# 实验一: 操作系统初步

**班级：安全1601 姓名：张天悦 学号：16281153**

(注意：本次所有实验都在Linux中完成)

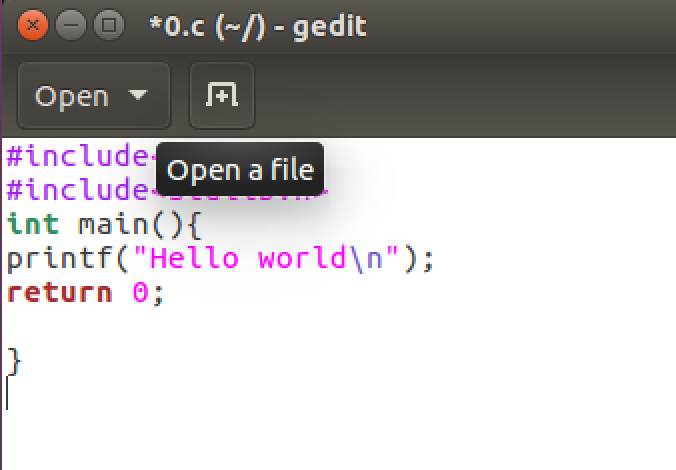
## 一、（系统调用实验）了解系统调用不同的封装形式。

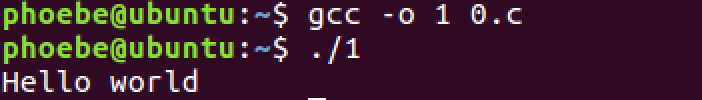
要求：

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？)。

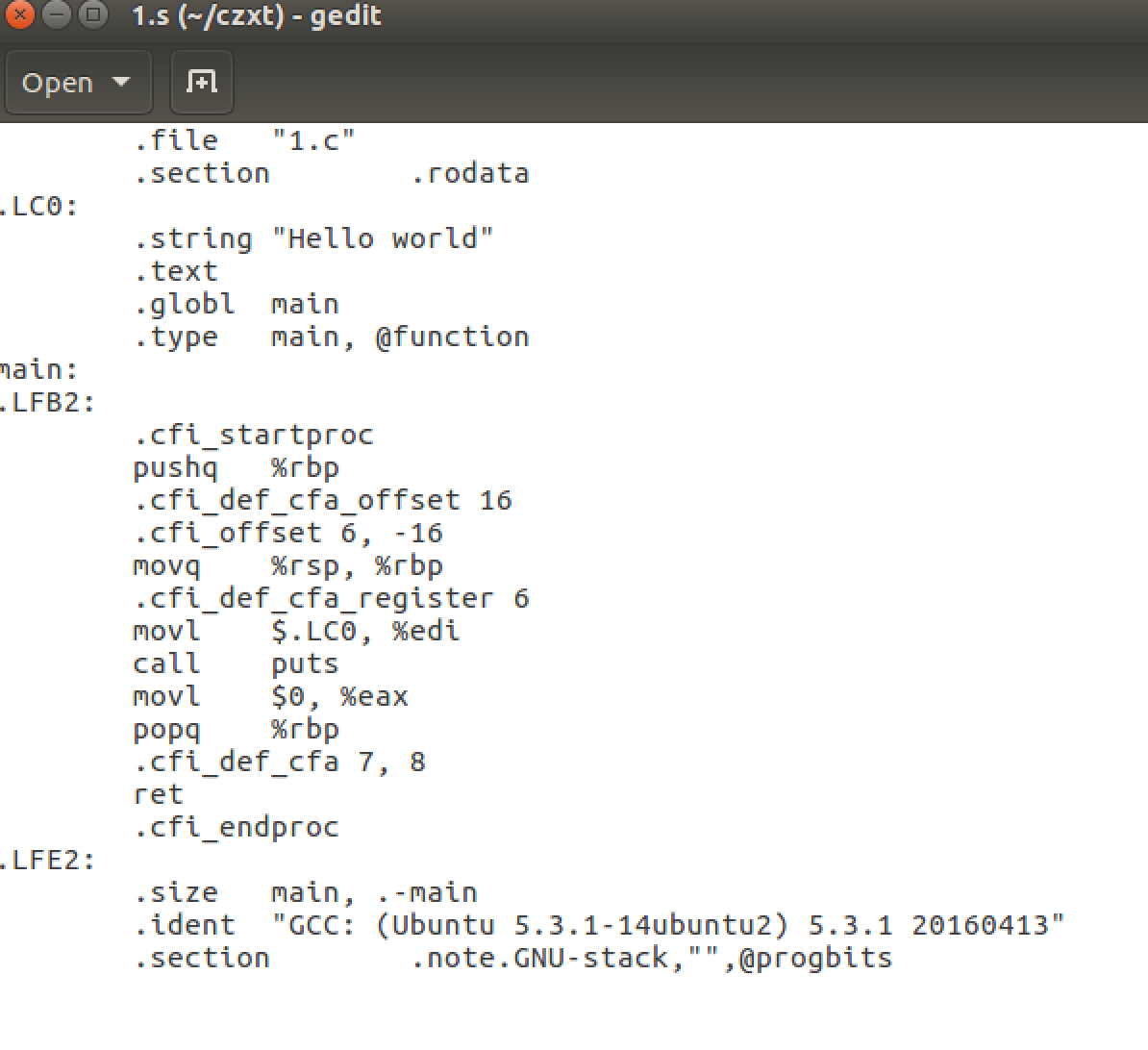
getpid的系统调用号是20，中断向量号80H，系统调用号14H。

