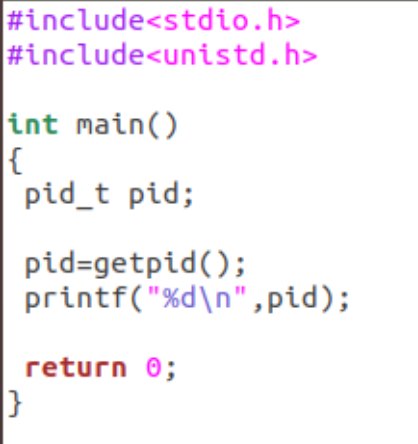
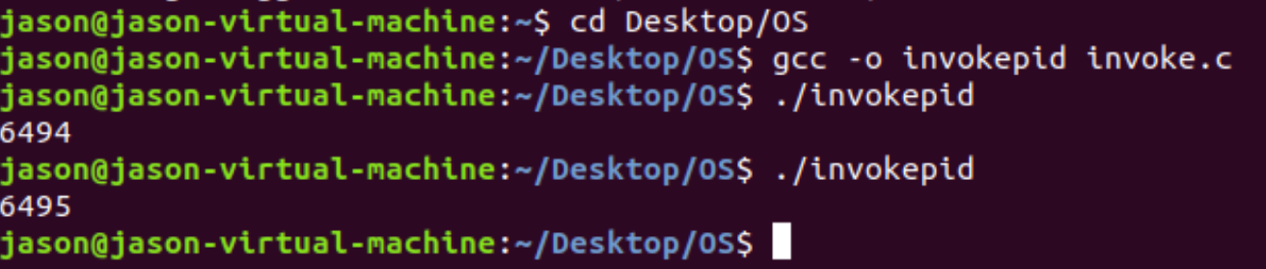
操作系统实验一实验报告：操作系统初步

16281012 聂小禹 计科1601

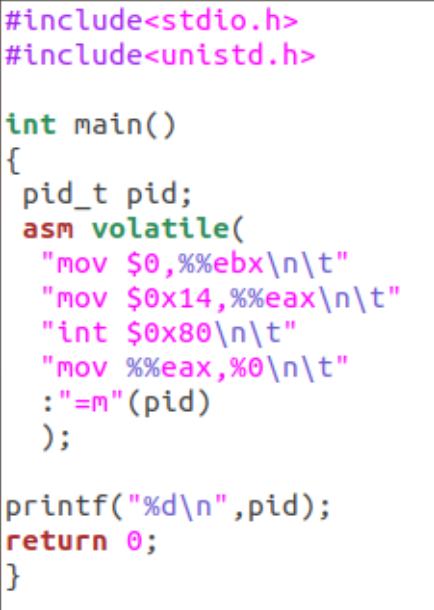
1. 系统调用实验
2. 利用API接口函数直接调用‘



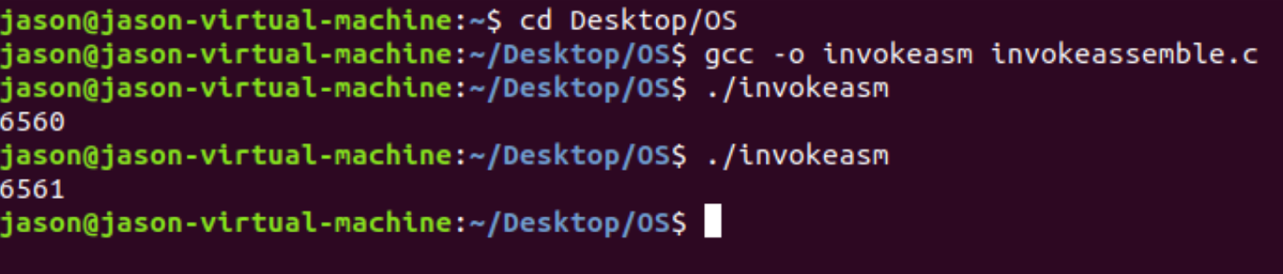
运行程序结果



1. 利用汇编中断调用



运行结果



getpid的系统调用号如图中显示，系统调用和普通函数完全不同，系统调用实际上是0x80号中断对应的中断处理程序的子程序。换句话说，在linux系统上，0x80中断是系统调用的统一入口。某个具体的系统调用是这个中断处理程序的子程序，进入具体某个系统调用是通过内核定义的系统调用号码来实现的。linux通过执行如下汇编代码陷入内核执行系统调用:

