**计算机操作系统**

**实验一报告**

课程名称： 计算机操作系统

任课教师： 何永忠

学生姓名： 杨珺瑶

学生学号： 16281210

专业班级： 安全1601

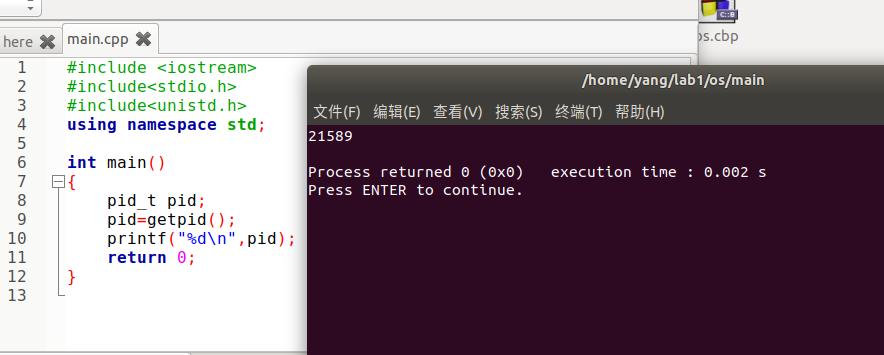
2019 年3月5日

# 一、系统调用实验

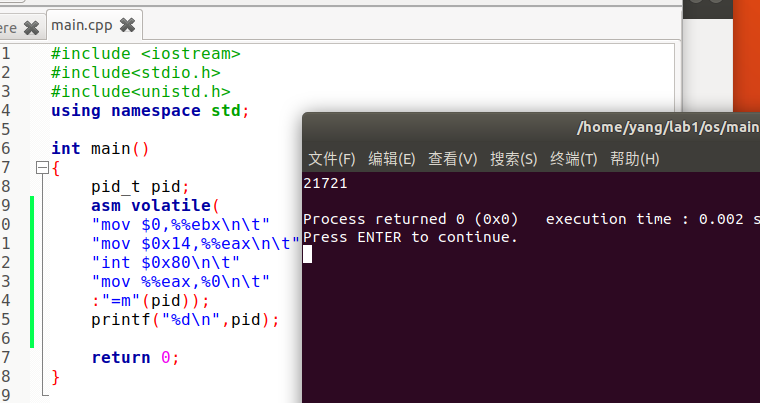
## 1. getpid

程序功能是查看当前进程号。

**（1）运行结果**



1-1 用API接口函数getpid()直接调用



1-2 汇编中断调用

**（2）问题**

**getpid的系统调用号是多少？**

由程序”mov $0x14……”可见，系统调用号为0x14，即20，经过查询可知为32位linux系统调用号。课上同学给的表格是64位的系统调用号，为39。

**linux系统调用的中断向量号是多少？**

0x80，由程序”int $0x80……”可得。进入内核态。

**Int与call区别？**

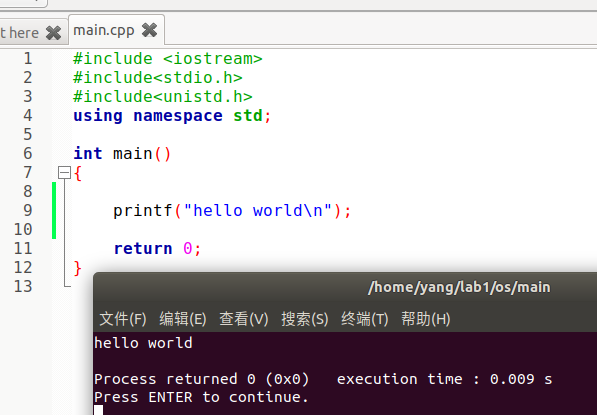
普通过程：call、jmp 直接转任意的子程序入口地址

软中断机制：int 0-255个中断向量。多了修改特权级和查找中断向量表的功能。

1.子程序完全为主程序服务的，两者属于主从关系，主程序需要子程序时就去调用子程序，并把调用结果带回主程序继续执行。而中断服务程序与主程序两者一般是无关的，不存在谁为谁服务的问题，两者是平行关系；2.主程序调用子程序过程完全属于软件处理过程，不需要专门的硬件电路，而中断处理系统是一个软、硬件结合系统，需要专门的硬件电路才能完成中断处理的过程；3.子程序嵌套可实现若干级，嵌套的最多级数由计算机内存开辟的堆栈大小限制，而中断嵌套级数主要由中断优先级数来决定，一般优先级数不会很大。

