒単位)
- tk_stp_alm　　アラームハンドラの動作停止
- tk_ref_alm　　アラームハンドラ状態参照
- tk_ref_alm_u　　アラームハンドラ状態参照(マイクロ秒単位)

[8] 割込み管理機能

- tk_def_int　　割込みハンドラ定義
- tk_ret_int　　割込みハンドラから復帰

[9] システム状態管理機能

- tk_rot_rdq　　タスクの優先順位の回転
- tk_get_tid　　実行状態タスクのタスクID参照
- tk_dis_dsp　　ディスパッチ禁止
- tk_ena_dsp　　ディスパッチ許可
- tk_ref_sys　　システム状態参照
- tk_set_pow　　省電力モード設定
- tk_ref_ver　　バージョン参照

[10] サブシステム管理機能

- tk_def_ssy　　サブシステム定義
- tk_sta_ssy　　スタートアップ関数呼出
- tk_cln_ssy　　クリーンアップ関数呼出
- tk_evt_ssy　　イベント処理関数呼出
- tk_ref_ssy　　サブシステム定義情報の参照
- tk_cre_res　　リソースグループの生成
- tk_del_res　　リソースグループの削除
- tk_get_res　　リソース管理ブロックの取得

付録B

T-Kernel/SMの拡張SVC・ライブラリ

T-Kernel/SMの機能

[1] システムメモリ管理機能

[2] アドレス空間管理機能

[3] デバイス管理機能

[4] 割込み管理機能

[5] I/Oポートアクセスサポート機能

[6] 省電力機能

[7] システム構成情報管理機能

[8] メモリキャッシュ制御機能

[9] 物理タイマ機能

[10] ユーティリティ機能

[1] システムメモリ管理機能
（システムメモリ割当て）

- tk_get_smb　　システムメモリの割当て
- tk_rel_smb　　システムメモリの解放
- tk_ref_smb　　システムメモリ情報取得

[1] システムメモリ管理機能
（メモリ割当てライブラリ）

- Vmalloc　　非常駐メモリの割当て
- Vcalloc　　非常駐メモリの割当て
- Vrealloc　　非常駐メモリの再割当て
- Vfree　　非常駐メモリの解放
- Kmalloc　　常駐メモリの割当て
- Kcalloc　　常駐メモリの割当て
- Krealloc　　常駐メモリの再割当て
- Kfree　　常駐メモリの解放

[2] アドレス空間管理機能
（アドレス空間設定）

- SetTaskSpace　　タスクのアドレス空間設定

[2] アドレス空間管理機能
（アドレス空間チェック）

- ChkSpaceR　　メモリ読込みアクセス権の検査
- ChkSpaceRW　　メモリ読込み書込みアクセス権の検査
- ChkSpaceRE　　メモリ読込みアクセス権および実行権の検査
- ChkSpaceBstrR　　文字列読込みアクセス権の検査
- ChkSpaceBstrRW　　文字列読込み書込みアクセス権の検査
- ChkSpaceTstrR　　TRONコード文字列読込みアクセス権の検査
- ChkSpaceTstrRW　　TRONコード文字列読込み書込みアクセス権の検査

[2] アドレス空間管理機能
（論理アドレス空間管理）

- LockSpace　　メモリ領域のロック
- UnlockSpace　　メモリ領域のアンロック
- CnvPhysicalAddr　　物理アドレスの取得
- MapMemory　　メモリのマップ
- UnmapMemory　　メモリのアンマップ
- GetSpaceInfo　　アドレス空間の各種情報の取得
- SetMemoryAccess　　メモリアクセス権の設定

[3] デバイス管理機能
（デバイスの入出力操作）

- tk_opn_dev　　デバイスのオープン
- tk_cls_dev　　デバイスのクローズ
- tk_rea_dev　　デバイスの読込み開始
- tk_rea_dev_du　　デバイスの読込み開始
  （64ビットマイクロ秒単位）
- tk_srea_dev　　デバイスの同期読込み
- tk_srea_dev_d　　デバイスの同期読込み(64ビット)
- tk_wri_dev　　デバイスの書込み開始
- tk_wri_dev_du　　デバイスの書込み開始
  （64ビットマイクロ秒単位）
- tk_swri_dev　　デバイスの同期書込み
- tk_swri_dev_d　　デバイスの同期書込み(64ビット)

