操作系统实验报告

Lab3

姓名：刘博

学号：141220065

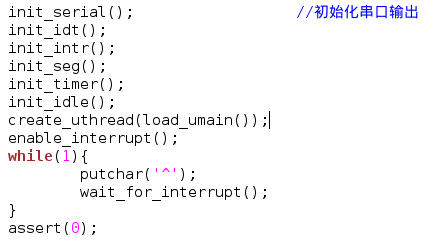
计算机科学与技术系

2016.4.23

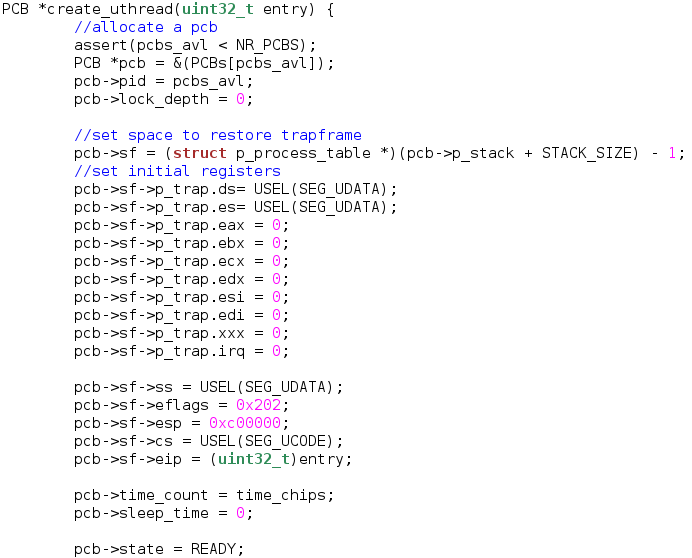
邮箱：1610266604@qq.com

1. **实验目的：**
2. 添加时钟中断
3. 加载用户程序
4. 初始化用户进程
5. 开中断
6. 从时钟中断调度到用户进程中
7. 用户进程调用fork（）系统调用创建子进程
8. 利用时钟中断调度父子进程
9. 一段时间后利用exit（）结束进程，返回内核进程
10. **实验过程：**

