実践報告

小規模リアルタイム・マルチタスク処理用OSの製作 （1）

東北ポリテクカレッジ　生産情報システム技術科　**谷本　富男**
（東北職業能力開発大学校）

1．はじめに

さまざまな機器に組み込まれているマイクロコンピュータ（マイコン）は，時代とともに多機能，高速，高性能などのインテリジェント化の要求を実現してきた。高速・多機能化の実現では，製品の中に複数のマイコンを組み込み，それぞれを連携させる方法と，1つのマイコンで，同時に複数の処理をさせることが可能なマルチタスクOSの導入などが考えられる。

組込みOSとしてリアルタイムOS（RTOS）製品を使用する場合，通常はブラックボックスされた状態で購入するため，高価格の割には，使いにくい，情報が入りにくい，理解しにくい等の不満が多かった。しかし近年のオープンソース化の流れによって状況が一変してきた。RTOSのソースコードも購入できるとか，製品版と同様のOSがノンサポートだが無料で手に入れることが可能となった。

しかし，そのソースコードはOSのさまざまな機能や，多くのターゲットを対象とするために書かれた膨大な情報であり，短時間で内容を理解するのは困難である。また，一般的なパソコンOS（Windows等）とは，仕組みが異なるOSであることや，プログラムする際の環境や方法になじみのないことなどが敷居を高くしている要因になっている。

そこで今回，OSの利用技術より構築技術を意識した教育訓練用教材の必要性を感じ，製作してみたのでここに紹介する。教育訓練用ということで，極限まで機能を絞り込んだため，OSというよりはタスクスケジューラのみの機能を実装している。第1回目の今回は事象駆動（イベントドリブン）型のスケジューリングをするOS（miniOSと呼ぶ）の製作を紹介する。2回目はプリエンプティブなスケジューリングを行うOSの製作を紹介する予定である。

2．組込み機器開発事情

マイクロコンピュータ（マイコン）は，携帯電話，エアコン，テレビ，電子レンジなどの家庭用電子機器をはじめとして，あらゆる分野の製品に組み込まれた状態で使用されてきた。そのため機器の筐体の中に隠れている場合が多く，設置スペース・消費電力などの制約条件や熱・雑音・振動・埃などの劣悪環境の中でも，高速・安定性が要求され続けてきた。

組込み機器開発の状況は，10年以上も前から組込み技術者不足が叫ばれている。製品の多品種・少量化，低価格化，設計・製造期間の短縮，さらなる高速・高機能化の流れのなかで，目先の対応に追われ，対外的な状況を把握し次の一手を考えている余裕などほとんどないのが実状となっている。そのことから，今後さらに深刻な事態が予測される。

一方，組込みソフトウェア開発の状況でいえば，トロン協会が実施している「リアルタイムOSの利用動向とITRONに関するアンケート調査」が数年前から実施されている。ここ数年の結果からは状況の進展（改善）が見られず，いまだ技術者不足，開発環境やツール不足が補えていない状況といえる。

2000年のアンケート調査結果では，リアルタイム

OSの問題点として最も多く選ばれたのは，前回までと同様に「使いこなせる技術者が不足またはいない」で，32.3%の回答者が最大の問題点であるとしている。それに次いで，「OSにより仕様の違いが大きく切り替えの負担が大きい」が10.0%，「初期および保守費用が高い」が10.0%，「開発環境やツールが不足」が9.0%，「使用リソースが大きすぎる」と「実行時のライセンスが高い」が6.3%となっている。「開発環境やツールの不足」をあげた回答者に対して，具体的にどのようなツールが不足しているかを具体的に記述してもらう項目に対しては，多くの回答者がデバッグ環境関連ツールをあげている。

多くの組込みソフトウェア開発組織では，バックログに対処するのが精いっぱいで，既存の成果物を再利用することや，現場で必要な技術を基礎から新人に教育することが不可能な状況といえる。

3．ターゲットのハードウェア構成

今回は小規模な組込み機器を意識したターゲットを用意した。すなわち限られた記憶容量，処理能力で，どの程度のことが可能かを検討してみた。図1に今回使用したターゲットのブロック構成図を示す。

