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　（2）

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

前編（小規模リアルタイム・マルチタスク処理用OSの製作（1））ではノンプリエンプティブなスケジューリングをするOSの製作を紹介したが，第2回目の今回は，プリエンプティブなスケジューリングを行うOS（miniOSと呼ぶ）の製作を紹介する。

前編で紹介したOSは，アプリケーションプログラムの協力によりリアルタイム処理の実現が可能となった。反対にいえば，リアルタイム処理させるにはアプリケーションプログラムを工夫する必要があった。

今回はプリエンプティブなスケジューリング機能を持っているので，実行中のタスクから強制的にCPUの実行権を奪い取ることが可能となる。

2．ターゲットのハードウェア構成

PICマイコン以外は，前編のハードウェア構成と同じである。PIC18FシリーズはMicrochip社のハイエンドに位置づけられたCPUで，PIC18F452は前編で使用したPIC16F877のピンコンパチブルかつ上位互換のCPUとなる。そのため，前編で使用したターゲットをそのまま使用可能であり，作成したソースコードもほとんど変更することなく動作させることができる。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。

3．PIC18F452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PIC18F452）の主な仕様を表1に示す。

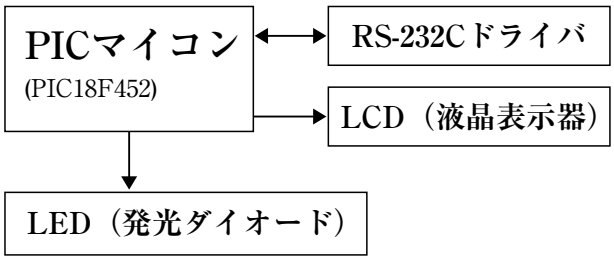

図1　ターゲットのブロック構成図

その他，割り込みレベルが高・低の2レベルに拡張され，基本レジスタ（WREG，STATUS，BSR）の退避復旧がハードウェア化された。これにより，より高速な割り込み処理が可能となった。

またスタックレベルが31レベルに拡張され，スタックポインタ（STKPTR）がPUSH命令，POP命令

表1　PIC18F452の主な仕様

Features	PIC18F452
Operating Frequency	DC-40 MHz
Program Memory （Bytes）	32K
Program Memory （Instructions）	16384
Data Memory （Bytes）	1536
Data EEPROM Memory （Bytes）	256
Interrupt Sources	18
I/O Ports	Ports A, B, C, D, E
Timers	4
Capture/Compare/PWM Modules	2
Serial Communications	MSSP, Addressable USART
Parallel Communications	PSP
10-bit Analog-to-Digital Module	8 input channels
RESETS （and Delays）	POR, BOR, RESET Instruction, Stack Full, Stack Underflow （PWRT, OST）
Programmable Low Voltage Detect	Yes
Programmable Brown-out Reset	Yes
Instruction Set	75 Instructions
Packages	40-pin DIP 44-pin PLCC 44-pin TQFP

で読み書きできるようになった。スタックをプログラムで操作できることによりプリエンプティブなOSを製作できた。

4．開発環境

前編と同様に，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。

図2に開発環境の構成図を示す。

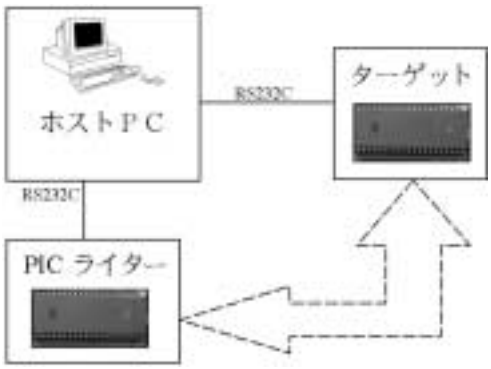

図2　開発環境の構成図

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コンパイラ（HI-TECH社 体験版PICC-18）で開発を行った。

