实验四：内核线程管理

1. 实验目的
2. 了解内核线程创建/执行的管理过程
3. 了解内核线程的切换和基本调度过程
4. 实验内容

实验2/3完成了物理和虚拟内存管理，这给创建内核线程（内核线程是一种特殊的进程）打下 了提供内存管理的基础。当一个程序加载到内存中运行时，首先通过ucore OS的内存管理子 系统分配合适的空间，然后就需要考虑如何分时使用CPU来“并发”执行多个程序，让每个运行 的程序（这里用线程或进程表示）“感到”它们各自拥有“自己”的CPU。

本次实验将首先接触的是内核线程的管理。内核线程是一种特殊的进程，内核线程与用户进 程的区别有两个：

内核线程只运行在内核态

用户进程会在在用户态和内核态交替运行

所有内核线程共用ucore内核内存空间，不需为每个内核线程维护单独的内存空间

而用户进程需要维护各自的用户内存空间

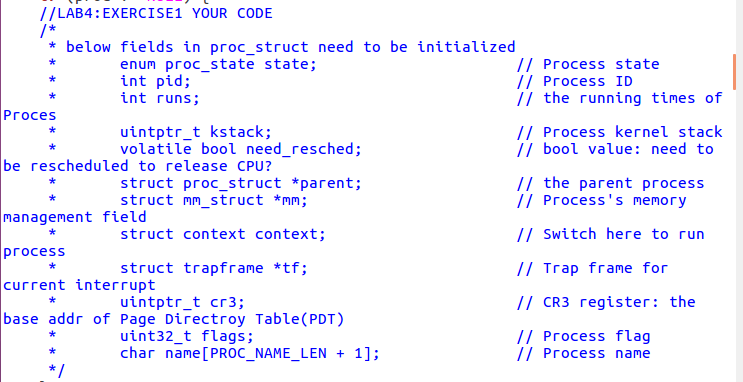
1. 练习

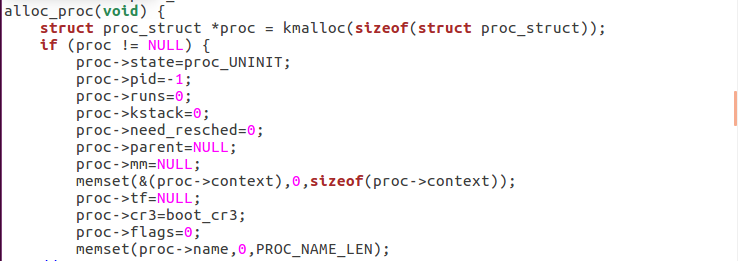
对实验报告的要求：

1. 基于markdown格式来完成，以文本方式为主
2. 填写各个基本练习中要求完成的报告内容
3. 完成实验后，请分析ucore\_lab中提供的参考答案，并请在实验报告中说明你的实现与参考答案的区别
4. 列出你认为本实验中重要的知识点，以及与对应的OS原理中的知识点，并简要说明你对 二者的含义，关系，差异等方面的理解（也可能出现实验中的知识点没有对应的原理知 识点）
5. 列出你认为OS原理中很重要，但在实验中没有对应上的知识点

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





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）

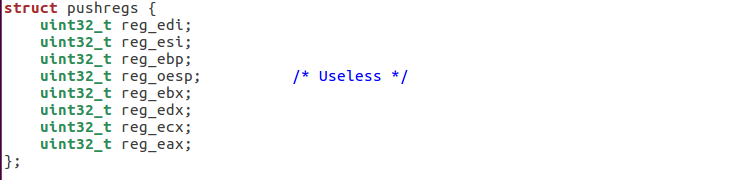
Context的作用:

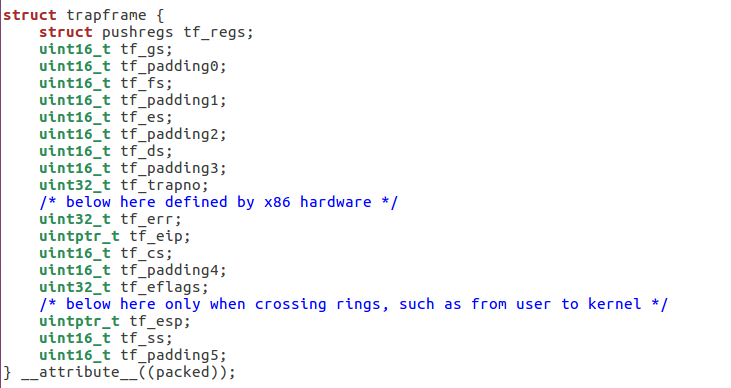
在ucore中，所有的进程在内核也是相对独立的。使用context保存寄存器的目的就在于在内核态中能够进行上下文之间的切换。



tf变量的作用：

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，因此需要构造出一个伪造的中断返回现场，也就trapframe，使得可以正确地将控制权转交给新的线程





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 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

