T-Kernel入門
～ユーザープログラムの作成とカスタマイズ～

T-Engineフォーラムは、組込みミドルウェア流通を最大の目的とする、T-Engineプロジェクトのいわば中核となるリアルタイムOS「T-Kernel」を2004年1月に公開しました[注1)]。T-Kernelの取得方法やビルドの方法については本誌VOL.86の「T-Kernelソースコード公開」にて説明されていますので、今回はT-Kernel上でのユーザープログラムの作成方法とT-Kernelのカスタマイズ（システムコールの追加・削除）方法を中心に説明します。

T-Kernelの取得・構築方法

まず、T-Kernelの取得ならびに構築方法について説明します。

T-KernelのソースコードはT-Engineフォーラムのウェブページ（http://www.t-engine.org/）から取得できます。ウェブページの「T-Kernel利用申し込み」を選び、ライセンスに同意し、必要事項を記入すると、ダウンロードページを示したメールが送られてきます。そのメールに記載されたページにアクセスすると、T-Kernelのソースコードをダウンロードすることができます。

ダウンロードしたソースコードから、実際にカーネルを構築し動かしてみましょう。ダウンロードしたT-Kernelをそのまま動かすにはT-Engine開発キットが必要です。ここでは開発キットに付属しているGNUベースの開発環境[注2)]を使用し、ホームディレクトリの直下でソースコードを解凍してSH7727用のT-Kernelを構築することにします。

表1　T-Kernel構築に必要な環境変数

環境変数名	本文中での設定値	補足
BD	~/tkernel_source	T-Kernel ソースコードのトップ
GNUs	/usr	GNU 関連ツール
GNU_BD	/usr/local/te/tool/Linux-i686	クロス開発用 GNU 関連ツールのベースディレクトリ
GNUsh	${GNU_BD}/sh-unknown-tkernel	SH7727 用のコンパイラなど
GCC_EXEC_PREFIX	${GNU_BD}/lib/gcc-lib	gcc 関連ディレクトリ

```
% cd
% tar xzvf tkernel.1.01.00.tar.gz
```

解凍に成功すると、tkernel_sourceというディレクトリが作成されます。これを環境変数BDに登録します。このほか、表1に示すような環境変数を設定しておきます。

T-Kernelは、ライブラリ、本体の順にコンパイルします。ライブラリは~/tkernel_source/lib/build/std_sh7727ディレクトリで、本体は~/tkernel_source/tkernel/sysmain/build/std_sh7727ディレクトリでmakeを実行して構築します。

注1）T-Kernelは2004年7月にバージョン1.01.00にアップデートされています。
注2）ここでは、T-Engine/SH7727開発キット（パーソナルメディア社提供）を使用しています。

```
% cd ~/tkernel_source/lib/
build/std_sh7727
% make
% cd ~/tkernel_source/
tkernel/sysmain/build/std_sh7727
% make
```

makeが終了しますと、カレントディレクトリ上にkernel-rom.motというファイルが生成されます。これがT-Kernelのオブジェクトファイルです。

次にRomInfoのオブジェクトを生成します。~/tkernel_source/config/build/std_sh7727ディレクトリ上でmakeを実行するとRomInfoのオブジェクトファイルであるrominfo.motが生成されます。

最後に生成したrominfo.mot、kernel-rom.motの2つのオブジェクトをこの順にT-EngineにフラッシュROMに転送します。フラッシュROMへの書き込みには通常T-MonitorのFlashLoad（FLLO）コマンドを使用します。T-Engineの種類やバージョンによって異なる場合がありますので、詳しくは開発キットのマニュアルをご覧ください。

T-Kernel上でのユーザープログラムの作成方法

ユーザープログラムは、基本的に~/tkernel_source/kernel/usermain/ディレクトリに作成します。

T-Kernelはカーネルの初期化を終えると、同ディレクトリのusermain.cにあるusermain()関数を呼び出します。T-Kernelをダウンロードして展開した状態では、このusermain()関数はリスト1のように定義されています。これは、“Push any key to shutdown the T-Kernel.”というメッセージをデバッグメッセージに出力し、デバッグコンソール上に入力からを待ち、入力があればT-Kernelを終了するというものです。

