オリジナルOS「MicrOS」の設計と実装

第1回 フリーのリアルタイムOSを使ってみよう!

田口 信夫

本誌2007年5月号付属のV850マイコン基板で動作するオリジナルOS「MicrOS（マイクロス）」が登場した．MicrOSはμITRONに似た機能をもち，それでいてソース・コードは5000行程度と非常にコンパクトである．本連載では，MicrOSを題材にして組み込み向けリアルタイムOSの設計と実装について見ていく．（編集部）

関連データ

1. MicrOSの概要とソースの概観

1-1. MicrOSとは何か

この度，本誌2007年5月号付属V850マイコン基板で動作するオリジナルOSを作成しました．「MicrOS」という名前で，読者の皆さんにも使える形で公開しています[注1]．

MicrOSは本連載で扱うOSの名前です．V850で動作するように作られているので，「V850-MicrOS」と呼んでいます[注2]．MicrOSは，組み込みOSの作り方に付けた名前でもあります．なぜ組み込みOSの作り方に名前を付けたのかというと，組み込みOSの作り方を説明したかったからです．MicrOS方式によって構造が単純化され，組み込みOSを簡単に構成することができます．

リスト1　aplMain部

MicrOS.c内．今回はApplication.c内のものを使用しているので未使用．

```
008: /************** Application main *****************/
009: #if 1  /*** 1: aplMain in another source, 0: in this
source ***/
010:
011: #if USING_TIMERTASK
012: extern _TCB        *__aplTCBList[];
013: #define tmrTCB __aplTCBList[0]         /* Timer Task */
014: #endif
015:
016: #else
017:
018: #define NTASKS     16
019: _TCB   *__aplTCBList[NTASKS];
020: #define tmrTCB __aplTCBList[0]         /* Timer Task */
021:
022: /* Uart buffer */
023: #define    UARTSSZ     256
024: #define    UARTRSZ     128
025: char   uartbuf[UARTSSZ+UARTSSZ];
026: /* Debug buffer */
027: #define    DBGBFSZ     2048
028: char   debugbuf[DBGBFSZ];
029: /*=====================================================+
030: |       application main                              |
031: +=====================================================*/
032: void aplMain_template(void)
033: {
```

V850-MicrOSのサービスは機能的にはμITRON 3.0に相当します．MicrOSと同じような機能をμITRONのインターフェースに書き換えたとしても，大した作業にはなりません．現在のV850-MicrOSのシステム・コールは，μITRONの機能のうち比較的使用すると予想される機能を取り出して，それにMicrOS独自の機能を幾つか付け加えたと思っていただければ結構です．

MicrOSにはアプリケーション・システム（ユーザ・アプリケーション）を構成する上でぜひ利用してもらいたい機能があります．それはMicrOSのデマンド・コールバック関数です．一言で言うと，デマンド型のシステム・コールとコールバック関数を組み合わせた機能で，タイマ制御やUARTの入力処理に利用できます．この機能を使うとタイマを使ったアプリケーションなどが単純化され，アプリケーション・システムの見通しをよくできます．このようにMicrOSは単なるμITRONのサブセットではなく，組み込みアプリケーション・システムの構成方法に提案を行っている部分も持っています．

1-2. V850-MicrOSの全体構成

● MicrOS.cのソース・コードを読む

MicrOSの説明に入る前に，V850-MicrOSのソース・

注1：MicrOSのソース・コードはCQ出版社のWebサイトよりダウンロードできる．
http://www.cqpub.co.jp/interface/download/contents.htm

注2：本連載では扱わないが，MicrOSにはV850アセンブリ・コード版，ARMアセンブリ・コード版も存在する．

KEYWORD —— MicrOS，リアルタイムOS，マルチプログラミング，マルチタスク，CA850，割り込み処理，システム・コール，V850マイコン基板，PM+，リンク・ディレクティブ・ファイル，デバッガ