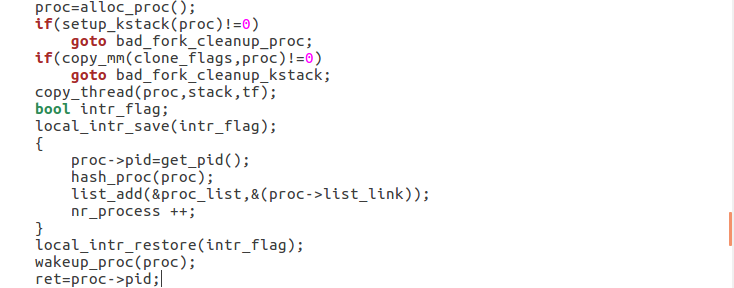
将新进程添加到进程列表

唤醒新进程

返回新进程号

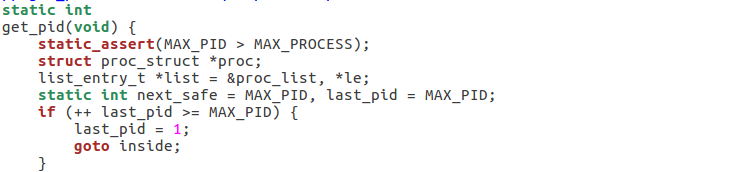
请在实验报告中简要说明你的设计实现过程。请回答如下问题：

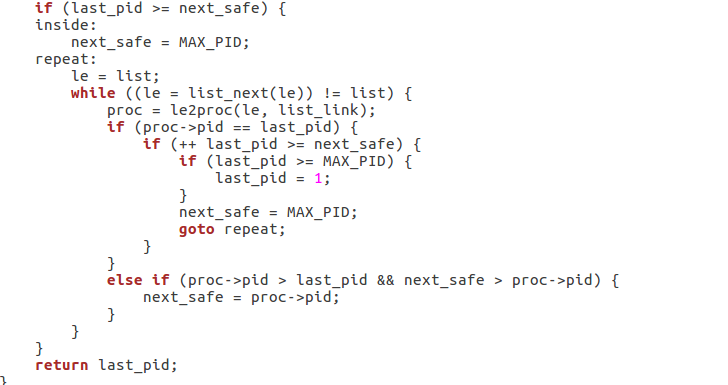
代码：



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。

找到get\_pid函数:

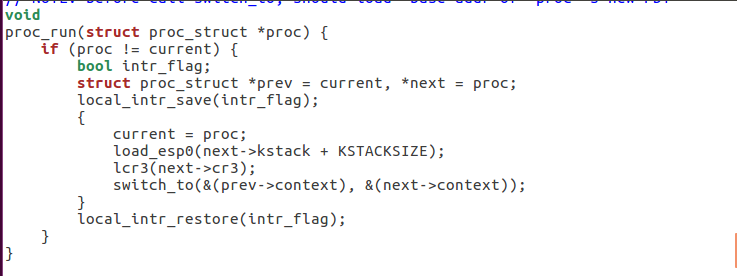




**可以看到get\_pid函数有做规避重复的措施，因此只要get\_pid互斥，就可以保证pid唯一。**

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）

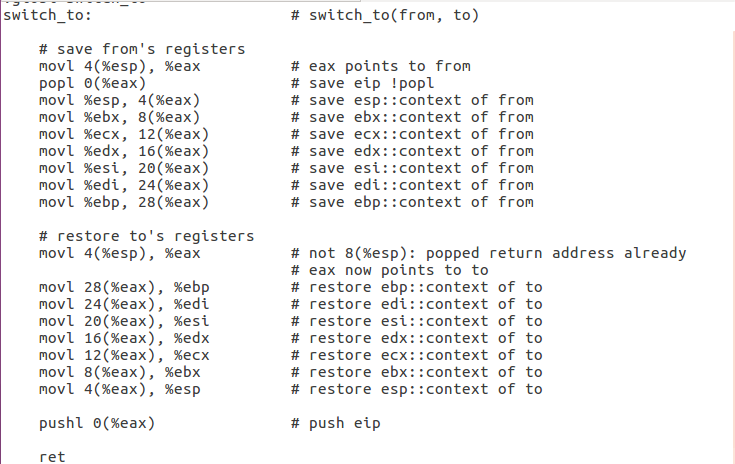
请在实验报告中简要说明你对proc\_run函数的分析。并回答如下问题：



**Pro\_run用于使一个线程在CPU中运行，可以看到它的主要过程为：**

1. **将当前进程设为传入的进程**
2. **修改esp指针的值**
3. **修改页表项**
4. **使用switch\_to进行上下文切换**

**Switch\_to:**



**首先把当前寄存器的值送到原线程的context中保存，再将新线程的context赋值各寄存器。**

在本实验的执行过程中，创建且运行了几个内核线程？

**两个内核线程：**

**创建第0个内核线程idleproc。**在 init.c::kern\_init 函数调用了 proc.c::proc\_init 函数。 proc\_init 函数启动了创建内核线程的步骤。首先当前的执行上下文（从 kern\_init 启动至今）就可以看成是 uCore 内核（也可看做是内核进程）中的一个内核线程的上下文。为此，uCore 通过给当前执行的上下文分配一个进程控制块以及对它进行相应初始化，将其打造成第 0 个内核线程 – idleproc。

**创建第 1 个内核线程 initproc。**第 0 个内核线程主要工作是完成内核中各个子系统的初始化，然后就通过执行 cpu\_idle 函数开始过退休生活了。所以 uCore 接下来还需创建其他进程来完成各种工作，但 idleproc 内核子线程自己不想做，于是就通过调用 kernel\_thread 函数创建了一个内核线程 init\_main。在实验四中，这个子内核线程的工作就是输出一些字符串，然后就返回了（参看 init\_main 函数）。但在后续的实验中，init\_main 的工作就是创建特定的其他内核线程或用户进程

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

**关闭中断和打开中断。有些过程是互斥的，只允许一个线程进入，因此需要关闭中断来处理临界区。**

