SW 実験Ⅱテーマ2,3

レポート

C過程 S-15組 1TE20137W 2023/01/18

柳 鷹

目次

[テーマ2:マルチタスクカーネルの制作 1](#_Toc125562104)

[テーマ2について 1](#_Toc125562105)

[テーマ2の実験全体に占める位置づけ 1](#_Toc125562106)

[プログラムのリスト及びその説明 2](#_Toc125562107)

[プログラムにおいて、注意したこと 17](#_Toc125562108)

[プログラムに発生した問題とその原因、解決方法 17](#_Toc125562109)

[テーマ3:応用 18](#_Toc125562110)

[テーマ3について 18](#_Toc125562111)

[テーマ3の実験全体に占める位置づけ 18](#_Toc125562112)

[プログラムのリスト及びその説明 18](#_Toc125562113)

[プログラムにおいて、注意したこと 30](#_Toc125562114)

[プログラムに発生した問題とその原因、解決方法 30](#_Toc125562115)

[個人課題について 30](#_Toc125562116)

[考察 31](#_Toc125562117)

# テーマ2:マルチタスクカーネルの制作

## テーマ2について

本実験のテーマ2では、テーマ1で移植したC言語標準入出力ライブラリを用いてマルチタスクカーネル処理、つまり複数の「仕事」を1つのシステム下で切り替えながら実行する処理を記述することが目的である。タスクの切り替えは以下の二つがある。

* タスク間で資源を共有する場合の排他制御（P・Vシステムコール）
* タイマ割り込みによるタスクの切り替え

これらを実装するために2つのキュー　－readyキュー、各共有資源のキュー　を用意する。前者は順番が来たらいつでも実行可能なタスクの待ち行列で、後者は各共有資源の（後述する）セマフォが持つキューで他のタスクが資源を占有しているために休眠、つまり実行を待機している、タスクの待ち行列である。

## テーマ2の実験全体に占める位置づけ

ここでは、マルチタスク処理を実行する上で必要な以下の4つの事項に注意する。

1. 特権命令の概念
2. 排他制御（P・Vシステムコール）の概念
3. 1つのシステムで複数タスク切り替えながら実行するために必要な処理
4. タスク切り替えのタイミングによるカーネル再入防止

1つ目の特権命令は、OSには実行可能で利用者プログラムでは実行不可となるような処理を実装する際に必要なものである。OSと利用者プログラムで実行可能性を区別するために2つのモードを用いる。それはすべての命令が実行可能であるスーパーバイザモードと一部の命令は実行不可能となるユーザーモードである。スーパーバイザモードで実行するものは、入出力とタスク切り替えを担うタイマのセット・リセットであり、具体的にはエコーバックプログラムで作成したTRAP命令を用いる。

2つ目の排他制御は、複数のタスクが共有資源を同時に操作しないように制御するものである。各共有資源にはセマフォと呼ばれる、カウンタとキューを持つ構造体を持たせる。カウンタは初期化時には1にセットしてある。そこであるタスクがその資源を使用する際に、カウンタの値を一つ減らす。これをP命令とする。そしてタスクが資源利用を終えるとカウンタを一つ増やす。これをV命令とする。このときカウンタに着目すると、1のときは資源利用可能、0のときは他のタスクが資源利用中、0未満のときは他のタスクが利用中かつさらに他のタスクが資源利用を待っている状態を示している。そして0,0未満のときに資源利用を待っているタスクをキューに保存して、順に取り出すことで排他制御を可能にしている。

3つ目の1つのシステムで複数タスク切り替えながら実行するために必要な処理は、タスクの実行が中断されreadyキューに計算機内部の状態が正しく保存され、逆にタその中断されていたタスクを回復する際に元の計算機内部の状態を復元することである。具体的には、実行中断時のPC（プログラムカウンタ）の値、レジスタの値、スタックの内容を保存・復元が必要である。

4つ目のタスク切り替えのタイミングによるカーネル再入防止は、本実験においてはP・V命令とタイマ割り込みの整合性をとるために、P・V命令の発効前に走行レベルを7にして割り込みを禁止する。

## プログラムのリスト及びその説明

本テーマではグループでの共同作業であったため、担当部分については詳細に、担当外の部分については概要を記す。なお当グループの担当箇所の分担は、マルチタスク制御、タイマ制御、セマフォ制御、タスク切り替え制御部の5つで、当人は**タイマ制御部**（init\_timer: , set\_timer:, reset\_timer:, hard\_clock）を担当した。