では、T-Kernel上でパズルゲームを作ってみましょう。図1のような構成で作ることにします。

初期タスクはT-Kernelが起動するusermain()関数です。リスト1のコメントにあるとおり、usermain()関数で行う処理はユーザープログラムの初期タスクを起動するだけにとどめることが推奨されています。このため、初期タスクはメインタスクを起動してその終了を待ちます。メインタスクは計時用の周期ハンドラとT-Engineにあるキー・タッチパネル監視タスクを起動します。周期ハンドラは100msごとに起動され、1秒ごとに画面を書き換えます。キー・タッチパネル監視タスクは、T-Engineに備え付けられているボタン（以下キーといいます）やタッチパネルからの入力を待ち、それをメインタスクに伝えるタスクです。キー・タッチパネル監視タスクからメインタスクにイベントを通知するために、メッセージバッファを使用します。

初期タスクは優先度138（最高優先度1、最低優先度140）という、非常に低い優先度で動きます。この優先度より高い優先度のタスクを起動すると初期タスクは実行可能

リスト1　ユーザープログラムのメイン関数usermain()（一部抜粋）

```
/*
 * Entry routine for the user application.
 * At this point, Initialize and start the user application.
 *
 * Entry routine is called from the initial task for Kernel,
 * so system call for stopping the task should not be issued
 * from the contexts of entry routine.
 * We recommend that:
 * (1)' usermain()'  only generates the user initial task.
 * (2)initialize and start the user application by the user
 * initial task.
 */

EXPORT INT usermain( void )
{
    tm_putstring( “Push any key to shutdown the T-Kernel.\n” );
    tm_getchar(-1);

    return 0;
}
```

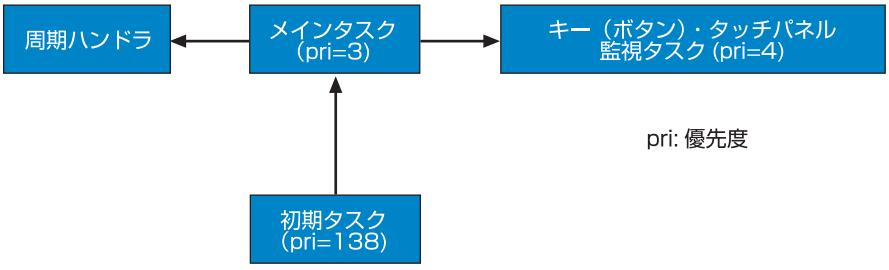

図1　パズルゲームの構成

状態のまま待機します。このあと初期タスクが実行状態になりますと、関数usermain()を抜け、T-Kernelはそのまま終了処理に移行します。そのため、T-Kernelを終了させるとき以外は初期タスクが実行状態にならないようにする必要があります。

リスト2はパズルゲームのusermain()関数です。ここでは、ユーザープログラムのメインタスクを起動後、tk_slp_tsk()システムコールを用いて初期タスクを待ち状態にしています。これは初期タスクが実行可能状態のままメインタスクがセマフォ待ちなどにより待ち状態に移行してしまい、初期タスクが実行状態になることを防ぐための措置です。ユーザープログラムが終了すると初期タスクを起床させ、正常にT-Kernel終了処理に移ります。

リスト2　パズルゲームのusermain()関数

```
/* 初期タスク */
IMPORT INT usermain( void );
/* ユーザー初期タスクメイン処理 */
IMPORT INT main_task( INT stcd, VP exinf );

/* メインタスク定義 */
EXPORT const T_CTSK mtask = {
       NULL,                    /* exinf */
       TA_HLNG|TA_RNG0,         /* tskatr */
       (FP)main_task,           /* task */
       3,                       /* itskpri */
       8*1024,                  /* stksz */
       0,                       /* sstksz */
       NULL,                    /* stkptr */
       NULL,                    /* uatb */
       0,                       /* lsid */
       0                        /* resid */
};

/* itskid - 初期タスク
 * mtskid - メインタスク、メイン処理タスク
 * ktskid - キー・タッチパネル監視タスク */
EXPORT ID itskid, mtskid, ktskid;

/* 初期タスク */
EXPORT INT usermain( void )
{
  /* 自タスクIDを取得しておく */
  itskid = tk_get_tid();

   /* メインタスクを生成し起動する */
   mtskid = tk_cre_tsk((T_CTSK*)&mtask);
   tk_sta_tsk(mtskid, 0);

   /* メインタスクを起動後、自分は眠る
    * メインタスクが終了時に起床してくれるのを待つ */
   tk_slp_tsk(TMO_FEVR);

   /* 終了 */
   return -1;
}
```