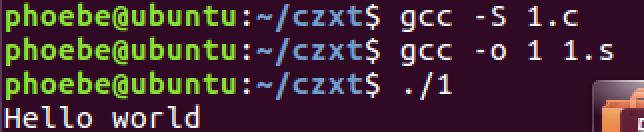
1. 上机完成习题1.13。





**汇编：**

****

****

3、阅读pintos操作系统源代码，画出系统调用实现的流程图。

系统调用

中断服务程序system call

查找系统调用表

在中断向量表中查

找

发起系统调用请求发起中断

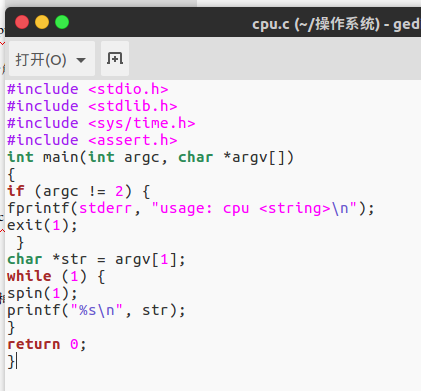
用户代码

## 二、（并发实验）根据以下代码完成下面的实验。

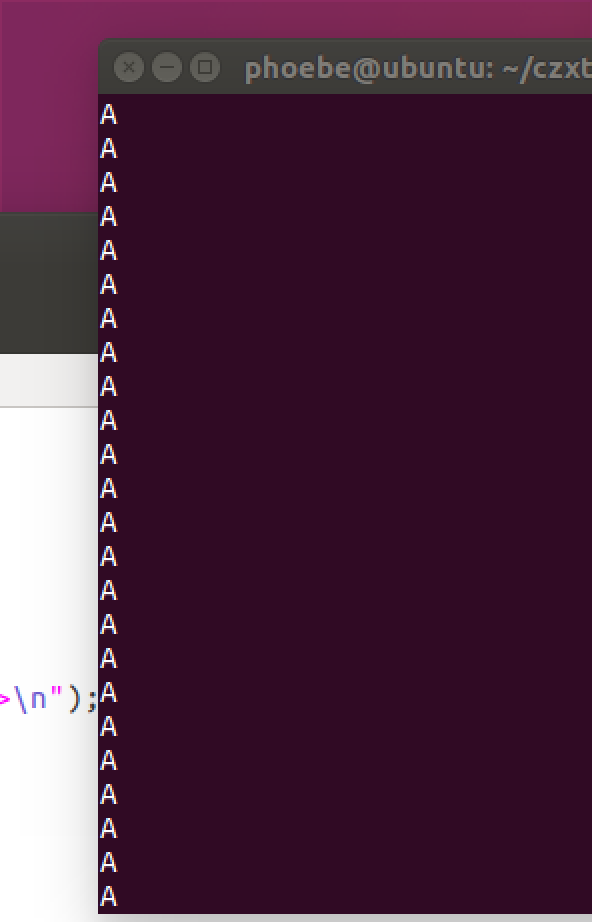
要求：

