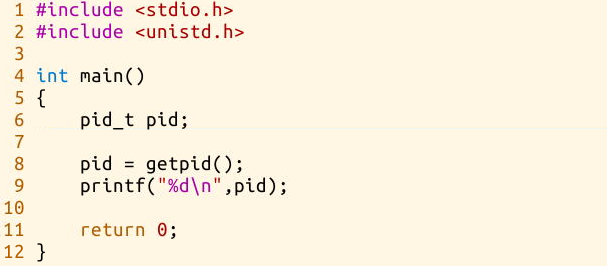
**实验一: 操作系统初步**

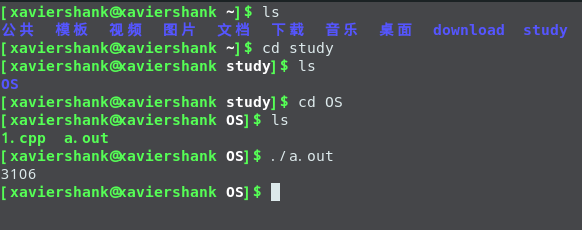
(注意：本次所有实验都在Linux中完成)

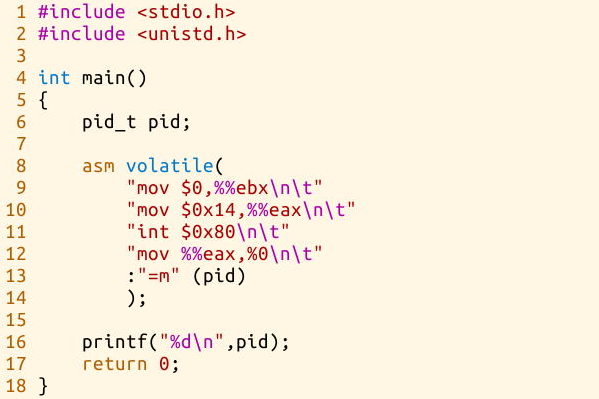
一、（系统调用实验）了解系统调用不同的封装形式。

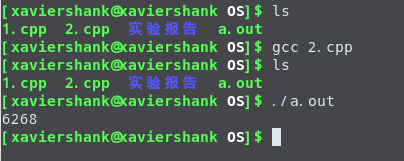
要求：1、参考下列网址中的程序。阅读分别运行用API接口函数getpid()直接调用和汇编中断调用两种方式调用Linux操作系统的同一个系统调用getpid的程序(请问getpid的系统调用号是多少？linux系统调用的中断向量号是多少？)。2、上机完成习题1.13。

<http://hgdcg14.blog.163.com/blog/static/23325005920152257504165/>









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。

要求：

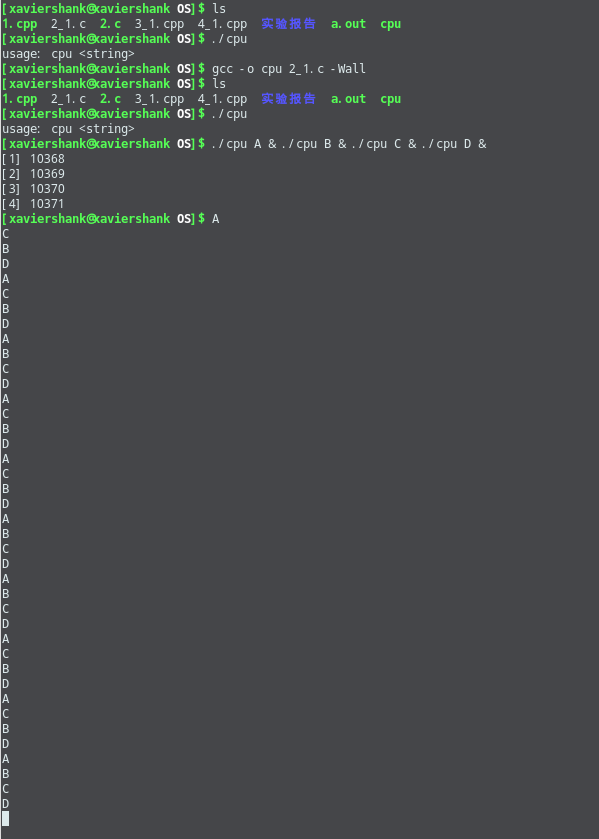
1. 编译运行该程序（cpu.c），观察输出结果，说明程序功能。

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

程序功能：循环输出传入参数到屏幕，如果无参数，则输出usage:cpu<string>

输出结果在下面：

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。



1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <sys/time.h>

4 #include <assert.h>

5 #include "common.h"