リスト3はメインタスク部分です。ここでは、周期ハンドラとキー・タッチパネル監視タスクを定義し、タッチパネル監視タスクを起動します。周期ハンドラの起動・停止はゲーム本体であるgame_proc()関数で行います。

このパズルゲームは、T-Engine開発キットに付属のサンプルコードを一部利用して実装しました。これをコンパイルして実行すると、図2のような画面が現れます。画面上には赤と青のパネルがあり、パネルをクリックすると自分と周囲のパネルの色が逆転します。すべての同じ色にするとクリアです。画面上の水色の四角はカーソルです。SW1のキーを用いてこれを動かし、SW2のキーを押して逆転させるパネルを選択することもできます。

●ユーザープログラムを複数のファイルに分割した場合の構築法

なお、ユーザープログラムが大きくなり、複数のファイルに分割する場合は、~/tkernel_source/kernel/usermain/Makefile.usermainというファイルを編集します。このファイルにはコンパイルするユーザープログラムのソースコードを示すSRCという変数があります。ダウンロードしたファイルではリスト4のようになっています。これにファイルsrc1.c、src2.cを追加する場合は、Makefile.usermainファイルのSRC変数定義をリスト5のように書き換えてコンパイルを行います。

T-Kernelのカスタマイズ

T-Kernelのライセンス「T-License」（詳しくは本号P.21「T-License」を参照）では、再配布しないかぎり、T-Kernelのソースコードを改変して利用することを許可しています。T-Kernelのソースコードを改変し、最初にT-Kernelを構築したときと同じ方法

でカーネルを再構築すると、改変したカーネルを利用できます。ここでは、システムコールの追加・削除の方法について説明します。

●T-Kernelのシステムコール呼び出し方法の分析

システムコールの追加・削除方法を説明する前に、T-Kernelでのシステムコールに呼び出し方を見てみましょう。T-Kernelでは、システムコールは割込みによって呼び出され、その方法はターゲットシステムによって異なります。ここでは、SH7727のシステムコール呼び出しについて説明します。

T-Kernelのシステムコールは、tk_cre_tsk()やtk_wai_sem()などのように関数呼び出しの形で提供されています。これらの関

リスト3　パズルゲームのメインタスク部分

```
/* キー・タッチパネル監視タスク定義 */
EXPORT const T_CTSK ktask = {
        NULL,                   /* exinf */
        TA_HLNG|TA_RNG0,        /* tskatr */
        (FP)&kbpd_task,         /* task */
        4,                      /* itskpri */
        8*1024,                 /* stksz */
        0,                      /* sstksz */
        NULL,                   /* stkptr */
        NULL,                   /* uatb */
        0,                      /* lsid */
        0                       /* resid */
};

/* 周期ハンドラ */
EXPORT ID cycid;
EXPORT T_CCYC ccyc = {
    NULL,             /* exinf */
    TA_HLNG,          /* cycatr */
    (FP)&cyclic_hdr,  /* cychdr */
    100,              /* cyctim */
    0                 /* cycphs */
};

/* メッセージバッファ */
EXPORT ID mbfid;
EXPORT T_CMBF cmbf = {
    NULL,      /* exinf */
    TA_TPRI,   /* mkfatr */
    32,        /* bufsz */
    4          /* maxmsz */
};

/* メインタスク */
EXPORT INT main_task( INT stcd, VP exinf )
{
    /*
------------------------------------------------------------
*/
    /* 各種初期化処理 */

    /* キーボード,LCDを初期化する */
    init_kbpd();
    init_screen();

    /* 入力監視用タスクを作成する */
    ktskid = tk_cre_tsk((T_CTSK*)&ktask);
    tk_sta_tsk(ktskid, 0);

    /* タイマーハンドラを作成 */
    cycid = tk_cre_cyc(&ccyc);

    /* メッセージバッファを作成 */
    mbfid = tk_cre_mbf(&cmbf);

    /*
------------------------------------------------------------ */
    /* メインのゲームループ */
    while(1)
    {
        /* 初期化 */
        init_game();
        /* メインのゲーム処理 */
        if(game_proc()==0) break;
        /* クリアした場合,レベルをあげて再スタート */
        lev++;
    }

    /*
------------------------------------------------------------ */
    /* 終了 */

    /* 資源を削除 */
    tk_del_cyc(cycid);
    tk_del_mbf(mbfid);
    tk_ter_tsk(ktskid);
    tk_del_tsk(ktskid);

    /* 初期タスクを起床して自分は終了 */
    tk_rel_wai(itskid);
    tk_exd_tsk();

    /* ここへは到達しない */
    return 0;
}
```

