# システムプログラミング 1
# レポート

氏名: 今田　将也 (IMADA, Masaya)
学生番号: 09430509

出題日: 2019 年 10 月 07 日
提出日: 2019 年 11 月 20 日
締切日: 2019 年 11 月 25 日

## 1　概要

本演習では，PIM という MIPS CPU シミュレータのハードウェア上に C 言語とアセンブリ言語を使用して文字の表示と入力のためのシステムコールライブラリを作成する． さらに，そのライブラリを使用して printf 及び gets 相当を C 言語で作成する． 最後に，それらを利用した応用プログラムを動作させる．

なお、与えられた課題内容を以下に述べる．

### 1.1　課題内容

以下の課題についてレポートをする． プログラムは，MIPS アセンブリ言語及び C 言語で記述し，SPIM を用いて動作を確認している．

**2-1** SPIM が提供するシステムコールを C 言語から実行できるようにしたい． 教科書 A.6 節「手続き呼出し規約」に従って，各種手続きをアセンブラで記述せよ． ファイル名は， syscalls.s とすること．また，記述した syscalls.s の関数を C 言語から呼び出すことで， ハノイの塔 (hanoi.c とする) を完成させよ．

```
 1: void hanoi(int n, int start, int finish, int extra)
 2: {
 3:   if (n != 0){
 4:     hanoi(n - 1, start, extra, finish);
 5:     print_string("Move disk ");
 6:     print_int(n);
 7:     print_string(" from peg ");
 8:     print_int(start);
 9:     print_string(" to peg ");
10:     print_int(finish);
11:     print_string(".\n");
12:     hanoi(n - 1, extra, finish, start);
```

```
13:   }
14: }
15: main()
16: {
17:   int n;
18:   print_string("Enter number of disks> ");
19:   n = read_int();
20:   hanoi(n, 1, 2, 3);
21: }
```

spim-gcc によって hanoi.s ができたら， hanoi.s, syscalls.s の順に SPIM 上でロードして実行.

実行例は以下の通り:

```
Enter number of disks> 3
Move disk 1 from peg 1 to peg 2.
Move disk 2 from peg 1 to peg 3.
Move disk 1 from peg 2 to peg 3.
Move disk 3 from peg 1 to peg 2.
Move disk 1 from peg 3 to peg 1.
Move disk 2 from peg 3 to peg 2.
Move disk 1 from peg 1 to peg 2.
```

**2-2** hanoi.s を例に spim-gcc の引数保存に関するスタックの利用方法について，説明せよ． そのことは，規約上許されるスタックフレームの最小値 24 とどう関係しているか． このスタックフレームの最小値規約を守らないとどのような問題が生じるかについて解説せよ.

hanoi.c のソースコード

**2-3** 以下のプログラム report2-1.c をコンパイルした結果をもとに， auto 変数と static 変数の違い，ポインタと配列の違いについてレポートせよ.

```
 1: int primes_stat[10];
 2:
 3: char * string_ptr   = "ABCDEFG";
 4: char   string_ary[] = "ABCDEFG";
 5:
 6: void print_var(char *name, int val)
 7: {
 8:   print_string(name);
 9:   print_string(" = ");
10:   print_int(val);
11:   print_string("\n");
12: }
```

```
13:
14: main()
15: {
16:   int primes_auto[10];
17:
18:   primes_stat[0] = 2;
19:   primes_auto[0] = 3;
20:
21:   print_var("primes_stat[0]", primes_stat[0]);
22:   print_var("primes_auto[0]", primes_auto[0]);
23: }
```

**2-4** printf など，一部の関数は，任意の数の引数を取ることができる． これらの関数を可変引数関数と呼ぶ． MIPS の C コンパイラにおいて可変引数関数の実現方法について考察し，解説せよ．

**2-5** printf のサブセットを実装し， SPIM 上でその動作を確認する応用プログラム (自由なデモプログラム) を作成せよ． フルセットにどれだけ近いか，あるいは，よく使う重要な仕様だけをうまく切り出して， 実用的なサブセットを実装しているかについて評価する． ただし，浮動小数は対応しなくてもよい (SPIM 自体がうまく対応していない)． 加えて，この printf を利用した応用プログラムの出来も評価の対象とする．