テーマ2で実装・変更したプログラムを以下に示す。

mtk\_c.h

|  |
| --- |
| /\*  \* mtk\_c.h  \*  \* 各種定数を定義する  \*/  #ifndef mtk\_c\_H  /\* if not defined this \*/  #define mtk\_c\_H  /\* define it \*/  /\* 定数の定義 \*/  #define NULLTASKID      0  /\* キューの終端 \*/  #define NUMTASK         7  /\* 最大タスク数 \*/  #define STKSIZE      1024  /\* スタックサイズ = 1KB \*/  #define NUMSEMAPHORE    3  /\* intの型エイリアス \*/  typedef int TASK\_ID\_TYPE;  /\* 各構造体の定義 \*/  typedef struct {      int          count;      TASK\_ID\_TYPE task\_list;  } SEMAPHORE\_TYPE;  typedef struct{      void         (\*task\_addr)();      void         \*stack\_ptr;      int          priority;      int          status;      TASK\_ID\_TYPE next;  } TCB\_TYPE;  typedef struct{      char ustack[STKSIZE];      char sstack[STKSIZE];  } STACK\_TYPE;  /\* 大域変数の宣言 \*/  TASK\_ID\_TYPE curr\_task;  TASK\_ID\_TYPE new\_task;  TASK\_ID\_TYPE next\_task;  TASK\_ID\_TYPE ready;  SEMAPHORE\_TYPE semaphore[NUMSEMAPHORE];  TCB\_TYPE task\_tab[NUMTASK + 1];  STACK\_TYPE stacks[NUMTASK];  /\* 関数のプロトタイプ宣言 \*/  void init\_kernel(void);  void set\_task(void \*p);  void\* init\_stack(int id);  void begin\_sch(void);  void sched(void);  TASK\_ID\_TYPE removeq(TASK\_ID\_TYPE \*q);  void addq(TASK\_ID\_TYPE \*q, TASK\_ID\_TYPE task\_id);  //void addq1(TASK\_ID\_TYPE \*q, TASK\_ID\_TYPE task\_id);  void sleep(int s\_id);  void wakeup(int s\_id);  void p\_body(int s\_id);  void v\_body(int s\_id);  /\* 関数のextern宣言 \*/  extern void first\_task(void);  extern void pv\_handler(void);  extern void P(int semaphore\_id);  extern void V(int semaphore\_id);  extern void swtch(void);  extern void hard\_clock(void);  extern void init\_timer(void);  #endif  /\* mtk\_c\_H \*/ |

ヘッダファイル” mtk\_c.h”では、” mtk\_c.c”と”test2.c”の外部で定義される関数のextern宣言とカーネル関連の大域変数を含んでいる。

大域変数

curr\_task 現在実行中のタスクID

new\_task 現在登録中のタスクID

next\_task 次の実行するタスクID

ready 実行待ちのタスクキュー

semaphore 共有資源のセマフォ

task\_tab タスクコントロールブロックの配列

stacks タスクスタックの配列

mtk\_c.c

|  |
| --- |
| /\*  \*  \* sched()   : タスクのスケジュール関数  \* removeq() : キューからタスクを取り出す  \* addq()    : キューの最後尾にタスクを登録  \* sleep()   : タスクを休眠状態にしてタスクスイッチをする  \* wakeup()  : 休眠状態のタスクを実行可能状態にする  \*  \* In the future, these functions will be included into mtk\_c.c  \*/  #include "mtk\_c.h"  #include <stdio.h>  void init\_kernel() {      ready = 0;                  //readyの初期化      \*(int\*)0x084 = (int)pv\_handler;  //pv\_handlerをtrap#1の割り込みベクタに登録      for (int i=0; i<NUMSEMAPHORE; i++) {          semaphore[i].count = 1;              semaphore[i].task\_list=0;      }          for (int i=0; i<NUMTASK; i++) {              task\_tab[i+1].task\_addr=0;              task\_tab[i+1].stack\_ptr=0;              task\_tab[i+1].priority=0;              task\_tab[i+1].status=0;              task\_tab[i+1].next=0;          }    }  void set\_task(void \*p) {      int i = 1;      while (1) {                                     //タスクIDの捜索          if (task\_tab[i].status == 0) {              break;          }          i = i+1;      }      new\_task = i;      task\_tab[new\_task].task\_addr = p;                        //TCBの更新      task\_tab[new\_task].status = 1;      task\_tab[new\_task].stack\_ptr = init\_stack( new\_task );          //スタックの初期化      addq(&ready, new\_task);                         //キューへの登録    }  void \*init\_stack(int id) {      int \*ssp1;                            //4バイト用int      unsigned short int \*ssp2;             //2バイト用unsigned short int      void \*back;                           //戻り値用      ssp1 = (int \*)&stacks[id-1].sstack[STKSIZE];    //stacks[id-1]に各値を代入      \*(--ssp1) = (int)task\_tab[id].task\_addr;      ssp2 = (unsigned short int \*)ssp1;    //2バイトに切り替え      \*(--ssp2) = 0x0000;      ssp1 = (int \*)ssp2;      for (int i=0; i<15; i++) {            //15\*4バイト戻す          \*(--ssp1);      }        \*(--ssp1) = (int \*)&stacks[id-1].ustack[STKSIZE];        back = (void \*)ssp1;      return back;  }  void begin\_sch()  {      int i;      i = removeq(&ready);      curr\_task = i;      init\_timer();      first\_task();      return;  }  TASK\_ID\_TYPE removeq(TASK\_ID\_TYPE \*q){      // 先頭タスクのidを保存      TASK\_ID\_TYPE head\_task\_id = \*q;      // 先頭タスクを除去して新たなタスクidを登録      \*q = task\_tab[head\_task\_id].next;        return head\_task\_id;  }  void sched(void) {      printf("\n\n timer ---------\n\n");      // readyキューの先頭タスクを取り出してnext\_taskへ      next\_task = removeq(&ready);      // next\_taskがNULLの場合はタイマ割り込みが来るまでループ      while(!next\_task);  }  void addq(TASK\_ID\_TYPE \*q, TASK\_ID\_TYPE task\_id){      // idとして先頭idを保存      TASK\_ID\_TYPE id = \*q;      \*(char \*)0x00d0002f = 'a';      if (!id) { // 渡されたqueueの先頭がNULLだった場合          \*q = task\_id;          task\_tab[task\_id].next = NULLTASKID;          return;      }      \*(char \*)0x00d0002d = 'a';      // task\_tabのnextがNULLになるまでたどる      while (task\_tab[id].next) id = task\_tab[id].next;      \*(char \*)0x00d00039 = 'a';      // 最後尾にタスクidを追加      task\_tab[id].next = task\_id;      task\_tab[task\_id].next = NULLTASKID;    }  void sleep(int s\_id){      TASK\_ID\_TYPE\* head\_q = &(semaphore[s\_id].task\_list);      // 該当セマフォの待ち行列に現タスクをつなぐ      addq(head\_q, curr\_task);      // 次に実行するタスクidをnext\_taskにセット      sched();      // タスクを切り替え      swtch();  }  void wakeup(int s\_id){      // セマフォの待ち行列の先頭タスクを取得      TASK\_ID\_TYPE\* head\_sem = &(semaphore[s\_id].task\_list);      // セマフォの待ち行列の先頭タスクをキューから除去      TASK\_ID\_TYPE head\_task = removeq(head\_sem);      // readyキューにその先頭タスクを追加      addq(&ready, head\_task);  }  void p\_body(int s\_id){       semaphore[s\_id].count--; //セマフォのcount値を1減らす     if (semaphore[s\_id].count < 0){           sleep(s\_id);       //セマフォが獲得できない場合は休眠状態に入る  }  }  void v\_body(int s\_id){     semaphore[s\_id].count++; //セマフォのcount値を１増やす     if (semaphore[s\_id].count <= 0){           wakeup(s\_id);      //セマフォが空けばそのセマフォを待っているタスクを１つ実行可能状態にする  }  } |