int 0x80; //这一句是进入系统调用统一入口。

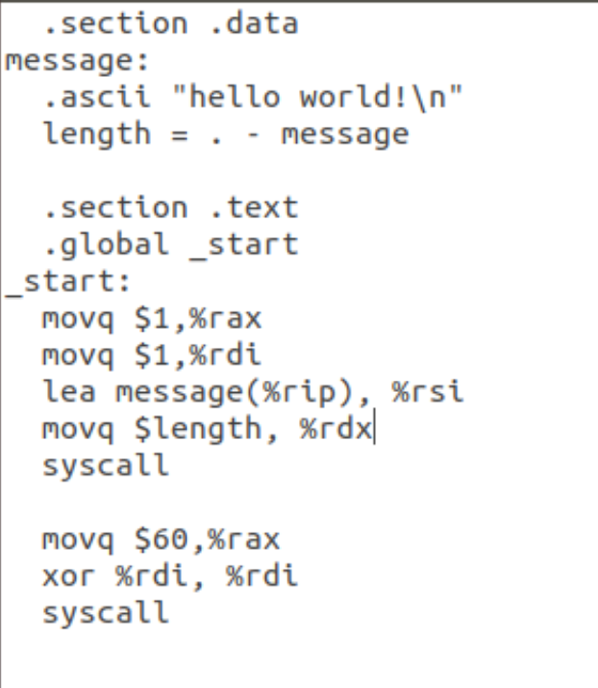
在执行"int 0x80;"进行中断之前，

1.把系统调用号码赋值给寄存器EAX；

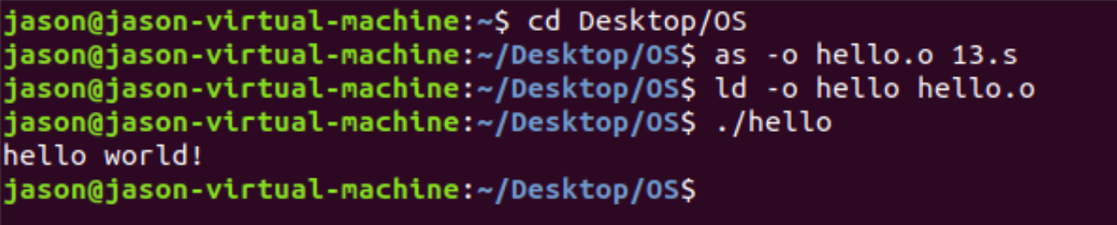
2.把系统调用需要的参数按次序赋值给寄存器EBX，ECX，EDX等等。

这样，等下0x80中断发生的时候，系统调用需要的全部信息就能通过这些寄存器传递给中断处理程序了。

* 1. 用汇编程序输出hello world!



