实验二实验报告

**江学强，PB16120100**

1. **阶段一：实现三个系统调用：fork、exec、join**

添加系统调用得一般方法与实验一相同，不再赘述，只叙述实现的过程和遇到的一些问题，下面对三个系统调用分别叙述。

**fork系统调用**：fork系统调用的功能是创建一个新的子进程，子进程继承父进程的代码、内存空间和寄存器状态，fork系统调用实现就是先要保存这些内容到子进程中，之后把子进程放到就绪队列，下面看nachos的实现。

实现fork系统调用的第一步就是新建一个Thread类作为子进程类，这里需要说明一下 我是怎么实现每个进程唯一的tid的，我在Thread类中定义了一个静态的unsigned int变量tidGenerate，初始化为0，在构造函数中把tidGenerate赋值给tid，并让tidGenerate递增，这样就实现了每个进程独特的tid。此外我在构造函数中把parent变量赋值为 kernel->currentThread,即完成了父子进程关系的联系以及把新建的队列加入到kernel的进程队列中，调用kernel->addThread。子进程新建后做一些保存的工作：保存内存空间，这里要注意保存之前要新建一个AddrSpace类实例给新建的对象，之后调用AddrSpace的类方法CopyMemory，之后调用SaveUserState保存寄存器的值，这些工作做完之后调用Thread类方法Fork，参数分别是一个函数指针，这个函数指针是处理子进程执行前的就绪过程的，就是forked函数，以及新建的Thread类。Fork方法的用法我是在网上看了一篇调试Nachos的博客中学习了用gdb调试Nachos来单步跟踪提供的程序知道的用法，事实上这是fork系统调用实现的最难的一个点。

**exec系统调用**：主要过程就是把执行的程序加载到内存并执行，下下面看nachos的实现。

exec的实现比较直接，情况内存空间调用AddrSpace->reset()方法即可，之后再根据文件名把程序加载到内存，调用 AddrSpace->Load(filename)即可，如果加载成功则执行，否则结束当前进程。

**Join系统调用**：join的实现个人感觉稍微绕一些，需要理解信号量的作用，下面具体叙述。

父进程调用join方法，本质上就是等待相应的子进程结束之后再从join中跳出来，也就是把父进程阻塞在join中等待相应的子进程结束。这个过程父子进程是通过信号量保持同步的，先判断这个子进程是不是在内核中，若在，则在join中新建一个SemaPhore类，信号量的值初始化为0 ，然后把这个信号量插入到父进程的信号量队列中，之后进行父进程进行p操作，阻塞，知道子进程结束时在其Finish方法中调用相应信号量的v操作才让父进程结束p操作并跳出，子进程已经结束，父进程删除相应信号量。过程描述的很详细，其中调用的函数或者方法都很简单直观，不再赘述。