[3] デバイス管理機能
（デバイスの入出力操作）

- tk_wai_dev　　デバイスの要求完了待ち
- tk_wai_dev_u　　デバイスの要求完了待ち(マイクロ秒単位)
- tk_sus_dev　　デバイスのサスペンド
- tk_get_dev　　デバイスのデバイス名取得
- tk_ref_dev　　デバイスのデバイス情報取得
- tk_oref_dev　　デバイスのデバイス情報取得
- tk_lst_dev　　登録済みデバイス一覧の取得
- tk_evt_dev　　デバイスにドライバ要求イベントを送信

[3] デバイス管理機能
（デバイスドライバの登録）

- tk_def_dev　　デバイスの登録
- tk_ref_idv　　デバイス初期情報の取得

[3] デバイス管理機能
（デバイスドライバインタフェース）

- openfn　　オープン関数
- closefn　　クローズ関数
- execfn　　処理開始関数
- waitfn　　完了待ち関数
- abortfn　　中止処理関数
- eventfn　　イベント関数

[4] 割込み管理機能（CPU割込み制御）

- DI　　外部割込み禁止
- EI　　外部割込み許可
- isDI　　外部割込み禁止状態の取得

[4] 割込み管理機能（割込みコントローラ制御）

- DINTNO　　割込みベクタから割込みハンドラ番号へ変換
- EnableInt　　割込み許可
- DisableInt　　割込み禁止
- ClearInt　　割込み発生のクリア
- EndOfInt　　割込みコントローラにEOI発行
- CheckInt　　割込み発生の検査
- SetIntMode　　割込みモード設定

[5] I/Oポートアクセスサポート機能
（I/Oポートアクセス）

- out_b　　I/Oポート書込み(バイト)
- out_h　　I/Oポート書込み(ハーフワード)
- out_w　　I/Oポート書込み(ワード)
- out_d　　I/Oポート書込み(ダブルワード)
- in_b　　I/Oポート読込み(バイト)
- in_h　　I/Oポート読込み(ハーフワード)
- in_w　　I/Oポート読込み(ワード)
- in_d　　I/Oポート読込み(ダブルワード)

[5] I/Oポートアクセスサポート機能（微小待ち）

- WaitUsec　　微小待ち(マイクロ秒)
- WaitNsec　　微小待ち(ナノ秒)

[6] 省電力機能

- low_pow　　システムを低消費電力モードに移行
- off_pow　　システムをサスペンド状態に移行

[7] システム構成情報管理機能
（システム構成情報の取得）

- tk_get_cfn　　システム構成情報から数値列取得
- tk_get_cfs　　システム構成情報から文字列取得

[8] メモリキャッシュ制御機能

- SetCacheMode　　キャッシュモードの設定
- ControlCache　　キャッシュの制御

[9] 物理タイマ機能

- StartPhysicalTimer　物理タイマの動作開始
- StopPhysicalTimer　物理タイマの動作停止
- GetPhysicalTimerCount　物理タイマのカウント値取得
- DefinePhysicalTimerHandler　物理タイマハンドラ定義
- GetPhysicalTimerConfig　物理タイマのコンフィグレーション情報取得

[10] ユーティリティ機能（オブジェクト名設定）

- SetOBJNAME　　オブジェクト名設定

[10] ユーティリティ機能（高速ロック・マルチロックライブラリ）

- CreateLock　高速ロックの生成
- DeleteLock　高速ロックの削除
- Lock　高速ロックのロック操作
- Unlock　高速ロックのロック解除操作
- CreateMLock　高速マルチロックの生成
- DeleteMLock　高速マルチロックの削除
- MLock　高速マルチロックのロック操作
- MLockTmo　高速マルチロックのロック操作(タイムアウト指定付き)
- MLockTmo_u　高速マルチロックのロック操作(タイムアウト指定付き、マイクロ秒単位)
- MUnlock　高速マルチロックのロック解除操作