## 2.上机题1.13

**（1）linux系统调用C函数形式**

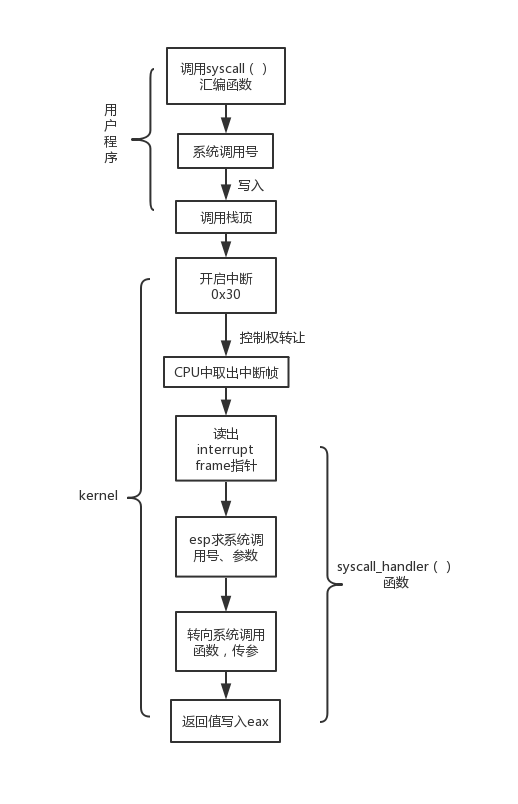


**（2）汇编代码形式**



这段代码在64位系统中无法直接编译，编译32位程序需要加上”-m32”。但是直接输入会提示错误，需要安装适配库” sudo apt-get install gcc-multilib g++-multilib module-assistant”。