三个系统调用的叙述中都未叙述中断处理程序的具体内容，因为这部分内容与实验一相同，都是在寄存器中获取参数，之后调用相关的系统调用或者函数，最后写回结果，故未赘述。

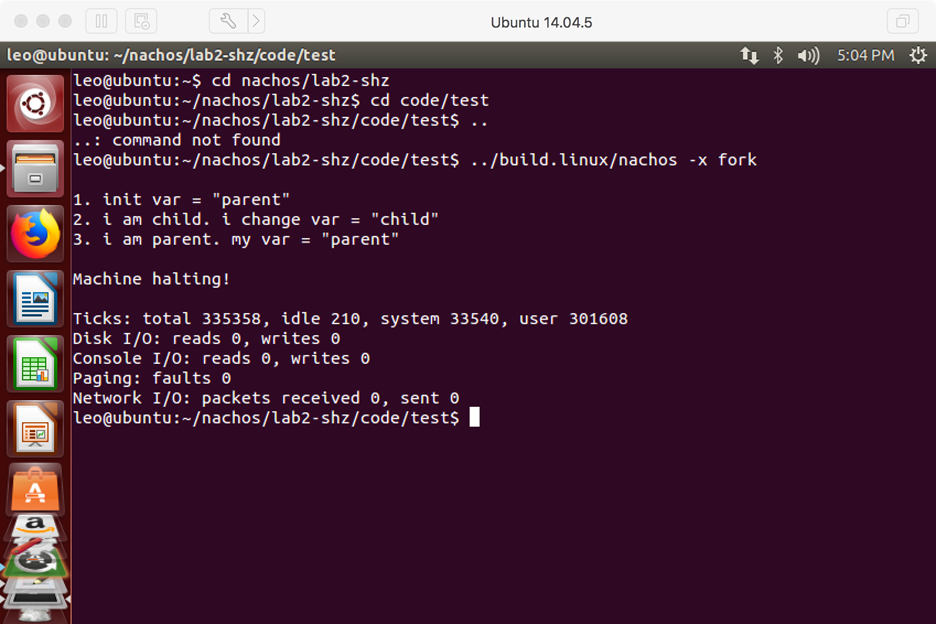
1. **阶段二：编写一个简单的shell并实现进程动态优先级调度**

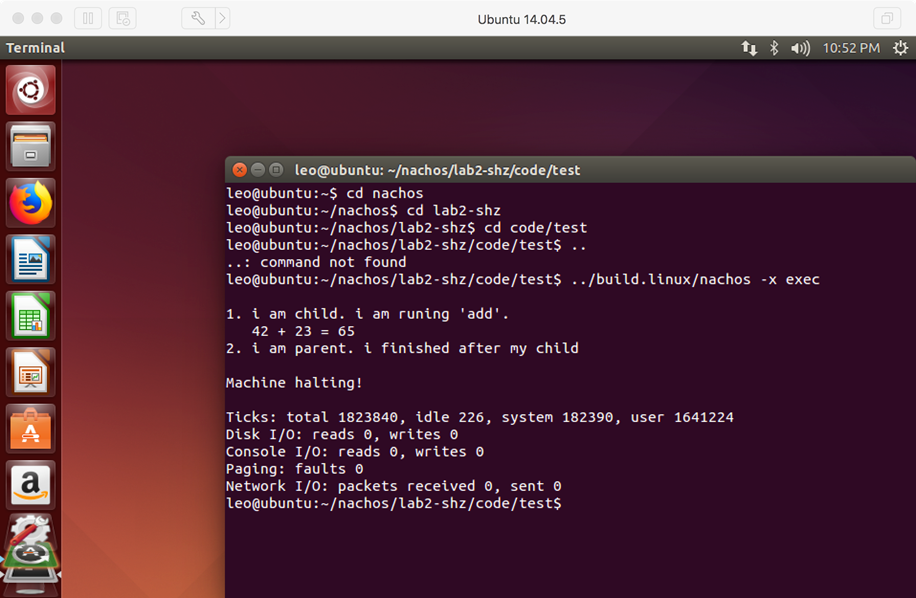
**shell的编写：**shell的编写助教已经做了绝大部分工作，如获取指令，分割指令等。

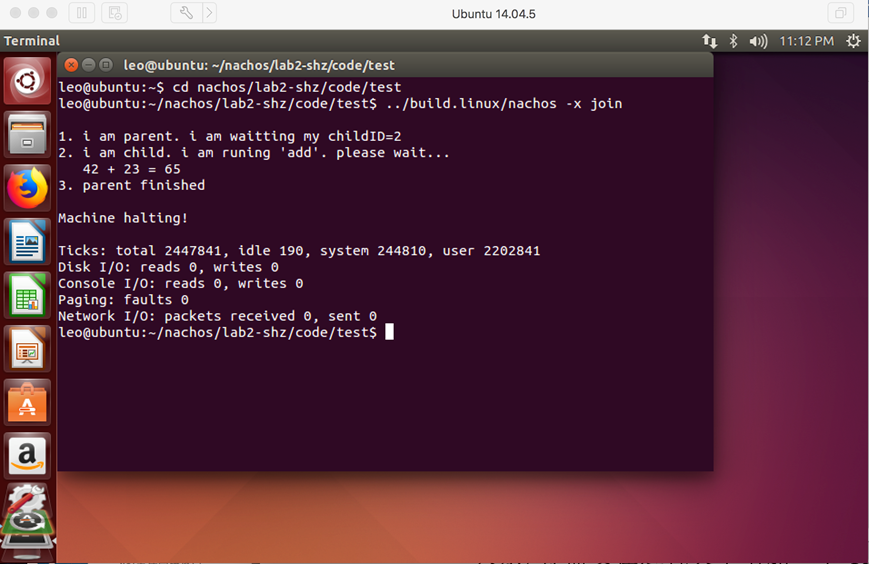
分割后的指令放在cmdLine数组中，指令数位cmdNum，实现shell是调用fork建立cmdNum个shell的子进程，之后再在fork返回的子进程（即返回值为0）中调用exec执行cmdLine，注意这里都是循环执行，在fork返回的父进程（即返回值不为0）中调用join，即等待子进程所以的死进程，亦需要循环调用。

