

**操作系统实验一: 操作系统初步**

**——报告**

指导老师：何永忠老师

作者：于小涵

学号： 16281021

班级： 计科1601

**北京交通大学**

2019年3月

**实验一: 操作系统初步**

一、（系统调用实验）了解系统调用不同的封装形式。

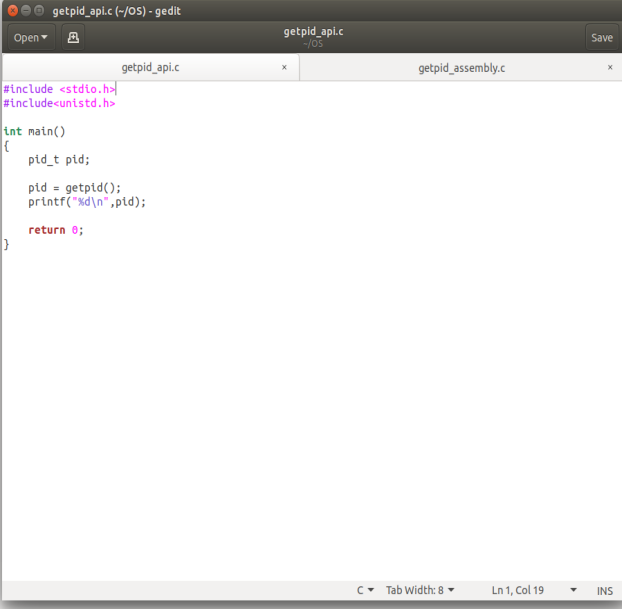
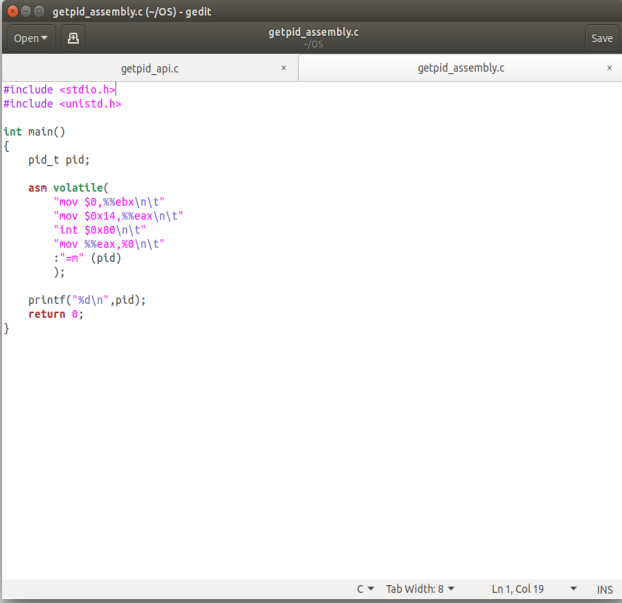
1、参考下列网址中的程序。阅读分别运行用API接口函数getpid()直接调用和汇编中断调用两种方式调用Linux操作系统的同一个系统调用getpid的程序(请问getpid的系统调用号是多少？linux系统调用的中断向量号是多少？)。