数は、libsvcライブラリが提供しており、ライブラリ内で割込みを行ってシステムコールの本体を実行させています。システムコールの本体は_tk_cre_tsk()などのように先頭に“_”の付いた関数です。

~/tkernel_source/lib/libsvc/src/sysdepend/std_sh7727/ディレクトリには、各システムコールにつき1つずつファイルが配置されています。ファイル名はtk_cre_tsk.Sのようにシステムコール名に拡張子“.S”が付いたものです。

具体的にtk_cre_tsk.Sを見てみるとリスト6のようになっています。リスト6では、レジスタr0にTFN_CRE_TSKというマクロ定義された定数を設定し、trapa命令によって割込みをかけています。trapaは例外を発生させる命令であり、これによってcall_entry関数が呼ばれます（リスト7）。

call_entry()関数はアセンブラで~/tkernel_source/kernel/sysdepend/cpu/sh7727/cpu_support.Sに記述されています。この関数では割込みのためのレジスタ退避などを行うほか、trapa実行時にレジスタr0に指定された値を用いて対応する関数を呼び出します。

リスト6で指定されている定数TFN_CRE_TSKはT-Kernelシステムコールの関数コードであり、~/tkernerl_source/include/sys/svc/tkfncd.hに定義されています（リスト8）。

この値は、表2のように定義されています。

また、リスト8のマクロ定義に合わせ、~/tkernerl_source/include/sys/svc/tksvctbl.hにはリスト9のようなマクロが定義されています。

リスト9にあるN_TFNは、システムコールの最大数です。_SVC_ENTRYは先頭にアンダーバー“_”を付けるマクロであるため、マクロ展開されると_tk_cre_tsk()、_tk_del_tsk()、_tk_sta_tsk()…という、システムコール本体の関数名が並ぶことになります。

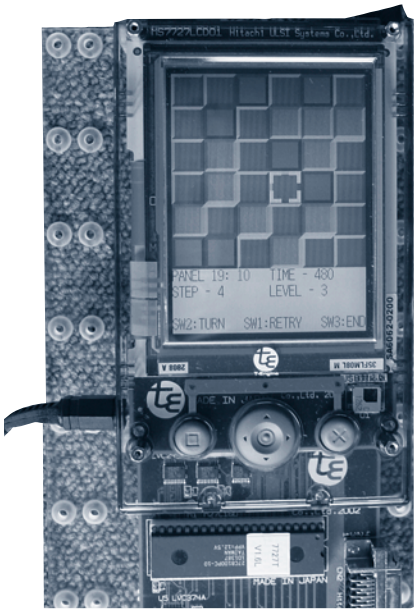

図2　パズルゲーム画面

リスト4　Makefile.usermainファイル（抜粋）

```
# source files
SRC += usermain.c
```

リスト5　Makefile.usermainファイルへのソースファイル追加定義

```
# source files
SRC += usermain.c src1.c src2.c
```