init\_kernel

TCB配列の初期化、readyキューの初期化、P・V命令のベクタ登録、セマフォの初期化を担う。

set\_task

ユーザータスク関数へのポインタを引数に取り、task\_tabの中から空きスロットを見つけ、new\_taskに代入する。それをもとにTCBの更新、スタックの初期化、readyキューへの登録を行う。これはtestファイルのmainの最初に呼び出される。

init\_stack

ユーザータスク用のスタックsspの初期化を担う

begin\_sch

readyキューからタスクを一つ取り出しcurr\_taskに代入し、タスク切り替えのタイマの開始、最初のタスクの起動を担う。

addq

引数にキューへのポインタとタスクIDを取り、TCBをキューの最後に登録する。

removeq

引数にキューへのポインタとタスクIDを取り、キューから先頭タスクIDを返す。

p\_body

セマフォのカウントを1つ減らした後、カウントが0未満ならそのタスクをキューに入れ、休眠状態にさせる。

v\_body

セマフォのカウントを1つ増やした後、カウントが0以下ならキューの先頭タスクを復帰させ、readyキューに入れる。

sched

readyキューの先頭タスクを取り出して、next\_taskに登録する。またnext\_taskがnullの場合はタイマ割り込みがあるまで無限ループを行う。

sleep

タスクを休眠状態にする。一セマフォの待ち行列に休眠すべきタスクをつなぎ、next\_taskのセット、タスク切り替えを行う。

wakeup

休眠状態のタスクを回復させる。一セマフォのキューの休眠状態だった先頭タスクをポップし、readyキューに追加する。

mtk\_asm.s

|  |
| --- |
| .section .bss  .global first\_task  .global pv\_handler  .global P  .global V  .global swtch  .global hard\_clock  .global init\_timer  .extern addq  .extern sched  .extern curr\_task  .extern next\_task  .extern task\_tab  .extern ready  .equ    SIZE\_OF\_TCB, 0x14  .equ IOBASE,    0x00d00000  .equ LED7,  IOBASE+0x000002f | ボード搭載のLED用レジスタ  .equ LED6,  IOBASE+0x000002d | 使用法については付録 A.4.3.1  .equ LED5,  IOBASE+0x000002b  .equ LED4,  IOBASE+0x0000029  .equ LED3,  IOBASE+0x000003f  .equ LED2,  IOBASE+0x000003d  .equ LED1,  IOBASE+0x000003b  .equ LED0,  IOBASE+0x0000039  .section .text  .even  swtch:                  \* %SR,全レジスタの値をスタックに積む                  move.w %SR, -(%SP)                  movem.l %D0-%D7/%A0-%A6, -(%SP)                  move.l  %USP, %a0                  move.l  %a0, -(%SP)                  jsr     get\_tcb\_ssp              | %a0 <- TCBのSSPの位置                  move.l  %SP, (%a0)               | SSPをTCBの所定の位置に格納          move.l  next\_task, curr\_task                  jsr     get\_tcb\_ssp              | %a0 <- TCBのSSPの位置                  move.l  (%a0), %SP               | TCBに記録されているSSPを回復                  bra     end\_swtch  get\_tcb\_ssp:                  move.l  curr\_task, %d0           | %d0 = 現タスクのID 例 d0 = 2                  move.l  #SIZE\_OF\_TCB, %d1        | %d1 = TCBのサイズ voidが４バイトで要素が５個だから4\*5                  mulu.w  %d0, %d1                 | 該当タスクのTCBにアクセスするためのオフセット                  lea.l   task\_tab, %a0            | task\_tabの先頭アドレス                  adda.l  %d1, %a0                 | アドレスとオフセットを加算= 現                  adda.l  #4,  %a0                 | task\_tab[ID].stack\_ptrの位置を取得                  rts  end\_swtch:          move.l  (%SP)+, %A0                  move.l  %A0, %USP                  movem.l (%SP)+, %D0-%D7/%A0-%A6 | 各レジスタを復帰                  rte  pv\_handler:       movem.l %D0-%D7/%A0-%A6, -(%sp) |使用可能性のあるレジスタの退避       move.w %SR, -(%sp)  |SRの退避       move.w #0x2700, %SR  |走行レベルを７へ       movem.l %D1, -(%sp)  |D1を引数としてスタックに積む         cmp.l  #0, %D0       |Pからの呼び出しの場合       beq    p\_order       cmp.l  #1, %D0       |Qからの呼び出しの場合       beq    v\_order    p\_order:       jsr p\_body      |p命令実行       bra end\_pv\_handler  v\_order:       jsr v\_body      |v命令実行       bra end\_pv\_handler    end\_pv\_handler:       add.l #4, %sp        |引数(d1)分spを上げる       move.w (%sp)+, %SR  |SRを復帰       movem.l (%sp)+, %D0-%D7/%A0-%A6  |レジスタの復帰       rte    P:       movem.l %D0-%D1, -(%sp)       move.l #0, %D0        |PシステムコールのID＝０をD0レジスタに       move.l  12(%sp) ,%D1   |スタックに入っている引数をD1に（d0-d1+戻り番地のPC= 12）       trap #1       movem.l (%sp)+, %D0-%d1       rts    V:       movem.l %D0-%D1, -(%sp)       move.l #1, %D0        |VシステムコールのID＝1をD0レジスタに       move.l  12(%sp) ,%D1  |スタックに入っている引数をD1に（d0-d1+戻り番地のPC= 12）       trap #1       movem.l (%sp)+, %D0-%d1       rts    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* hard\_clock  \*\* SW実験1で作成したタイマ用ハードウェア割り込み処理インターフェースから呼び出される  \*\*  \*\* このルーチン内で使用するレジスタをスーパーバイザスタックに積むが、  \*\* タイマ割り込みで実行されるルーチンであるため退避忘れが多いらしい  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  hard\_clock:          movem.l %D0-%D7/%A0-%A6, -(%sp)         | 使用可能性のあるレジスタの退避          move.w  %SR, %D0                        | %SRを%D0へ          btst.l  #13, %D0                        | SRの13bit目を見てスーパーバイザか否かを確認          beq     end\_hard\_clock                  | 13bit目が1でなければ、スーパーバイザモードでなければ終了          move.l  curr\_task, -(%sp)          move.l  #ready,   -(%sp)         move.b   #'a', LED1          jsr     addq                            | current\_taskをreadyの末尾に追加          add.l   #8, %sp          move.b  #'a', LED2          jsr     sched                           | schedを起動し、次に実行されるタスクIDがnext\_taskにセット          move.b  #'a', LED3          jsr     swtch                           | swtchの起動          move.b  #'a', LED4  end\_hard\_clock:          movem.l (%sp)+, %D0-%D7/%A0-%A6         |レジスタ復帰退避          rts  init\_timer:          movem.l %D0-%D2, -(%sp)         | 使用可能性のあるレジスタの退避          move.l  #3, %D0                 | システムコールRESET\_TIMERの番号          trap    #0                      | RESET\_TIMER呼び出し          move.l  #4, %D0                 | システムコールSET\_TIMERの番号          move.w  #10000, %D1             | 割り込み発生周期は1秒に設定(p38)          move.l  #hard\_clock, %d2        | 割り込み時に起動するルーチンの先頭アドレス          trap    #0                      | SET\_TIMER呼び出し          movem.l (%sp)+,%D0-%D2          | SRの退避          rts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*first\_task  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  first\_task:              move.l curr\_task, %d0     /\* TCB先頭番地の計算 \*/          lea.l  task\_tab, %a1      /\* 見つけたアドレスを%a1に \*/  loop\_first\_task:          subq.l #1, %d0            /\* 配列の何番目なのか計算 \*/          add.l  #0x14, %a1         /\* %a1:先頭アドレス \*/          cmp    #0, %d0          bne    loop\_first\_task          /\*USP,SSPの値の回復\*/          add.l   #4, %a1           /\* a1にTCBのSSP \*/          move.l  (%a1), %ssp          movem.l (%sp)+, %a0          move.l  %a0, %usp          movem.l (%sp)+, %d0-%d7/%a0-%a6                  rte                       /\*rte\*/ |

first\_task

最初のタスクの起動を担う。

pv\_handler

走行レベルを7にして他の割り込み（特に、タイマ割り込み）を禁止してP・V命令を発行する。

P

Pシステムコールの入口で、pv\_handlerから呼ばれてTRAP #1命令を実行する。

V

Ｖシステムコールの入口で、pv\_handlerから呼ばれてTRAP #1命令を実行する。

swtch

タスク切り替えを起こす関数で、PC（プログラムカウンタ）の値、レジスタの値、スタックの内容を保存・復元を行う。

**hard\_clock**

スーパーバイザモードでタイマ割り込みを行うサブルーチンである。まず、addqを呼び出して実行中のタスクをreadyキューの末尾に追加する。次にshedを呼び出しreadyキューの先頭にあるタスクをnext\_taskにセットしてswtchを起動することでタイマ割り込みを可能にしている。

