**Lab4：内核线程管理**

实验目的：

了解内核线程创建/执行的管理过程

了解内核线程的切换和基本调度过程

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。

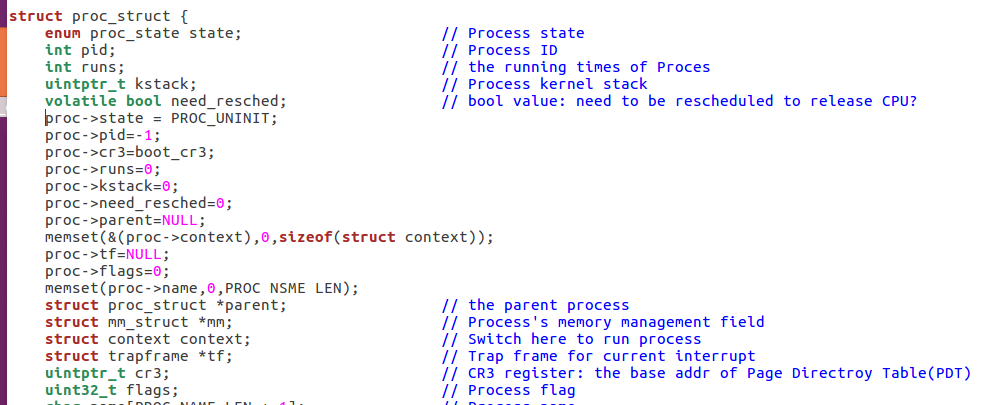
本次实验将首先接触的是内核线程的管理。内核线程是一种特殊的进程，内核线程与用户进 程的区别有两个：

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

实验结果

练习1：

补全代码如下：



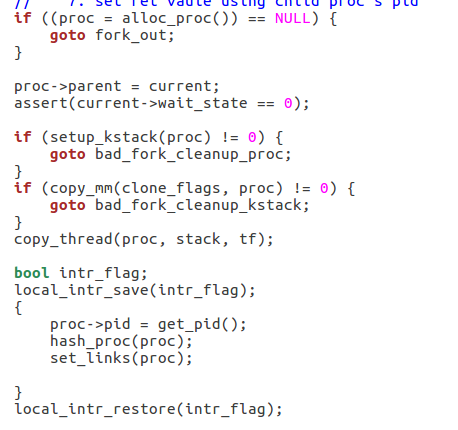
问题：请说明proc\_struct中 struct context context 和 struct trapframe \*tf 成员变量含义和在 本实验中的作用是啥？（提示通过看代码和编程调试可以判断出来）

答案：context：进程的上下文，用于进程切换。起到的作用就是保存了现场。在 ucore中，所有的进程在内核中也是相对独立的，因此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switch\_to。

② tf：中断帧的指针，总是指向内核栈的某个位置：当进程从用户空间跳到内核空间时，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值。除此之外，ucore内核允许嵌套中断。因此为了保证嵌套中断发生时tf 总是能够指向当前的tf，ucore 在内核栈上维护了 tf 的链。

练习2：

设计实现：



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。

答案：可以做到，在使用 fork 或 clone 系统调用时产生的进程均会由内核分配一个新的唯一的PID值。具体来说，就是在分配PID时，设置一个保护锁，暂时不允许中断，这样在就唯一地分配了一个PID。

练习3：

Make qemu

