实验三 内核线程管理

171491328 罗晓璐

一、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

二、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

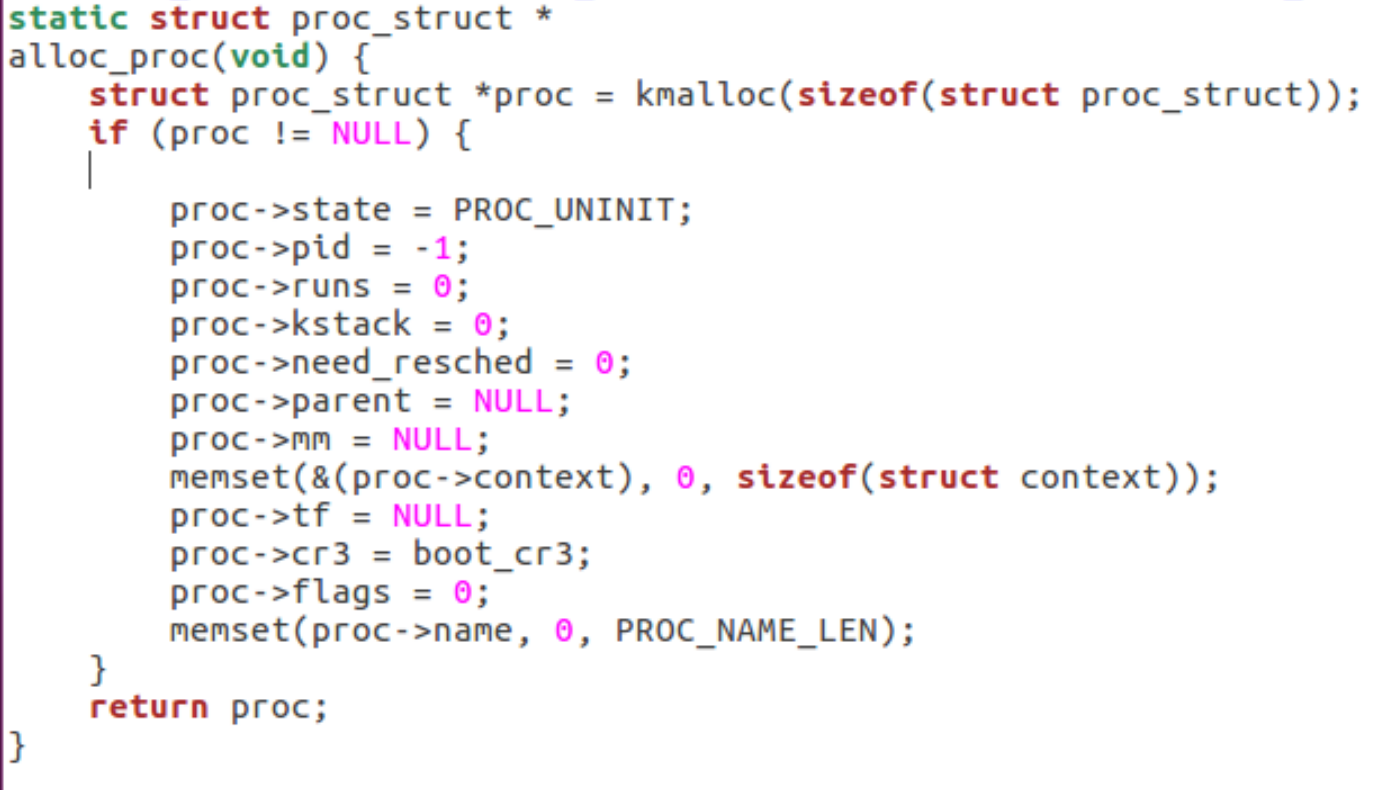
所有内核线程共用ucore内核内存空间，不需为每个内核线程维护单独的内存空间而用户进程需要维护各自的用户内存空间。

