# Lab2

**练习1：**

context和\*tf的作用分析

①context：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 为新创建的内核线程分配资源**

**1.**第一步：申请内存块，如果失败，直接返回处理

第二步：为进程分配一个内核栈

第三步：复制父进程的内存信息到子进程

第四步：复制父进程相关寄存器信息（上下文）

第五步：将新进程添加到进程列表（此过程需要加保护锁）

第六步：一切准备就绪，唤醒子进程

第七步：设置返回的子进程号

**2.**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，保证了ID的唯一性。上述操作真正完成了资源分配的工作，与第一步中的工作有着明显的区别。do\_fork只是创建当前进程的副本，他们执行的上下文，寄存器，代码都是一样的。

**练习3 理解proc\_run和它调用的函数如何完成进程切换的**

通过proc\_run和进一步的switch\_to函数完成两个执行现场的切换，具体流程如下：

①让current指向next内核线程initproc；

②设置任务状态段ts中特权态0下的栈顶指针esp0为next内核线程initproc的内核栈的栈顶，即next->kstack + KSTACKSIZE ；

③设置CR3寄存器的值为next内核线程initproc的页目录表起始地址next->cr3，这实际上是完成进程间的页表切换；

由switch\_to函数完成具体的两个线程的执行现场切换，即切换各个寄存器，当switch\_to函数执行完“ret”指令后，就切换到initproc执行