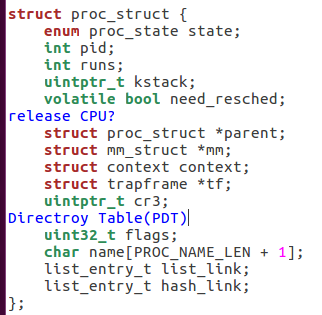
练习1：

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。

【提示】在alloc\_proc函数的实现中，需要初始化的proc\_struct结构中的成员变量至少包括： state/pid/runs/kstack/need\_resched/parent/mm/context/tf/cr3/flags/name。

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：

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）



context：进程的上下文，用于进程切换（参见switch.S）。在ucore中，所有的进程在内核中也是相对独立的（例如独立的内核堆栈以及上下文等）。使用context保存寄存器的目的就在于在内核态中能够进行上下文之间的切换。实际利用context进行上下文切换的函数是在kern/process/switch.S中定义的switch\_to。

tf：中断帧的指针，**总是指向内核栈的某个位置**：当进程从用户控件跳到内核空间时，中断帧记录了进程在被中断前的状态。当内核需要跳回用户空间时，需要调整中断帧以恢复让进程继续执行的各寄存器的值。

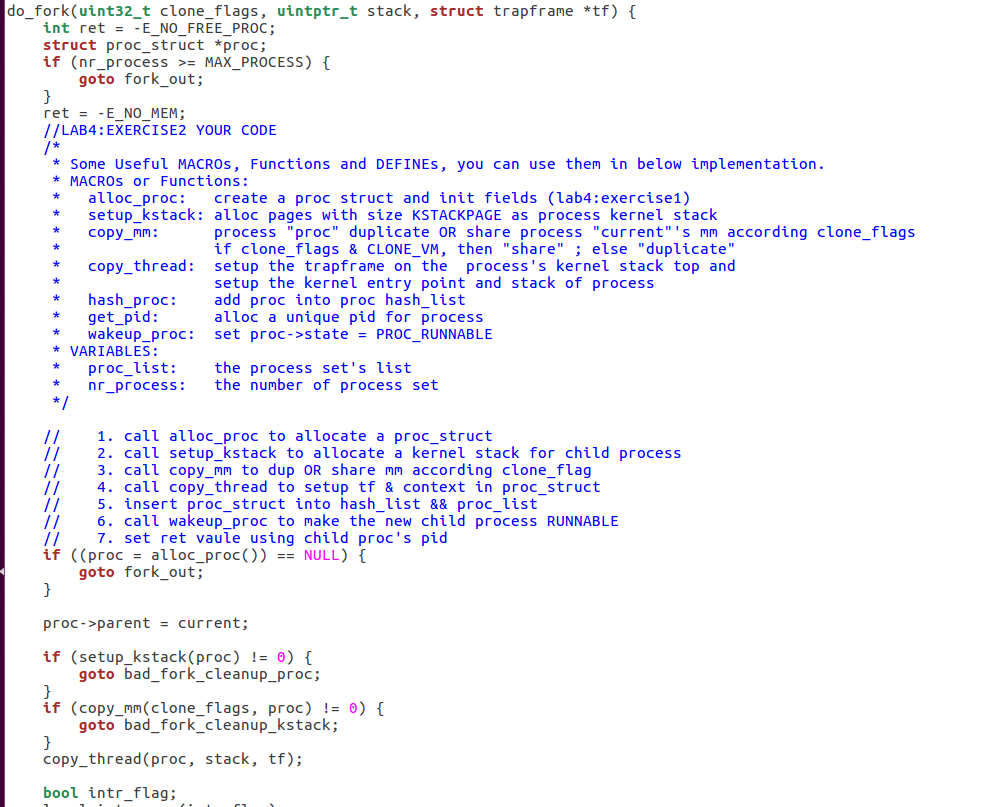
练习2：

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函数完成具体内核线程的创建工作。 do\_kernel函数会调用alloc\_proc函数来分配并初始化一个进程控制块，但alloc\_proc只是找到了一小块内存用以记录进程的必 要信息，并没有实际分配这些资源。ucore一般通过do\_fork实际创建新的内核线程。do\_fork的作用是，创建当前内核线程的 一个副本，它们的执行上下文、代码、数据都一样，但是存储位置不同。在这个过程中，需要给新内核线程分配资源，并且 复制原进程的状态。你需要完成在kern/process/proc.c中的do\_fork函数中的处理过程。它的大致执行步骤包括：

调用alloc\_proc，首先获得一块用户信息块。 为进程分配一个内核栈。 复制原进程的内存管理信息到新进程（但内核线程不必做此事） 复制原进程上下文到新进程 将新进程添加到进程列表 唤醒新进程 返回新进程号

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：

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。

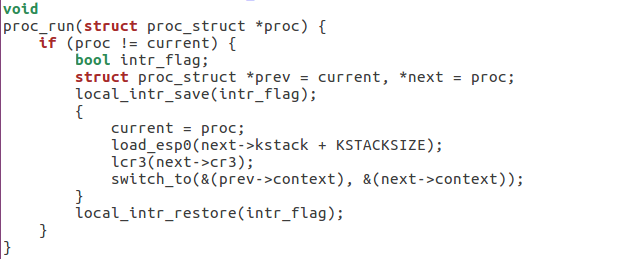


新进程的id是通过get\_pid来获得的，通过查看该函数代码，发现它通过last\_pid和next\_safe两个变量来取得一个正确的id，我们可以发现这样进出比较和++处理之后，返回的last\_

Pid都是唯一的没有重复的。

练习3：

请在实验报告中简要说明你对proc\_run函数的分析。并回答如下问题:在本实验的执行过程中,创建且运行了几个内核线程?语句local\_intr\_save(intr\_flag);....local\_intr\_restore(intr\_flag);在这里有何作用?请说明理由。



共运行了两个线程，两个函数的作用是关闭中断