



《操作系统》课程作业

学 号： 16281006

姓 名： 贾飞阳

专 业： 计算机科学与技术

学 院： 计算机与信息技术学院

提交日期： 2019 年 03 月 11 日

实验一：操作系统初步

作业题目：

1.1 （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？)。

（2）上机完成习题 1.13。

（3）阅读 **pintos** 操作系统源代码，画出系统调用实现的流程图。

1.2 （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`）

（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。

1.3 （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`）

（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 &`

1.4 （共享的问题）根据以下代码完成实验。

(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`) (执行命令 1: `./thread 1000`)

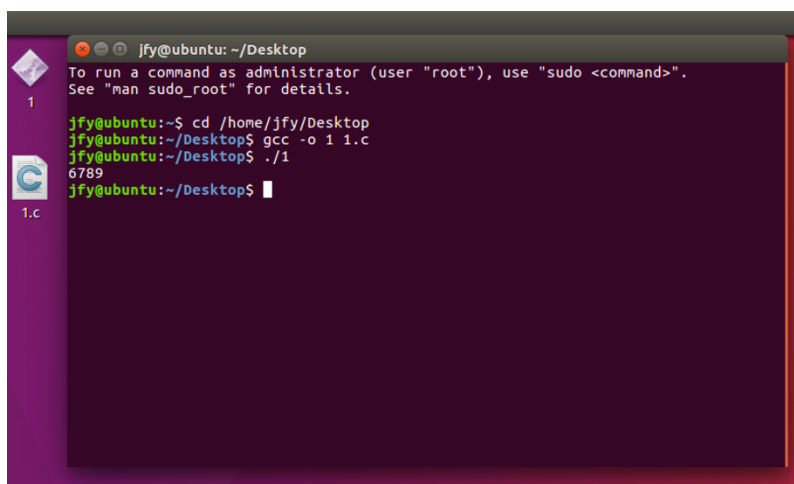
(2) 尝试其他输入参数并执行，并总结执行结果的有何规律？你能尝试解释它吗？(例如执行命令 2: `./thread 100000`) (或者其他参数。)

(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不会导致意想不到的问题。

作业解答：

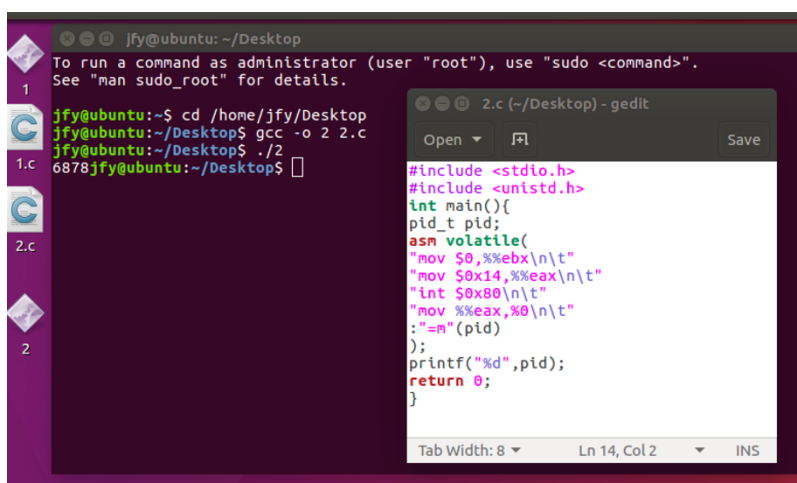
第 1.1 题解答：

(1) 从所给代码中可以看出 `getpid` 的系统调用号是 `0x14`，linux 系统调用的中断向量号是 `0x80`，两种调用方式如下所示：



```
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ gcc -o 1 1.c
jfy@ubuntu:~/Desktop$ ./1
6789
jfy@ubuntu:~/Desktop$
```

