实验一：操作系统初步

安全 1601　15282028　赵旭宏

一、（系统调用实验）了解系统调用不同的封装形式。

1 请问 getpid 的系统调用号是多少？linux 系统调用的中断向量号是多少？

系统调用号 39；

中断向量号为 0x80；

```
pixiao@ubuntu:~/OS$ vim getpid.c
pixiao@ubuntu:~/OS$ gcc getpid.c
pixiao@ubuntu:~/OS$ ./a.out
68566
pixiao@ubuntu:~/OS$ vim Getpid.c
pixiao@ubuntu:~/OS$ ls
a.out  getpid.c  Getpid.c
pixiao@ubuntu:~/OS$ gcc Getpid.c
pixiao@ubuntu:~/OS$ ./a.out
68623
```

2、上机作业 1.13

Linux 调用 c 函数

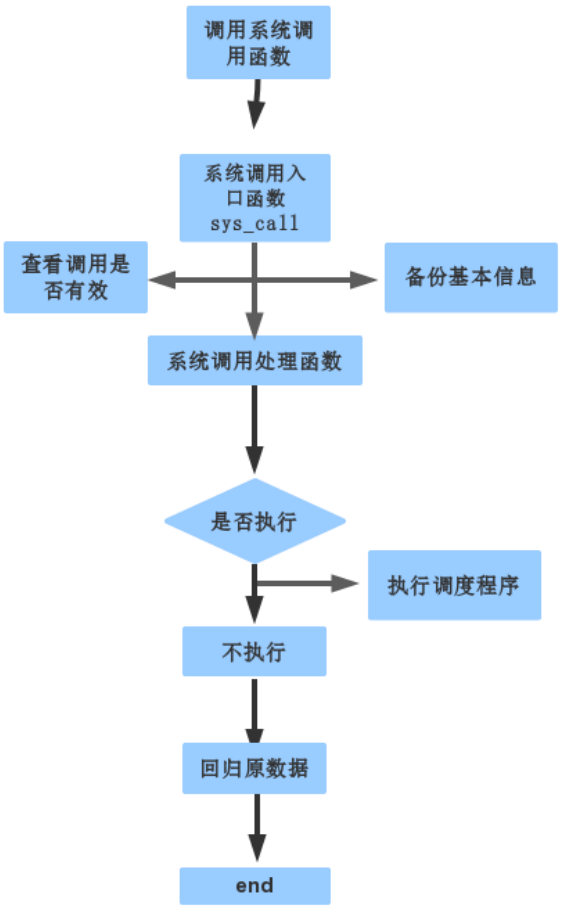
```
Hello Worldpixiao@ubuntu:~/OS$ gcc hello.c
pixiao@ubuntu:~/OS$ ./a.out
```

Linux 汇编方式

```
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;
}
```

```
$ nasm -f elf64 hello.asm
$ gcc -o hello hello.o
$ ./hello

$
```

3. 阅读 pintos 操作系统源代码，画出系统调用实现的流程图。



二、（并发实验）根据以下代码完成下面的实验。

1.编译运行该程序（cpu.c），观察输出结果，说明程序功能。

(编译命令： gcc -o cpu cpu.c －Wall)（执行命令：./cpu）

```c
#include<unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>

 int main(int argc, char *argv[])
{
if (argc != 2) {
fprintf(stderr, "usage: cpu <string>\n");
exit(1);
}
char *str = argv[1];
while (1) {
sleep(1);
printf("%s\n", str);
}
return 0;
}
```

判断参数个数，若不为 2，则输出 usage:cpu<string>;若为 2，则重复输出第一个字符。

```
pixiao@ubuntu:~/OS$ gcc -o cpu cpu.c
pixiao@ubuntu:~/OS$ ./cpu
usage: cpu <string>
pixiao@ubuntu:~/OS$
```

2.再次按下面的运行并观察结果：执行命令：./cpu A &；./cpu B &；./cpu C &；./cpu D &程序 cpu 运行了几次？他们运行的顺序有何特点和规律？请结合操作系统的特征进行解释。

运行 4 次，运行顺序交替，在一个时间段并发执行，在一个时间点一个执行

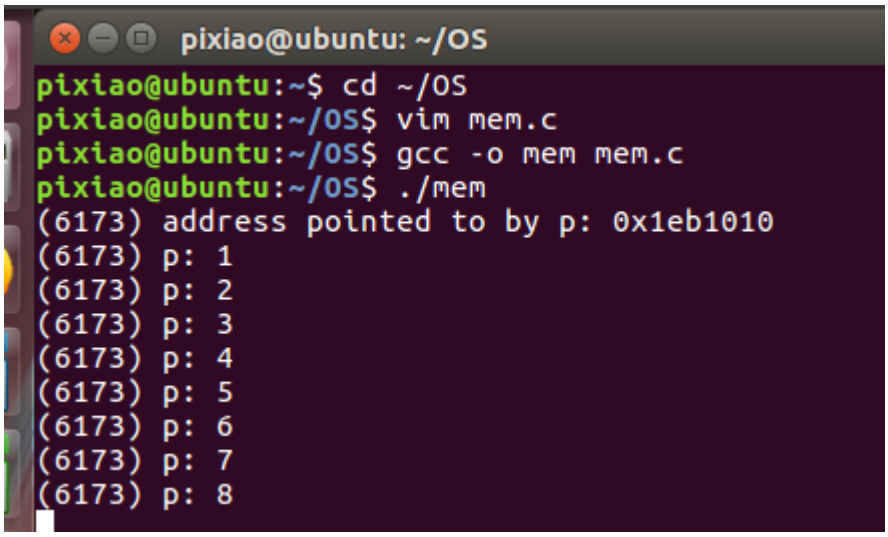
```
pixiao@ubuntu:~/OS$ ./cpu A& ./cpu B& ./cpu C& ./cpu D&
[1] 5979
[2] 5980
[3] 5981
[4] 5982
pixiao@ubuntu:~/OS$ B
A
D
C
B
C
A
D
B
A
D
C
B
A
C
D
B
```