输出结果



pintos

系统调用

🡪int 0x80中断向量

🡪System-Call系统调用入口

🡪SAVE\_ALL保存现场

🡪进行相应系统调用

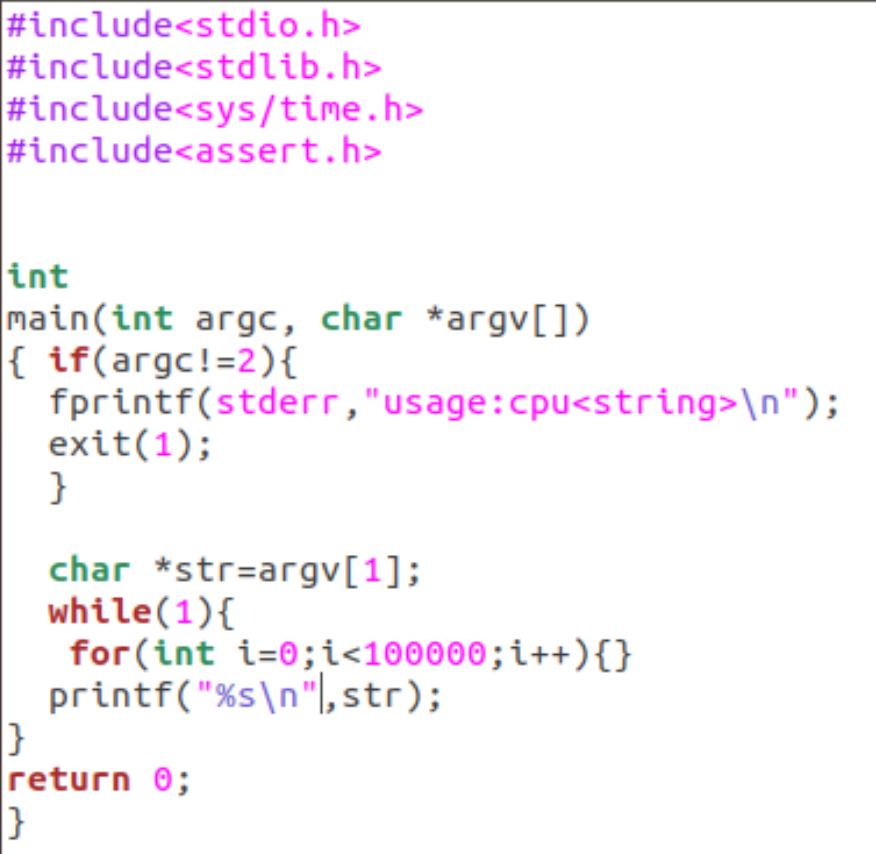
🡪是否进行信号处理和调度

----------------------------------------

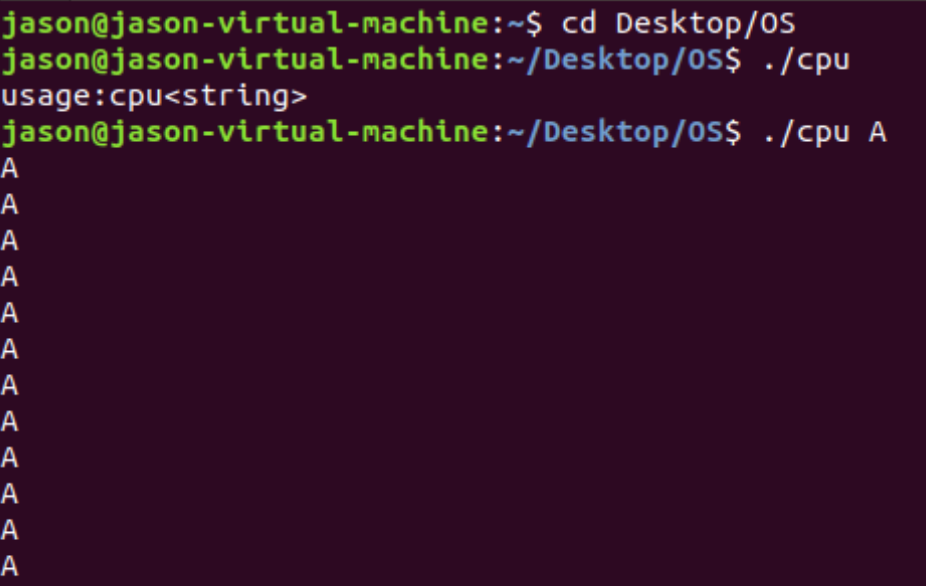
🡪是 进行信号处理进程调度

🡪否 恢复现场🡪返回用户态

1. 并发实验



运行结果

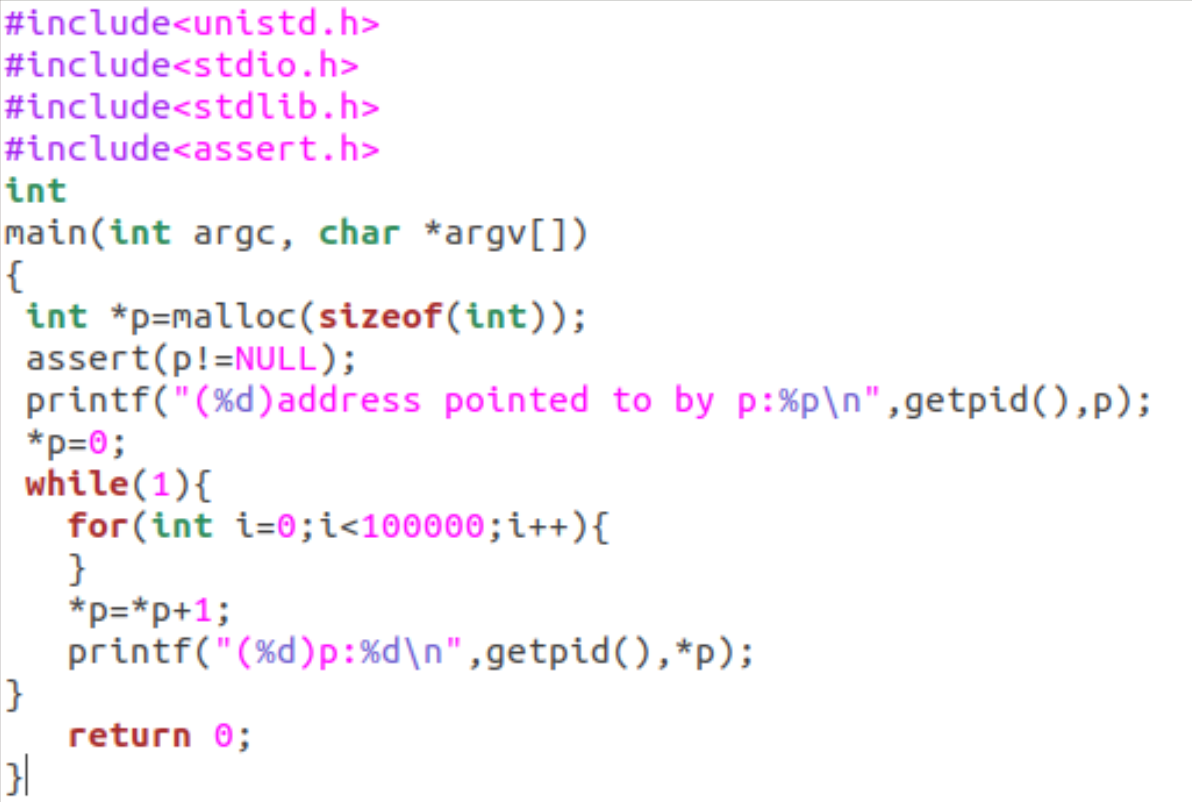


当传入参数A的时候，每隔一段循环时间，就会输出这个字符A

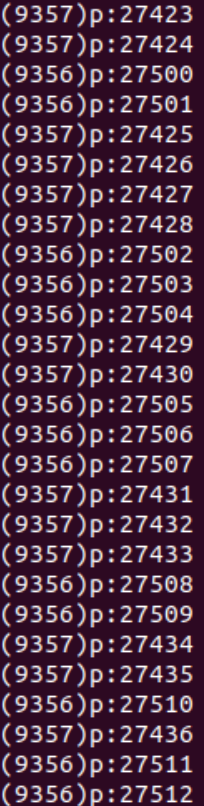
/Users/Nie/Desktop/Screen Shot 2019-03-12 at 8.26.43 AM.png

当运行指令 ./cpu A & ./cpu B & ./cpu C & ；时，四个程序分别使用参数ABCD开始运行，操作系统在硬件的帮助下负责程序的并发执行，让用户感受到CPU的虚拟化。将单个CPU转换为看似无限数量的CPU，看起来像是一起在运行，称为虚拟化。

