操作系统第三次实验报告

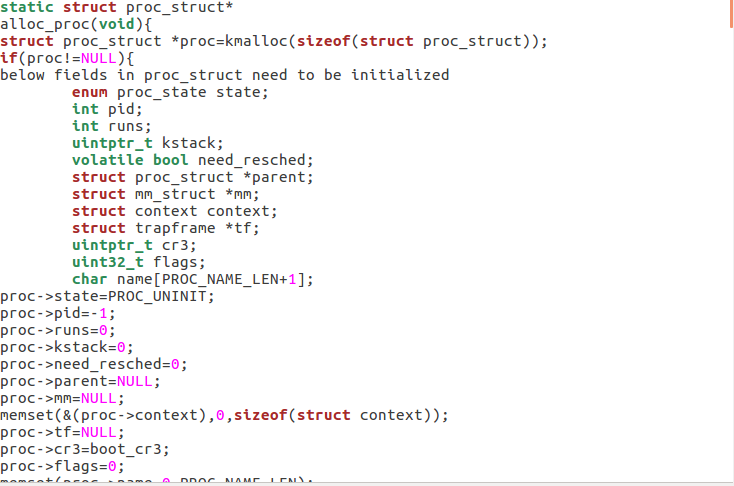
练习一：分配并初始化一个进程控制块

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，而用户进程会在在用户态和内核态交替运行； 所有内核线程直接使用共同 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。 这里主要是从 kern\_init 函数的物理内存管理初始化开始的，按照函数的次序 做了一个简单的总结： 1、pmm\_init() (1) 初始化物理内存管理器。 (2) 初始化空闲页，主要是初始化物理页的 Page 数据结构，以及建立页目录表 和页表。 (3) 初始化 boot\_cr3 使之指向了 ucore 内核虚拟空间的页目录表首地址，即 一级页表的起始物理地址。 (4) 初始化第一个页表 boot\_pgdir。 (5) 初始化 GDT，即全局描述符表。 2、pic\_init() 初始化 8259A 中断控制器 3、idt\_init() 初始化 IDT，即中断描述符表 4、vmm\_init() 主要就是实验了一个 do\_pgfault() 函数达到页错误异常处理功能，以及虚拟内 存相关的 mm,vma 结构数据的创建/销毁/查找/插入等函数 5、proc\_init() 这个函数启动了创建内核线程的步骤，完成了 idleproc 内核线程和 initproc 内核线程的创建或复制工作，这是本次实验分析的重点，后面将详细分析。 6、ide\_init() 完成对用于页换入换出的硬盘(简称 swap 硬盘)的初始化工作 7、swap\_init() swap\_init() 函 数 首 先 建 立 完 成 页 面 替 换 过 程 的 主 要 功 能 模 块 ， 即 swap\_manager，其中包含了页面置换算法的实现 操作系统是以进程为中心设计的，所以其首要任务是为进程建立档案，进程档案 用于表示、标识或描述进程，即进程控制块。这里需要完成的就是一个进程控制 块的初始化。 而这里我们分配的是一个内核线程的 PCB，它通常只是内核中的一小段代码或者 函数，没有用户空间。而由于在操作系统启动后，已经对整个核心内存空间进行 了管理，通过设置页表建立了核心虚拟空间(即 boot\_cr3 指向的二级页表描述 的空间)。所以内核中的所有线程都不需要再建立各自的页表，只需共享这个核 心虚拟空间就可以访问整个物理内存了。

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。

实验代码：



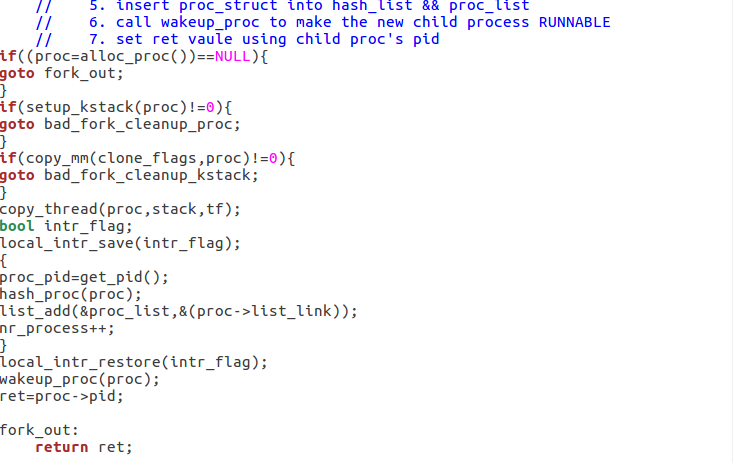
请说明proc\_struct中 struct context context 和 struct trapframe \*tf 成员变量含义和在本实验中的作用是啥？