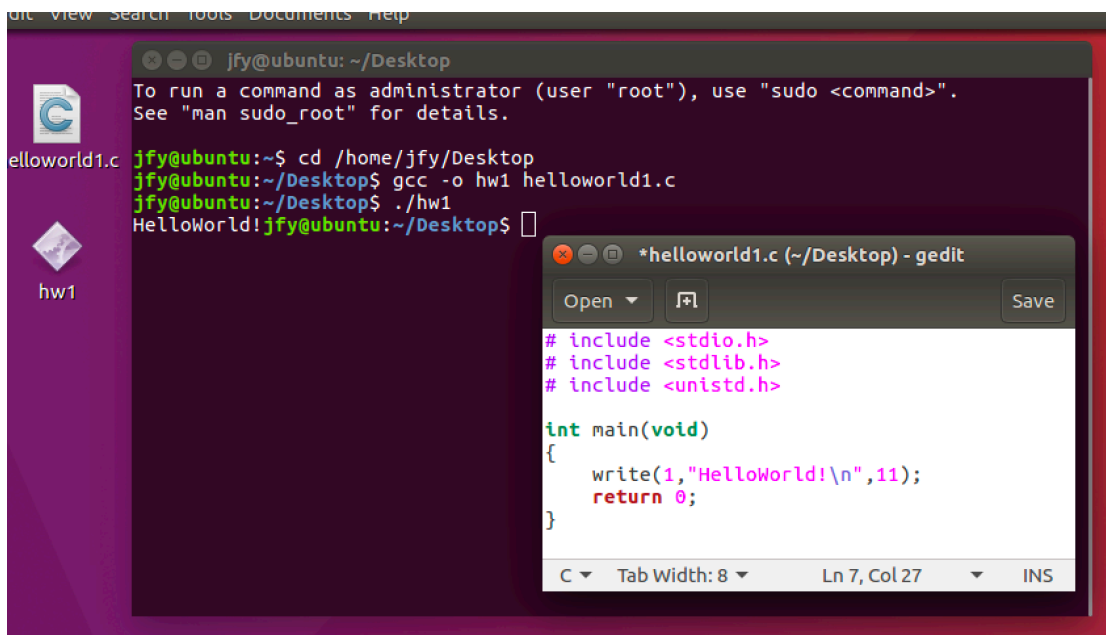


```
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ gcc -o 2 2.c
jfy@ubuntu:~/Desktop$ ./2
6878jfy@ubuntu:~/Desktop$
```

```
#include <stdio.h>
#include <unistd.h>
int main(){
    pid_t pid;
    asm volatile(
        "mov $0,%%ebx\n\t"
        "mov $0x14,%%eax\n\t"
        "int $0x80\n\t"
        "mov %%eax,%0\n\t"
        : "=m"(pid)
    );
    printf("%d",pid);
    return 0;
}
```

(2) 两种调用方式如下。其中，linux 的系统调用使用 `write` 函数。此函数有三个参数，第一个参数为 `1` 时是控制台输出。



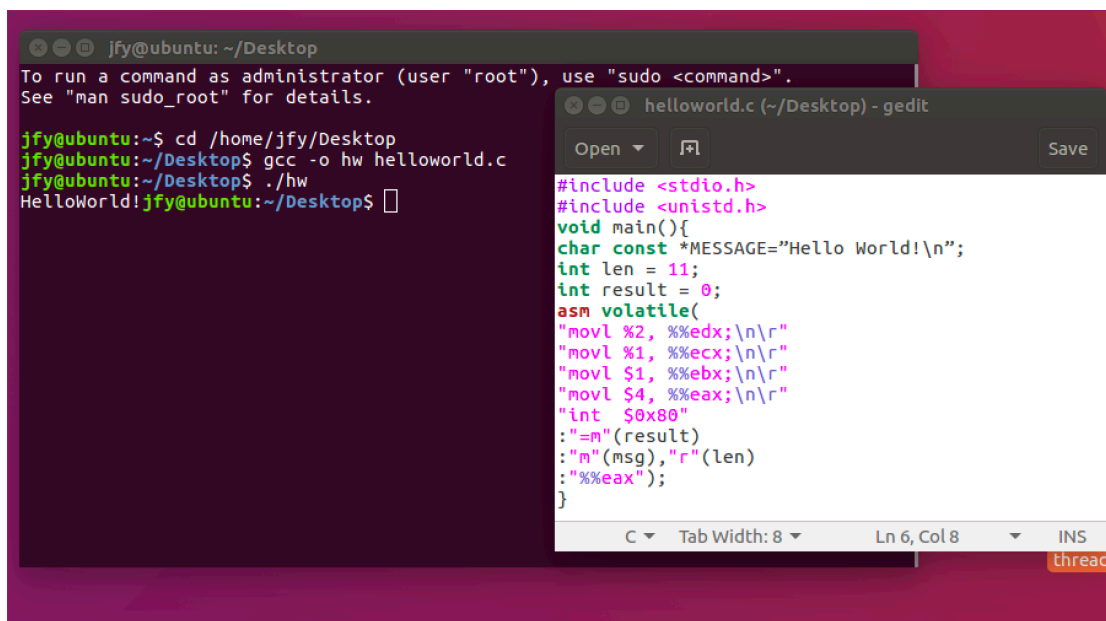
```
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ gcc -o hw1 helloworld1.c
jfy@ubuntu:~/Desktop$ ./hw1
HelloWorld!jfy@ubuntu:~/Desktop$
```