**init\_timer**

上述のhard\_clockをベクタテーブルに登録するサブルーチンである。mon.s（エコーバックプログラム）のTrap #0命令でシステムコール3,4番のRESET\_TIMERとSET\_TIMERを呼び出している。なお、割り込み周期は目視しやすいように1秒（=10000[0.1ms]）としている。

test2.c

|  |
| --- |
| #include <stdio.h>  #include "mtk\_c.h"  extern void init\_kernel();  extern void set\_task(void \*a);  extern void begin\_sch();  #define L  500  #define M  1000  #define N  2000  void task1(){  while(1){    // task1-a    for (int j = 0;j<L;j++){      printf("task1-a %d \n", j);      /\* L: 1 秒間ループ、この待機ループの間にタイマ割込みは生じない \*/    }    P(0);    // task1-b    for (int j = 0;j<L;j++){      printf("task1-b %d \n", j);      /\* L: 1 秒間ループ、この待機ループの間にタイマ割込みは生じない \*/    }    V(0);    P(1);    // task2-c    for (int i = 0;i<M;i++){      printf("task1-c %d \n", i);      /\* M: 2 秒間ループ、この待機ループの間にタイマ割込みが必ず生じる \*/    }    V(1);  }}  void task2(){  while(1){    P(0);    // task2-a    for (int i = 0;i<M;i++){      printf("task2-a %d \n", i);      /\* M: 2 秒間ループ、この待機ループの間にタイマ割込みが必ず生じる \*/    }    V(0);    P(1);    // task2-b    for (int i = 0;i<N;i++){      printf("task2-b %d \n", i);      /\* N: 3 秒間ループ、この待機ループの間にタイマ割込みが必ず生じる \*/    }    V(1);    // task2-c    for (int i = 0;i<M;i++){      printf("task2-c %d \n", i);      /\* M: 2 秒間ループ、この待機ループの間にタイマ割込みが必ず生じる \*/    }  }}  void task3(){  while(1){    // task3-a    P(0);    for (int i = 0;i<M;i++){      printf("task3-a %d \n", i);      /\* M: タスクループ1回につき、この待機ループで 1 回タイマ割込みが必ず生      じる \*/    }    V(0);    // task3-b    for (int j = 0;j<L;j++){      printf("task3-b %d \n", j);      /\* L: 1 秒間ループ、この待機ループの間にタイマ割込みは生じない \*/    }    P(1);    // taask3-c    for (int j = 0;j<L;j++){      printf("task3-c %d \n", j);      /\* L: 1 秒間ループ、この待機ループの間にタイマ割込みは生じない \*/    }    V(1);  }}  int main(void){  init\_kernel();  set\_task(task1);  set\_task(task2);  set\_task(task3);  begin\_sch();  return 0;  } |

test2.cではテーマ1のレポートで作成した思考実験を改良してマルチタスク環境下でセマフォによる動作確認をしている。

## プログラムにおいて、注意したこと

本テーマで注意したことは、コメントアウトを残すことである。共同開発で進めていくため、単体テスト（主に文法的なエラーをなくす程度）では通っても結合テストを行う場合、他の作業の責任者にプログラムの概略を記す必要がある。もちろん、各担当箇所以外の作業内容を理解しようと実験テキストに目を通すが、やはり実際にソースコードとして実装すると全体の流れは分かっていても一行一行で何をしているかについては責任者以外の理解は曖昧である。したがって、担当箇所についてはほとんど行にコメントアウトを残した。

## プログラムに発生した問題とその原因、解決方法

本テーマのD1~D5の5つのデバッグ作業で何度も問題に直面した。ただ、その直面の仕方は主に2パターンであった。それはコンパイル前の文法的なエラー（及び警告）とコンパイル後の例外エラーであった。特にバスエラーの等の例外エラーでは、printfデバッグを用いることが難しかった。そのため問題発生後、原因追求と解決のためには

* 文法的なエラーを減らしコンパイルを通す
* LEDデバッグで問題発生個所を発見する
* pritnfデバッグを行う

という流れで行った。

例えば、デバッグD1の作業では何度もバスエラーが起きて各担当者に思い当たる問題がなく、pritnfデバッグも動作しない状態で四苦八苦していた。そこでC言語ファイル・アセンブラファイルに関わらず、まず文法的なエラーを担当者がなくした後、分担してソースコード一行一行にLEDデバッグを挟み、原因追及を行った。結局、D1ではbegin\_sch()で呼び出したinit\_timer()から返ってこず、init\_timerで”rts”ではなく”rte”になっていることが分かった。それはinit\_timerの最終行でのLEDデバッグには成功し、begin\_sch()のinit\_timerの後、first\_task()の前のLEDデバッグができなかったからであった。

# テーマ3:応用

## テーマ3について

テーマ3では、テーマ2までに作成したC言語標準入出力ライブラリを用いたマルチタスク処理環境において、以下のようなプログラムを実行することが目的である。

* 3つ以上のタスクを順に実行する
* マルチタスクとして2つのシリアルポートを用いて2つのディスプレイに出力するタスクを実行する
* 一方のポートの入力をもう一方のポートで出力するようなタスクを作成し、排他制御を行う