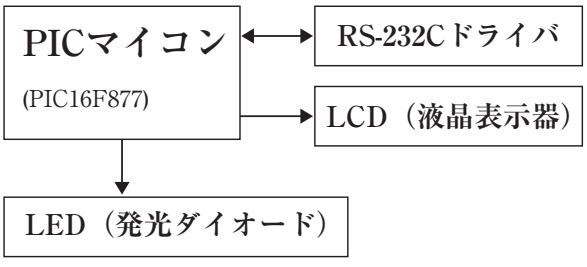

図1　ターゲットのブロック構成図

3.1　PIC16F877の仕様

今回ターゲットボードに使用したマイコン（Microchip社製PIC16F877）の主な仕様を以下に示す。

・高性能RISC CPU
・命令数：35
・命令実行サイクル：1サイクル（分岐命令は2サイクル）
・最高動作速度 20MHz クロック入力時 200ns 命令サイクル
・FLASH プログラムメモリ：8K ワード（14bit）
・データメモリ（RAM）：368 バイト
・EEPROM データメモリ：256 バイト
・割り込み要因：14
・ハードウェアスタック： 8 レベル
・直接，間接，相対の各アドレッシングモード
・専用のオンチップRC発振器付きウォッチドッグタイマー（WDT）

周辺機能の特徴

・タイマー 0: 8 ビット・プリスケーラ付き 8 ビットタイマー/ カウンタ
・タイマー 1: プリスケーラ付き16 ビットタイマー/ カウンタ，外部水晶/ クロックによりスリープ中もインクリメントが可能
・タイマー 2: 8 ビットの周期レジスタ，プリスケーラ，ポストスケーラ付きの 8 ビットタイマー/ カウンタ
・キャプチャー，コンペア，PWM モジュール
・10 ビットマルチチャネルAD 変換器
・同期シリアルポート（SSP）　SPITM（マスター／スレーブ），I2CTM（マスター/ スレーブ）
・ユニバーサル同期･非同期レシーバー・トランスミッター（USART/SCI）9 ビット・アドレス検出

4．開発環境

開発方法としては，ホストPCでターゲット側プログラムを作成し，デバッグを行った後PICライターで書き込み，PICをターゲットボードに差し替え動作確認をするという手順を繰り返しながら作成した。差し替えが頻繁となるので，PICライターやターゲットのICソケットにゼロプレッシャソケットを用いている。図2に開発環境の構成図を示す。

ホストPCには,開発統合環境（Microchip社MPLAB IDE）をインストールし，C言語コンパイラ（HI-TECH社 PICC）で開発を行った。

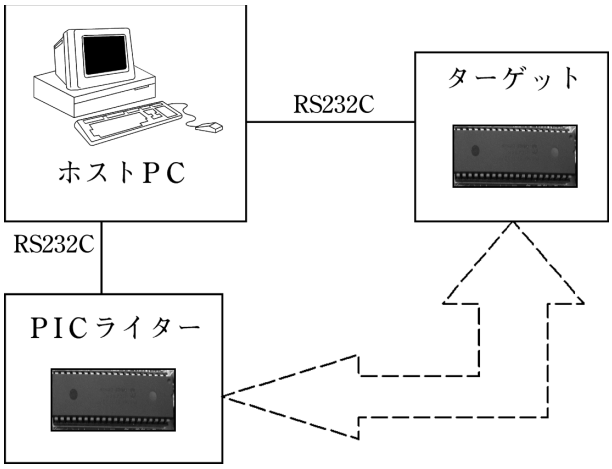

図2　開発環境の構成図

5．ソフトウェア仕様

今回製作したminiOSのソフトウェア仕様を以下に示す。

・OSの種類	優先度による事象駆動型
・タスクの状態数	4
・タスク優先度数	8（最大16）
・API	μITRON風API
・タスク数	6（優先度8までの場合）
・プログラム容量	ROM　1.4k words
	RAM　93 bytes
・ディスパッチ時間	60μS以内
・システムクロック	1mS

5.1　タスク・コントロール・ブロック（TCB）

TCBの構造を図3に示す。

長方形の枠内は，それぞれ2バイト幅のデータとなるので，1つのTCBの容量は11バイトとなる。優先度（priority）と状態（status）は4ビットずつの領域とした。

PIC877のRAMは最大368バイトであるが，4つのバンクに分かれている。PICCではバンクを超えた参照が不自由（固定）なため，BANK1のRAM 80バイトにTCBや待ちキューを押し込める必要があった。そのため，タスク優先度を8までに限定しても，6つのTCBしか作成できなかった。

15　　8	7　　0
struct def_tcb	*next;
struct def_tcb	*prev;
const char	*task_name;
FP	task;
DLYTIME	wait_times;
	UB priority UB status

