

# 实验四 内核线程管理

计31 张正 2013011418

## 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。

### 练习0：填写已有实验

本实验依赖实验1/2/3。请把你做的实验1/2/3的代码填入本实验中代码中有“LAB1”，“LAB2”，“LAB3”的注释相应部分。

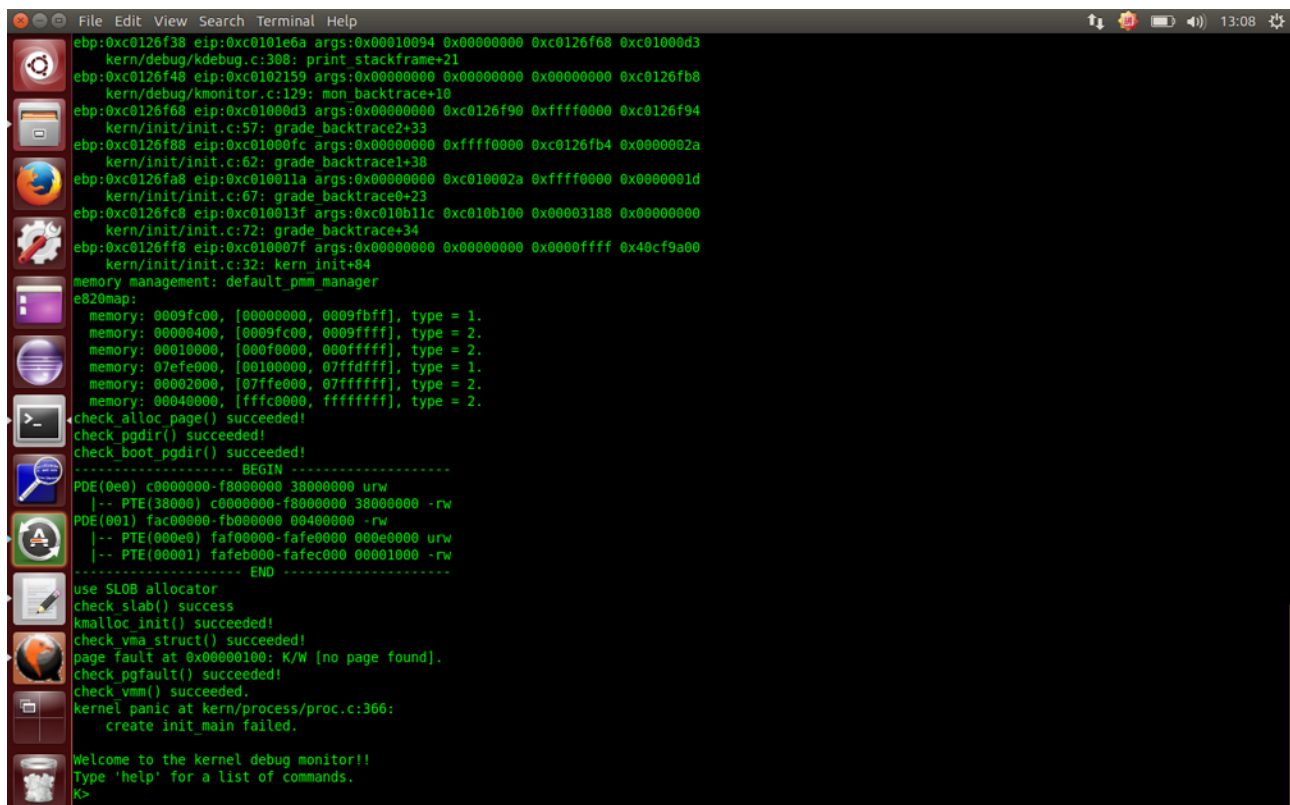
### 练习1：分配并初始化一个进程控制块（需要编码）

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。

在相应位置填入如下代码

```
proc->state = PROC_UNINIT;
proc->pid = -1;
proc->runs = 0;
proc->kstack = 0;
proc->need_resched = 0;
proc->parent = NULL;
proc->mm = NULL;
memset(&(proc->context), 0, sizeof(struct context)); //初始化进程上下文
proc->tf = NULL; //初始化中断帧，用于记录进程发生中断前的状态
proc->cr3 = boot_cr3; //因为是内核线程，所以CR3=boot_cr3
proc->flags = 0;
memset(proc->name, 0, PROC_NAME_LEN);
```

