**Linux操作系统分析**

**实验一**

**Mykernel**

**学号：SG13225146**

**姓名：张 添 程**

**日期：2013. 03. 11**

**Lab1：Mykernel**

**学号：SG13225146 姓名：张添程**

目录

[一、实验环境 3](#_Toc382338323)

[1.1硬件平台： 3](#_Toc382338324)

[1.2软件平台： 3](#_Toc382338325)

[二、实验目的 3](#_Toc382338326)

[三、实验内容 4](#_Toc382338327)

[3.1编译内核 4](#_Toc382338328)

[3.2使用qemu查看内核工作状态 9](#_Toc382338329)

[3.3多进程时间片轮转分析 14](#_Toc382338330)

[3.3.1内核工作方式分析 14](#_Toc382338331)

[3.3.2代码分析 17](#_Toc382338332)

[3.3.3过程分析 25](#_Toc382338333)

[四、实验总结 29](#_Toc382338334)

[五、实验中遇到的问题 30](#_Toc382338335)

[5.1qemu的使用 30](#_Toc382338336)

[5.2代码中的不易理解的问题 30](#_Toc382338337)

[5.2.1$1f 30](#_Toc382338338)

[5.2.2ret 30](#_Toc382338339)

[六、参考文献 31](#_Toc382338340)

# 一、实验环境

## 1.1硬件平台：

Lenovo昭阳E46L

CPU：(英特尔)Pentium(R) Dual-Core CPU T4500

主板：联想 LE6

内存：2.00 GB (三星 PC3-8500 DDR3 SDRAM SO-DIMM 1067MHz)

硬盘：320 GB (西数 WD-WXM1A60481)