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	優先度によるプリエンプト型
・タスクの状態数	4
・タスク優先度数	16
・ API	μITRON　ライク
・タスク数	11
・プログラム容量	ROM　8557 bytes
	RAM　425 bytes
・ディスパッチ時間	100μS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。

1つのTCBの容量は16バイトとなる。タスクごと

15　8	7　0
struct def_tcb	*next;
struct def_tcb	*prev;
const char	*task_name;
FP	taskini;
FP	task;
DLYTIME	wait_times;
UB w;	UB status;
UB bsr;	UB priority : 4 ; UB status : 4 ;

図3　TCBの構造

のコンテキストとして基本レジスタ（WREG，STATUS，BSR）の退避・復旧用の領域をTCB内に確保した。優先度（priority）と状態（status）は4ビットの領域とした。

TCBにはタスクの実行アドレスを2つ用意している。taskiniはタスクの先頭アドレスで，taskは再実行される場合の開始アドレスを示す。

PIC18F452になってRAMは1536バイトに拡張され，BANKも15に拡張されている。特にBANK1～5は256バイトごとのRAM空間となっている。PICC-18ではBANKを超えた参照が不自由（固定）なため，BANK1のRAM 256バイトにTCBや待ちキューを押し込める必要がある。優先順位ごとの待ちキューが4バイト×16優先度，11個分のTCBが16バイト×11個となり，合計246バイトとなる。そのため，11個のTCBしか作成できなかった。待ちキューとTCBのBANKを分離すれば15個のTCBが作成できる。また，BANKを超えた参照が自由な仕組みを実装すればさらにタスク数を増やすことが可能となるが，まだ挑戦していない。PIC18F452のRAM空間を図4に示す。

5.2　タスクの状態とキュー

前編と同様に，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5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

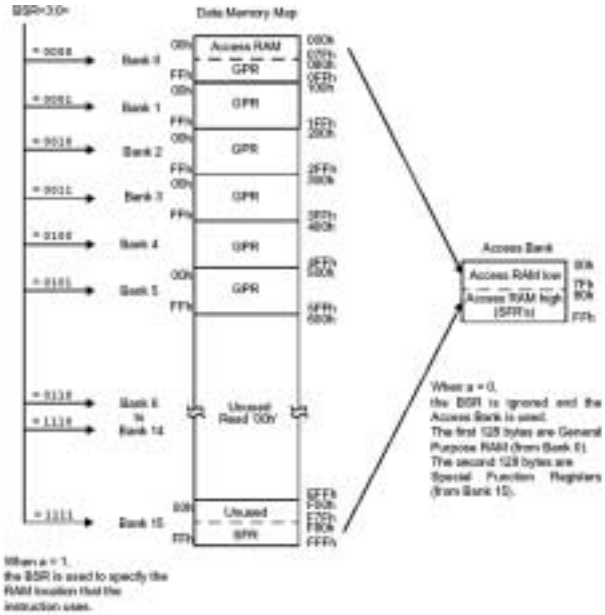

図 4　PIC18F452のRAM空間

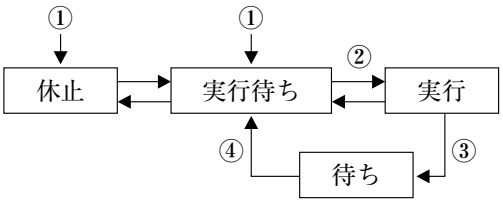

図 5　タスク状態遷移図

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 6 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