The code in `helloworld1.c` is:

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

int main(void)
{
    write(1, "HelloWorld!\n", 11);
    return 0;
}
```



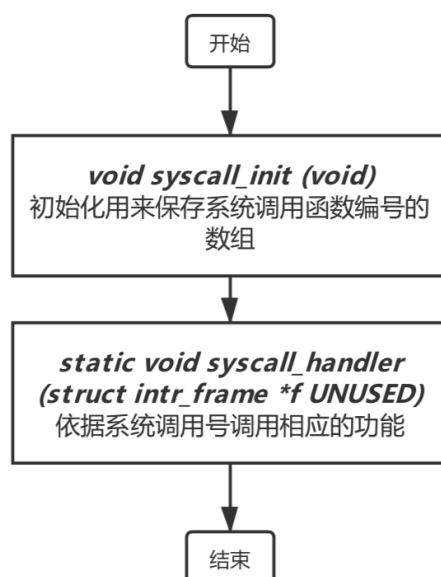
```
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ gcc -o hw helloworld.c
jfy@ubuntu:~/Desktop$ ./hw
HelloWorld!jfy@ubuntu:~/Desktop$
```

The code in `helloworld.c` is:

```
#include <stdio.h>
#include <unistd.h>
void main(){
    char const *MESSAGE="Hello World!\n";
    int len = 11;
    int result = 0;
    asm volatile(
        "movl %2, %%edx;\n\r"
        "movl %1, %%ecx;\n\r"
        "movl $1, %%ebx;\n\r"
        "movl $4, %%eax;\n\r"
        "int $0x80"
        : "=m"(result)
        : "m"(MESSAGE), "r"(len)
        : "%%eax");
}
```

(3) 通过阅读../userprog/syscall.c 文件, 可知 pintos 操作系统的系统调用的实现如下:



第 1.2 题解答:

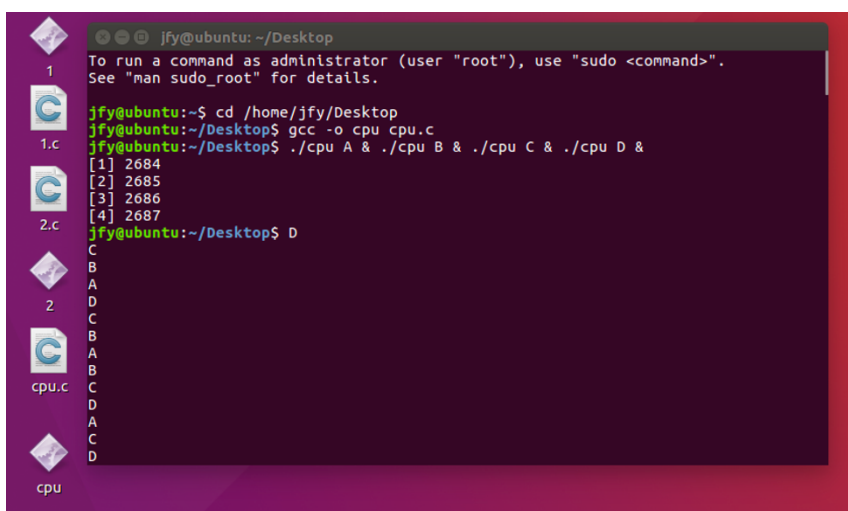
(1) 进行如题所示的编译命令和执行命令, 结果如下。根据代码和输出的信息可以看出, 这个程序的作用是输出命令行参数中的内容。如果输入的参数个数为 0 (代码中设置的条件为不等于 2, 因为函数的名称在是命令行参数中的第一个, 用户参数从第二个算起), 则输出提示信息。