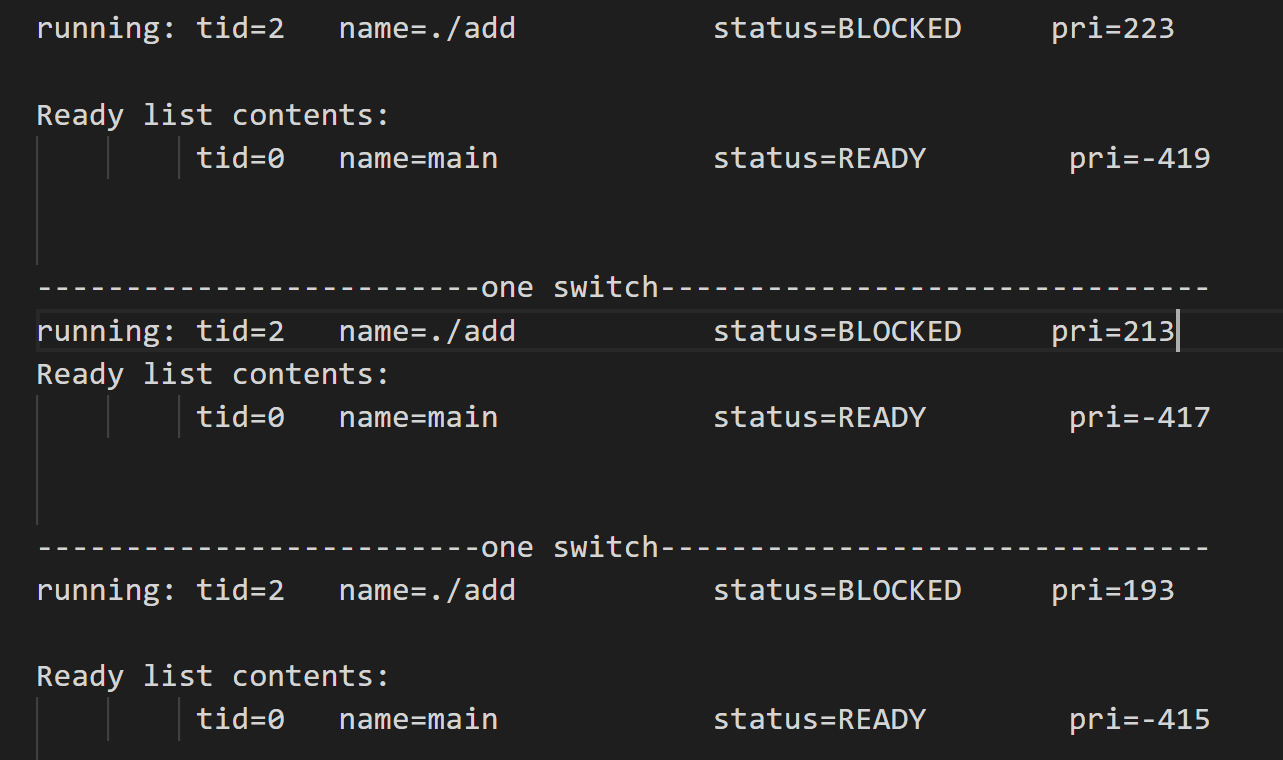
**动态优先级调度**：我们需要编写的函数是Schedule类的findNextToRun方法，方法调用flushpriority函数动态更新所有就绪队列的优先级以及正在执行的进程的优先级，更新的数值规定助教已经规定，当前进程优先级priority=priority -距离上次调度间隔/10，距离上次调度时间距离是总的时间 kernel->stats->totalTick,上次调度时间，kernel->schedual->lastSwitchTick,对就绪队列中的进程是每个进程的优先级加上AdaptPace常数，方法是写了一个接受进程的函数flush，在函数中将优先级加上AdaptPace，之后在flushpriority中调用readyList->Apply(flush)，对每个进程进程操作。优先级调整后，需要判断是否满足调度条件，即就绪队列不空并且距离上次调度时间不过短，一个简单的判断即可，若不满足调度条件，返回空，即继续执行原来的进程，否则用就绪队列中优先级最大的（front）的优先级与当前进程的优先级比较，若就绪队列中大，则进行上下文弹出front并返回，返回之前要更新调度时间lastSwitchTick，否则返回空。

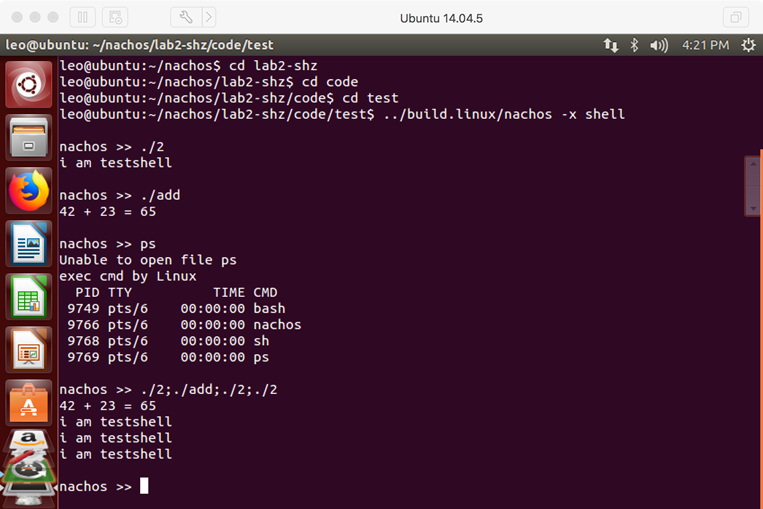
**3． 实验结果：**

****









**4.实验感想与收获**

这次实验由于助教做了大量的工作所以这次实验的代码编写不难，但是要理清整个进程的内容还是有些难度，我的主要时间都花在了阅读nachos源码中，对每个类在某个阶段的作用都掌握了（自我感觉），所以虽然代码量不大，但是收获还是很大的。