## 1.2软件平台：

操作系统：win7 32位旗舰版

虚拟机工具：VMware Workstation 10.0.1

实验系统：Ubuntu 12.04 LTS

实验软件：qemu、gedit

实验语言：C、汇编

相关软件：Microsoft Office Visio 2013、Word 2013、PowerPoint 2013、QQ截图工具、画图

# 二、实验目的

1、熟悉、理解Linux内核工作方式

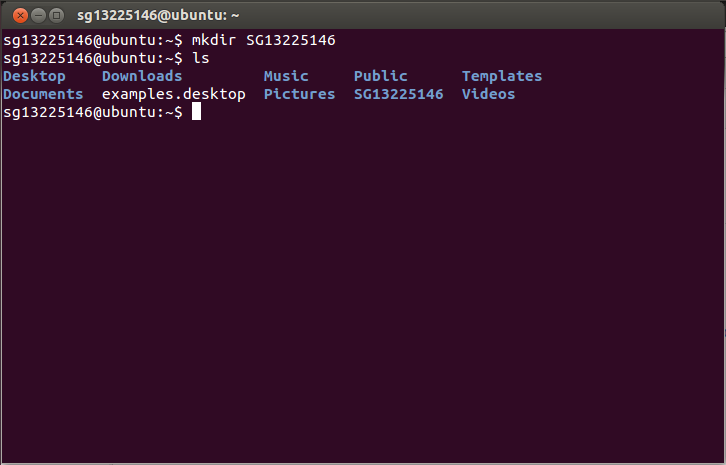
2、尝试编写自己的内核

3、理解多进程时间片轮转的工作方式

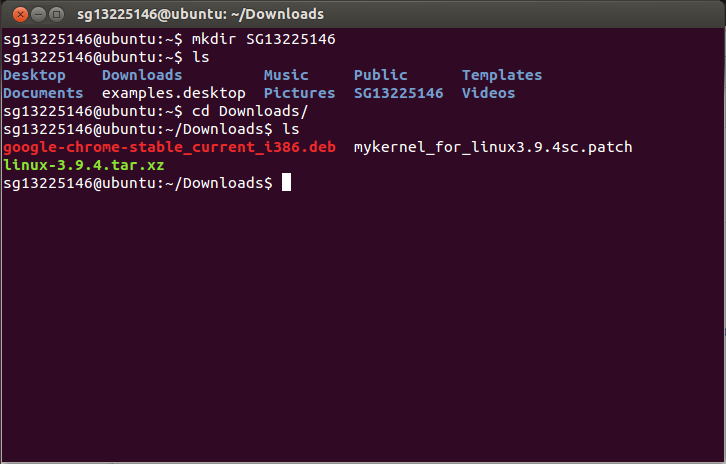
# 三、实验内容

## 3.1编译内核

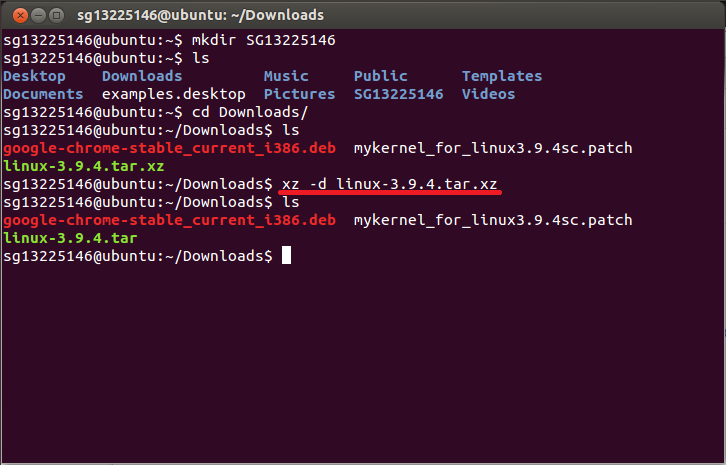
创建SG13225146文件夹作为工作区



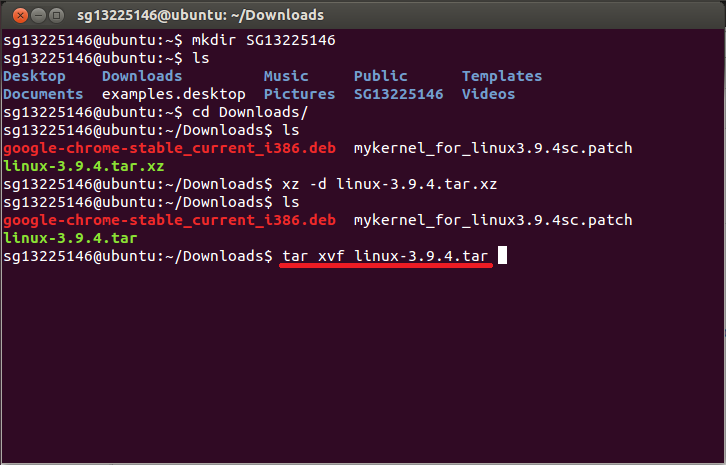