首先将kernel的函数进行更新，初始化time时钟中断，初始化用户pcb，打开中断，并从时钟中断进入用户函数；

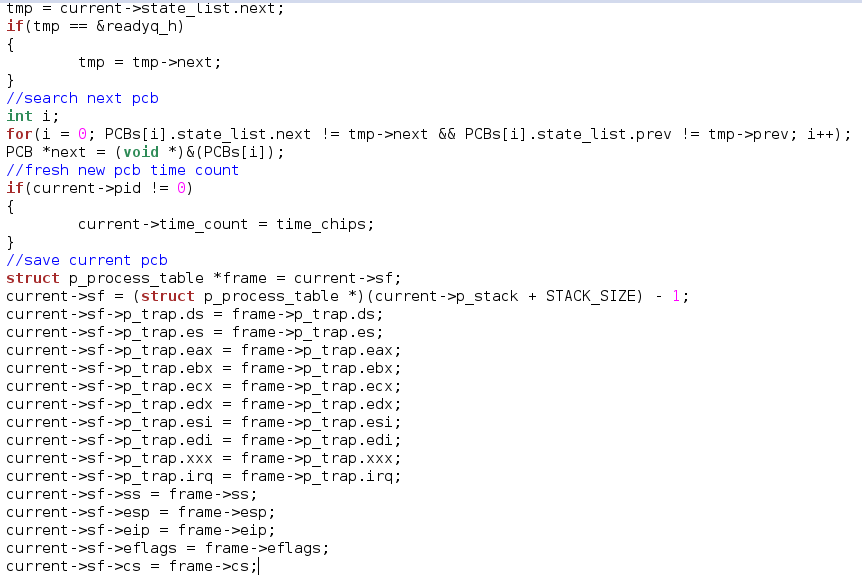


其中load\_umain（）函数是用来加载用户代码（与lab2一致），其中create\_uthread（）函数用来创建用户的pcb，函数如下：

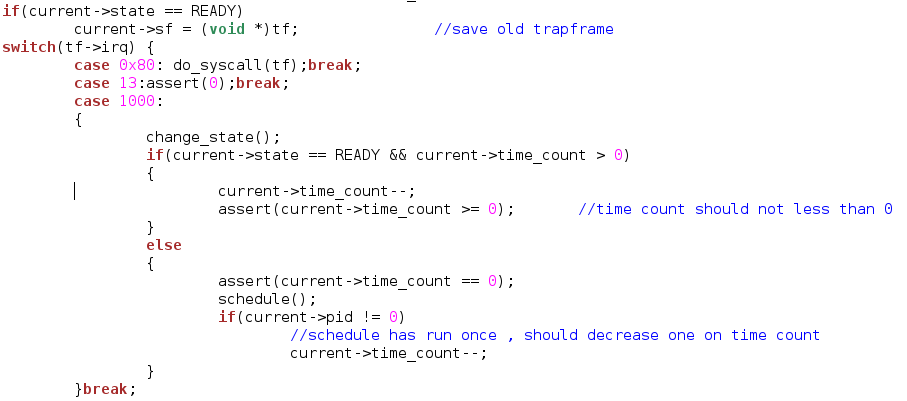


将所用的pcb中的通用寄存器以及栈帧初始化（注意eflags要初始化成0x202（打开if位以便检测时钟中断的到来），将eip赋值成为elf->entry，以便函数在中断返回时可以成功跳入用户进程代码处；

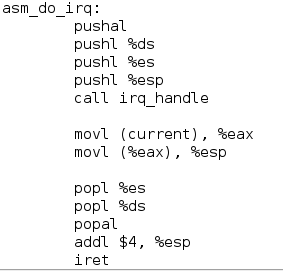
在等待时钟中断时，cpu会接受到第一个时钟中断，然后判断其状态，如果是ready，那么可以判断其时间片是否到期，如果该进程时间片到期，则可以进行调度算法；



如果需要进行调度，则可以直接将current当前的状态保存到当前的pcb中，然后将current赋值更新成为就绪队列的下一个pcb；



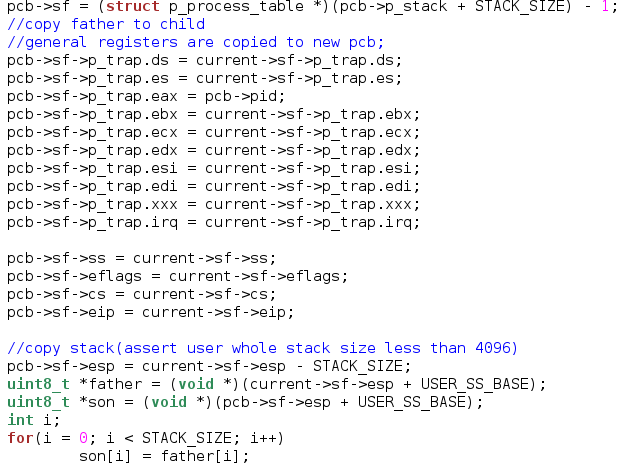
判断当前是否需要调度；



更改完current指针后，就可以将current中的陷阱帧结构与原来的tf结构进行替换，这样原来的进程就会通过iret指令跳转到下一个进程的现场，并且执行下一个进程的代码；这样就完成了基本的进程切换；

Fork（）：

Fork函数本身并不改变current的值，所以fork函数执行过后进程没有切换；所以forkj函数仅仅是创建了一个新的pcb以便调用；所以该过程与原来的创建pcb的过程相似，所以只是赋值以便后续调用；

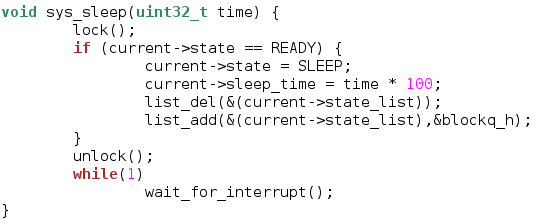


注意fork函数应该复制一个与原来进程一模一样的pcb，所以除了返回值eax之外，所有的pcb值都应该与原来的pcb一样，而且注意要将用户进程的栈帧复制到另外一个地址中；