リスト2　タイマ関連

```
105: #if 0
106: /** Write/Read Hardware timer & Arrenge Timervalue **/
107: /*=================================================+
108: |   write Hardware Timer     return:remain value    |
109: +=================================================*/
110: unsigned int _writeTimerValue_template(unsigned int
value)
111: {
112: #if SYSCLK_READ_OK
113:     TP5CTL0 = 0;        /* TMP5 reset */
114:     TP5CTL1 = 0x00;     /* set TMP5 to interval mode */
115:     TP5OPT0 = 0x00;
```

コードを見てみましょう．MicrOS.cをエディタで開いてみてください．

ここにはMicrOSのシステム・コールのほとんどが収められています．各機能は，

/*********** <機能名> ***********/

で区切られているので，各システム・コールの処理量は一目で分かるようになっています．

最初は__aplMainという関数です（**リスト1**）．この関数はアプリケーション・システムの初期設定を行う関数のテンプレートです．#if 1を #if 0に変更すればここに__aplMainの実体が展開されます．

次にタイマ関係の関数があります（**リスト2**）．これはハードウェアの違いを吸収するための関数でMicrOSのタイマ制御で用いられます．

ここまでの関数はアプリケーション・システムによって変更される可能性のあるプログラムです．

その次に現れるのはMicrOSの中で比較的共通に使われる関数群（**リスト3**）で，アプリケーション・システムがこれらの関数を直接コールすることはありません．この中では_SCB，_RQB，_TCBといった構造体が使われています．これらはMicrOSを構成するときに鍵となる構造体なのですが，その説明は後回しにして，ここではさらにMicrOS.cを眺めていきましょう．

この後はシステム・コールのコーディングです（**リスト4**）．この辺りの関数は数行の小さなものばかりです．どんどん先に進みます．しばらく進むと400行目あたりに__wait関数（**リスト5**）と__active関数があります．この関数はMicrOSを構成するときの鍵となる関数ですから名称だけを記憶しておいてください．どんどんいきます．

550行目あたりにメモリ・アロケーションがあります（**リスト6**）．これは可変長のメモリ・アサインです．malloc

リスト3　共通関数群（FIFOの処理）

```
191: /****** OS function(interrupt in inhibit) *******/
192: /*=================================================+
193: |       FirstIn FirstOut Link                      |
194: +=================================================*/
195: void fifo(void *scb, void *rqb)
196: {
197:     _SCB            *s;
198:     _RQB            *r;
199:
200:     s = scb;
201:     r = rqb;
202:     r->next = 0;
203:     if(s->head == 0){
204:         s->head = r;
205:         s->tail = r;
206:         r->prior = (_RQB *)s;
207:     } else {
```

リスト4　システム・コールの関数（プライオリティの変更）

```
324: /*=================================================+
325: |       Change Priority                             |
326: +=================================================*/
327: int __chgPri(unsigned char pri)
328: {
329:     int             irc;
330:     _TCB            *t;
331:
332:     if(pri >= SYSNPRI) return(E_PRI);
```

リスト5　__wait関数

```
411: int __wait(unsigned int waitstatus)
412: {
413:     _TCB            *t;
414:     int             psw, irc;
415:
416:     while(__cpumode() != 0){};
/* if(not UserMode) then eternal loop */
417:     irc = E_WAITFLAG;
418:     psw = __DIC();
419:     if((_sysctrl->cur == (_TCB *)0)||( waitstatus == 0))
goto p9;
420:     /*** set TCB status ***/
```

リスト6　メモリ関連の関数（malloc）

```
548: /*********** Memory Allocation ************/
549: /*=================================================+
550: |       Memory Allocation
|
551: +=================================================*/
552: static void *malloc(unsigned int size)
553: {
554:     union{
555:         void            *v;
556:         unsigned int    *pui;
557:         unsigned int    ui;
558:     } top, lmt, adr, w;
559:     unsigned int        sz;
560:
561:     size = (size + (3+8)) & (~0x03);
```