### 1.2 xspim の実行方法

```
$ xspim -mapped_io&
```

でコンソール上で実行後，必要なアセンブリファイルを load し，run することで実行した．

### 1.3 c ソースコードからアセンブリファイルへの変換方法

```
$ spim-gcc file.c
```

でコンソール上で実行後，file.c に対応する file.s というアセンブリファイルが作られる．

## 2 課題レポート

### 2.1 2-1

以下に作成したプログラムを記載する．

#### 2.1.1 作成したプログラム

syscalls.s

```
 1         .text
 2         .align  2
 3
 4 _print_int:
 5         subu  $sp, $sp, 24
 6         sw    $ra, 20($sp)
 7
 8         li    $v0, 1  # 1: print_int
 9         syscall
10
11         lw    $ra, 20($sp)
12         addu  $sp, $sp, 24
13         j     $ra
14
15 _print_string:
16         subu  $sp, $sp, 24
17         sw    $ra, 20($sp)
18
19         li    $v0, 4  # 4: print_string
20         syscall
21
22         lw    $ra, 20($sp)
23         addu  $sp, $sp, 24
24         j     $ra
25
26 _read_int:
27         subu  $sp, $sp, 24
28         sw    $ra, 20($sp)
29
30         li    $v0, 5  # 5: read_int
31         syscall
32
33         lw    $ra, 20($sp)
34         addu  $sp, $sp, 24
35         j     $ra
36
37 _read_string:
38         subu  $sp, $sp, 24
39         sw    $ra, 20($sp)
40
41         li    $v0, 8  # 8: read_string
42         syscall
43
44         lw    $ra, 20($sp)
```

```
45          addu  $sp, $sp, 24
46          j     $ra
47
48  _exit:
49          subu  $sp, $sp, 24
50          sw    $ra, 20($sp)
51
52          li    $v0, 10  # 10: exit
53          syscall
54
55          lw    $ra, 20($sp)
56          addu  $sp, $sp, 24
57          j     $ra
58
59  _print_char:
60          subu  $sp, $sp, 24
61          sw    $ra, 20($sp)
62
63          li    $v0, 11  # 11: print_char
64          syscall
65
66          lw    $ra, 20($sp)
67          addu  $sp, $sp, 24
68          j     $ra
69
70  _read_char:
71          subu  $sp, $sp, 24
72          sw    $ra, 20($sp)
73
74          li    $v0, 12  # 12: _read_char
75          syscall
76
77          lw    $ra, 20($sp)
78          addu  $sp, $sp, 24
79          j     $ra
```

### 2.1.2 考察

作成は，手続き呼出し規約に基づいて，各ルーチンごとにスタックポインタをルーチンの開始時に確保し，終了時に破棄して呼び出された関数に戻る設計にしている．syscall でカーネルに所望することを$v0 レジスタへ格納し，syscall を呼び出している．

print_int に対応する関数は，4 行目から 13 行目に記載している．print_string に対応する関数は，15 行目から 24 行目に記載している．read_int に対応する関数は，26 行目から 35 行目に記載している．read_string に対応する関数は，37 行目から 46 行目に記載している．exit に

対応する関数は，48 行目から 57 行目に記載している．print_char に対応する関数は，59 行目から 68 行目に記載している．read_char に対応する関数は，70 行目から 79 行目に記載している．

なお、今回の hanoi.c には用いないが，文字列をユーザから受け付ける read_string，数値をユーザから受け付ける read_int と文字を表示する print_char と文字をユーザから受け付ける read_char，そして，プログラムを終了する exit を作成した．

作成したプログラム中のラベルの先頭にアンダーバーをつけているがこれは，本演習で用いた gcc のルールでコンパイラに依存するものであるが，アセンブリ中で_function_name と記述しておくと，C 言語から function_name で呼び出すことができるからである．

## 2.2 2-2

### 2.2.1 考察

考察のために，以下に図や表をつくってみて比較して考察をしてみる。自分で書いた C のプログラムとかで考察するのもよい。

## 2.3 2-3

### 2.3.1 作成したプログラム

### 2.3.2 考察

図や表をつくってみて比較して考察をしてみる。自分で書いた C のプログラムとかで考察するのもよい。【auto と static の違い】１９行目確保したスタック上に存在している。スタックの絵を書いてみよう。static 関数はプログラムの終わりまで存在するが、auto 変数は該当の関数呼び出しが終わったら領域が開放されて値が使えなくなる。

同じ名前の変数が用意された異なるファイルを 2 つ読み込むとどうなるのかについて調べてみる。

【ポインタと配列の違い】

## 2.4 2-4

### 2.4.1 作成したプログラム

### 2.4.2 考察

## 2.5 2-5

### 2.5.1 作成したプログラム

### 2.5.2 考察

# 3 感想