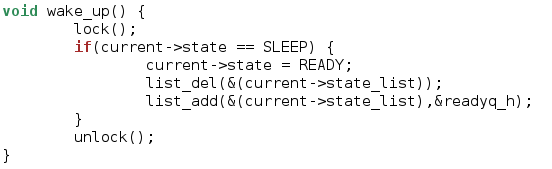
Sleep（）：

Sleep函数是用来对当前进程进行阻塞的，调用sleep函数后，程序进入中断，并且在中断当中停留一定时间后才可以继续执行，注意这个时候cpu是被占用的，不可以进行其他进程的切换和抢占；

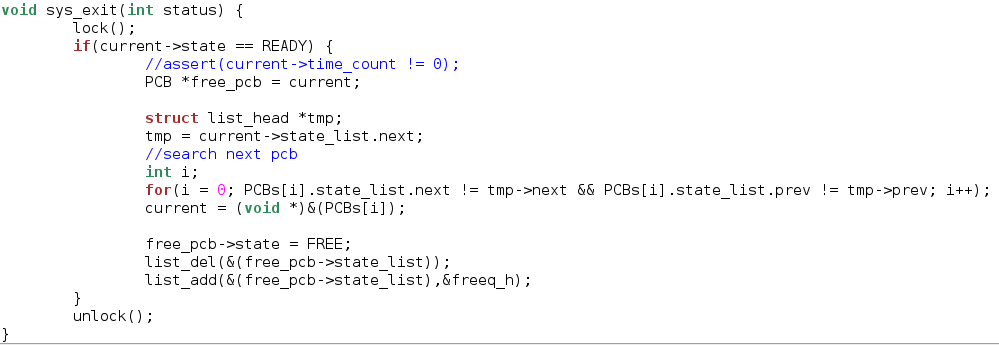
（sleep函数进入中断后不到一点的时间是不能返回的）



这里利用while（1）进行无限循环等待时钟中断，时钟中断可以在某一个时刻调用wake\_up函数来对阻塞的sleep程序进行唤醒；



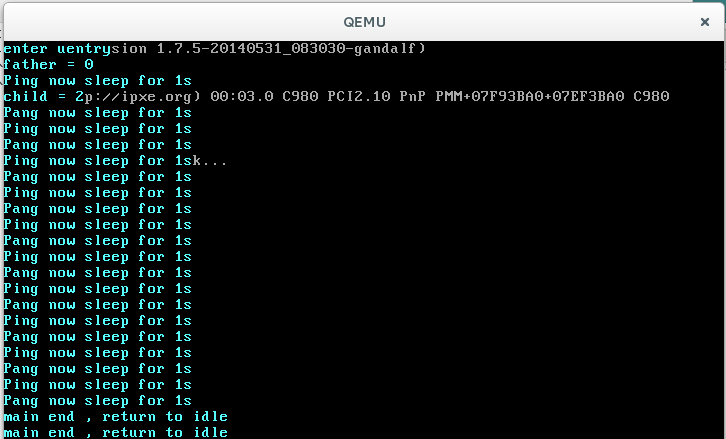
在sleep和wake\_up函数进行时应该关闭外部中断的响应；



Exit函数与此类似，当前进程如果调用exit函数，则将当前进程的pcb从就绪队列中删除，并且将current指针向后偏移，使其进行下一个程序的执行和调度；

1. **实验结果：**

当每个进程时间片为1，循环10次，每次停留1s时：



此时可以看到ping和pang循环出现，也就是父子进程循环进行

1. **实验疑问：**

对于tss中的esp0决定了每个进程中断时的陷阱帧的位置，那么如果不会进行嵌套中断时，是否不需要进行esp0的切换，由于每个pcb都有专用的内核栈，所以是否只要将中断时的陷阱帧存放到pcb中即可？这样会不会出现问题；

对于段寄存器，是否需要ldt，在实验中，可以通过固定栈帧大小来确定拷贝栈帧的位置，而实际在linux中，我们需要ldt表来进行不同进程的地址切换，所以在本次试验中是否也应该通过ldt表来进行加载？