查看Downloads文件内容



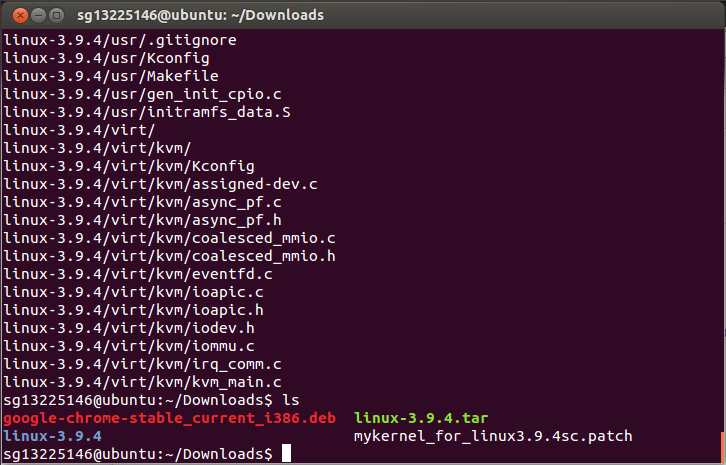
解压linux-3.9.4.tar.xz



解压刚解压出来的linux-3.9.4.tar

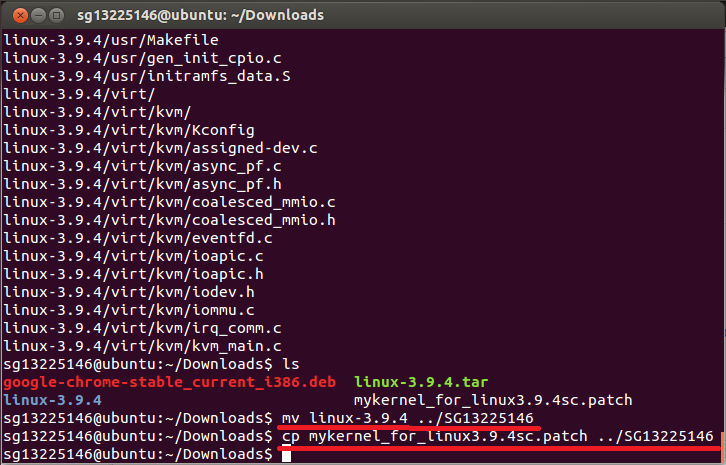


查看Downloads文件夹内容



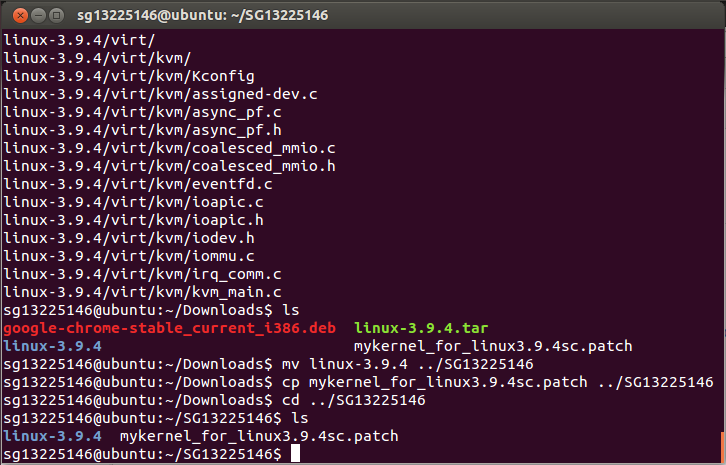
将linux-3.9.4文件夹剪切到刚创建工作区SG13225146

将mykernel\_for\_linux3.9.4sc.patch复制到工作区SG13225146

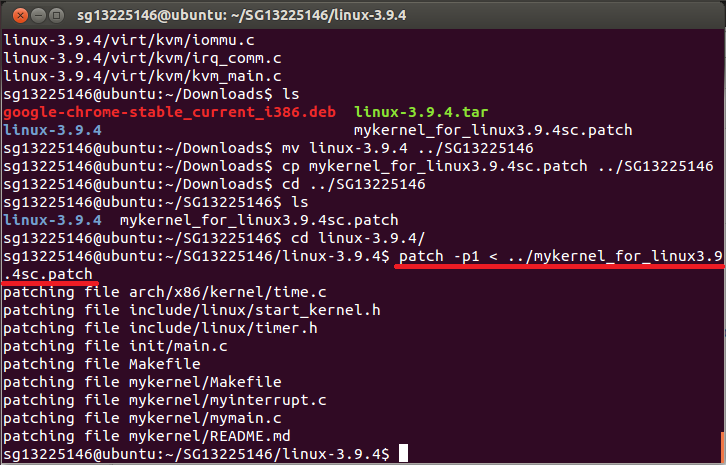


注因为linux-3.9.4文件内容复杂庞大，如果复制会花费大量时间，时间操作了解压与剪切的总和，故这里使用的是剪切，如果需要重新布置，在解压下linux-3.9.4.tar就好。