```
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ ./cpu
usage: cpu <string>
jfy@ubuntu:~/Desktop$
```

The screenshot shows a terminal window with the following content: The user is in the directory ~/Desktop. They run the command ./cpu, which outputs 'usage: cpu <string>'. The terminal window has a purple title bar and a sidebar with icons for files 1, 1.c, 2.c, 2, cpu.c, and cpu.

(2) 进行如题所示的编译命令和执行命令，结果如下。可以明显的看到输出的顺序与输入参数的顺序 (ABCD) 是不一致的。且这个不一致是变化的，并不仅是除了 ABCD 这个顺序之外的单个顺序。四个程序在宏观上是一同运行的，微观上还是在 `cpu` 交替运行的，他们的优先级相同，因此先进行的程序不一定是先结束，这就造成了这四个程序同时运行的时候，结束的顺序是不确定的，那么输出顺序也是不一定的。

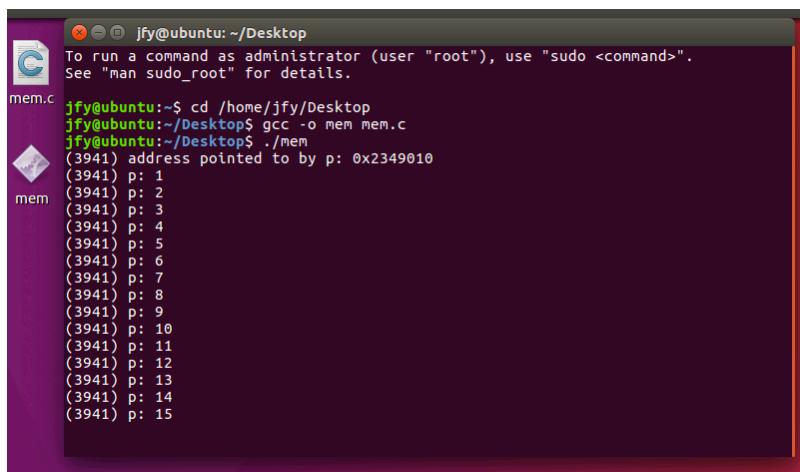


```
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ gcc -o cpu cpu.c
jfy@ubuntu:~/Desktop$ ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 2684
[2] 2685
[3] 2686
[4] 2687
jfy@ubuntu:~/Desktop$ D
C
B
A
D
C
B
A
B
C
C
D
D
A
C
D
cpu
```

第 1.3 题解答:

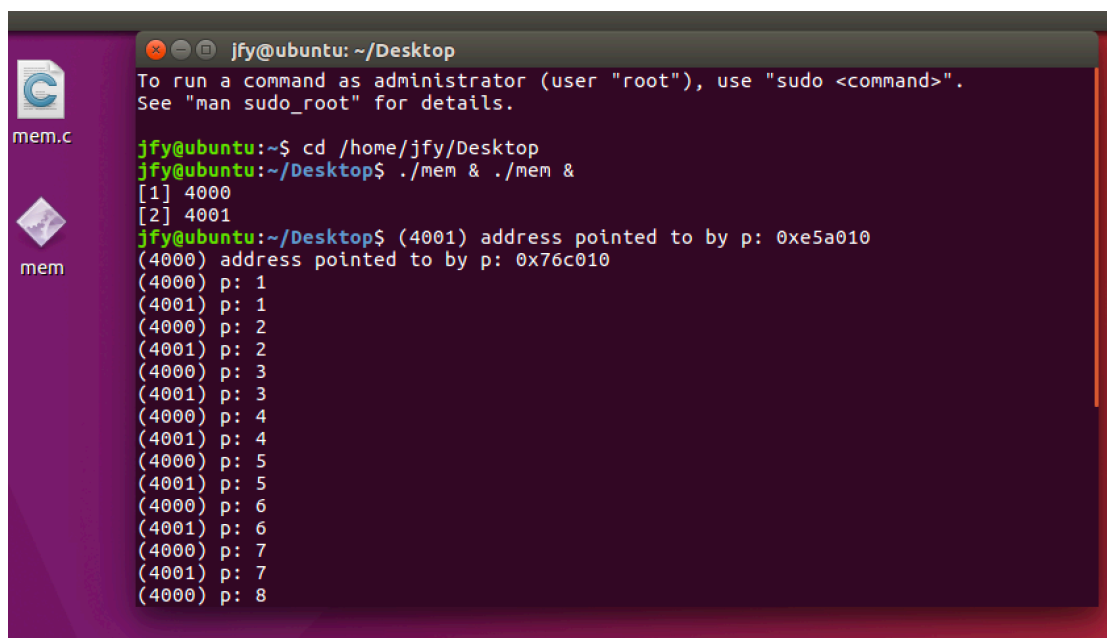
(1) 进行如题所示的编译命令和执行命令，结果如下。该程序做了如下几件事：程序分配了内存并打印出内存地址，然后将 `0` 放入内存的第一个位置。在一个无限循环中，延迟一秒并递增存储在 `p` 中保存的地址的值。对于每个 `print` 语句，它还会打印出正在运行的程序的进程标识符（每个运行过程中都唯一）。