リスト6　tk_cre_tsk.Sのアセンブラコード（一部抜粋）

```
    .globl  Csym(tk_cre_tsk)
    .type   Csym(tk_cre_tsk), @function

Csym(tk_cre_tsk):
    mov.l   fno, r0
    trapa   #TRAP_SVC
    rts
    nop

    .balign     4
fno:    .long   TFN_CRE_TSK
```

この並び順はリスト8で定義される各定数マクロ中の第2バイトの値の順番と一致します。

call_entry()関数は、trapaした際の数値の第2バイトに対応する位置の関数を呼ぶ処理を行います。これにより最終的にC言語で記述されたシステムコールの関数の呼び出しが行われます。

●T-Kernelシステムコールの追加・削除方法

T-Kernelからランデブを参照するシステムコールtk_ref_porを削除してみましょう。先に見ましたT-Kernelでのシステムコール呼び出し方法の解析より、T-Kernelからシステムコールtk_aaa_bbbを削除するためには、次の処理が必要であることがわかります。

①~/tkernerl_source/include/sys/svc/tkfncd.hからマクロ定数TFN_AAA_BBBを削除し、その後に定義されたマクロ定数の2バイト目の数値を順番になるように修正する。
②~/tkernerl_source/include/sys/svc/tksvctbl.hから_SVC_ENTRY（tk_aaa_bbb）を削除し、N_TFNマクロ定数の値を1減らす。
③trapa命令を発行するためのtk_aaa_bbb.Sファイルをlibsvcディレクトリ配下から削除する。
④~/tkernerl_source/include/tk/syscall.hからtk_aaa_bbb()のプロトタイプ宣言を削除する。
⑤実際にカーネルソースコード中から関数_tk_aaa_bbb()を削除する。

この処理のうち、最初の3つは追加または追加する関数の内容にあまり依存しません。そのため、T-Kernelのソースコードには最初の3つの処理を自動で行うスクリプト~/tkernel_source/etc/mktksvcが付いています。このスクリプトは、~/tkernel_source/include/tk/syscall.h内のシステムコール一覧（リスト10）から自動でマクロ定数やtrapa命令を発行するアセンブラファイルを生成します。

スクリプトmktksvcは/* [BEGIN SYSCALLS] */から/* [END SYSCALLS] */の間をシステムコールのプロトタイプ定義とみなします。

このスクリプトを用いてシステムコールtk_ref_porを削除してみましょう。まず、~/tkernerl_source/include/tk/syscall.h（リスト10）からtk_ref_porのプロトタイプを削除します（リスト11）。

この状態で、libsvcのビルドディレクトリ~/tkernel_source/lib/libsvc/build/std_sh7727/に移動し、make sourceでソースファイルを生成できます。make clean_sourceで以前のソースを削除できます。

リスト7　trapa割込みによってcall_entry()関数を呼び出す部分
（~/tkernel_source/kernel/sysdepend/cpu/sh7727/cpu_init.cから抜粋）

```
/* Register exception handler used on OS */

define_inthdr(TRAP_SVC,      call_entry);
define_inthdr(TRAP_RETINT,   _tk_ret_int);
define_inthdr(TRAP_DISPATCH, dispatch_entry);
define_inthdr(TRAP_LOADSR,   load_SR);
```

リスト8　T-Kernel関数コード定義
（~/tkernerl_source/include/sys/svc/tkfncd.hより抜粋）

```
/*
 *       T-Kernel function code
 */

#define TFN_CRE_TSK       0x80010100
#define TFN_DEL_TSK       0x80020100
#define TFN_STA_TSK       0x80030200
#define TFN_EXT_TSK       0x80040000
#define TFN_EXD_TSK       0x80050000
#define TFN_TER_TSK       0x80060100
#define TFN_DIS_DSP       0x80070000
#define TFN_ENA_DSP       0x80080000
#define TFN_CHG_PRI       0x80090200
```

表2　システムコール関数コードの定義

バイト位置	値
最上位バイト	常に 0x80
2 番目のバイト	関数の定義順に 1、2、3…と付けられる
3 番目のバイト	関数の引数の数
最下位バイト	常に 0x00