## テーマ3の実験全体に占める位置づけ

## プログラムのリスト及びその説明

テーマ3で実装・変更したプログラムを以下に示す。

csys68k.h

|  |
| --- |
| #include <stdarg.h>  #include <fcntl.h>  #include <errno.h>  extern void outbyte(int a,unsigned char c);  extern char inbyte(int b);  int read(int fd, char \*buf, int nbytes){    int ch;    switch(fd){    case 0:    case 3:        ch = 0;        break;      case 4:        ch = 1;        break;      default:        return EBADF;        break;    }    char c;    int  i;    for (i = 0; i < nbytes; i++) {      c = inbyte(ch);      if (c == '\r' || c == '\n'){ /\* CR -> CRLF \*/          outbyte(ch, '\r');          outbyte(ch, '\n');          \*(buf + i) = '\n';   /\* } else if (c == '\x8'){ \*/  /\* backspace \x8 \*/      } else if (c == '\x7f'){    /\* backspace \x8 -> \x7f (by terminal config.) \*/        if (i > 0){            outbyte(ch, '\x8'); /\* bs  \*/            outbyte(ch, ' ');   /\* spc \*/            outbyte(ch, '\x8'); /\* bs  \*/          i--;        }        i--;        continue;      } else {          outbyte(ch, c);          \*(buf + i) = c;      }      if (\*(buf + i) == '\n'){          return (i + 1);      }    }    return (i);  }  int write (int fd, char \*buf, int nbytes){    int i, j, ch;    switch(fd){     case 1:     case 2:     case 3:         ch = 0;         break;       case 4:         ch = 1;         break;       default:         return EBADF;    }    for (i = 0; i < nbytes; i++) {      if (\*(buf + i) == '\n') {        outbyte (ch, '\r');          /\* LF -> CRLF \*/      }      outbyte (ch, \*(buf + i));      for (j = 0; j < 300; j++);    }    return (nbytes);  }  int fcntl(int fd, int cmd, ...) {    if (cmd == F\_GETFL) {      return O\_RDWR;    } else {      return 0;    }  } |

read, write

2ポート入出力を可能にするため、ストリーム入出力関数のscanf(), printf()を構成する低レベル関数read(),write() に引数としてファイルディスクリプタを渡している。テーマ2までに実装したscanf(), printf()も引き続き利用できるように

* ファイルディスクリプタ0=標準入力に対してはscanf()
* ファイルディスクリプタ1,2=標準出力・標準エラー出力に対してはprintf()

となるように変更した。また、

* UART1（つまりポート0）にはファイルディスクリプタ3を割り当て出力
* UART2（つまりポート1）にはファイルディスクリプタ4を割り当て出力

するように実装することで2ポートでの入出力を可能にした。

fcntl

前項の通り、ストリーム関数の入力元・出力先をファイルディスクリプタで判別させるにはCライブラリ関数fdopen(int fd, const char \*mode)で入出力ストリームをファイルディスクリプタへの割り当てが必要となる。例えば本実験の場合、ファイルディスクリプタ3に、読み込みを示す”r”と指定した場合、UART1からの読み込みとなる。このfdopenが呼び出す低レベル関数がfcntlである。これはこのファイルディスクリプタと指定する入出力モードが二つのみを引数にとり、前者の入出力属性と後者が一致するかを確認する。一致した場合「入出力可能」を返し、それ以外は単に0を返すように実装した。

inchrw.c

|  |
| --- |
| .global inbyte  .text  .even  inbyte:      movem.l %D1-%D3, -(%sp)  inbyte\_loop:      move.l #1,  %D0         | GETSTRINGを呼び出す      move.l 16(%sp), %D1     | ch   = 0 or 1      move.l #In\_BUF, %D2     | p    = #BUF      move.l #1, %D3          | size = 1      trap   #0          cmpi   #1, %d0       | GETSTRINGの返り値が1と一致するか確認      bne    inbyte\_loop   | 一致しなければ再受信      move.b In\_BUF, %d0   | 戻り値をd0に追加      movem.l (%sp)+, %D1-%D3          rts  .section .bss  In\_BUF:      .ds.b 1 |

inbyte

テーマ2までとの変更点は、ポート番号を引数に取ったことである。引数であるポート番号はスタックポインタから戻り先のPC（プログラムカウンタ）と%D1~%D3レジスタの値を考慮して参照している。

outchr.s

|  |
| --- |
| .global outbyte  .section .text  .even  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* outbyte  \*\* 引数：  \*\*       char型のデータ(スタックに格納済)  \*\* 出力：  \*\*       シリアルポート0に出力  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  outbyte:      movem.l %d0-%d3, -(%sp)  out\_byte\_loop:      move.l #SYSCALL\_NUM\_PUTSTRING, %d0 | PUTSTRINGを指定      move.l 20(%sp), %d1  | ch   = 0 or 1      move.l %sp, %d2      | data = char[1]      add.l  #27, %d2      move.l #1, %d3       | size = 1      trap   #0      cmpi.l #1, %d0       | PUTSTRINGの返り値が送信数と一致するか確認      bne    out\_byte\_loop | 一致しなければ再送信      movem.l (%sp)+, %d0-%d3      rts  .equ SYSCALL\_NUM\_PUTSTRING,    2 |

outbyte

テーマ2までとの変更点は、引数にchar型のデータに加えポート番号取ったことである。データ読み込み先の先頭アドレスとポート番号はスタックポインタが下から積まれるので、ポート番号、データの読み込み先の順番で、戻り先のPC（プログラムカウンタ）と%D0~%D3レジスタの値を考慮してデータの読み込み先として参照している。

