

“操作系统原理与实践”实验报告

基于内核栈切换的进程切换

本实验完整代码在 http://git.shiyanlou.com/gaoyuanhezhihao/shiyanlou_cs115/src/master/Lab5 中。本次实验参考了 [qiuy_mac](https://www.shiyanlou.com/user/34029)同学（<https://www.shiyanlou.com/user/34029>）的实验报告和代码，在此表示衷心的感谢。1.首先改写switch_to函数，因为linux-0.11使用的是一个宏，所以我们这里需要用汇编编写一个switch_to函数。之前我把switch_to以及first_return_from_kernel写在一个新建的文件中，结果有问题，在参考别的同学的实验报告后，把这两个函数写到了kernel/system_calls中，同时需要注意的是，为了能够在schedule函数，copy_process函数中“看到”这两个函数，我们需要用.globl声明这两个函数符号，同时在sched.c,fork.c中声明这两个函数。具体可以参考源代码

```
.globl switch_to,first_return_from_kernel

.align 2
switch_to:

    pushl %ebp
    movl %esp,%ebp
    pushl %ecx
    pushl %ebx
    pushl %eax
    movl 8(%ebp),%ebx
    cmpl %ebx,current
    je 1f
    movl %ebx,%eax
    xchgl %eax,current
    movl tss,%ecx
    addl $4096,%ebx
    movl %ebx,ESP0(%ecx)
    movl %esp,KERNEL_STACK(%eax)
    movl 8(%ebp),%ebx
    movl KERNEL_STACK(%ebx),%esp
    movl 12(%ebp),%ecx
    lldt %cx
    movl $0x17,%ecx
    mov %cx,%fs
```

2.修改任务数据结构：task_struct，在其中加入内核栈指针成员。

```
struct task_struct {
    /* these are hardcoded - don't touch */
    long state; /* -1 unrunnable, 0 runnable, >0 stopped */
    long counter;
    long priority;
    long kernelstack;
};
```

数据：同时需要修改的是0号进程的初始化

```
#define INIT_TASK \
/* state etc */ { 0, 15, 15, PAGE_SIZE+( long )&init_task,\
/* signals */ 0,{{}},, 0, \
/* ec,brk... */ 0, 0, 0, 0, 0, \
```

3.修改copy_process函数。首先申请一页内存：

```
p = (struct task_struct *) get_free_page();
```

这段代码就是新建的进程保存信息的地方，原本的copy_process中从父进程复制了几乎所有的寄存器保存在这里，其中有一个很大的不同就是eax被写为了0，起始这就是fork()函数的返回值，这样在最后fork()返回的时候就可以实现与父进程不同的用户代码走向。

```
p->tss.eax = 0;
```

现在我们不在使用tss进程切换方式，所以这里保存tss内容的代码可以不要。重要的是在我们的栈中手动压入一些信息。这一点在实验提示中已经有了详细的说明。需要注意的是，其实我们的这个栈就在这个新建的p指向的页中，前面

```
p = (struct task_struct *) get_free_page();
```

获得了一页内存，大小为4096字节。前面保存任务数据结构只用了很少的一部分，剩下的内容完全可以用来作为新建进程的内核栈。所以我们将这页内存最后的地址作为栈顶：

```
long *krnstack;
krnstack = ( long )(PAGE_SIZE+( long )p);
```

然后再这个内核栈中手动“压入”一些内容，使得它在后面切换的时候能够顺利的进入first_return_from_kernel函数，最后返回到用户程序中。

```
*(-- krnstack) = ss & 0xffff ;
*(-- krnstack) = esp;
*(-- krnstack) = eflags;
*(-- krnstack) = cs & 0xffff ;
*(-- krnstack) = eip;
*(-- krnstack) = ds & 0xffff ;
*(-- krnstack) = es & 0xffff ;
*(-- krnstack) = fs & 0xffff ;
*(-- krnstack) = gs & 0xffff ;
*(-- krnstack) = esi;
*(-- krnstack) = edi;
*(-- krnstack) = edx;
*(-- krnstack) = ( long )first_return_from_kernel;
*(-- krnstack) = ebp;
*(-- krnstack) = ecx;
*(-- krnstack) = ebx;
*(-- krnstack) = 0;
```

这个时候，栈顶的地址就可以保存到任务数据结构中了

```
p->kernelstack = krnstack;
```