リスト9　システムコールのエントリ定義
（~/tkernerl_source/include/sys/svc/tksvctbl.hより抜粋）

```
#define _SVC_ENTRY(name)  .int     Csym(_##name)

#define N_TFN     109

          _SVC_ENTRY(tk_cre_tsk)
          _SVC_ENTRY(tk_del_tsk)
          _SVC_ENTRY(tk_sta_tsk)
          _SVC_ENTRY(tk_ext_tsk)
          _SVC_ENTRY(tk_exd_tsk)
          _SVC_ENTRY(tk_ter_tsk)
          _SVC_ENTRY(tk_dis_dsp)
          _SVC_ENTRY(tk_ena_dsp)
          _SVC_ENTRY(tk_chg_pri)
```

注3）機能追加はシステムコールを直接追加するのではなく、拡張SVCを使って追加するのが作法です。

makeの結果としていくつかのヘッダファイルとtrapa命令を発行するアセンブラのファイルがシステムコールと同じ数だけ生成されます。このあと、make installを実行します。make installは生成されたアセンブラファイルをコンパイルしてlibsvc.aを生成し、適切な場所にコピーします。しかし、生成したヘッダファイルは自動的には適切な場所にはコピーされません。生成されたヘッダファイルは~/tkernel_source/lib/libsvc/src/sysdepend/include/に置かれますので、このディレクトリの中身をすべて~/tkernel_source/include/sys/svc/にコピーするか、またはシンボリックリンクを貼ってください。

また、システムコールtk_ref_porの本体_tk_ref_por()は~/tkernel_source/kernel/tkernel/src/rendezvous.cに定義されています。これを削除してカーネルを最構築すればシステムコールtk_ref_porのないT-Kernelが生成されます。

システムコールを削除する場合、削除したシステムコールを利用しているシステムコールがあるとリンクに失敗します。あらかじめ削除しようとしているシステムコールを使っているシステムコールがないか確認する必要があります。

システムコールを追加する場合[注3)]も同様に~/tkernel_source/include/tk/syscall.hの編集を行ってlibsvcの再ビルドを行った後、システムコールの関数定義を追加しカーネルを再構築します。

本稿で紹介しているサンプル（パズルゲーム）のソースプログラムとMakefile.usermainを本号の付録CD-ROMに収録しました。詳しくはP.57「付録CD-ROMの使い方」をご覧ください。

リスト10　システムコール一覧定義（~/tkernerl_source/include/tk/syscall.hから抜粋）

```
/* ------------------------------------------------------------------------
*/
/*
 * Definition for interface library automatic generation (mktksvc)
 */
/*** DEFINE_TKSVC ***/

/* [BEGIN SYSCALLS] */
IMPORT ID tk_cre_tsk( T_CTSK *pk_ctsk );
IMPORT ER tk_del_tsk( ID tskid );
IMPORT ER tk_sta_tsk( ID tskid, INT stacd );
...（中略）...
IMPORT ER tk_rpl_rdv( RNO rdvno, VP msg, INT rmsgsz );
IMPORT ER tk_ref_por( ID porid, T_RPOR *pk_rpor );
IMPORT ER tk_def_int( UINT dintno, T_DINT *pk_dint );
...（中略）...
IMPORT ER tk_del_res( ID resid );
IMPORT ER tk_get_res( ID resid, ID ssid, VP *p_resblk );
IMPORT ER tk_set_pow( UINT powmode );
/* [END SYSCALLS] */
```

リスト11　システムコール一覧定義からtk_ref_porを削除

```
/* [BEGIN SYSCALLS] */
IMPORT ID tk_cre_tsk( T_CTSK *pk_ctsk );
IMPORT ER tk_del_tsk( ID tskid );
IMPORT ER tk_sta_tsk( ID tskid, INT stacd );
...（中略）...
IMPORT ER tk_rpl_rdv( RNO rdvno, VP msg, INT rmsgsz );
IMPORT ER tk_def_int( UINT dintno, T_DINT *pk_dint );
...（中略）...
IMPORT ER tk_del_res( ID resid );
IMPORT ER tk_get_res( ID resid, ID ssid, VP *p_resblk );
IMPORT ER tk_set_pow( UINT powmode );
/* [END SYSCALLS] */
```