図3　TCBの構造

5.2　タスクの状態とキュー

生成されたタスクは，実行（RUN），実行待ち（READY），待ち（WAIT），休止（DORMANT）の4状態のなかのどれかに所属する。タスクの状態遷移を図4に示す。

①生成されたタスクは休止状態または実行待ち状態となる。②実行権がカーネルに戻ると，実行待ち状態から優先順位の高い（ここでは大きい値）タスクを探して実行する。③実行中のタスクの遅延（dly_tsk）が実行されると，待ち状態に移され，その間別の実行待ちのタスクが実行される。④タスクの遅延（dly_tsk）命令で指定した時間が経過すると実行待ち状態に移される。

MiniOSでは，休止，実行待ち，待ち状態用の3種類のキューを使用している。実行待ちキューは優先順位ごとに別のキューとなる。キューとTCB間またはTCBとTCB間は双方向のリンクで接続される。

図5に優先度0の実行待ちキューにTCB1とTCB2がつながっている状態を示す。ただしTCB1とTCB2は同じ優先順位であるが，TCB1がTCB2より後から実行待ちキューにつながったことになる。

5.3　アプリケーション・プログラム・インタ

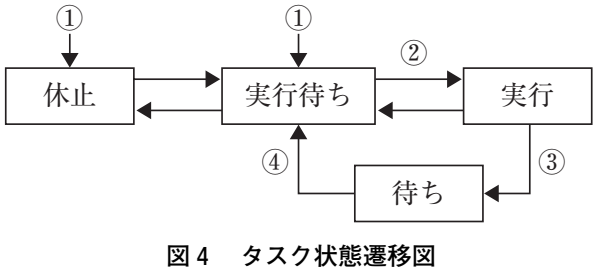

図4　タスク状態遷移図

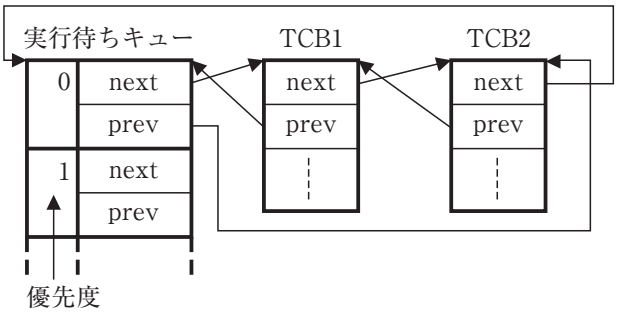

図５　実行待ちキューの様子

ーフェース（API）

一般的にOS操作用APIはシステムコールと呼ばれ，TRAPなどのソフトウェア割り込みを使用してエレガントに仕上げたいところだが，PICにはソフトウェア割り込みが存在しないので，通常の関数呼び出しと同じ仕組みでOSの操作をしている。リスト１に実装している関数一覧を示す。

```
1.  void cre_tsk(ID taskid ,const char* taskname,FP
    task ,PRI priority ,ATR status); // タスクの生成
2.  void act_tsk(ID taskid);          // タスクの起動
3.  void ext_tsk(void);      // 自タスクの終了
4.  void exd_tsk(void);      //自タスクの終了と削除
5.  void ter_tsk(ID taskid); // タスクの終了と削除
6.  void chg_pri(ID taskid, PRI priority);
                    // タスク優先度の変更
7.  void ref_tsk(ID taskid ,const char* taskname ,
    UB* info);      // タスクの状態参照
8.  void dly_tsk(DLYTIME); // 自タスクの遅延
9.  void print(void); // タスク遷移状態の表示(特別)
10. void logS(const unsigned char *); // ログメッセ
    ージの送信(特別)
```

リスト１　関数一覧

６．サンプルプログラム

サンプルプログラムでは，複数のタスクを並列実行させて動作確認を行った。

６.１　サンプル１

メインタスクで４つのタスクを生成し，それぞれのタスクが指定した時間間隔でポートＢのそれぞれのビットを点滅させる。リスト２を以下に示す。

```
//   minOS 用サンプルプログラム
#include <pic.h>
#include "task.h"
#include "minos.h"

void task1(void){
        PORTB ^= 0x10;
        dly_tsk(50);
}

void task2(void){
        PORTB ^= 0x20;
        dly_tsk(100);
}

void task3(void){
        PORTB ^= 0x40;
        dly_tsk(150);
}

void task4(void){
        PORTB ^= 0x80;
        dly_tsk(200);
}

void main_task(void){
        cre_task(1 ,  "task1" ,(FP) task1 , 6,TTS_RDY);
        cre_task(2 ,  "task2" ,(FP) task2 , 6,TTS_RDY);
        cre_task(3 ,  "task3" ,(FP) task3 , 6,TTS_RDY);
        cre_task(4 ,  "task4" ,(FP) task4 , 6,TTS_RDY);

        exd_tsk();
}
```