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。

【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。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trapframe，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（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。

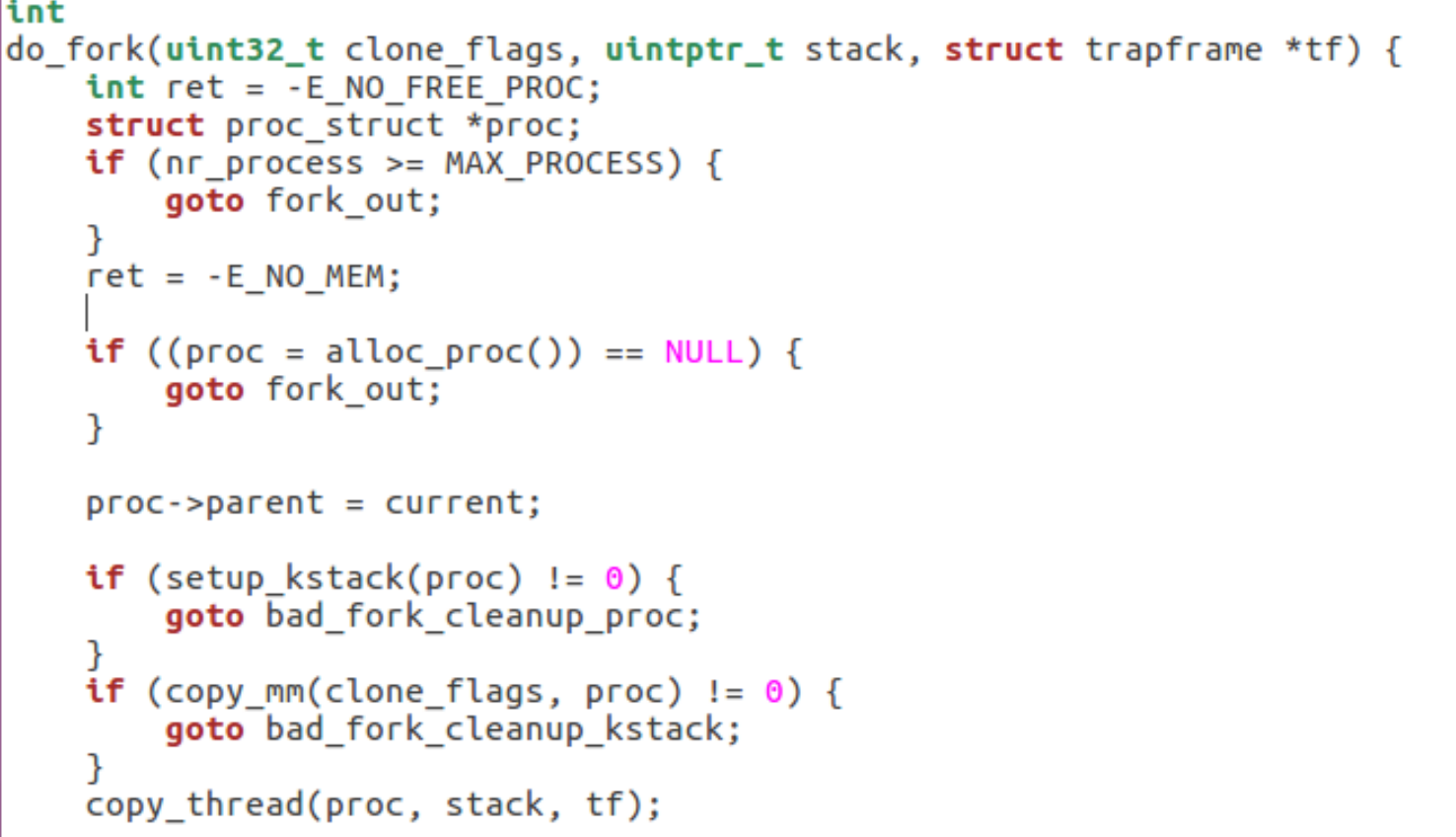