运行如下图



● 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？

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。

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。

## 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。你需要完成在kern/process/proc.c中的do\_fork函数中的处理过程。

do\_fork（）函数完成具体内核线程的创建工作，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。大致执行步骤如下：

1. 调用alloc\_proc，首先获得一块用户信息块
2. 为进程分配一个内核栈
3. 复制原进程的内存管理信息到新进程（内核线程不用做）
4. 复制原进程的上下文到新进程
5. 将新进程添加到进程列表
6. 唤醒新进程
7. 返回新进程号

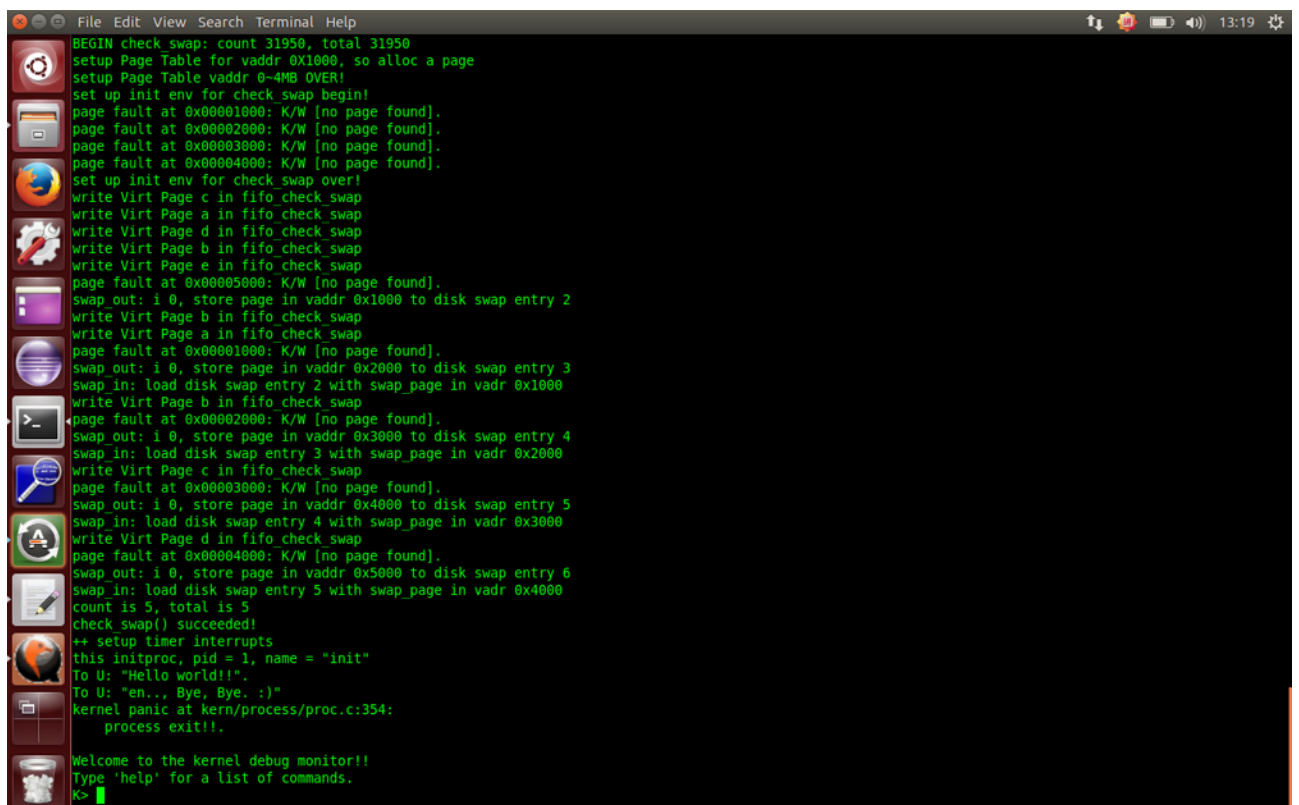
添加如下代码

```

if ((proc = alloc_proc()) == NULL) { //获得用户信息块
    goto fork_out;
}
proc->parent = current; //设置父进程
if (setup_kstack(proc) != 0) { //分配了2页内核栈
    goto bad_fork_cleanup_proc;
}
if (copy_mm(clone_flags, proc) != 0) {
    goto bad_fork_cleanup_kstack;
}
copy_thread(proc, stack, tf); //设置中断帧
bool intr_flag;
local_intr_save(intr_flag);
{
    proc->pid = get_pid();
    hash_proc(proc); //将新进程加入hash_list
    list_add(&proc_list, &(proc->list_link));
    nr_process ++;
}
local_intr_restore(intr_flag); //将新进程加入proc_list
wakeup_proc(proc); //唤醒进程，等待调度
ret = proc->pid;