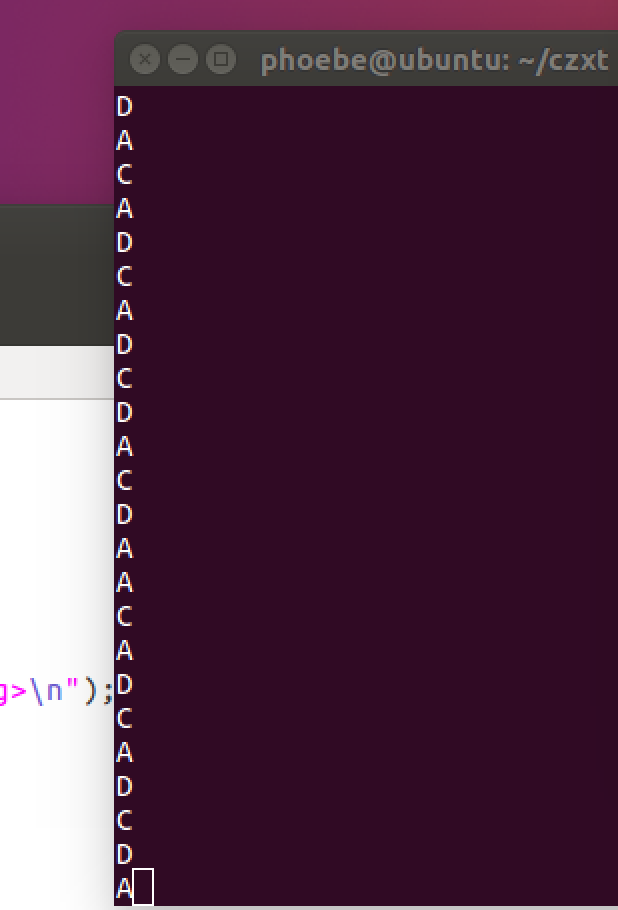
1. 编译运行该程序（cpu.c），观察输出结果，说明程序功能。

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



程序功能：打印出用户在命令行上输入的字符串，不停止。代码将打印用户输入的字符串，一直重复，程序将永远运行; 只有“Control-c”才能暂停程序。每隔1s输出传入参数，若传入参数正确，则输出；若传入参数不正确，则输出错误提示usage: cpu <string>。

1. 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。
2. 