为进程分配一个内核栈。

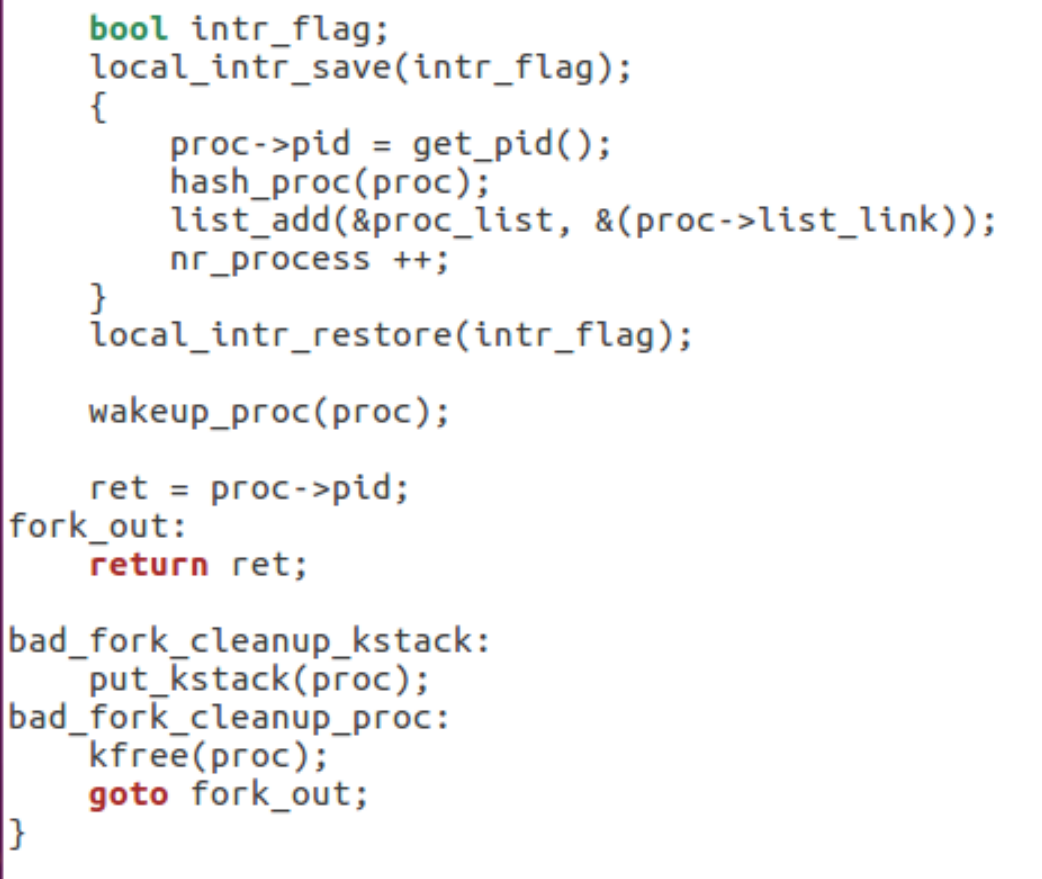
复制原进程上下文到新进程

将新进程添加到进程列表

唤醒新进程

返回新进程号





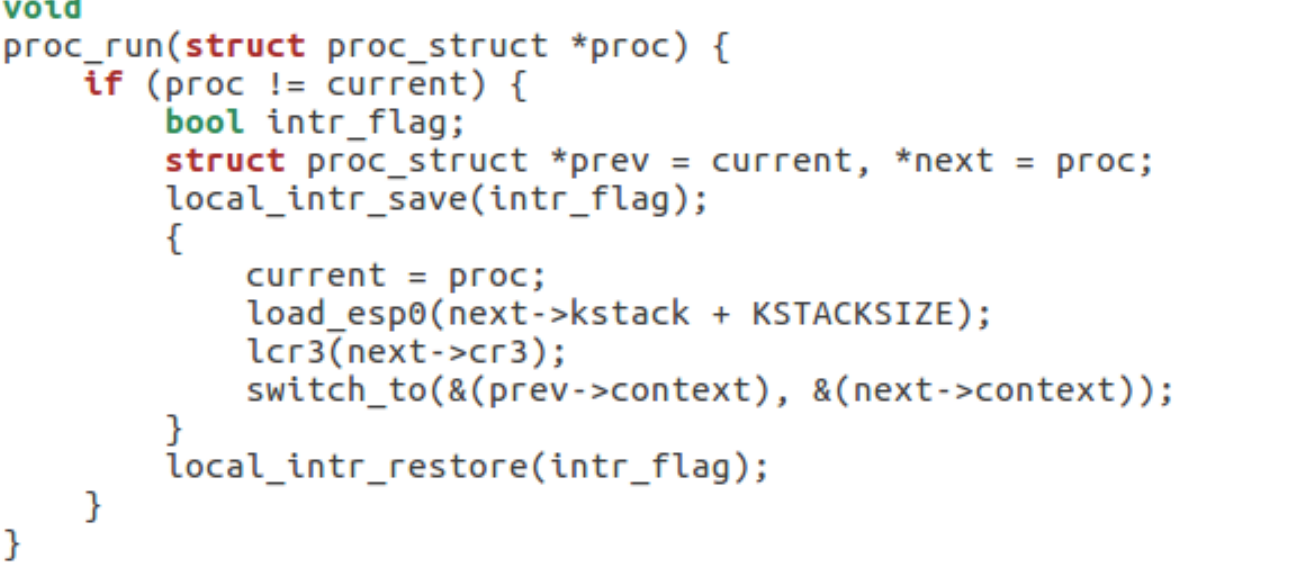
请在实验报告中简要说明你的设计实现过程。请回答如下问题：

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。

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：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：



proc\_run中首先进行了TSS以及cr3寄存器的设置，然后调用到了swtich\_to函数来切换线程，根据上文中对switch\_to函数的分析可以知道，在调用该函数之后，首先会恢复要运行的线程的上下文，然后由于恢复的上下文中已经将返回地址（copy\_thread函数中完成）修改成了forkret函数的地址(如果这个线程是第一运行的话，否则就是切换到这个线程被切换出来的地址)，也就是会跳转到这个函数，最后进一步跳转到了\_\_trapsret函数，调用iret最终将控制权切换到新的线程；

