第3章 タスク・スケジューラ内部の仕組み

MiniOSを作る

菅原 政義

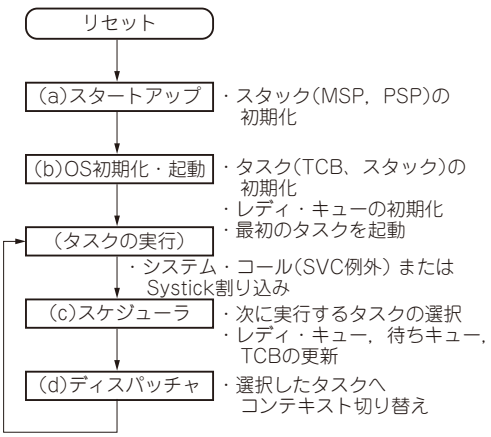

図1 MiniOSの処理の流れ

ここでは，MiniOSの内部の仕組みと実装について，詳しく解説していきます．Cortex-M4搭載マイコンで動作するタスク・スケジューラ内部の仕組みについて，理解できることを目指します．

■ MiniOSの処理の流れ

MiniOSの処理の流れを**図1**に示します．CPUのリセット後，(a)のスタートアップを経てC言語のmain関数に到達します．main関数で(b)のOSの初期化・起動を行い，MiniOSのタスク情報を初期化します．初期状態では，全タスクを実行可能状態とするのでレディ・キューに登録を行っています．その中で最も優先順位の高いタスクを起動して，MiniOSの動作を開始します．その後，タスク実行中に，システム・コールを呼び出したり，SysTick割り込みが発生したりすると，MiniOSのタスク・スケジューラおよびディスパッチャが実行されます．(c)のスケジューラでは，システム・コールの種類やシステム状態によってタスクの状態を更新し，次に実行するタスクを選択します．(d)のディスパッチャでは，スケジューラで選択したタスクに切り替え(ディスパッチ)を行います．続いて，各処理について詳しく説明します．

リスト1 ベクタ・テーブル(アセンブリ)

```
__Vectors:
    .long   __main_stack_start
                    /* Main stack pointer (MSP) */
    .long   Reset_Handler       /* Reset Handler */
```

リスト2 MiniOSのTCB定義

```
typedef struct tcb {
  uint32_t *  p_sp;     // 各タスクのスタックへのポインタ
  struct tcb *p_prev;   // 前のタスク (レディキュー，
                                        待ちキューで使用)
  struct tcb *p_next;   // 次のタスク
  int32_t     id;       // タスクID
  int32_t     pri;      // タスク優先度
  int32_t     state;    // タスクの状態
  uint32_t    time_up;  // 起動時刻(dly_tsk()で使用)
} tcb_t;

tcb_t _tcb_tbl[TASK_NUM];
```

■ (a)スタートアップ

リセットからmain関数の呼び出しまでCPUがリセットされると，ベクタ・テーブルに登録されているリセット・ハンドラReset_Handler()から処理を開始します．このとき，ベクタ・テーブルの先頭の値がスタック・ポインタ(MSP)として設定されます(**リスト1**)．リセット・ハンドラ内でメモリ領域(bss，data)の初期化を行った後，main()を呼び出します．

■ (b)OS初期化・起動

● TCBとタスク・スタック

マルチタスクのOSでは，タスクの管理情報として，Task Control Block(TCB)という情報を各タスクにつき1つずつタスクの数だけ保持します．TCBには，各タスクのスタック・ポインタ，キューの接続情報，タスクの属性情報を持ちます．MiniOSのTCB定義を**リスト2**に示します．

それぞれのタスクは，スタックを共有せずに，独立したスタック領域を持ちます(**リスト3**)．スタックには，タスクの実行中に変数を格納したり，タスク切り替え時にコンテキスト情報を格納したりします．タスクごとにスタックを切り替えて利用することで，タス