三、（内存分配实验）根据以下代码完成实验。

1.阅读并编译运行该程序(mem.c)，观察输出结果，说明程序功能。(命令： gcc -o mem mem.c －Wall)

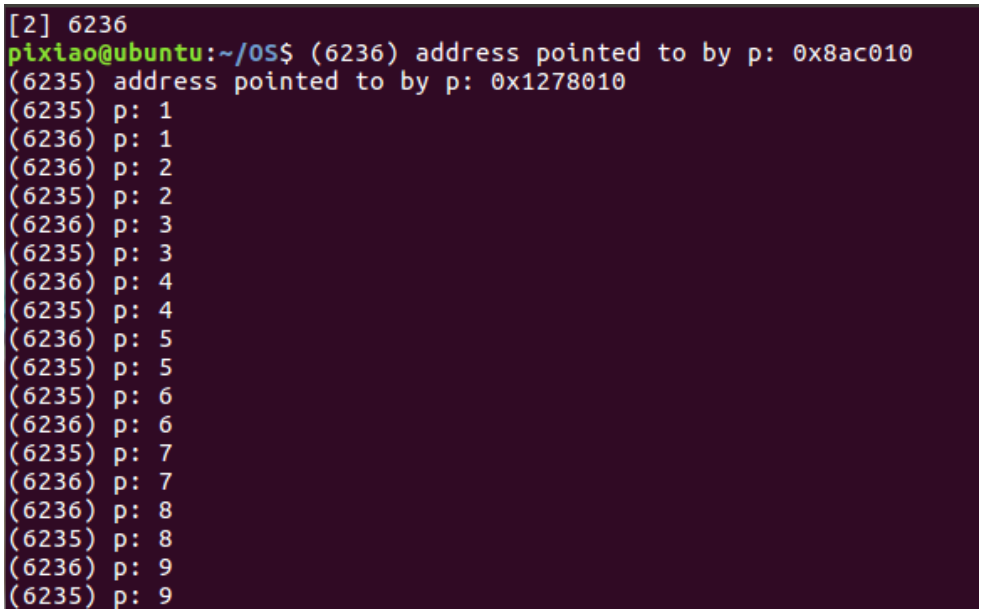
程序功能：申请一个 int 类型大小的内存空间，打印 pid 和内存地址，从 1 开始打印变量，每次使变量加一

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

int main(int argc, char *argv[])
 {
 int *p = malloc(sizeof(int)); // a1
 assert(p != NULL);
 printf("(%d) address pointed to by p: %p\n",
 getpid(), p); // a2
 *p = 0; // a3
 while (1) {
 sleep(1);
 *p = *p + 1;
 printf("(%d) p: %d\n", getpid(), *p); // a4
 }
 return 0;}
```



2.再次按下面的命令运行并观察结果。两个分别运行的程序分配的内存地址是否相同？是否共享同一块物理内存区域？为什么？命令：./mem &; ./mem &

两者都不相同，两次命令分别装载到两个地址



四、（共享的问题）根据以下代码完成实验。

4.1 阅读并编译运行该程序，观察输出结果，说明程序功能。（编译命令：gcc -o thread thread.c -Wall －pthread）（执行命令 1：./thread 1000）

计算程序运行次数

```
pixiao@ubuntu:~/OS$ gcc thread.c -o test -lpthread
pixiao@ubuntu:~/OS$ ./test
usage: threads <value>
pixiao@ubuntu:~/OS$ ./test 1000
Initial value : 0
Final value : 2000
pixiao@ubuntu:~/OS$ ./test 10000
Initial value : 0
Final value : 20000
pixiao@ubuntu:~/OS$
```

4.2 尝试其他输入参数并执行，并总结执行结果的有何规律？你能尝试解释它吗？（例如执行命令 2：./thread 100000）（或者其他参数。）

因为有两个进程所以输出的值为参数的二倍


```
pixiao@ubuntu:~/OS$ gcc thread.c -o test -lpthread
pixiao@ubuntu:~/OS$ ./test
usage: threads <value>
pixiao@ubuntu:~/OS$ ./test 1000
Initial value : 0
Final value : 2000
pixiao@ubuntu:~/OS$ ./test 10000
Initial value : 0
Final value : 20000
pixiao@ubuntu:~/OS$
```


4.3 提示：哪些变量是各个线程共享的，线程并发执行时访问共享变量会不会导致意想不到的问题。

Couter 变量是共享的，会造成问题。


```
pixiao@ubuntu:~/OS$ gcc thread.c -o test -lpthread
pixiao@ubuntu:~/OS$ ./test
usage: threads <value>
pixiao@ubuntu:~/OS$ ./test 1000
Initial value : 0
Final value : 2000
pixiao@ubuntu:~/OS$ ./test 10000
Initial value : 0
Final value : 20000
pixiao@ubuntu:~/OS$
```