test3.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include "mtk\_c.h"  FILE\* com0in;  FILE\* com0out;  FILE\* com1in;  FILE\* com1out;  void init\_file(){     com0in = fdopen(3, "r");     com0out = fdopen(3, "w");     com1in = fdopen(4, "r");     com1out = fdopen(4, "w");  }  void show(int width, int height, char question[height][width], char column[width], int w\_answer, int h\_answer, int a1){      int w,h;      // 描画部分      // 列名および空白描画      fprintf(com0out, "     |");      fprintf(com1out, "     |");      for (w = 1; w < width-1; w++){          fprintf(com0out, "%c", column[2\*w]);          fprintf(com1out ,"%c", column[2\*w]);          fprintf(com0out, "%c|", column[2\*w+1]);          fprintf(com1out, "%c|", column[2\*w+1]);      }      fprintf(com0out, "\n   ");      fprintf(com1out, "\n   ");      for (w = 0; w < width-1; w++){          fprintf(com0out, "  |");          fprintf(com1out, "  |");      }      fprintf(com0out, "\n");      fprintf(com1out, "\n");      // 間違い本体      for(h=0; h<height; h++){          // 列名を描画するかどうか          for (w = 0; w < width; w++){            if(h == h\_answer && w == w\_answer && a1 == 0 ){                fprintf(com0out, "\x1b[101m%2c\x1b[0m|", question[h][w]);                fprintf(com1out, "\x1b[101m%2c\x1b[0m|", question[h][w]);            continue;            }            if(h == h\_answer && w == w\_answer && a1 == 1 ){                fprintf(com0out, "\x1b[102m%2c\x1b[0m|", question[h][w]);                fprintf(com1out, "\x1b[102m%2c\x1b[0m|", question[h][w]);            continue;            }            else{                fprintf(com0out, "%2c|", question[h][w]);                fprintf(com1out, "%2c|", question[h][w]);            }          }          fprintf(com0out, "\n");          fprintf(com1out, "\n");    }  }  char \*check\_answer;  int answerer = 2;  void task0(){  while(1){      char s0;      // 入力受付      fscanf(com0in, "%s", &s0);      // トーク画面制限      P(0);      // 自分の入力値削除      fprintf(com0out, "\e[A\e[K");      // 自分をフォントを赤色にして自分の右側に表示      fprintf(com0out, "\x1b[31m\n%80s\n\x1b[0m", &s0);      // 相手のの入力値削除      fprintf(com1out, "\e[A\e[K");      // 自分をフォントを赤色にして相手の左側に表示      fprintf(com1out, "\x1b[31m\n%s\n\x1b[0m\n", &s0);      // ゲームスタートコマンドかどうか      if(strncmp(&s0, "start",5) == 0){          int width, height;          int width\_input, height\_input;          fprintf(com1out, "Player0 is now setting rule ...");          fprintf(com0out, "\nThe size of widths(letters) more than 2, less than 10 ");          scanf("%d", &width\_input);          width = width\_input;          fprintf(com1out, "\nThe size of width = %d\n", width);          width += 2;            fprintf(com0out, "The size of heights(letters) less than 26 ");          scanf("%d", &height\_input);          fprintf(com0out, "\n");          height = height\_input;          fprintf(com1out, "\nThe height of width = %d\n", width);          int h,w;          char column[2\*width-1];          char question[height][width];          char q = 'a';          char a = 'e';          int a1 = 2;            int w\_answer, h\_answer;          w\_answer = rand() % (width - 2) + 2;          h\_answer = rand() % height;            // 列(int)を作成          for(w=0; w < width-1; w++){              if(w<10){                  column[2\*w] = '0';                  column[2\*w+1] = '0' + w;              }              else{                  column[2\*w] = '0' + (w/10 % 10);                  column[2\*w+1] = '0' + (w % 10);              }          }          // 行(char)を作成（各列の先頭要素を操作）          for(h=0; h < height; h++){              question[h][0] = 65 + h;              // 各列の2列目は空白              question[h][1] = ' ';          }            // 間違い本体部分          for(h=0; h<height; h++){              for (w = 2; w < width; w++)              {                  if(h == h\_answer && w == w\_answer){                      question[h][w] = a;                  }                  else{                      question[h][w] = q;                  }              }          }          show(width, height, question, column, w\_answer,h\_answer,a1);          char \*check0 = &question[h\_answer][0];          char \*check1 = &column[2\*(w\_answer-1)];          char \*check2 = &column[2\*(w\_answer-1)+1];          char \*check;          sprintf(check, "%c%c%c\n", check0[0], check1[0], check2[0]);          check\_answer = check;          fprintf(com0out, "\nGive cell name faster than your opponent! ex)A01\n");          fprintf(com1out, "\nGive cell name faster than your opponent! ex)A01\n");          // トーク画面解放          V(0);          while(1){              // 入力受付              fscanf(com0in, "%s", &s0);              // トーク画面制限              P(0);              // 自分の入力値削除              fprintf(com0out, "\e[A\e[K");              // 自分をフォントを赤色にして自分の右側に表示              fprintf(com0out, "\x1b[31m\n%80s\n\x1b[0m", &s0);              // 相手のの入力値削除              fprintf(com1out, "\e[A\e[K");              // 自分をフォントを赤色にして相手の左側に表示              fprintf(com1out, "\x1b[31m\n%s\n\x1b[0m\n", &s0);              if(answerer == 1){                  fprintf(com0out, "\n\n \x1b[32mPlayer1\x1b[0m solved faster!\n\n");              fprintf(com1out, "\n\n \x1b[32mPlayer1\x1b[0m solved faster!\n\n");                  show(width, height, question, column, w\_answer,h\_answer,1);                  break;              }              if(strncmp(&s0, check\_answer, 3) == 0){                  answerer = 0;                  fprintf(com0out, "\n\n \x1b[31mPlayer0\x1b[0m solved faster!\n\n");              fprintf(com1out, "\n\n \x1b[31mPlayer0\x1b[0m solved faster!\n\n");                  show(width, height, question, column, w\_answer,h\_answer,0);                  break;              }              if(strncmp(&s0, "restart",7)==0){                  break;              }          }          // トーク画面解放          V(0);      }      else{          // トーク画面解放          V(0);      }  }  }  void task1(){  while(1){      char s1;      // 入力受付      fscanf(com1in, "%s", &s1);        // トーク画面制限      P(0);      // 自分の入力値削除      fprintf(com1out, "\e[A\e[K");      // 自分をフォントを緑色にして自分の右側に表示      fprintf(com1out, "\x1b[32m\n%80s\n\x1b[0m", &s1);      // 相手のの入力値削除      fprintf(com0out, "\e[A\e[K");      // 自分をフォントを緑色にして相手の左側に表示      fprintf(com0out, "\x1b[32m\n%s\n\x1b[0m\n", &s1);      if(!check\_answer){          strncmp(&s1, check\_answer, 3) == 0;          answerer = 1;      }      // トーク画面解放      V(0);  }  }    int main(void){    init\_kernel();    init\_file();    set\_task(task0);    set\_task(task1);    fprintf(com0out, "\nFind 'e'\n");    fprintf(com1out, "\nFind 'e'\n");    fprintf(com0out, "\nYou're Player0 You can set rule\n");    fprintf(com0out, "Give 'start' to play!\n");    fprintf(com1out, "\nYou're Player1 You can't set rule\n");    fprintf(com0out, "\nYou can also enjoy chatting\n\n");    fprintf(com1out, "\nYou can also enjoy chatting\n\n");    begin\_sch();    return 0;  } |