リスト２　sample1.c

6.2　サンプル2

ターゲットにキャラクタ液晶表示器を接続し，初期化処理と1秒ごとのカウントアップ表示をさせるlcdタスクを追加した。リスト3に追加部分（lcdタスクの内容）を示す。

できるだけリアルタイム性を損なわないようにするために，頻繁に実行権をOSに戻すことが必要となる。

```
unsigned char step , count;

void
lcd_task(void)
{

  switch(step){
    /* initialise the LCD - put into  4  bit mode */
    case  0 :
        TRISB =  0 ;      //PORT  B  all output
        LCD_RS =  0 ;    // write control bytes
        PORTB = 0x30;  // attention!
        LCD_STROBE;
        dly_tsk(5);
        step =  1 ;
        break;
    case  1 :
        LCD_STROBE;
        Delay10Us(10);
        LCD_STROBE;
        dly_tsk(1);
        step =  2 ;
        break;
    case  2 :
        PORTB = 0x20;  // set  4  bit mode
        LCD_STROBE;
        Delay10Us(4);
        lcd_write(0x28);  //  4  bit mode, 5x8 font
        lcd_write(0x08);  // display off
        lcd_write(0x01);  // clear display
        dly_tsk(2);
        step =  3 ;
        break;
    case  3 :
        lcd_write(0x06);  // entry mode
        lcd_write(0x0F);  // display on, blink
        dly_tsk(2);
        step =  4 ;
        break;
    case  4 :
        lcd_puts("test program ");
        b2c(count);
        dly_tsk(1000);
        step =  5 ;
        break;
    case  5 :
        lcd_clear();
        dly_tsk(10);
        count ++;
        step =  4 ;
        break;
  }
}
```

リスト3　lcdタスクの内容

6.3　サンプル3

シリアルRS232Cよりリモート制御するためにメインタスクを改良している。メインタスクをリスト4

```
unsigned char sw= 0 ;

void main_task(void){
  static ID task =  1 ;
  char* taskname ;
  UB* info;
  PRI priority;

  switch (sw){
    case  0 :   // タスクの起動と初期化
        cre_tsk( 1 ,"task1(PB4)",(FP)task1,6,TTS_RDY);
        cre_tsk( 2 ,"task2(PB5)",(FP)task2,6,TTS_RDY);
        cre_tsk( 3 ,"task3(PB6)",(FP)task3,6,TTS_RDY);
        cre_tsk( 4 ,"task4(PB7)",(FP)task4,6,TTS_RDY);

        step =count = 0 ; // lcd_tsk 用の初期値設定
        cre_tsk( 5 ,"task5(LCD)",(FP)lcd_task, 3 ,  ¥
              TTS_DMT);
        sw =  1 ;
        break;
```

```
case 1:    // モニタ(main_task)コマンド表示

    logS("////////////////////////////////////////////////¥r¥n");
    logS("//*      リモート制御 9600bps¥r¥n");
    logS("//  comannd¥r¥n");
    logS("//  1..5     タスク指定(print)¥r¥n");
    logS("//   a     タスクの起動(act_tsk)¥r¥n");
    logS("//   t     タスクの終了(ter_tsk)¥r¥n");
    logS("//   u     タスク優先度のUp¥r¥n");
    logS("//   d     タスク優先度のDown¥r¥n");
    logS("*/¥r¥n");

    sw = 2;
    break;

case 2:
    if (RCIF){  // シリアル受信チェック
        switch(RCREG){
            case '1':
                task = 1;
                break;
            case '2':
                task = 2;
                break;
            case '3':
                task = 3;
                break;
            case '4':
                task = 4;
                break;
            case '5':
                task = 5;
                break;

            case 'a':
                act_tsk(task);
                break;
            case 't':
                ter_tsk(task);
                break;
            case 'u':
                ref_tsk(task, taskname , info);
                priority = (PRI) ((*info >> 4)+ 1);
                if(priority< (TCB_PRIORITY- 1))
                        chg_pri(task , priority+ 1);
                break;
            case 'd':
                ref_tsk(task, taskname , info);
                priority = (PRI) ((*info >> 4)+ 1);
                if (priority > 0)
                        chg_pri(task , priority- 1);
                break;
            }
            print();
            RCIF = 0;
        }
    }
    dly_tsk(100);
}
```