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

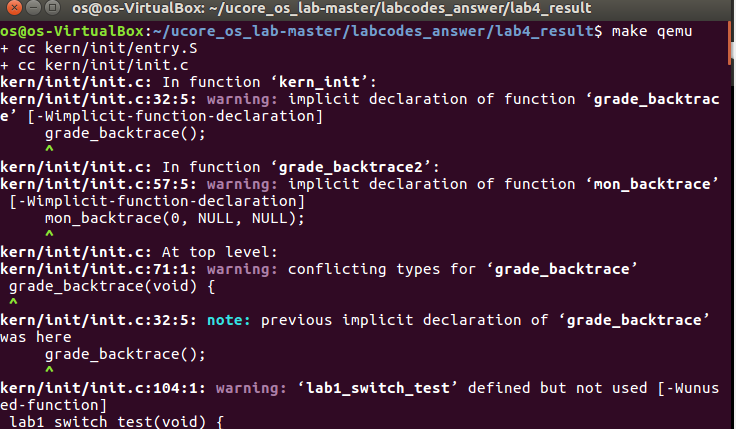
alloc\_proc 实质只是找到了一小块内存用以记录进程的必要信息，并没有实际 分配这些资源，而练习 2 完成的 do\_fork 才是真正完成了资源分配的工作，当 然，do\_fork 也只是创建当前内核线程的一个副本，它们的执行上下文、代码、 数据都一样，但是存储位置不同。 根据提示及阅读源码可知，它完成的工作主要如下： 1、分配并初始化进程控制块（ alloc\_proc 函数）; 2、分配并初始化内核栈，为内核进程（线程）建立栈空间（ setup\_stack 函数） ; 3、根据 clone\_flag 标志复制或共享进程内存管理结构（ copy\_mm 函数）; 4、设置进程在内核（将来也包括用户态）正常运行和调度所需的中断帧和执行 上下文 （ copy\_thread 函数）; 5、为进程分配一个 PID（ get\_pid() 函数）; 6、把设置好的进程控制块放入 hash\_list 和 proc\_list 两个全局进程链表中; 7、自此，进程已经准备好执行了，把进程状态设置为“就绪”态; 8、设置返回码为子进程的 PID 号。 实现过程如下： \* 实现思路： 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。 proc = alloc\_proc(); // 为要创建的新的线程分配线程控制块的空间 if (proc == NULL) goto fork\_out; // 判断是否分配到内存空间 assert(setup\_kstack(proc) == 0); // 为新的线程设置栈，在本实验中，每 个线程的栈的大小初始均为 2 个 Page，即 8KB assert(copy\_mm(clone\_flags, proc) == 0); // 对虚拟内存空间进行拷贝， 由于在本实验中，内核线程之间共享一个虚拟内存空间，因此实际上该函数不需 要进行任何操作 copy\_thread(proc, stack, tf); // 在新创建的内核线程的栈上面设置伪造好 的中端帧，便于后文中利用 iret 命令将控制权转移给新的线程 proc->pid = get\_pid(); // 为新的线程创建 pid hash\_proc(proc); // 将线程放入使用 hash 组织的链表中，便于加速以后对某 个指定的线程的查找 nr\_process ++; // 将全局线程的数目加 1 list\_add(&proc\_list, &proc->list\_link); // 将线程加入到所有线程的链表 中，便于进行调度

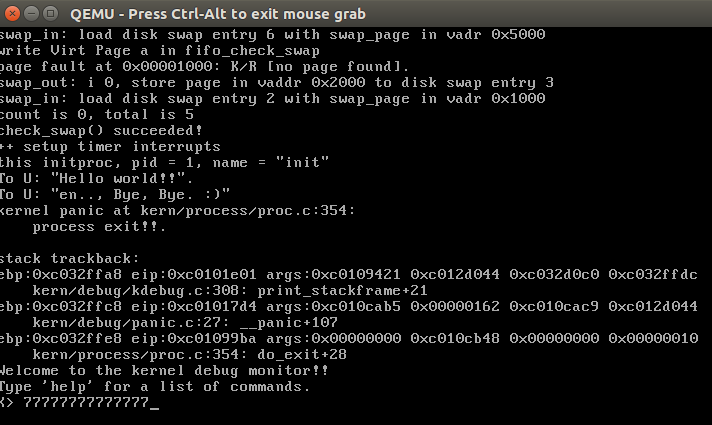


请说明ucore是否做到给每个新fork的线程一个唯一的id？请说明你的分析和理由

首先，本实验不提供线程释放的功能，意味着pid只分配不回收。当fork的线程总数小于MAX\_PID时，每个线程的pid是唯一的。当fork的线程总数大于MAX\_PID时，后面fork的线程的pid可能与前面的线程重复（暂不确定）。

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。

在指定文件下输入 make qemu 得到以下结果



在本实验的执行过程中，创建且运行了几个内核线程？

本实验创建且运行了两个内核线程，分别是idle和init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

避免在进程切换过程中处理中断。