实验一：操作系统初步

安全 1601 沈香港 16281077

实验环境：ubuntu16.04

一、（系统调用实验）了解系统调用不同的封装形式

1.程序代码（注：将两种调用方式在同一个 C 程序中实现）

```
shen@shen-Lenovo-Rescuer-15ISK: ~/操作系统
#include<stdio.h>
#include<unistd.h>
int main(){
    pid_t pid,pidasm;
    pid=getpid();
    printf("use getpid():current process pid is %d\n",pid);

    asm volatile(
            "mov $0,%%ebx\n\t"
            "mov $0x14,%%eax\n\t"
            "int $0x80\n\t"
            "mov %%eax,%0\n\t"
            :"=m"(pidasm)
            );
    printf("use asm:current process pid-asm is %d\n",pidasm);
}

~
~
```

程序运行结果

```
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ vim test1.c
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./test1
use getpid():current process pid is 10018
use asm:current process pid-asm is 10018
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$
```

中断向量号是 0x80

系统调用号是 0x14

2.习题 1.13

（1）C 语言程序实现代码

```
#include "stdio.h"
#include "string.h"

int main()
{
    char* msg = "Hello World";
    printf("%s", msg);
        return 0;
}
~
```

（2）汇编实现代码

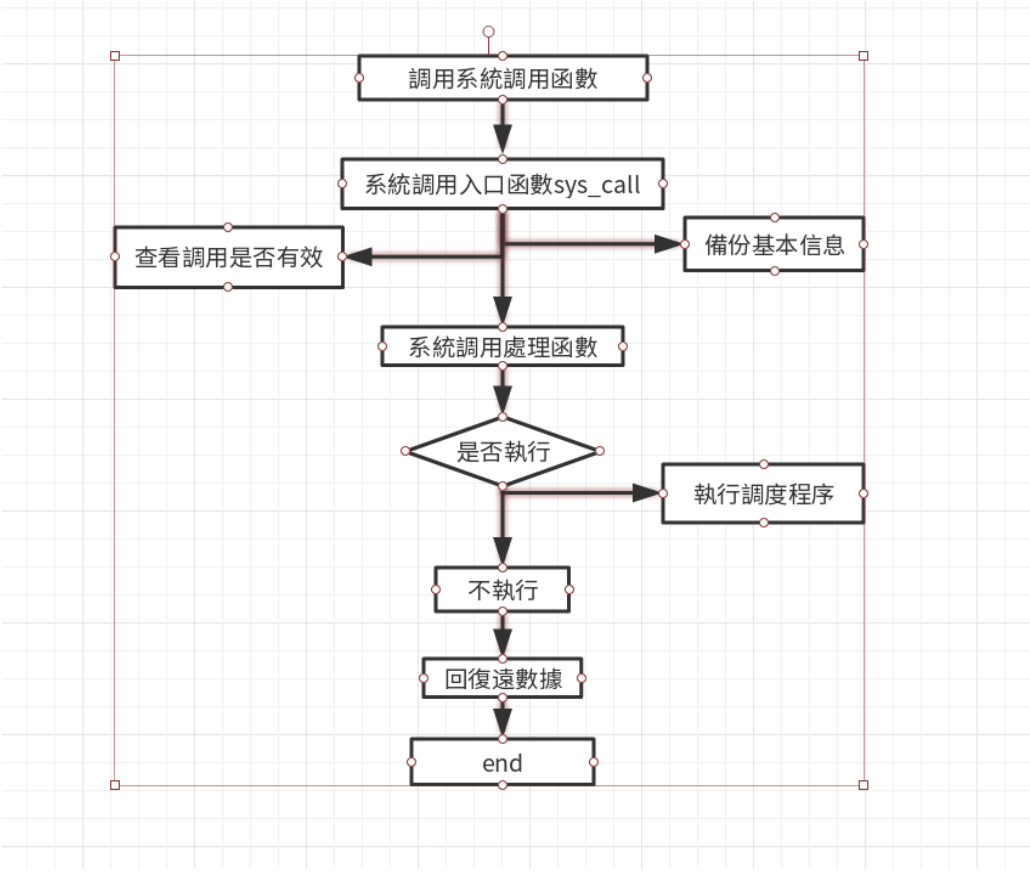
shen@shen-Lenovo-Rescuer-15ISK: ~/操作系统

```
int main()
{
    char* msg = "Hello World";
        int len = 11;
        int result = 0;

    __asm__ __volatile__("movl %2, %%edx;\n\r" /*传入参数：要显示的字符串长度*/
             "movl %1, %%ecx;\n\r" /*传入参赛：文件描述符（stdout）*/
                          "movl $1, %%ebx;\n\r" /*传入参数：要显示的字符串*/
                          "movl $4, %%eax;\n\r" /*系统调用号：4 sys_write*/
                          "int  $0x80" /*触发系统调用中断*/
            :"=m"(result) /*输出部分：本例并未使用*/
            :"m"(msg),"r"(len)  /*输入部分：绑定字符串和字符串长度变量*/
            :"%eax");

        return 0;
}
~
~
~
```