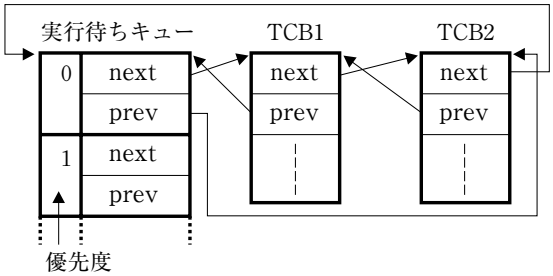

図 6　実行待ちキューの様子

```
1.  cre_tsk(ID taskid ,const char* taskname,FP task
    ,PRI priority ,ATR status); // タスクの生成
2.  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,
    PRI* ,ATR*);               // 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)
11.  void taskinfo1(ID taskid); // タスク名の表示
12.  void taskinfo2(ID taskid); // 優先度の表示
13.  void taskinfo3(ID taskid); // ステータスの表示
```

リスト 1　関数一覧

実装している関数一覧を示す。

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。前編のプログラムと比較してもらいたい。

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。

```
//  miniOS 用サンプルプログラム
#include <pic18.h>
```

```
#include "task.h"
#include "minos.h"


void task1(void){
    while(1){
        PORTB ^= 0x10;
        dly_tsk(500);
    }
    ext_tsk();
}

void task2(void){
    while(1){
        PORTB ^= 0x20;
        dly_tsk(1000);
    }
    ext_tsk();
}

void task3(void){
    while(1){
        PORTB ^= 0x40;
                dly_tsk(1500);
         }
         ext_tsk();
}

void task4(void){
         while(1){
                PORTB ^= 0x80;
                dly_tsk(2000);
         }
         ext_tsk();
}

void main_task(void){

      cre_task(1 ,  "task1" ,(FP) task1 , 9,TTS_RDY);
      cre_task(2 ,  "task2" ,(FP) task2 , 8,TTS_RDY);
      cre_task(3 ,  "task3" ,(FP) task3 , 7,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。

```
void
lcd_task(void)
{
        unsigned char count=0;

        /* initialise the LCD - put into 4 bit mode */
        TRISB = 0;          //PORT B all output
        LCD_RS = 0;         // write control bytes
        dly_tsk(15);        // power on delay
        PORTB = 0x30;       // attention!
        LCD_STROBE;
        dly_tsk(5);
        LCD_STROBE;

        DI;         // 割り込み禁止
        Delay10Us(10);
        EI;         // 割り込み許可

        LCD_STROBE;
        dly_tsk(1);
        PORTB = 0x20;       // set 4 bit mode
        LCD_STROBE;

        DI;     // 割り込み禁止
        Delay10Us(4);
        lcd_write(0x28);        // 4 bit mode, 5x8 font
        lcd_write(0x08);        // display off
        lcd_write(0x01);        // clear display
        EI;     // 割り込み許可

        dly_tsk(2);

        DI;         // 割り込み禁止
        lcd_write(0x06);            // entry mode
        lcd_write(0x0F);            // display on, blink curson on
        EI;         // 割り込み許可

        dly_tsk(2);

        while(1){
                DI;         // 割り込み禁止
                lcd_puts("test program ");
                b2c(count);
                EI;         // 割り込み許可
```

```
                dly_tsk(1000);

                DI;         // 割り込み禁止
                lcd_clear();
                EI;         // 割り込み許可

                dly_tsk(2);
                count ++;
        }
}
```

リスト 3　lcdタスクの内容

6.3　サンプル 3

ユーザプログラム側のタスクとして，

・Task1　　1 秒毎のPB4のLED点滅
・Task2　　1 秒毎のPB5のLED点滅
・Task3　　1 秒毎のPB6のLED点滅
・Task4　　1 秒毎のPB7のLED点滅
・LCD_task　　0 . 5 秒毎のカウントアップ値の液晶表示
・Load_task　　負荷タスク（無限ループ）
・Monita_task　ユーザインターフェース

を作成した。miniOS側のタスクとして，

・Syslog_task　シリアルポートからの文字出力

を作成した。

シリアルポートから文字列を出力する場合，直接ポートにアクセスすると，途中でディスパッチが発生し，別のタスクによって文字が送信されると送信文字列が意味の通らないことになる。この場合，同期を取るためにセマフォやミューテックス等を使用してロックするかディスパッチを禁止する必要がある。miniOSではセマフォやミューテックスのような同期オブジェクトを実装してないので，ディスパッチ禁止期間にしている。ディスパッチ禁止期間はリアルタイム性を損なう原因になるので，できるだけ少なく，短くすることが必要となる。Syslog_taskは，シスログバッファを通してシリアルポートから出力するタスクである。その様子を図 7 に示す。

シリアルRS232Cを使用してタスクをリモート制御するために，モニタタスクを作成している。モニタタスクをリスト 4 に示す。

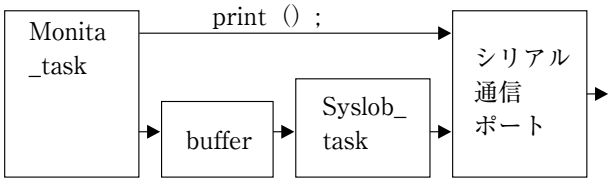

図 7　シリアル送信の仕組み