## 3.系统调用实现流程图

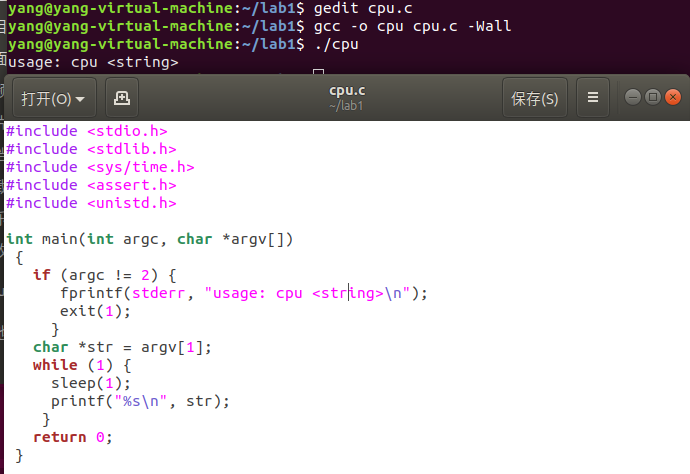


# 二、并发实验

## 1.编译运行，观察结果，说明功能

**“**编译命令： gcc -o cpu cpu.c **–Wall”**

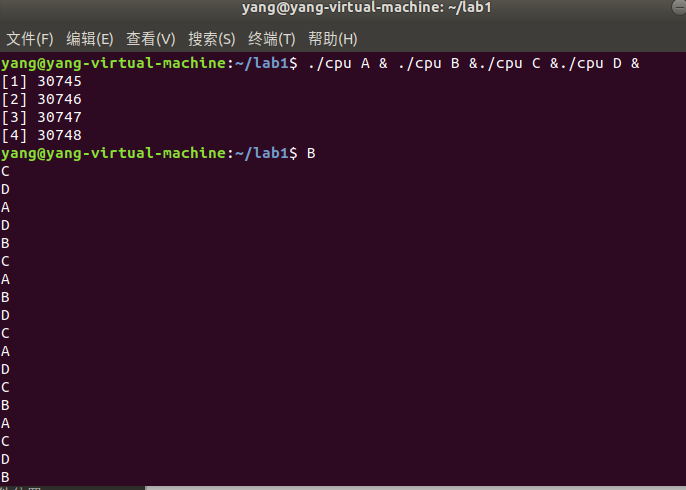
gcc有一些编译选项能够产生有用的警告信息，这些选项大多以-W开头，其中最有价值的当数-Wall，使用它能够使gcc产生尽可能多的警告信息。对Linux程序员来讲，gcc给出的警告信息是很有价值的，它们不仅可以帮助程序员写出更加健壮的程序，而且还是跟踪和调试程序的有力工具。建议在用gcc编译源代码时始终带上-Wall选项，并把它逐渐培养成为一种习惯，这对找出常见的隐式编程错误很有帮助。



2-1 代码、编译、运行

输出了一行"usage: cpu <string>\n"。本程序接收输入参数，循环延迟一秒打印参数。由于执行命令“./cpu”只输入了程序名，故只打印"usage: cpu <string>\n"。

## 2.再次运行



2-2 执行结果

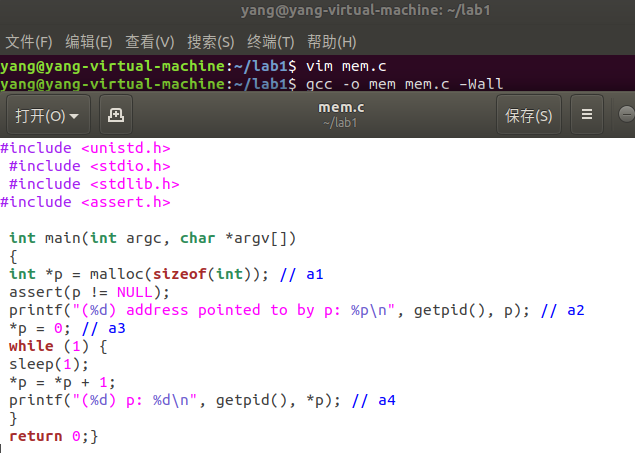
**程序cpu运行了几次？他们运行的顺序有何特点和规律？请结合操作系统的特征进行解释。**

运行了四次。暂未发现顺序规律，但基本是ABCD四个一组出现，其中A的时间间隔比较稳定。

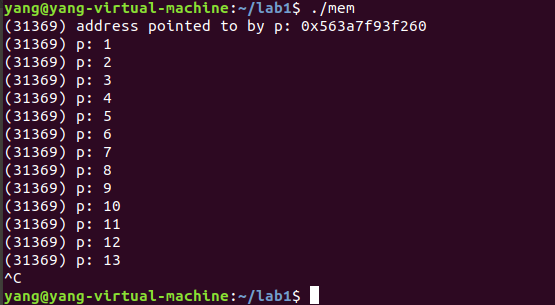
并发是操作系统的一大特性，指一个时间段中有几个程序都处于已启动运行到运行完毕之间，且这几个程序都是在同一个处理机上运行，但任一个时刻点上只有一个程序在处理机上运行。当有多个线程在操作时,如果系统只有一个CPU,则它根本不可能真正同时进行一个以上的线程，它只能把CPU运行时间划分成若干个时间段,再将时间段分配给各个线程执行，在一个时间段的线程代码运行时，其它线程处于挂起状。将单个CPU（或一小组CPU）转换为看似无限数量的CPU，从而允许许多程序看起来一次运行，这就是我们所谓的虚拟化CPU。