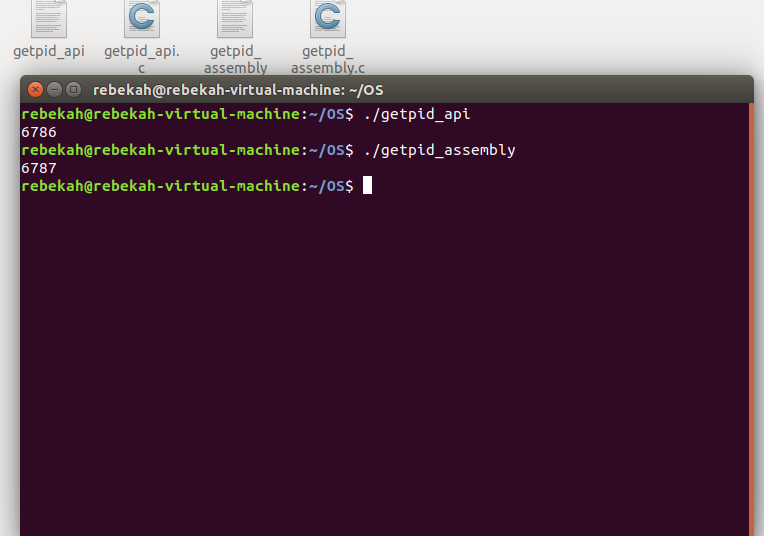
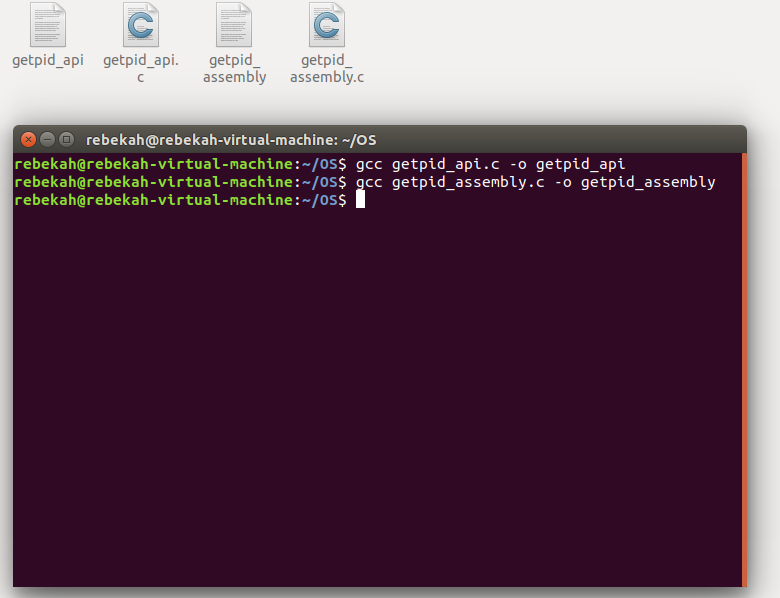
答：

Linux系统中断向量号是0x80

getpid的系统调用号是20

实验结果如下：



1. 上机完成习题1.13。

答：

命令printf(“Hello World!\n”)可归入 C标准函数、GNU C 函数库。

理由：C标准函数库（ANSI C）是任何C编译器都应该遵循的C语言协议。它是基本的C语言函数库，包含了C语言最基本的库函数。这个库可以根据头文件划分为 15 个部分。

GNU C是在ANSI C的基础上增添了一些基于Linux操作系统的扩展，例如对ASM语法的支持、编译器支持内联函数、支持\_\_attribute\_\_机制

Linux API是程序员与系统调用之间的接口，系统调用并不直接与程序员进行交互，它仅仅是一个通过软中断机制向内核提交请求以获取内核服务的接口。实际使用中，程序员调用的通常都是用户编程接口（API）。

Linux系统调用：

#include <unistd.h>

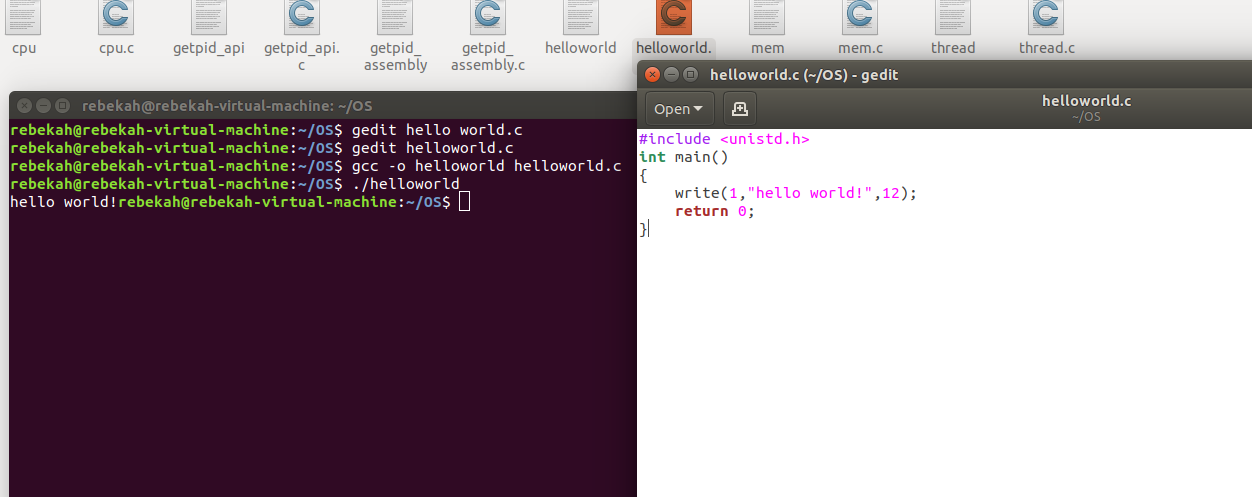
int main()