1. 内存分配实验



当运行单个mem程序的时候，内存是顺序分配的。下面看两个mem程序运行的情形。

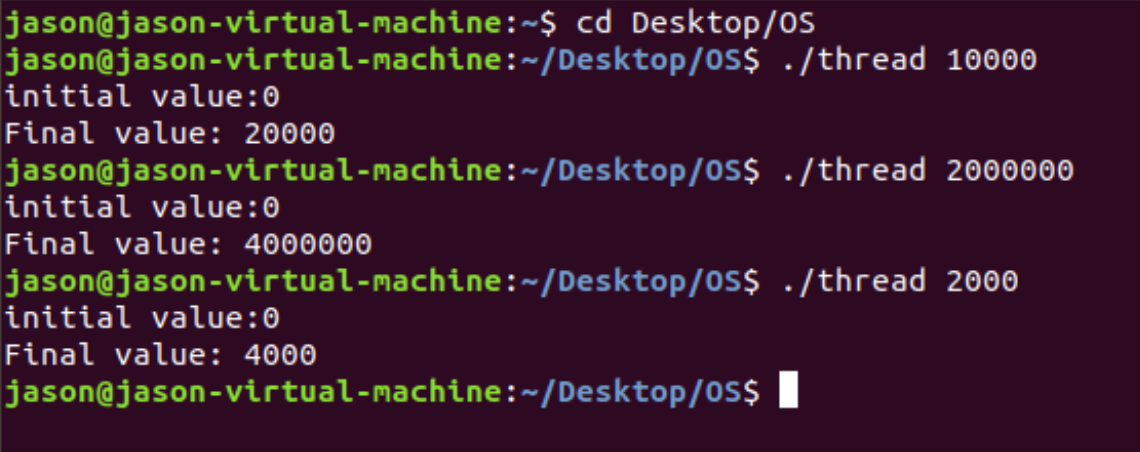


分配的地址不同，但似乎使用了同一块物理内存。

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。究竟如何实现所有这些也是本书第一部分关于虚拟化主题的内容。

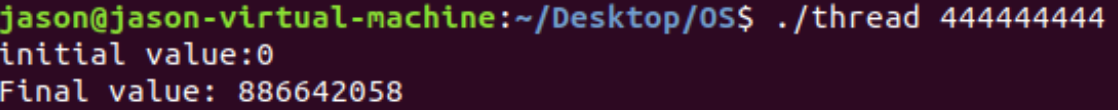
1. 共享





最终值是两倍输入值

final最终值不是两倍输入值



这是一个多线程的程序，主程序使用pthread.create()创建两个线程。您可以将线程视为在与其它函数相同的内存空间中运行的函数，其中一次激活多个函数。在这个程序中，每个线程开始在一个名为worker的例程中运行，在该例程中，它只是循环递归一个统计循环次数的计数器。

共享计数器递增，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原子地执行，所以有时会出现异常。