```
void monita_task(void){
  static ID  taskno=1;
  char* taskname ;
  unsigned char c;
  PRI priority;
  ATR status;

  logS("//////// モニタ（monita_task///////¥r¥n"）;
  dly_tsk(100);    // 1 秒で10000文字程度送信可能

  while(1){
    // 受信オーバーランエラーの検出とクリア
    if (OERR){
      CREN = 0;
      OERR = 0;
      CREN = 1;
    }

    if (RCIF){// シリアル受信チェック
      c = RCREG;
      switch(c){  // タスク指定
        case '1':
          taskno = 1;
          break;
        case '2':
          taskno = 2;
          break;
        case '3':
          taskno = 3;
          break;
        case '4':
          taskno = 4;
          break;
        case '5':
          taskno = 5;
          break;
        case '6':
          taskno = 6;
          break;

        case 'a':  // タスクの起動
```

```
        act_tsk(taskno);
        break;

      case 't':  // タスクの終了
        ter_tsk(taskno);
        break;

      case 'u':  // タスク優先度のUp
        ref_tsk(taskno, taskname , &priority ,&status);

        if (priority < (TCB_PRIORITY-1))
          chg_pri(taskno , (priority+1));
        break;

      case 'd':  // タスク優先度のDown
        ref_tsk(taskno, taskname , &priority , &status);

        if (priority > 0)
          chg_pri(taskno , (priority-1));
        break;

      case 'i':  // タスク情報（タスク名,優先度,ステータス）
        taskinfo1(taskno); // タスク名
        dly_tsk(50);
        logS(",");
        dly_tsk(50);
        taskinfo2(taskno); // 優先度
        dly_tsk(50);
        logS(",");
        dly_tsk(50);
        taskinfo3(taskno); // ステータス
        dly_tsk(50);
        logS("¥r¥n");
        dly_tsk(50);
        break;

      case 'p':  // タスク遷移状態一覧
        print();
        break;

      case 'r':  // 優先度 0 のラウンドロビン
        if (tss == 0){
          tss = 1;
          logS("TSS = ON¥r¥n");
          dly_tsk(20);
        }else{
           tss = 0;
          logS("TSS = OFF¥r¥n");
          dly_tsk(20);
        }
        break;

      case 'h':  // モニタコマンド表示
        logS("//////// ﾓﾆﾀｺﾏﾝﾄﾞ一覧///////¥r¥n");
        dly_tsk(100);  // 1秒で1000文字程度送信可能
        logS("// リモート制御 9600bps¥r¥n");
        dly_tsk(100);
        logS("//  comannd¥r¥n");
        dly_tsk(20);
        logS("//  1..6 タスク指定¥r¥n");
        dly_tsk(100);
        logS("//  a タスクの起動¥r¥n");
        dly_tsk(100);
        logS("//  t タスクの終了¥r¥n");
        dly_tsk(100);
        logS("//  u タスク優先度Up¥r¥n");
        dly_tsk(100);
        logS("//  d タスク優先度Down¥r¥n");
        dly_tsk(100);
        logS("//  i タスク情報の表示¥r¥n");
        dly_tsk(100);
        logS("//  p タスク遷移状態一覧¥r¥n");
        dly_tsk(100);
        logS("//  r 優先度 0 のﾗｳﾝﾄﾞﾛﾋﾞﾝ切り替え¥r¥n");
        dly_tsk(100);
        logS("//¥r¥n");
        dly_tsk(10);
        logS("//  h   ヘルプ¥r¥n");
        dly_tsk(100);
        logS("//¥r¥n");
        dly_tsk(10);
        break;

