实验四：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

相关原理介绍可看附录B：【原理】进程/线程的属性与特征解析。

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）

本练习的编码工作集中在proc.c中的alloc\_proc函数中，该函数的具体含义为创建一个新的进程控制块，并且对控制块中的所有成员变量进行初始化，根据实验指导书中的要求，除了指定的若干个成员变量之外，其他成员变量均初始化为0，取特殊值的成员变量如下所示：

proc->state = PROC\_UNINIT;

proc->pid = -1;

proc->cr3 = boot\_cr3;

struct proc\_struct \*proc = kmalloc(sizeof(struct proc\_struct)); // 为线程控制块分配空间

if (proc != NULL) {

proc->state = PROC\_UNINIT; // 初始化具有特殊值的成员变量

proc->cr3 = boot\_cr3;

proc->pid = -1;

proc->runs = 0; // 对其他成员变量清零处理

proc->kstack = 0;

proc->need\_resched = 0;

proc->parent = NULL;

proc->mm = NULL;

memset(&proc->context, 0, sizeof(struct context)); // 使用memset函数清零占用空间较大的成员变量，如数组，结构体等

proc->tf = NULL;

proc->flags = 0;

memset(proc->name, 0, PROC\_NAME\_LEN);

}

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：

练习

211

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：

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。

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

wakeup\_proc(proc); // 唤醒该线程，即将该线程的状态设置为可以运行

ret = proc->pid; // 返回新线程的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：

在本实验的执行过程中，创建且运行了几个内核线程？

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

说明理由

完成代码编写后，编译并运行代码：make qemu

如果可以得到如 附录A所示的显示内容（仅供参考，不是标准答案输出），则基本正确。

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；

总共创建了两个内核线程，分别为：

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；

initproc: 被创建用于打印"Hello World"的线程；