リスト 4　mainタスクの内容

に示す。

7．プログラム容量

今回作成したminOSはサンプルプログラム 3 を含めて以下のようになった。

ROM　2709 Words （全体の33.1%）
RAM　108 Bytes （全体の29.3%）

デバッグ情報を抜いてコンパイルすると以下のようになった。

ROM　2165 Words （全体の27.1%）
RAM　108 Bytes （全体の29.3%）

サンプルプログラムを抜いたOS部分だけだと以下のようになった。

ROM　1417 Words （全体の17.3%）
RAM　93 Bytes （全体の25.3%）

同様にデバッグ情報を抜いてコンパイルすると以下のようになった。

ROM　1134 Words （全体の13.8%）
RAM　93 Bytes （全体の25.3%）

8．最後に

市販されているOSやフリーのオープンソースのOS

は，複数の人手によって数百時間以上の時間をかけて作成されているため，多機能であるが複雑で，解析には多くの時間と努力が必要となる。数十時間程度の訓練に使用するのに適当な教材があればという思いで製作してみた。OSというには大いに物足りなさを感じるが，メモリが半分以上残っているので，必要な機能を追加することも期待している。ターゲットにPICを使用しているのは，マイコン制御等の訓練でPICを使用している場合が多いのではないかとの予測で採用した。

リアルタイム性については，実行待ちキューから優先順に実行タスクを探す処理が最大60μSなので，ユーザプログラムの作成の仕方によっては1mS以下のリアルタイムシステムとすることも可能だが，あまり小さくしすぎるとユーザプログラムの作成が大変になることやオーバヘッドが増大し効率が悪くなる。

事象駆動型のOSなので，無限ループのタスクがシステムに大きな負荷となり，簡単に制御不能に陥ることが体験できる。

またデバッグモード（#define DEBUGを宣言する）でコンパイルするとタスクの状態遷移がリモートより確認できるようにしてある。以下がその内容である（リスト5）。

次回は，PIC18F452を使用したプリエンプティブ・マルチタスクOSの製作を紹介したい。

```
Debuging Start 9600bps

Run TCB = main_task
Ready Queue
｜｜｜｜｜｜｜main_task-> ｜
Timer Wait Queue=>

Run TCB = task4
Ready Queue
｜task4->task3->task2->task1-> ｜｜｜｜｜｜｜
Timer Wait Queue=>

Run TCB =
Ready Queue
｜task3->task2->task1-> ｜｜｜｜｜｜｜
Timer Wait Queue=>task4->

Run TCB = task3
Ready Queue
 ｜task3->task2->task1->｜｜｜｜｜｜｜
Timer Wait Queue=>task4->

Run TCB =
Ready Queue
｜task2->task1->｜｜｜｜｜｜｜
Timer Wait Queue=>task3->task4->

Run TCB = task2
Ready Queue
｜task2->task1->｜｜｜｜｜｜｜
Timer Wait Queue=>task3->task4->

Run TCB =
Ready Queue
｜task1->｜｜｜｜｜｜｜
Timer Wait Queue=>task2->task3->task4->
```

リスト5　タスク遷移ログの内容

＜参考文献＞

1）「電子工作の実験室」　http://www.picfun.com/
2）PIC16F87X Data sheet（Microchip社）
3）「組込みソフトウェア管理者・技術者育成研究会」（SESSAME: Society of Embedded Software Skill Acquisition for Managers and Engineers）
http://blues.tqm.t.u-tokyo.ac.jp/esw/
4）リアルタイムOSの利用動向とITRONに関するアンケート調査
http://www.ertl.jp/ITRON/survey-j.html
5）永井正武監修：『実用組込みOS構築技法』，共立出版(株).
6）藤倉俊幸：『リアルタイム／マルチタスクシステムの徹底研究』，CQ出版社.
7）TOPPERS　プロジェクト
http://www.ertl.ics.tut.ac.jp/TOPPERS/
8）AzkiRTOS
http://www2.noritz.co.jp/anchor/ashp/azrtos/azindex.html