（3）编译后运行的结果

```
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./c-hello
Hello Worldshen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./api-hello
Hello Worldshen@shen-Lenovo-Rescuer-15ISK:~/操作系统$
```

3.系统调用程序流程图



二、（并发实验）根据以下代码完成下面的实验。

```
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./cpu
usage: cpu <string>
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$
```

```
shen@shen-Lenovo-Rescuer-15ISK: ~/操作系统
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./cpu A &./cpu B &./cpu C &./cpu D &
[1] 11006
[2] 11007
[3] 11008
[4] 11009
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ A
B
C
D
A
B
C
D
A
B
C
D
A
B
C
D
```

1、执行命令后提示缺少参数,程序通过 C 语言”printf”函数实现调用 cpu 的功能

2、cpu 运行了一次,每次调用输出顺序不变都是 A,B,C,D,并且同时输出,证明多个程序运行是在同一时间间隔内,但是他们输出的顺序没有改变 cpu 在这个时间间隔内运行程序还是有顺序的,这个现象符合并发的概念。Linux 为分时操作系统,所以程序执行有时间先后。

三、（内存分配实验）根据以下代码完成实验

程序运行结果

```
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./cpu
usage: cpu <string>
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./mem & ./mem &
[1] 11254
[2] 11255
(11254) address pointed to by p: 0x1995010
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ (11255) address pointed to by p: 0x25
39010
(11254) p: 1
(11255) p: 1
(11254) p: 2
(11255) p: 2
(11254) p: 3
(11255) p: 3
(11254) p: 4
(11255) p: 4
```

1、程序观察程序运行时变量被分配的内存地址。
2、两次运行程序分配的内存地址不相同(因为变量 p 分配的地址不同)
,不共享同一块物
理内存区域,两个程序分配的 pid 不同所以程序不在同一个内存区域上。

四、（共享的问题）根据以下代码完成实验

运行结果

```
shen@shen-Lenovo-Rescuer-15ISK: ~/操作系统
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./thread
usage: threads <value>
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./thread 1000
Initial value : 0
Final value : 1636
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$ ./thread 100000
Initial value : 0
Final value : 141166
shen@shen-Lenovo-Rescuer-15ISK:~/操作系统$
```

1、程序可以在执行过程中创建两条线程进行运行,并且调用共享的变量以观察多线程调用
的结果
2、执行结果程序运行中开了两个线程所以 counter 的输出应该改为输入数的二倍加入输入
为 n,则输出应该为 2n,带式实际结果出现为部分输出小于 2n 当然输出结果不可能大于
2n。结果出现小于 2n 的证明多线程调用初夏你了问题,发生了冲突。
3、线程共享变量:loops;counter;会导致意想不到的问题,事实证明多线程调用可能存
在数据丢失或者数据出错的问题。