```
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

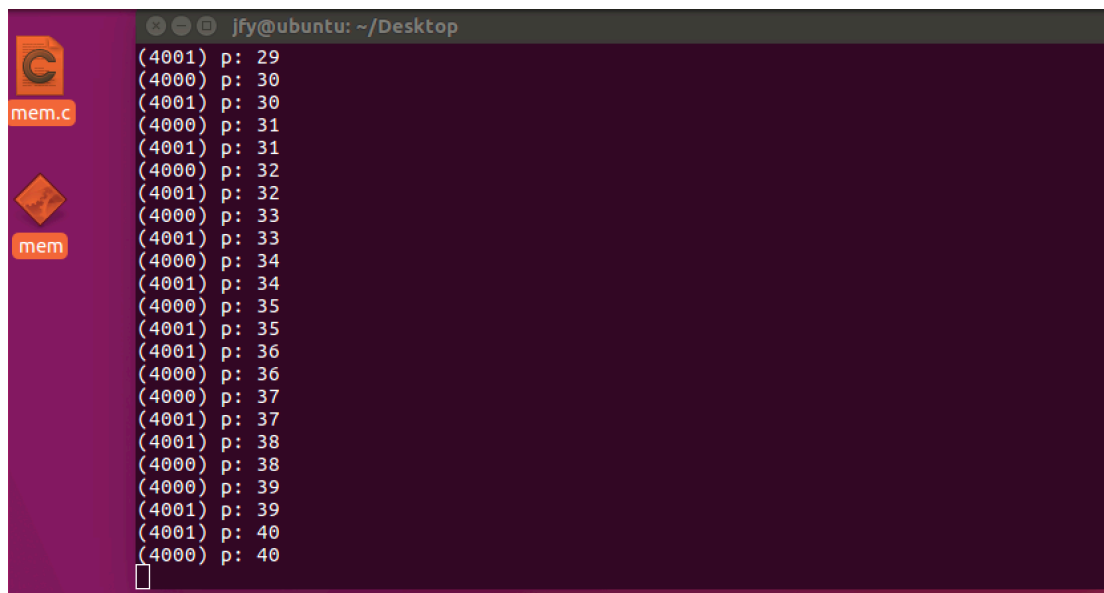
mem.c
jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ gcc -o mem mem.c
jfy@ubuntu:~/Desktop$ ./mem
(3941) address pointed to by p: 0x2349010
(3941) p: 1
(3941) p: 2
(3941) p: 3
(3941) p: 4
(3941) p: 5
(3941) p: 6
(3941) p: 7
(3941) p: 8
(3941) p: 9
(3941) p: 10
(3941) p: 11
(3941) p: 12
(3941) p: 13
(3941) p: 14
(3941) p: 15
```

(2) 进行如题所示的编译命令和执行命令，结果如下。可以从运行结果看到，两个独立运行的程序并不共享一块物理地址。这是因为操作系统虚拟化内存。每个进程访问自己的私有虚拟地址空间，操作系统以某种方式映射到机器的物理内存。一个正在运行的程序中的内存引用不会影响其他进程（或 OS 本身）的地址空间。