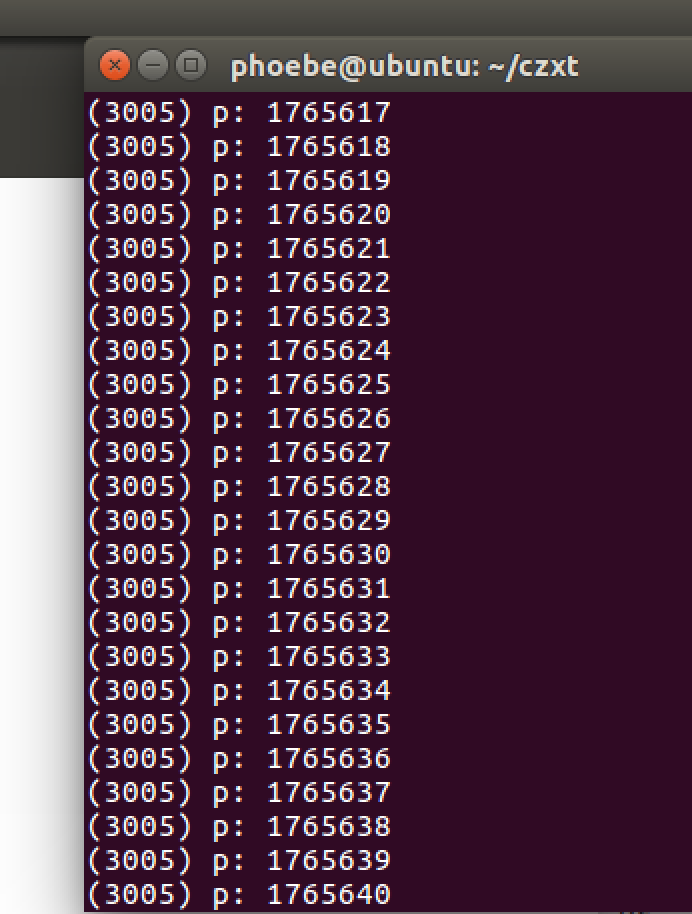


只有一个处理器，几个cpu程序在同时进行，运行有先后顺序，A先运行，后有ACD，这可以用操作系统中的并发行来解释，在操作系统中，并发是指一个时间段中有几个程序都处于已启动运行到运行完毕之间，且这几个程序都是在同一个处理机上运行，但任一个时刻点上只有一个程序在处理机上运行。

## 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)



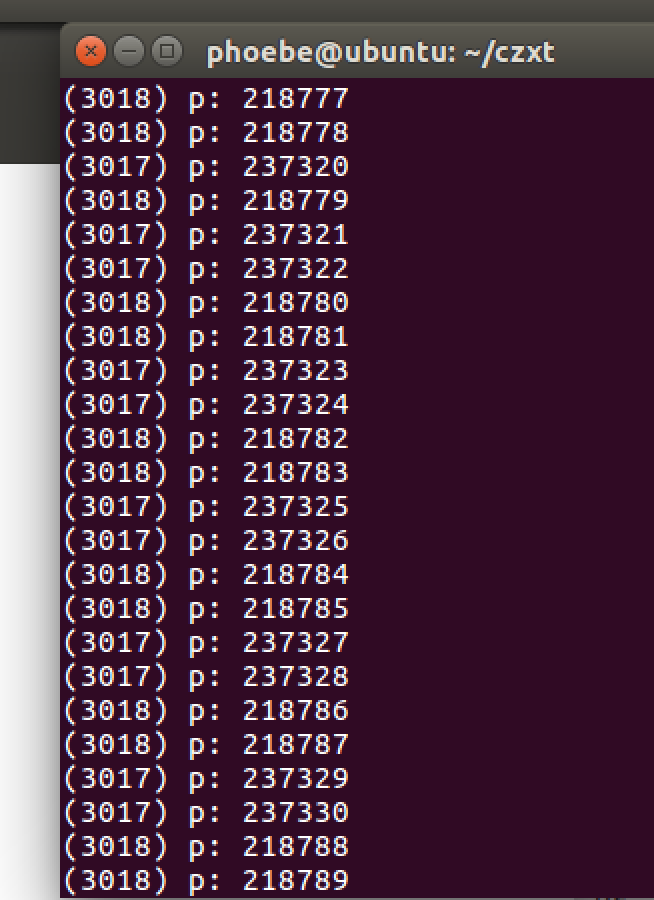
1、分配内存。

2、打印出内存地址，然后将数字0放入新分配的内存的第一个位置。

3、循环并递增存储在p中保存的地址的值。

对于每个printf语句，还会打印出正在运行的程序的进程标识符。 该标识符在每个运行过程中都是唯一的。程序运行时，更新值并打印出结果。

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 &



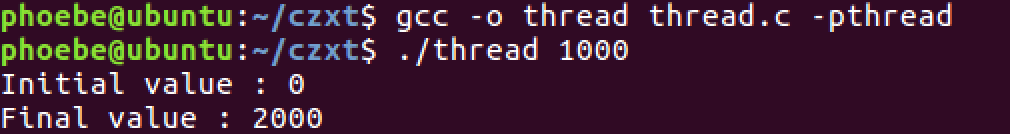
每个正在运行的程序都在同一地址分配了内存，但每个程序似乎都在独立更新这个地址的值。这是因为操作系统虚拟化了内存。 每个进程访问自己的私有虚拟地址空间（有时只称为其地址空间），操作系统以某种方式映射到机器的物理内存。 一个正在运行的程序中的内存引用不会影响其他进程（或OS本身）的地址空间；就运行程序而言，它拥有所有的物理内存。 然而，现实是物理内存是由操作系统管理的共享资源**。**