      }
      RCIF = 0;
    }

    dly_tsk(100);
    }

}
```

リスト 4　モニタタスクの内容

7．プログラム容量

今回作成したminiOS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　8557 Bytes （全体の26.1 %）

RAM　425 Bytes （全体の27.7％）

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　7783 Bytes （全体の23.8％）

RAM　423 Bytes （全体の27.5％）

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　5663 Bytes （全体の17.3％）

RAM　422 Bytes （全体の27.5％）

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　5123 Bytes （全体の15.6％）

RAM　421 Bytes （全体の27.4％）

8．最後に

前編と今回の2回に分けて，2種類のスケジューリングを行うOSの製作を紹介してきた。同じ動作をさせるにも，OSのスケジューリングが変わればアプリケーションにも多少なり影響する部分が出てくる。

今回のOSでは，プリエンプティブなスケジューリングを行っているが，PICマイコンの制約でいくつかの制限が生じている。

① RAM（1536 bytes）のBANKによる制限

上述しているが，RAM領域が複数のBANKに分かれているため，フラットな領域として使用できていない。そのためRAM領域全体を有効利用できていない。

② スタックの制限

PIC18F452になって，スタックをプログラムから操作できるようになったが，スタック領域を変更できない。そのため，タスクごとのスタック領域を用意できていない。タスクからの関数呼び出し期間中にディスパッチされないように工夫が必要となる。今回のプログラムでは，システムクロック（1mS毎）の割り込み後にディスパッチされるように遅延しているので，単に割り込み禁止期間としている。その分リアルタイム性を損なう原因となりやすい。共有スタックとして実装する方法（TCBにタスク毎のスタック情報を持たせる方法）もあるが，仕組みが少々複雑となり，ディスパッチ時間，TCBの増大につながることなどから採用していない。

③ コンテキストの制限

OSとして，どの範囲までのコンテキストを保存する必要があるか仕様の問題でもあるが，すべてのレジスタを範囲にするのが理想的といえる。

今回のプログラムでは基本レジスタ（WREG，STATUS，BSR）のみ退避させている。もっと退避する範囲を広げたい場合は，割り込みの飛び先（000008hまたは000018h）にレジスタの退避プログラムを記述することになる。TCBをできるだけ小さくしたかったので，基本レジスタのみ退避している。基本レジスタは関数呼び出しや割り込み時にハードウェアによって自動的にFast Register Stackへ退避･復旧される。Fast Register Stackの直接操作はできないが，復帰したときにタスクコンテキストとしてTCBに保存している。

また前編と同様に，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。

今回製作したminiOSは，100％C言語で製作したので可読性に優れているといえる。利用法にもよるが，20～40時間程度のリアルタイム・マルチタスク制御実習教材にすることが可能と考えられる。特にPICを使用した制御実習の延長に効果的と考える。今後は60～120時間程度の実習で使用可能な教材の作成を検討したい。

＜参考文献＞

1）「電子工作の実験室」　http://www.picfun.com/

2）PIC18FXX2 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』，共立出版株式会社.

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）AzukiRTOS
http://www2.noritz.co.jp/anchor/ashp/azrtos/azindex.html

9）「小規模リアルタイム・マルチタスク処理用OSの製作(1)」，『技能と技術』，5/2003，p51～p57.