```

运行结果如下



```

File Edit View Search Terminal Help
BEGIN check swap: count 31950, total 31950
setup Page Table for vaddr 0x1000, so alloc a page
setup Page Table vaddr 0-4MB OVER!
set up init env for check_swap begin!
page fault at 0x00001000: K/W [no page found].
page fault at 0x00002000: K/W [no page found].
page fault at 0x00003000: K/W [no page found].
page fault at 0x00004000: K/W [no page found].
set up init env for check_swap over!
write Virt Page c in fifo_check_swap
write Virt Page a in fifo_check_swap
write Virt Page d in fifo_check_swap
write Virt Page b in fifo_check_swap
write Virt Page e in fifo_check_swap
page fault at 0x00005000: K/W [no page found].
swap out: i 0, store page in vaddr 0x1000 to disk swap entry 2
write Virt Page b in fifo_check_swap
write Virt Page a in fifo_check_swap
page fault at 0x00001000: K/W [no page found].
swap out: i 0, store page in vaddr 0x2000 to disk swap entry 3
swap in: load disk swap entry 2 with swap_page in vadr 0x1000
write Virt Page b in fifo_check_swap
page fault at 0x00002000: K/W [no page found].
swap out: i 0, store page in vaddr 0x3000 to disk swap entry 4
swap in: load disk swap entry 3 with swap_page in vadr 0x2000
write Virt Page c in fifo_check_swap
page fault at 0x00003000: K/W [no page found].
swap out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page d in fifo_check_swap
page fault at 0x00004000: K/W [no page found].
swap out: i 0, store page in vaddr 0x5000 to disk swap entry 6
swap in: load disk swap entry 5 with swap_page in vadr 0x4000
count is 5, total is 5
check_swap() succeeded!
++ setup timer interrupts
this initproc, pid = 1, name = "init"
To U: "Hello world!!".
To U: "en..., Bye, Bye. :)"
kernel panic at kern/process/proc.c:354:
process exit!!

Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K>

```

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，每一个进程git\_pid得到唯一的一个id.

**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。

- 在本实验的执行过程中，创建且运行了几个内核线程？

创建了两个内核线程，空闲进程的创建：idleproc proc\_init()，创建第一个内核线程：initproc proc\_init()。ucore创建的第一个进程通过proc\_inti创建 idleproc，init\_proc 执行的函数为init\_main。

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

local\_intr\_save(intr\_flag)关中断

local\_intr\_restore(intr\_flag)开中断