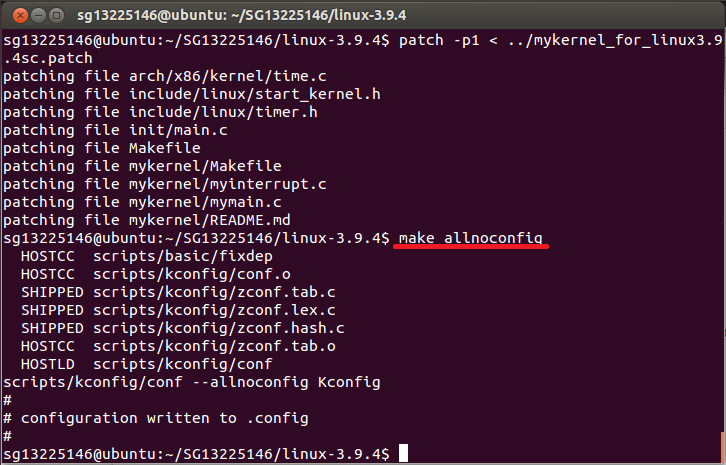
查看工作区内容



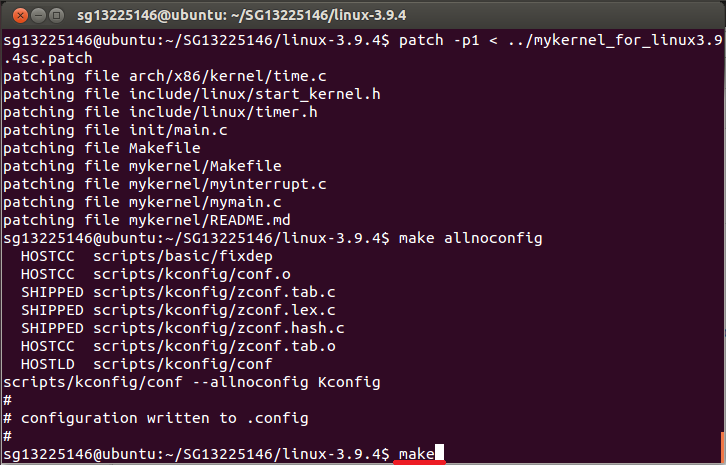
patch -p1 < ../mykernel\_for\_linux3.9.4sc.patch



