练习**1**：分配并初始化一个进程控制块（需要编码）

alloc\_proc函数（位于kern/process/proc.c中）负责分配并返回一个新的struct proc\_struct结

构，用于存储新建立的内核线程的管理信息。ucore需要对这个结构进行最基本的初始化，你

需要完成这个初始化过程。

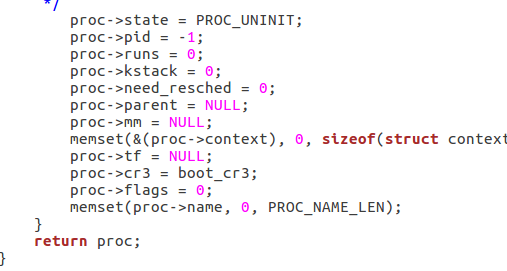
答：

本练习的编码工作集中在proc.c中的alloc\_proc函数中，该函数的具体含义为创建一个新的进程控制块，并且对控制块中的所有成员变量进行初始化，根据实验指导书中的要求，除了指定的若干个成员变量之外，其他成员变量均初始化为0，取特殊值的成员变量如下所示：

PROC->STATE=PROC\_UNINIT

Proc->PID=-1

PROC->CR3=BOOT\_CR3



请说明proc\_struct中 struct context context 和 struct trapframe \*tf 成员变量含义和在

本实验中的作用是啥？（提示通过看代码和编程调试可以判断出来）

答：

struct context context

在结构体中存储这除了eax之外的所有通用寄存器以及eip的数值，这就提示我们这个线程控制块中的context很有可能是保存的线程运行的上下文信息；context变量的意义就在于内核线程之间进行切换的时候，将原先的线程运行的上下文保存下来.由于eax是一个caller-save寄存器，并且在函数里eax的数值一直都可以在栈上找到对应，因此没有比较对其进行保存。

struct trapframe \*tf

tf变量的作用在于在构造出了新的线程的时候，如果要将控制权交给这个线程，是使用中断返回的方式进行的（跟lab1中切换特权级类似的技巧），因此需要构造出一个伪造的中断返回现场，也就是trapframe，使得可以正确地将控制权转交给新的线程；具体切换到新的线程的做法为，调用switch\_to函数，然后在该函数中进行函数返回，直接跳转到forkret函数，最终进行中断返回函数\_\_trapret，之后便可以根据tf中构造的中断返回地址，切换到新的线程了；

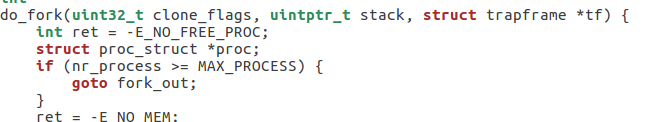
练习**2**：为新创建的内核线程分配资源（需要编码）

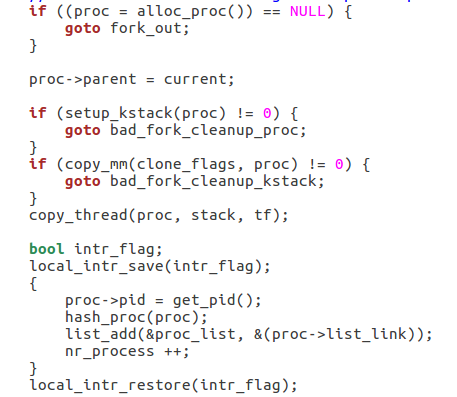
创建一个内核线程需要分配和设置好很多资源。kernel\_thread函数通过调用**do\_fork**函数完成

请在实验报告中简要说明你的设计实现过程。请回答如下问题：

答：

在本次练习中，主要需要实现的代码位于proc.c的do\_fork函数中，该函数的语义为为内核线程创建新的线程控制块，并且对控制块中的每个成员变量进行正确的设置，使得之后可以正确切换到对应的线程中执行；接下来将结合具体的代码来说明本次练习的具体实现过程





请说明ucore是否做到给每个新fork的线程一个唯一的id？请说明你的分析和理由。

可以。ucore中为新的fork的线程分配pid的函数为get\_pid在该函数中使用到了两个静态的局部变量next\_safe和last\_pid，根据命名推测，在每次进入get\_pid函数的时候，这两个变量的数值之间的取值均是合法的pid（也就是说没有被使用过），这样的话，如果有严格的next\_safe > last\_pid + 1，那么久可以直接取last\_pid + 1作为新的pid（需要last\_pid没有超出MAX\_PID从而变成1）； 如果在进入函数的时候，这两个变量之后没有合法的取值，也就是说next\_safe > last\_pid + 1不成立，那么进入循环，在循环之中首先通过if (proc->pid == last\_pid)这一分支确保了不存在任何进程的pid与last\_pid重合，然后再通过if (proc->pid > last\_pid && next\_safe > proc->pid)这一判断语句保证了不存在任何已经存在的pid满足：last\_pid<pid<next\_safe，这样就确保了最后能够找到这么一个满足条件的区间，获得合法的pid； 之所以在该函数中使用了如此曲折的方法，维护一个合法的pid的区间，是为了优化时间效率，如果简单的暴力的话，每次需要枚举所有的pid，并且遍历所有的线程，这就使得时间代价过大，并且不同的调用get\_pid函数的时候不能利用到先前调用这个函数的中间结果；

练习**3**：阅读代码，理解 **proc\_run** 函数和它调用的函数如何完成

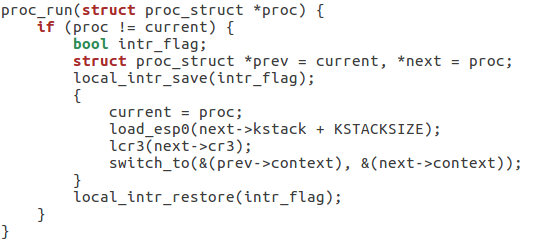
进程切换的。（无编码工作）

请在实验报告中简要说明你对proc\_run函数的分析

答：

**在本实验框架中，唯一调用到这个函数是在线程调度器的schedule函数中，也就是可以推测proc\_run的语义就是将当前的CPU的控制权交给指定的线程；**proc\_run中首先进行了TSS以及cr3寄存器的设置，然后调用到了swtich\_to函数来切换线程，根据上文中对switch\_to函数的分析可以知道，在调用该函数之后，首先会恢复要运行的线程的上下文，然后由于恢复的上下文中已经将返回地址（copy\_thread函数中完成）修改成了forkret函数的地址(如果这个线程是第一运行的话，否则就是切换到这个线程被切换出来的地址)，也就是会跳转到这个函数，最后进一步跳转到了\_\_trapsret函数，调用iret最终将控制权切换到新的线程；

函数如下：



在本实验的执行过程中，创建且运行了几个内核线程？

答：

两个

语句 local\_intr\_save(intr\_flag);....local\_intr\_restore(intr\_flag); 在这里有何作用?请

说明理由

答：

该语句的左右是关闭中断，使得在这个语句块内的内容不会被中断打断，是一个原子操作； 这就使得某些关键的代码不会被打断，从而不会一起不必要的错误；

完成代码编写后，编译并运行代码：make qemu