```
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

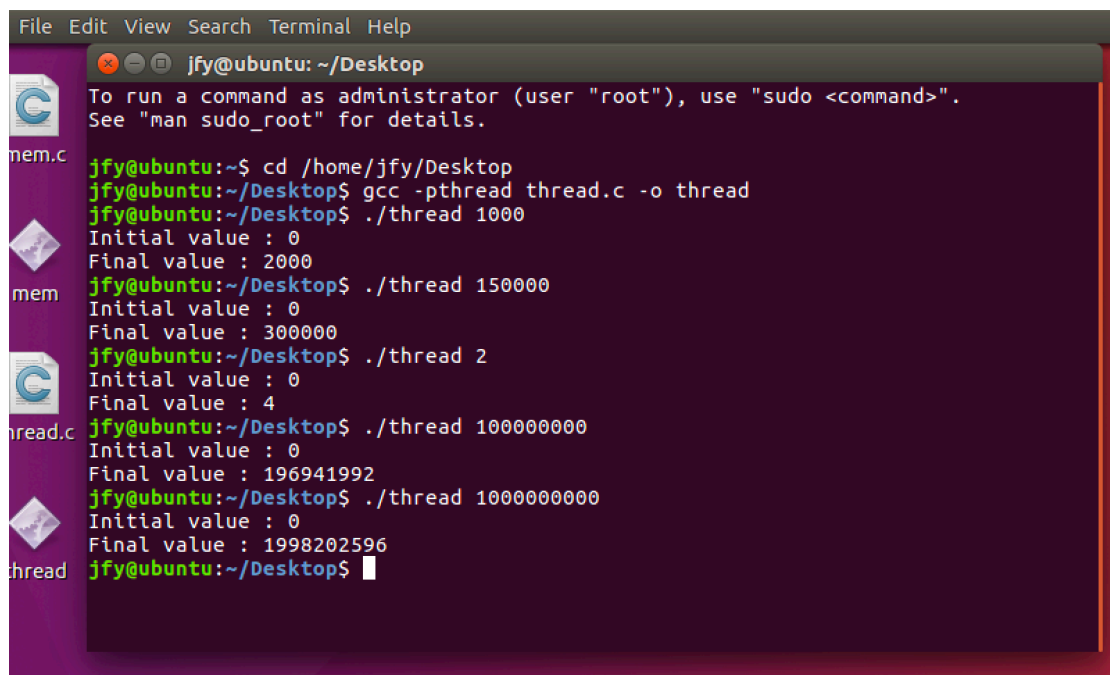
jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ ./mem & ./mem &
[1] 4000
[2] 4001
jfy@ubuntu:~/Desktop$ (4001) address pointed to by p: 0xe5a010
(4000) address pointed to by p: 0x76c010
(4000) p: 1
(4001) p: 1
(4000) p: 2
(4001) p: 2
(4000) p: 3
(4001) p: 3
(4000) p: 4
(4001) p: 4
(4000) p: 5
(4001) p: 5
(4000) p: 6
(4001) p: 6
(4000) p: 7
(4001) p: 7
(4000) p: 8
```



```
jfy@ubuntu: ~/Desktop
(4001) p: 29
(4000) p: 30
(4001) p: 30
(4000) p: 31
(4001) p: 31
(4000) p: 32
(4001) p: 32
(4000) p: 33
(4001) p: 33
(4000) p: 34
(4001) p: 34
(4000) p: 35
(4001) p: 35
(4001) p: 36
(4000) p: 36
(4000) p: 37
(4001) p: 37
(4001) p: 38
(4000) p: 38
(4000) p: 39
(4001) p: 39
(4001) p: 40
(4000) p: 40
```


第 1.4 题解答:

(1) 进行如题所示的编译命令和执行命令, 结果如下。调用时输入的参数为 1000。程序使用 Pthread `.create()` 创建两个线程, 每个线程在 `worker` () 的例程中运行, 该函数的作用是循环递增的计数器, 计数区间为 1。当两个线程完成时, 计数器的最终值为 2000, 因为每个线程将计数器递 1000 次。所以可以推断当循环的输入值设置为 N 时, 程序的期望输出为 2N。



```
File Edit View Search Terminal Help
jfy@ubuntu: ~/Desktop
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
jfy@ubuntu:~$ cd /home/jfy/Desktop
jfy@ubuntu:~/Desktop$ gcc -pthread thread.c -o thread
jfy@ubuntu:~/Desktop$ ./thread 1000
Initial value : 0
Final value : 2000
jfy@ubuntu:~/Desktop$ ./thread 150000
Initial value : 0
Final value : 300000
jfy@ubuntu:~/Desktop$ ./thread 2
Initial value : 0
Final value : 4
jfy@ubuntu:~/Desktop$ ./thread 100000000
Initial value : 0
Final value : 196941992
jfy@ubuntu:~/Desktop$ ./thread 1000000000
Initial value : 0
Final value : 1998202596
jfy@ubuntu:~/Desktop$
```

(2) 在以上解释的基础下, 增大输入的参数, 会发现得到的输入不再是 2N。原因是: 计数器递增, 需要三个指令: 计数器的值从存储器加载到寄存器中、递增、存储回内存。这意味着这三条指令不是同时执行的, 所以这“一组”操作与其他操作之间的执行顺序是不固定的。而且 `countor`、`loops` 这两个全局变量是被这两个线程共享的。当两个的线程一起被调用执行的时候, 共享全局变量 `countor`, 在编写多线程的程序时, 同一个变量可能被多个线程修改, 而程序通过该变量同步各个线程。所以当参数变得很大时, 就会因为指令调用顺序和参数共享的缘故导致错误。