make allnoconfig 复位

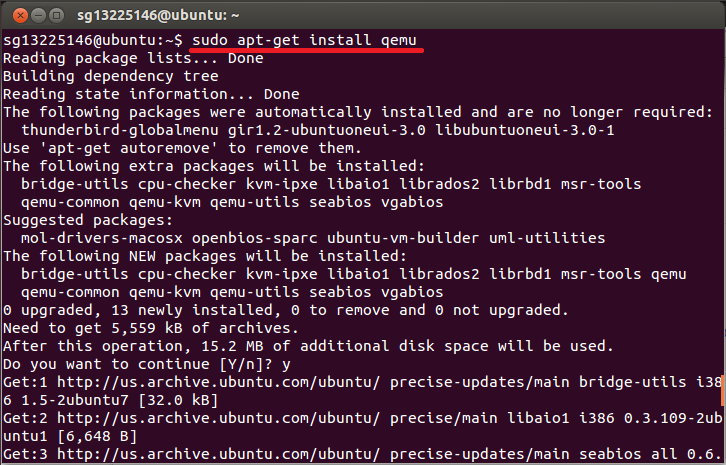


make 编译

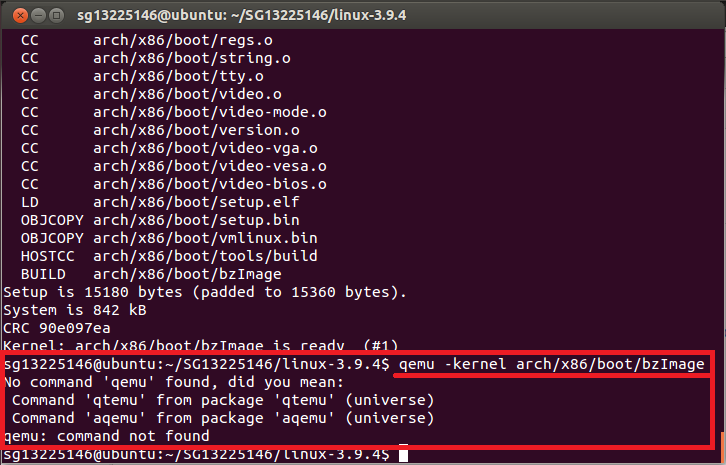


## 3.2使用qemu查看内核工作状态

安装qemu

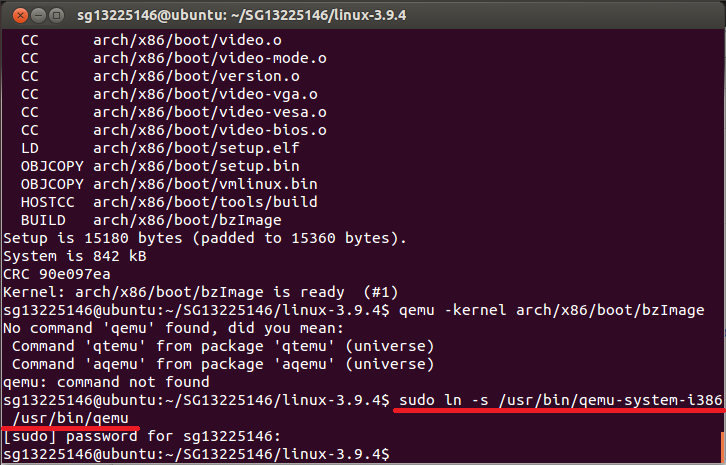


使用qemu查看内核

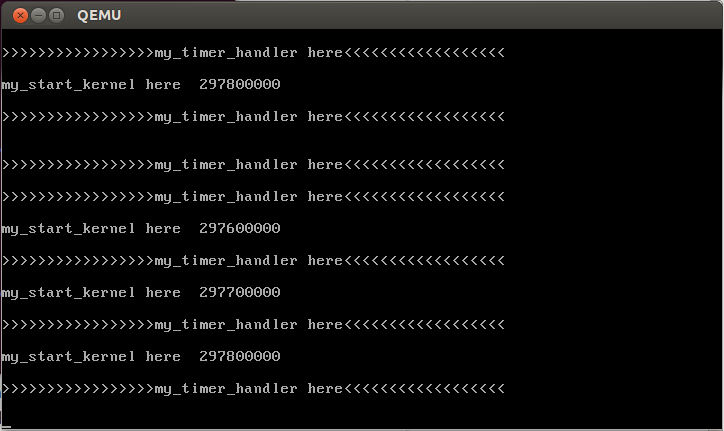
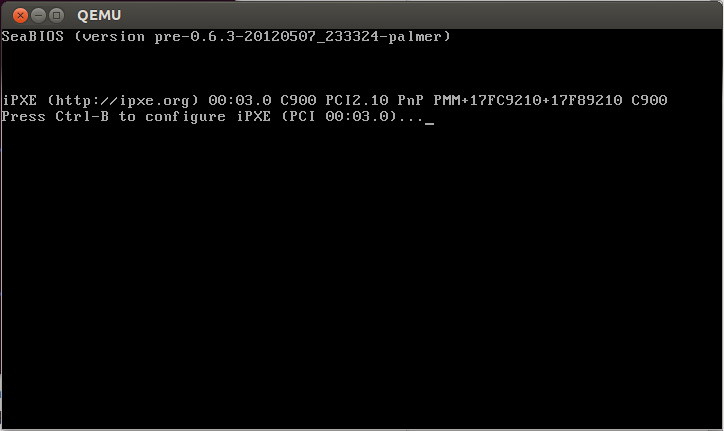
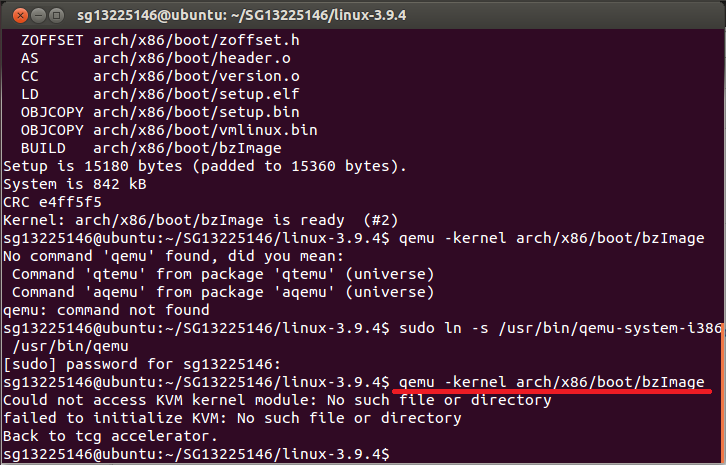