付録C

T-Kernel/DSのシステムコール

T-Kernel/DSの機能

[1] カーネル内部状態取得機能

[2] 実行トレース機能

[1] カーネル内部状態取得機能

- td_lst_tsk　　タスクIDのリスト参照
- td_lst_sem　　セマフォIDのリスト参照
- td_lst_flg　　イベントフラグIDのリスト参照
- td_lst_mbx　　メールボックスIDのリスト参照
- td_lst_mtx　　ミューテックスIDのリスト参照
- td_lst_mbf　　メッセージバッファIDのリスト参照
- td_lst_por　　ランデブポートIDのリスト参照
- td_lst_mpf　　固定長メモリプールIDのリスト参照
- td_lst_mpl　　可変長メモリプールIDのリスト参照
- td_lst_cyc　　周期ハンドラIDのリスト参照
- td_lst_alm　　アラームハンドラIDのリスト参照
- td_lst_ssy　　サブシステムIDのリスト参照

[1] カーネル内部状態取得機能

- td_rdy_que　　タスクの優先順位の参照
- td_sem_que　　セマフォの待ち行列の参照
- td_flg_que　　イベントフラグの待ち行列の参照
- td_mbx_que　　メールボックスの待ち行列の参照
- td_mtx_que　　ミューテックスの待ち行列の参照
- td_smbf_que　　メッセージバッファの送信待ち行列の参照
- td_rmbf_que　　メッセージバッファの受信待ち行列の参照
- td_cal_que　　ランデブ呼出待ち行列の参照
- td_acp_que　　ランデブ受付待ち行列の参照
- td_mpf_que　　固定長メモリプールの待ち行列の参照
- td_mpl_que　　可変長メモリプールの待ち行列の参照

[1] カーネル内部状態取得機能

- td_ref_tsk　　タスク状態参照
- td_ref_tsk_u　　タスク状態参照(マイクロ秒単位)
- td_ref_tex　　タスク例外の状態参照
- td_ref_sem　　セマフォ状態参照
- td_ref_flg　　イベントフラグ状態参照
- td_ref_mbx　　メールボックス状態参照
- td_ref_mtx　　ミューテックス状態参照
- td_ref_mbf　　メッセージバッファ状態参照
- td_ref_por　　ランデブポート状態参照
- td_ref_mpf　　固定長メモリプール状態参照
- td_ref_mpl　　可変長メモリプール状態参照

[1] カーネル内部状態取得機能

- td_ref_cyc　　周期ハンドラ状態参照
- td_ref_cyc_u　　周期ハンドラ状態参照(マイクロ秒単位)
- td_ref_alm　　アラームハンドラ状態参照
- td_ref_alm_u　　アラームハンドラ状態参照(マイクロ秒単位)
- td_ref_sys　　システム状態参照
- td_ref_ssy　　サブシステム定義情報の参照
- td_inf_tsk　　タスク統計情報参照
- td_inf_tsk_u　　タスク統計情報参照(マイクロ秒単位)
- td_get_reg　　タスクレジスタの参照
- td_set_reg　　タスクレジスタの設定
- td_get_tim　　システム時刻参照
- td_get_tim_u　　システム時刻参照(マイクロ秒単位)

[1] カーネル内部状態取得機能

- td_get_otm　　システム稼働時間参照
- td_get_otm_u　　システム稼働時間参照(マイクロ秒単位)
- td_ref_dsname　　DSオブジェクト名称の参照
- td_set_dsname　　DSオブジェクト名称の設定

[2] 実行トレース機能

- td_hok_svc　　システムコール・拡張SVCのフックルーチン定義
- td_hok_dsp　　タスクディスパッチのフックルーチン定義
- td_hok_int　　割込みハンドラのフックルーチン定義

【実習】T-Kernel 2.0入門テキスト「T-Kernel 2.0の概要」

著者　T-Engine Forum