SW 実験2テーマ1レポート

C過程 S-15組 1TE20137W 2022/12/19

柳 鷹

内容

[テーマ1:C言語ライブラリ関数の移植 1](#_Toc122443128)

[テーマ1について 1](#_Toc122443129)

[テーマ1の実験全体に占める位置づけ 1](#_Toc122443130)

[プログラムの説明 2](#_Toc122443131)

[プログラムのリスト 2](#_Toc122443132)

[プログラムにおいて、注意したこと 4](#_Toc122443133)

[プログラムに発生した問題とその原因・解決 5](#_Toc122443134)

[考察 5](#_Toc122443135)

[（テーマ2: マルチタスクカーネルの制作　の準備） 5](#_Toc122443136)

[readyキューを有する時分割マルチタスク環境下の思考実験 5](#_Toc122443137)

# テーマ1:C言語ライブラリ関数の移植

## テーマ1について

本実験のテーマ1では、68000互換CPU搭載のターゲットボード上でC言語処理系（標準入出力ライブラリ）をクロスコンパイラとしてカスタマイズするために入出力関数が動作することが目的である。具体的には以下のような流れでシリアルポートを経由してディスプレイに文字列が表示させることである。

|  |  |
| --- | --- |
| 出力 | 入力 |
| main()  test1.c | |
| printf() | scanf() |
| libc.a | |
| write() | read() |
| csys68k.c | |
| outbyte:  outchr.s | inbyte:  inchrw.s |
| trap#0 | |
| PUTSTRING | GETSTRING |
| mon.s | |
| シリアルポート  Hardware | |

m68k-elf-gccのlibc.aに収められているライブラリ関数printf(), scanf()をtest1.cのmain()内で呼び出す。これらの入出力関数はcsys68k.c内のwrite(), read()関数を用いて実装されている。このとき、シリアルポートに対して、write(), read()関数を使うためにソフトウェア実験Ⅰで作成したエコーバックプログラムののシステムコールGETSTRING, PUTSTRINGを呼び出すようにする。

## テーマ1の実験全体に占める位置づけ

本実験のテーマ1では、後のテーマ2,3に向けて以下を確認することができる

* プログラムの分割コンパイル・アセンブル
* アセンブリ言語プログラムとC言語プログラムとのリンク
* C言語プログラムの起動

## プログラムの説明

テーマ1では上述のようにpritnf(), scanf()を用いるために、まずソフトウェア実験Ⅰで作成したエコーバックプログラムをmon.sとしてハードウェアを初期化、システムコールのベクタの登録をする。その後、outchr.s, inchrw.sにアセンブラ関数（サブルーチン）として%D0レジスタにシステムコール番号4を入れ1文字表示するoutbyte:と 、%D0レジスタにシステムコール番号3を入れ1文字入力を受け付けるbinbyte:を実装した。

## プログラムのリスト

テーマ1で実装・変更したプログラムを以下に示す。

mon.s （初期化部分だけ）

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 初期化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .section .text  .even  **.extern start           | crt0.s内のstartをextern**  **.global monitor\_begin   | 大域変数（関数）の宣言**  monitor\_begin:      \* スーパーバイザ & 各種設定を行っている最中の割込禁止      move.w #0x2000,%SR      lea.l SYS\_STK\_TOP, %SP | Set SSP      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      \*\* 割り込みコントローラの初期化      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      move.b #0x40, IVR      | ユーザ割り込みベクタ番号を                             | 0x40+level に設定.      move.l #0x00ff3ff9,IMR | 全割り込みマスクMUART=>0,MTMR1=>1      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      \*\* 送受信 (UART1) 関係の初期化 ( 割り込みレベルは 4 に固定されている )      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      move.l #UART1\_interrupt, 0x110  | 受信割り込みベクタをセット      move.w #0x0000, USTCNT1 | リセット      move.w #0xe108, USTCNT1 | 送受信可能 , パリティなし , 1 stop, 8 bit,            | 受信割り込み許可, 送信割り込み禁止      move.w #0x0038, UBAUD1  | baud rate = 230400 bps      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      \*\* タイマ関係の初期化 ( 割り込みレベルは 6 に固定されている )      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      move.w #0x0004, TCTL1   | restart, 割り込み不可 ,            | システムクロックの 1/16 を単位として計時,            | タイマ使用停止      move.l #COMPARE\_INTERPUT, 0x118 /\* level 6 \*/      move.l #SYS\_CALL, 0x080 /\*SYS\_CALLの割り込みベクタ設定\*/      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      \*\* キューの初期化      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      lea.l  top0, %a2      lea.l  top1, %a3      move.l %a2, out0      move.l %a3, out1      move.l %a2, in0      move.l %a3, in1      move.l #0, s0      move.l #0, s1    **jmp start** |

inchrw.s

|  |
| --- |
| .global inbyte  .text  .even  inbyte:      movem.l %D1-%D3, -(%sp)  inbyte\_loop:      move.l #1, %D0          |GETSTRINGを呼び出す      move.l #0,   %D1        | ch   = 0      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 |

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 #0, %d1       | ch   = 0      move.l %sp, %d2   | data = char[1]      add.l  #23, %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  end\_outbyte:      movem.l (%sp)+, %d0-%d3      rts  .equ SYSCALL\_NUM\_PUTSTRING,    2 |

## プログラムにおいて、注意したこと

mon.sの初期化の実装、GETSTRINGで1文字を呼び出すinbyte:の実装の理解に問題はなかった。

本テーマのハイライトはoutbyte: でPUTSTRINGを呼び出す際の引数の取り方であったと考えている。PUTSTRINGをtrap命令で呼び出すときの引数(括弧内は固有値)は4つで、システムコール番号(2)、チャンネル番号(0)、データの読み込み先の先頭アドレス(変数)、送信するデータ数(1)をそれぞれ%D0~%D3レジスタに入れる。ここで気を付けるのは%D2レジスタに入れるデータの読み込み先の先頭アドレスである。このoutbyte:を呼び出すとき、スタックポインタには戻り先のPC（プログラムカウンタ）と%D0~%D3レジスタの値がスタックポインタに積まれているので、データの読み込み先の先頭アドレスを取り出すには下に積んであるロングサイズのレジスタ4個のPC分を考慮して、(スタックポインタの示す値+23)のアドレスを参照した。

## プログラムに発生した問題とその原因・解決

本プログラムを作成中、outbyteのスタックポインタに積まれた退避したレジスタの値とPCに気を付けていたが、どうしてもターミナル上で表示できない問題が発生した。そこでmon.s（ソフトウェア実験Ⅰで作成したエコーバックプログラム）のPUTSTRINGのレジスタ退避部に問題があることが分かり、修正すると無事に動いた。

（確認はしていないが、mon.sの差し替えでも解決したかもしれない。）

## 考察

本テーマでは、今までアセンブリ言語で開発していた68000互換CPU搭載のターゲットボード上でC言語処理系も動作確認することができた。このアセンブリ言語プログラムとC言語プログラムとのリンクができたことで、レジスタやメモリを意識したプログラムが書きやすいアセンブリ言語と、forやwhileなどの条件分岐・繰り返しや入出力関数を書きやすいC言語それぞれのメリットを生かした分割コンパイル・アセンブルの開発ができるようになったと感じている。しかしながら、outbyte:での引数の渡し方のように別々のプログラミング言語で開発を並行して進めるうえでのデメリットも発生するため、注意する必要があると感じた。

# 【準備】テーマ2: マルチタスクカーネルの制作

## readyキューを有する時分割マルチタスク環境下の思考実験