{

write(1,"hello world!",12);

return 0;

}



汇编：

section .data ;section declaration

msg db "Hello, world!",0xA ;our dear string

len equ $ - msg ;length of our dear string

section .text ;section declaration

;we must export the entry point to the ELF linker or

global \_start ;loader. They conventionally recognize \_start as their

;entry point. Use ld -e foo to override the default.

\_start:

;write our string to stdout

mov eax,4 ;system call number (sys\_write)

mov ebx,1 ;first argument: file handle (stdout)

mov ecx,msg ;second argument: pointer to message to write

mov edx,len ;third argument: message length

int 0x80 ;call kernel

;and exit

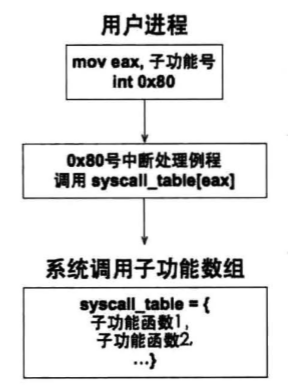
mov eax,1 ;system call number (sys\_exit)

xor ebx,ebx ;first syscall argument: exit code

int 0x80 ;call kernel

1. 阅读pintos操作系统源代码，画出系统调用实现的流程图。

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

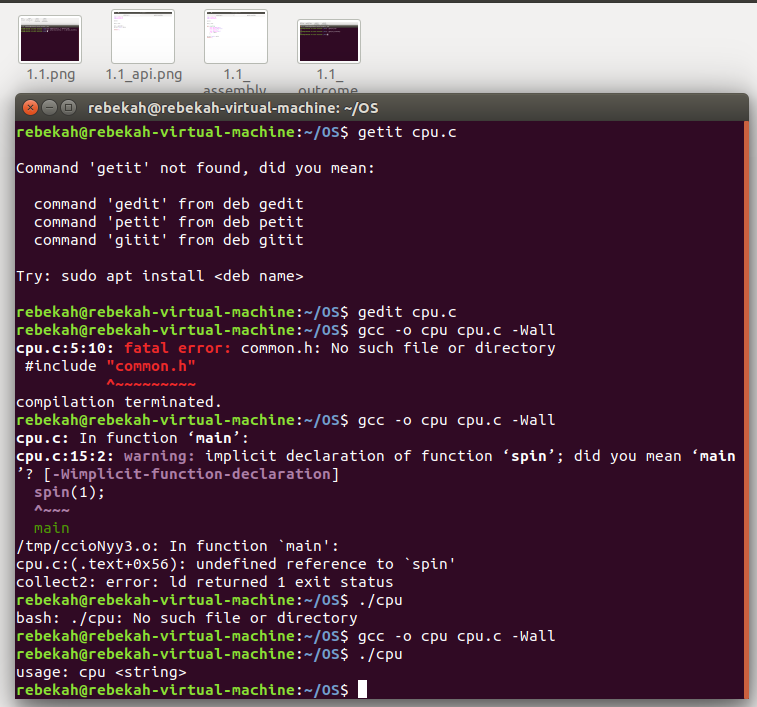


二、（并发实验）根据以下代码完成下面的实验。

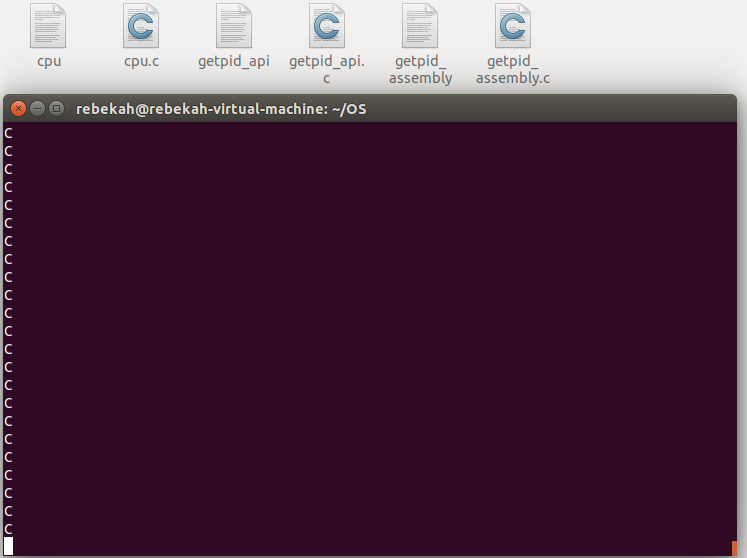
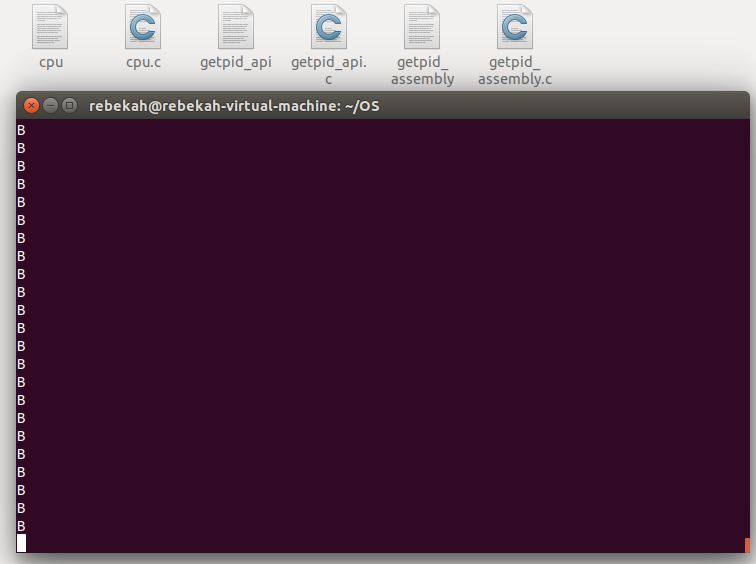
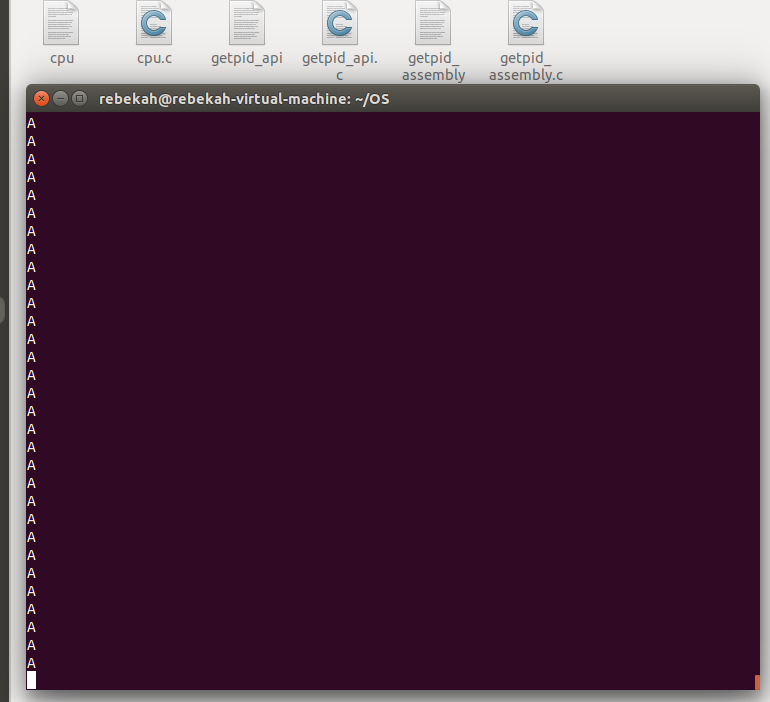
要求：