com0in, com0out, com1in, com1out

RS232ポートに対するファイルハンドルを保持するFILE構造体へのポインタの大域変数

個人課題

後述する

## プログラムにおいて、注意したこと

本テーマでは2ポートの入出力を実装するために、ファイルディスクリプタの割り当て、inbyte, outbyteでスタックからの引数を取り方に気を付けてプログラムを作成した。またtestファイルでは２ポートからの入出力、セマフォの利用、タイマ割り込みの３つをすべて利用するようなプログラムの作成を行ったが、エスケープシーケンスを用いることで個人課題部分を工夫するだけでなく、デバッグ作業を効率的行うことができた。

## プログラムに発生した問題とその原因、解決方法

本テーマでは、ファイルディスクリプタを用いた2ポートへの割り当てがset\_taskを用いて登録したタスクに関しては行われない問題が発生した。csys68k.h等の関連するソースコードを何度も見直したが、原因は分からなかった。そこまではmon.oをSW実験1で作成したエコーバックプログラムを改良したmon.sから作成していたものから、推奨されていた配布されたmon.0との差し替えをすると、解決した。原因がmon.sにあることは分からなかったが根本的な原因追求はできなかった。

## 個人課題について

test3.cではマルチタスク環境を用いて、「仲間外れ探し（”a”という文字の中から文字”e”を探す）+チャット」を作成した。ここで**mtk\_asm.sでタイマ割り込み周期を1[0.1ms]**に設定しておく。このとき、登録した二つのタスクではそれぞれポート0,1から入力を受け付け、入力がなければ一方のタスクに遷移し、入力があれば資源（０）を占有し、入力値の削除・出力を行った後で資源（０）の解放を行っている。そうすることで疑似的に両ポートから同時入力を可能にしている。このとき、自分側の入力は右側に相手の入力は左側になるように出力する、LINEライクな仕様になっている。

また、ポート0側でstartと入力すると、力があれば資源（０）を占有し、入力値の削除・出力を行った後で資源の解放を行うのではなく、仲間外れ探しが開始する。ポート0側では仲間外れ探しをするときのマップの設定を横・縦の文字数で指定する。なお、このときにポート1からの入力値は資源がポート0（のタスク）に占有されているため起こらない。ポート0での設定が終わると資源（０）を開放してゲームが始まる。このとき、再度両ポートからの入力をゲーム開始前と同様に受け付けるが、今度は毎度出力の旅に間違い”e”のある座標かどうか判定し、正しい座標が入ると答えが表示され、勝者が表示される仕様になっている。（厳密にいうと、どちらかが正しい入力を行い、ポートの入力を受け付けているタスク1が出力を行う際に答えが表示される）そして、またチャット機能へと戻り、何度でも仲間外れ探しをすることができる。

# 考察

本レポートで触れたテーマ2,3ではマルチタスク環境の作成とその応用を行った。入出力ストリーム関数のように便利な関数のあるC言語と、スタックやキューをレジスタ・メモリ単位で意識したプログラム作成が可能なアセンブリ言語それぞれの特性を生かして開発をすることができたと考えている。ただ、二つの言語を横断して、時に共同開発として分担して作業することで違う言語の関数を参照するときに問題が起こりやすいというのも事実であった。しかしテーマ3のファイルディスクリプタを用いて2ポート入出力のように流れを理解していないと実装できないものがあったからこそ理解が曖昧であったところが明確になり、学びにつながったと考えている。