在本实验的执行过程中，创建且运行了几个内核线程？

1.在本实验的执行过程中，创建且运行了几个内核线程？

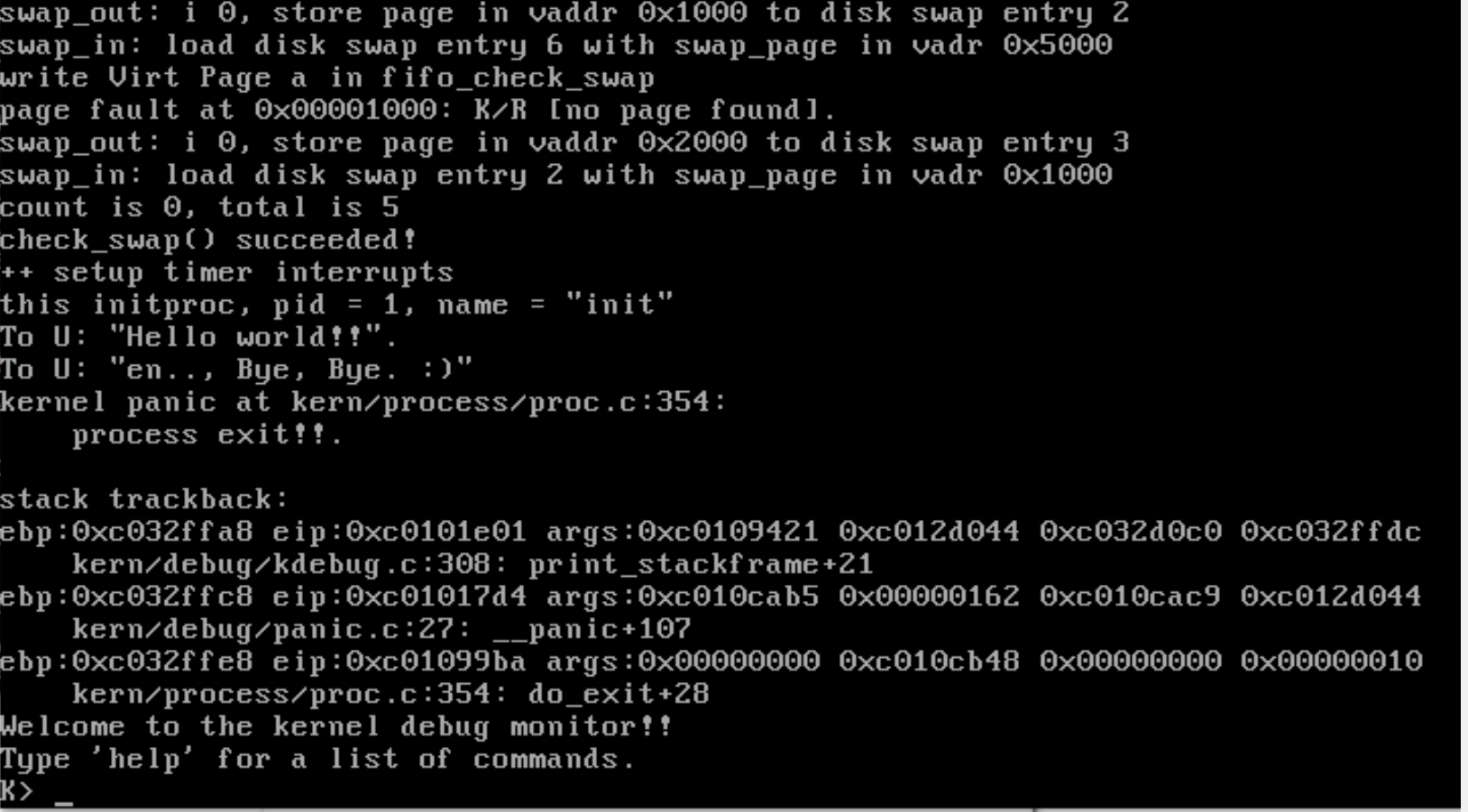
总共创建了两个内核线程，分别为：

1. idleproc: 最初的内核线程，在完成新的内核线程的创建以及各种初始化工作之后，进入死循环，用于调度其他线程；
2. initproc: 被创建用于打印"Hello World"的线程；

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

该语句的左右是关闭中断，使得在这个语句块内的内容不会被中断打断，这就使得某些关键的代码不会被打断，从而不会一起不必要的错误；

完成代码编写后，编译并运行代码：make qemu



在本次实验中设计到的知识点有：

1. 线程控制块的概念以及组成；
2. 切换不同线程的方法；

对应到的OS中的知识点有：

对内核线程的管理；

对内核线程之间的切换；

这两者之间的关系为，前者为后者在OS中的具体实现提供了基础；

实验中未涉及的知识点：

OS的启动过程；

OS中对物理、虚拟内存的管理；

OS中对用户进程的管理；

OS中对线程/进程的调度；

三、总结

本次实验主要针对内核线程的管理，所有内核线程直接使用共同的ucore内核内存空间，而用户进程需要维护各自的用户内存空间。以及了解到了进程切换的相关细节操作，更加深一步的了解了操作系统。