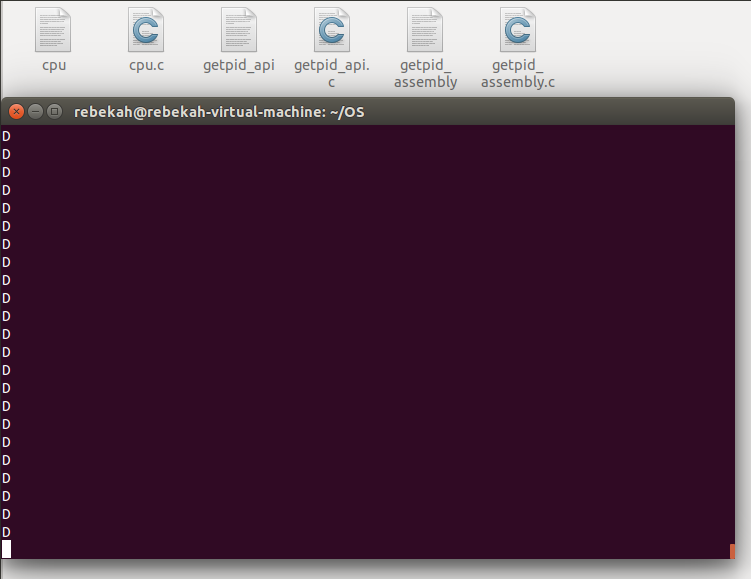
1. 编译运行该程序（cpu.c），观察输出结果，说明程序功能。

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



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);

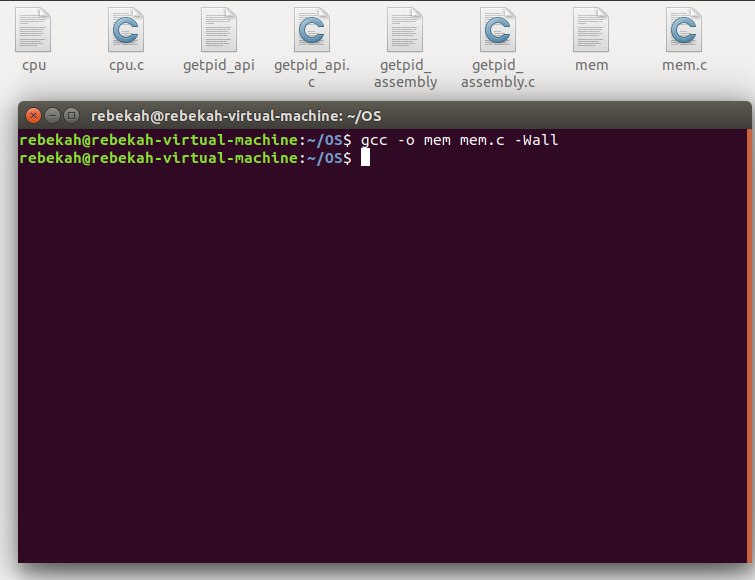
18 }

1. eturn 0;

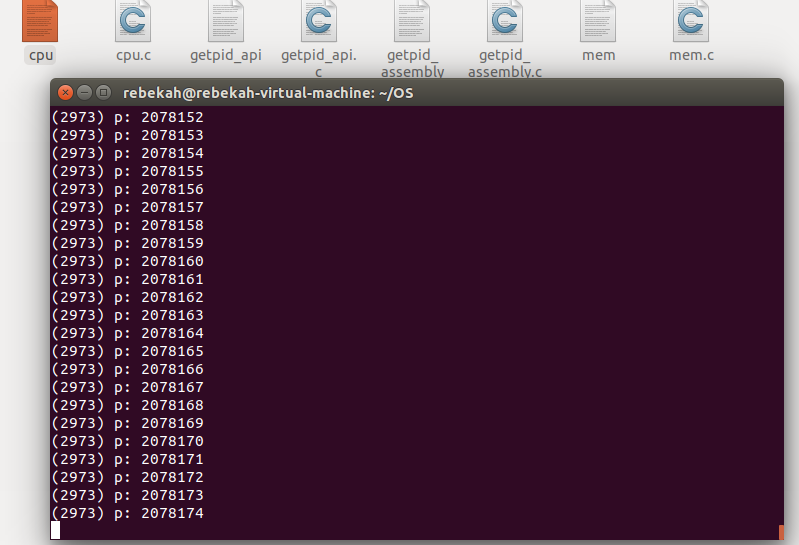
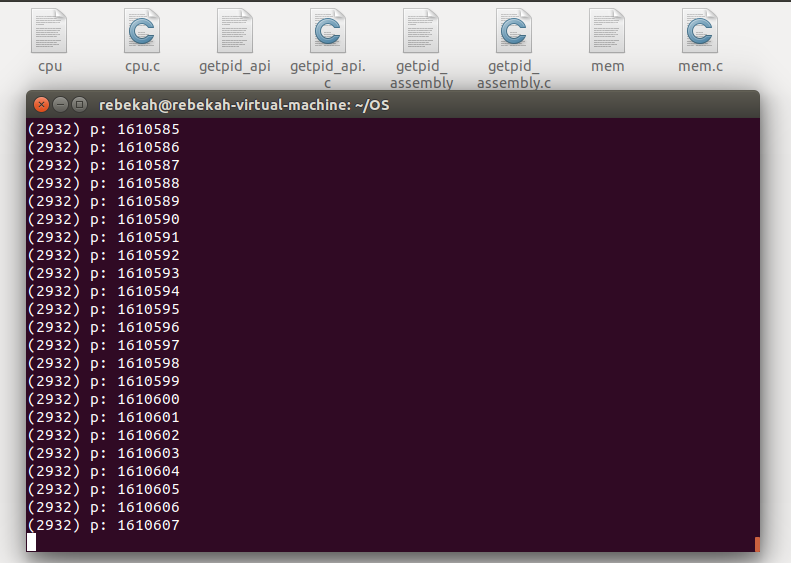
三、（内存分配实验）根据以下代码完成实验。