6

7 int

8 main(int argc, char \*argv[])

9 {

10 if (argc != 2) {

11 fprintf(stderr, "usage: cpu <string>\n");

12 exit(1);

13 }

14 char \*str = argv[1];

15 while (1) {

16 spin(1);

17 printf("%s\n", str);

18 }

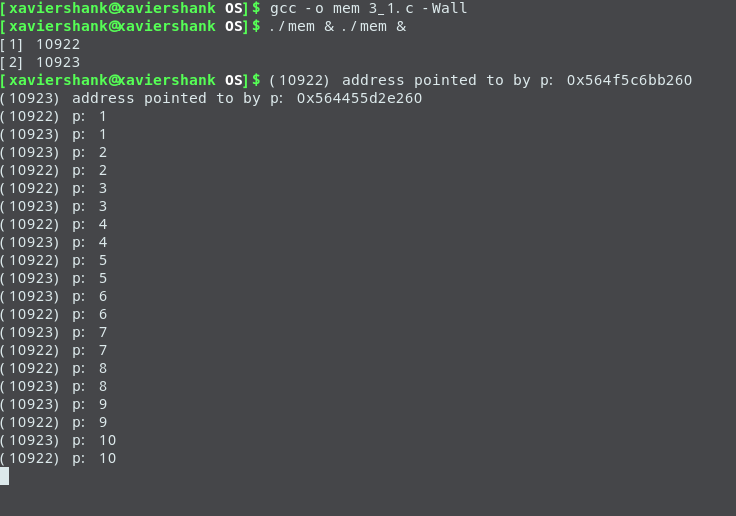
1. eturn 0;

三、（内存分配实验）根据以下代码完成实验。

要求：

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)

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 &



1 #include <unistd.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #include "common.h"

5

6 int

7 main(int argc, char \*argv[])

8 {

9 int \*p = malloc(sizeof(int)); // a1

10 assert(p != NULL);

11 printf("(%d) address pointed to by p: %p\n",

12 getpid(), p); // a2

13 \*p = 0; // a3

14 while (1) {

15 Spin(1);

16 \*p = \*p + 1;

17 printf("(%d) p: %d\n", getpid(), \*p); // a4

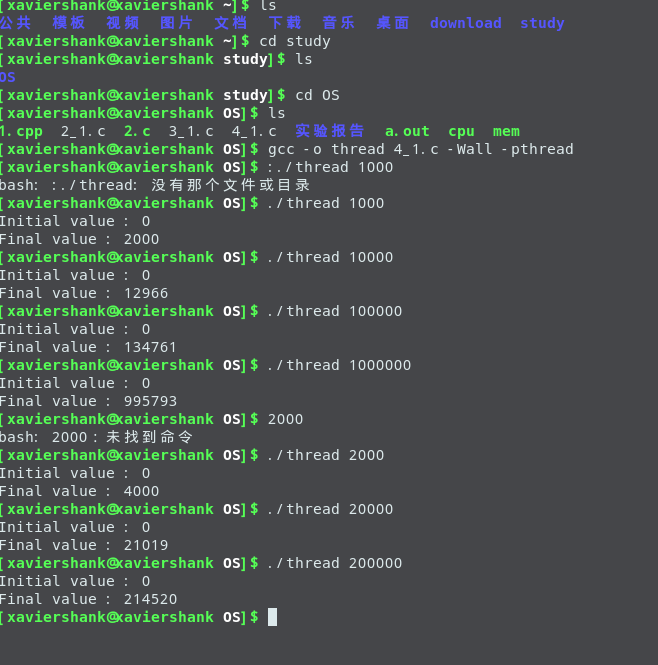
18 }

19 return 0;

四、（共享的问题）根据以下代码完成实验。

要求：

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）
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。）
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。



1 #include <stdio.h>

2 #include <stdlib.h>

3 #include "common.h"

4

5 volatile int counter = 0;

6 int loops;

7

8 void \*worker(void \*arg) {

9 int i;

10 for (i = 0; i < loops; i++) {

11 counter++;

12 }

13 return NULL;

14 }

15

16 int

17 main(int argc, char \*argv[])

18 {

19 if (argc != 2) {

20 fprintf(stderr, "usage: threads <value>\n");

21 exit(1);

22 }

23 loops = atoi(argv[1]);

24 pthread\_t p1, p2;

25 printf("Initial value : %d\n", counter);

26

27 Pthread\_create(&p1, NULL, worker, NULL);

28 Pthread\_create(&p2, NULL, worker, NULL);

29 Pthread\_join(p1, NULL);

30 Pthread\_join(p2, NULL);

31 printf("Final value : %d\n", counter);

32 return 0;