## 四、（共享的问题）根据以下代码完成实验。

要求：

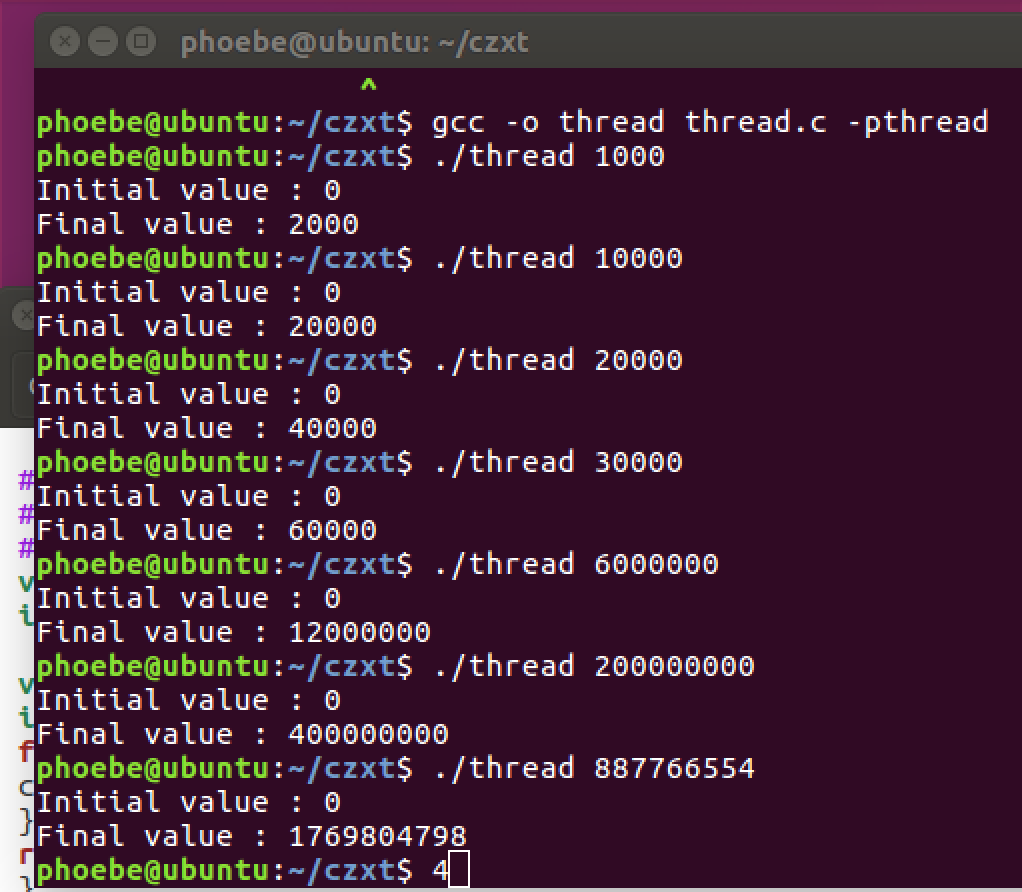
1. 阅读并编译运行该程序，观察输出结果，说明程序功能。（编译命令：gcc -o thread thread.c -Wall –pthread）（执行命令1：./thread 1000）





循环输入值为n，程序创建了两个线程，对同一地址内的数进行自增，输出初始数值和两次自增后的结果2n。

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。）



规律：final value在输入参数的2倍左右，不一定是2倍。

解释：原因与指令的执行方式有关。上面程序的一个关键部分共享计数器递增，这部分需要三个指令：一个用于将计数器的值从存储器加载到寄存器中，一个用于递增它，一个用于将其存储回内存。 因为这三个指令不是一次全部执行完毕，所以才会发生这样的现象。

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。

counter和loops是各个线程共享的。两个线程同时访问可能会导致数据的丢失，导致两次自增表现为一次自增。