# 三、内存分配实验

## 1.编译运行，观察结果，说明功能



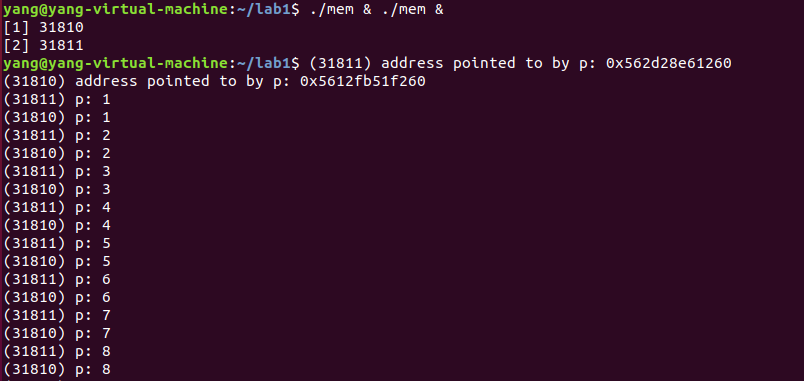
3-1 代码



3-2 运行结果

本程序首先给p分配一些内存（a1），打印出进程号和内存地址（a2），然后将数字0放入新分配的内存的第一个位置（a3）， 最后循环延迟一秒并递增存储p中保存的地址的值，打印此进程PID。

## 2.再次运行



3-3 执行结果

**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？**

不同。是。

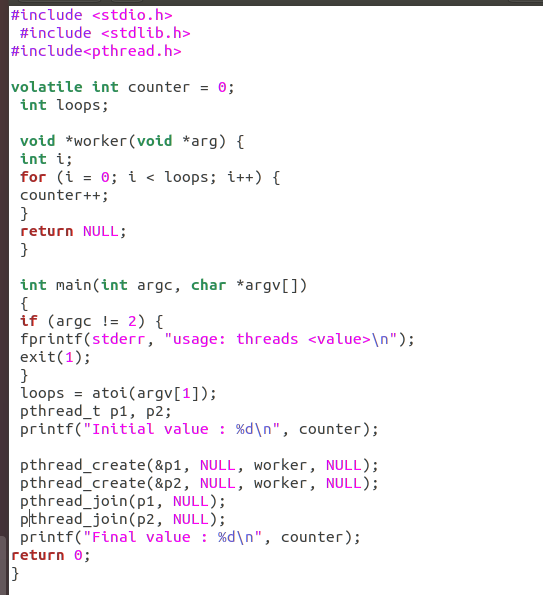
操作系统虚拟化了内存。每个进程访问自己的私有虚拟地址空间，操作系统以某种方式映射到机器的物理内存。一个正在运行的程序中的内存引用不会影响其他进程（或OS本身）的地址空间；就运行程序而言，它拥有所有的物理内存。现实是物理内存是由操作系统管理的共享资源。

# 四、共享的问题

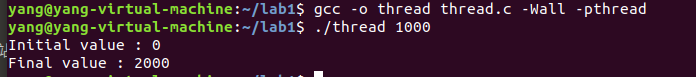
## 1.编译运行，观察结果，说明功能

gcc -o thread thread.c -Wall **–pthread**

链接 POSIX thread 库



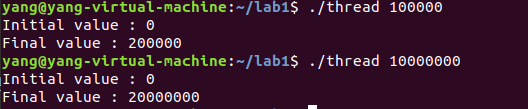
4-1 代码



4-2 执行结果

本程序使用pthread .create()创建两个线程，然后运行worker()，根据输入参数统计循环次数。最后输出两个线程循环总次数初始值和最终值。

## 2.再次运行



4-2 执行结果

**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？**

规律：输入n，输出2n。

counter、loops 是共享的。因为两个线程并发运行，counter的值是两个线程循环次数的和。

会导致其他问题。counter递增需要三个指令：1.将counter的值从存储器加载到寄存器中，2.递增它，3.将其存储回内存。 因为这三个指令不是一次全部执行，后面的操作在执行时可能某个共享数据被其他线程修改，而该修改并未同步到当前线程中，导致当前线程操作的数据与实际不符，所以可能会发生问题。