プロローグ いきなりマルチコアOSはちょっと…のビギナ向け

1500行OS「Try Kernel」でOSの基礎を学ぶ

豊山 祐一

ファイル名	行数	
application		
app.h	34	アプリケーション
gsns.c	127	
lcd.c	121	
lsns.c	43	
usermain.c	48	
boot		
boot2.c	23	起動部
reset_hdr.c	206	
vector_tbl.c	66	
device		
adc		
adc_rp2040.c	99	デバイス・ドライバ
adc_sysdep.h	43	
dev_adc.h	6	
devmgr		
device.c	71	
device.h	22	
device_tbl.c	11	
i2c		
dev_i2c.h	7	
i2c_rp2040.c	243	
i2c_sysdep.h	160	
include		
apidef.h	93	ヘッダ・ファイル
config.h	13	
error.h	31	
knldef.h	108	
sysdef.h	171	
syslib.h	63	
trykernel.h	6	
typedef.h	56	
kernel		
context.c	30	カーネル
dispatch.S	72	
eventflag.c	144	
inittsk.c	34	
scheduler.c	30	
semaphore.c	107	
syslib.c	29	
systimer.c	25	
task_mange.c	85	
task_queue.c	52	
task_sync.c	81	
linker		
pico_memmap.ld	59	リンカ・スクリプト

アプリケーションとデバイス・ドライバを除くOS部分は1584行
全体で2619行

図1 ソース・ファイルの一覧
wc -lで計測

特設で紹介するOS “Try Kernel”の概要

● コア機能だけだから容易に理解・自作できる

組み込みシステムでは主にリアルタイムOS（RTOS）と呼ばれるOSが使用されます．RTOSはOSのコア（中核）の機能だけを実装したコンパクトなOSです．よってOSを自作し仕組みを理解するには最適です．

またRTOSは，PCやサーバなどで使われているWindowsやLinuxなどの汎用OSと比べると，その機能も用途も大きく異なります．RTOSと汎用OSの違い，そしてRTOSの特徴と言われるリアルタイムとはどのように実現されるのか，具体的に理解するのはなかなか難しいと思います．

本特設では，本誌2023年7月号で作成したRTOS「Try Kernel」を題材に，その特徴や実現するための仕組みを説明していきます．

● 作成するRTOSは国際標準規格のサブセット

本記事で作成するTry Kernelは基本機能のみを実装したシンプルなRTOSです．Try Kernelの仕様は記事中で詳しく説明しますが，RTOSの国際標準規格IEEE 2050-2018のサブセットとなっているので，RTOSの学習にも最適です．

またサブセットと言っても実用的に使えることを意識しました．本特集の中で実際のセンサ制御なども実現してみます．

● とても読みやすいソースコード

図1は今回作るTry Kernelの全ソース・ファイルです．アプリケーションとデバイス・ドライバを除くOS部分はわずか1584行，全体でも2619行とOSとしては大変短くなっています．

リスト1は実際のソースコードの抜粋です．どの関数も30行程度と短く，`if`文や代入と関数呼び出しといったC言語の基本的な構文で構成され，平易な記述を心がけています．また，複雑な`#ifdef`がないこ

リスト1　ソースコードの抜粋（kernel¥task_sync.c）

```
ER tk_wup_tsk( ID tskid )
{
    TCB     *tcb;
    UINT    intsts;
    ER      err = E_OK;

    if(tskid <= 0 || tskid > CNF_MAX_TSKID)
        return E_ID;     /* ID番号チェック */

    DI(intsts);      // 割り込みの禁止
    tcb = &tcb_tbl[tskid-1];
    // tk_slp_tskで待ち状態か？
    if((tcb->state == TS_WAIT) &&
       (tcb->waifct == TWFCT_SLP)) {
        // タスクをウェイト・キューから外す
        tqueue_remove_entry(&wait_queue, tcb);
        /* TCBの各種情報を変更する */
        tcb->state = TS_READY;
        tcb->waifct    = TWFCT_NON;
        // タスクをレディ・キューに繋ぐ
        tqueue_add_entry(&ready_queue[tcb->itskpri],
                         tcb);
        scheduler();         // スケジューラの実行
    } else if(tcb->state == TS_READY || tcb->state
           == TS_WAIT) {     // 実行できる状態の場合
        tcb->wupcnt++;       // 起床要求数を増やす
    } else {
        err = E_OBJ; // 起床できるタスクの状態ではない
    }
    EI(intsts);       // 割り込みの許可
    return err;
}
```

とも読みやすさに寄与しています．

● マイコン学習の基礎としても役立つ

OS自体のプログラムはマイコンのハードウェアの上で直接動作します．つまり，OSの作成はベアメタル・プログラミングそのものと言えます．

Try Kernelでは，既存のSDK（ソフトウェア開発キット）やHAL（ハードウェア抽象化レイヤ）などのソフトウェアは原則使用せずに，マイコンを一から制御してOSを動かしていきます．マイコンのハードウェア制御を理解するのにも役に立つことと思います．

● 動作プラットホームはラズベリー・パイPico

Try Kernelは以下の観点から，本誌でも何度も登場しているラズベリー・パイPico（以降Pico）で動作させることとします．

- マイコンの中核であるCPUが，組み込みシステムで広く普及しているArm Cortex-Mコアを使っている．
- ボードの入手が比較的容易であり，RTOSを使ったいろいろな実験が行いやすい．

なお，Try Kernelはできるかぎり特定のマイコンに依存しないように設計していきますので，他のマイコン・ボードに実装することもそれほど難しくないはずです．

● ソースコードをウェブで配布します

ソースコードは本誌のダウンロード・ページで配布します．

https://www.cqpub.co.jp/interface/download/contents2024.htm

本記事では段階的にRTOSを作成していきますので，各部章ごとに完成したソースコードとなります．

また，最新版のソースコードは筆者のGitHubで公開します．

https://github.com/ytoyoyama/interface_trykernel

とよやま・ゆういち