システムプログラミング 2
期末レポート

氏名: 山田　敬汰 (Yamada，Keita)
学生番号: 09430559

出題日: 2019 年 12 月 9 日
提出日: 2019 年??月??日
締切日: 2020 年 1 月 27 日

1　概要

2　課題 2-1

2.1　課題内容

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．

2.2　C 言語で記述したプログラム例

```
 1 #include <stdio.h>
 2
 3 void hanoi(int n, int start, int finish, int extra)
 4 {
 5     if (n != 0)
 6     {
 7         hanoi(n - 1, start, extra, finish);
 8         print_string("Move disk ");
 9         print_int(n);
10         print_string(" from peg ");
11         print_int(start);
12         print_string(" to peg ");
13         print_int(finish);
14         print_string(".\n");
15         hanoi(n - 1, extra, finish, start);
16     }
```

```
17 }
18 main()
19 {
20     int n;
21     print_string("Enter number of disks> ");
22     n = read_int();
23     hanoi(n, 1, 2, 3);
24 }
```

2.3 作成したプログラム

```
 1 .text
 2 .align 2
 3
 4 _print_int:
 5  subu    $sp,$sp,24
 6  sw      $ra,20($sp)
 7
 8  li      $v0, 1
 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
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
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
42  syscall
43
44  lw      $ra,20($sp)
45  addu    $sp,$sp,24
46  j       $ra
47
48 _exit:
49  li      $v0, 10
50  syscall
51
```

2.4 実行結果

```
Enter number of disks>  4
Move disk 1 from peg 1 to peg 3.
Move disk 2 from peg 1 to peg 2.
Move disk 1 from peg 3 to peg 2.
Move disk 3 from peg 1 to peg 3.
Move disk 1 from peg 2 to peg 1.
Move disk 2 from peg 2 to peg 3.
Move disk 1 from peg 1 to peg 3.
Move disk 4 from peg 1 to peg 2.
Move disk 1 from peg 3 to peg 2.
Move disk 2 from peg 3 to peg 1.
Move disk 1 from peg 2 to peg 1.
Move disk 3 from peg 3 to peg 2.
Move disk 1 from peg 1 to peg 3.
Move disk 2 from peg 1 to peg 2.
Move disk 1 from peg 3 to peg 2.
```

2.5 プログラムの解説

ここでは_print_int 部分での手続きを例に解説する.

1. スタックの領域を 24 バイト確保し，戻りアドレスを格納しておく．戻りアドレスを退避させている理由としては，syscall 内の手続きが OS に一任され，アセンブリのプログラムか

ら分からないようになっているからである.（つまり，OS が勝手に $\$ra$ レジスタの値を壊している可能性を考慮している.）

2. $\$v0$ レジスタに適切な番号（`_print_int` の場合は 1）を格納し，`syscall` 命令を発行する．

3. スタックに格納しておいた戻りアドレスを $\$ra$ レジスタに再び格納し，呼び出し元に帰る．

その他の手続きも $\$v0$ レジスタの値を変更することで，同様の手順で実行することができる．（OS によって抽象化されている.）

また，`exit` 手続きのみ，スタックに戻りアドレスを格納していない．その理由としては `syscall` の発行によってプロセスが終了するので，値を退避させたところで復帰させる方法がないからである．

2.6 考察

今回のプログラムでは `syscall` を呼び出す部分のみをアセンブリ言語で記述している．これは，C 言語の中から直接 `syscall` 命令を発行する方法が存在しないからである．

また，手続き呼び出し規約によって「引数はどのレジスタに入っていて，戻り値はこのレジスタに入っている」というのが決められているので，この規約を守っている限りは C 言語とアセンブリ言語との連携を円滑に行うことができる．

3 課題 2-2

3.1 課題内容

4 課題 2-3

4.1 課題内容

5 課題 2-4

5.1 課題内容

6 課題 2-5

6.1 課題内容

7 感想