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)



1. 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

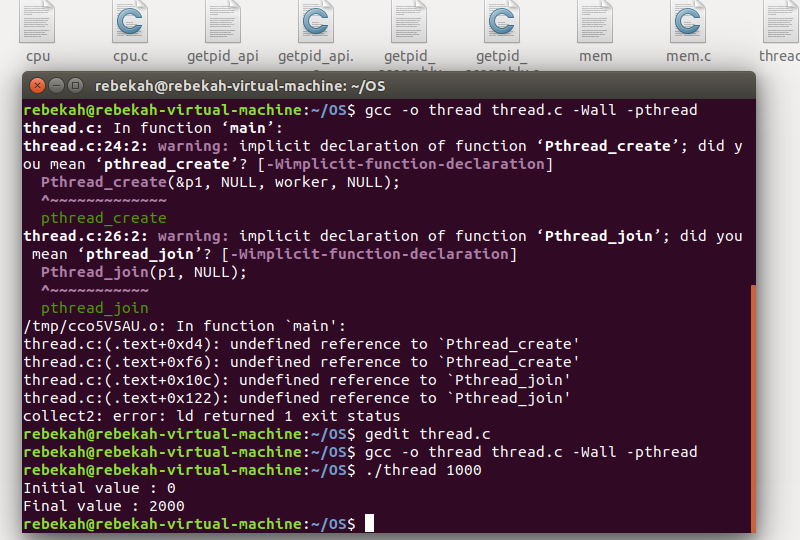
18 }

19 return 0;

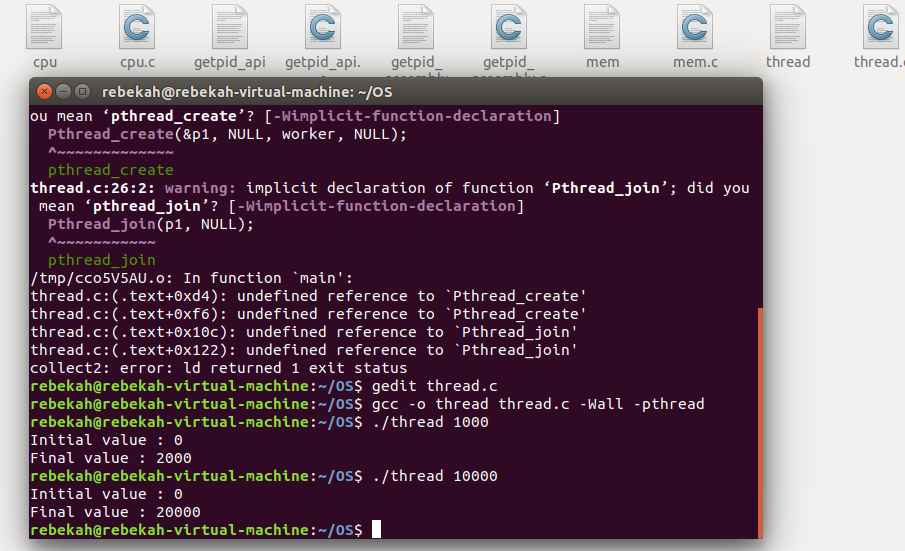
四、（共享的问题）根据以下代码完成实验。

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）



1. 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。）



1. 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");0

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;