查阅资料发现

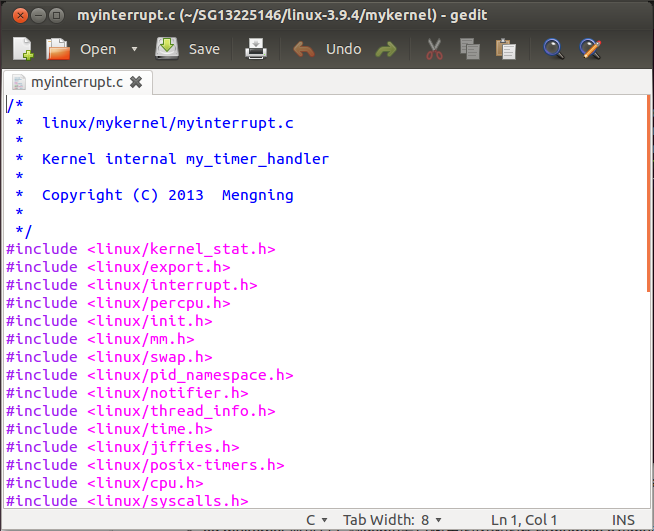
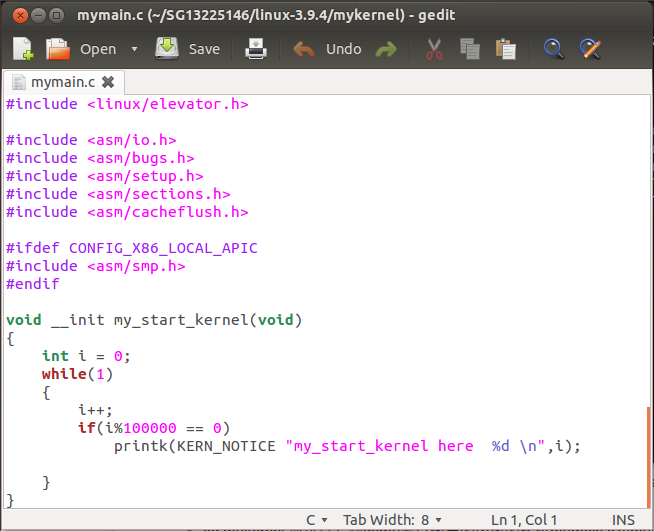
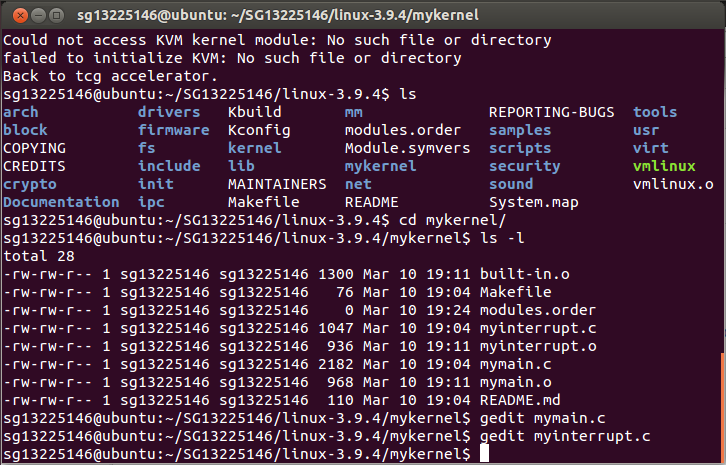
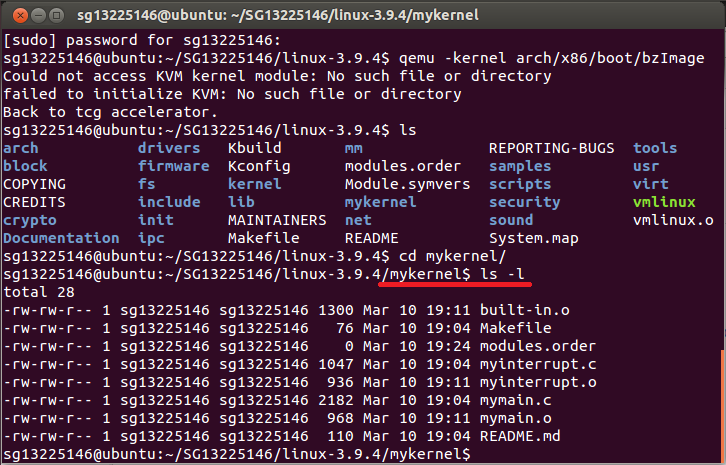
对于Ubuntu 12.04LTS没有qemu命令，可以使用qemu-system-i386（32位）或qemu-system-x86\_64（64位）替换，可以建立一条连接sudo ln –s /usr/bin/qemu-system-i386 /usr/bin/qemu3.2，再使用qemu查看内核工作方式



使用qemu查看内核



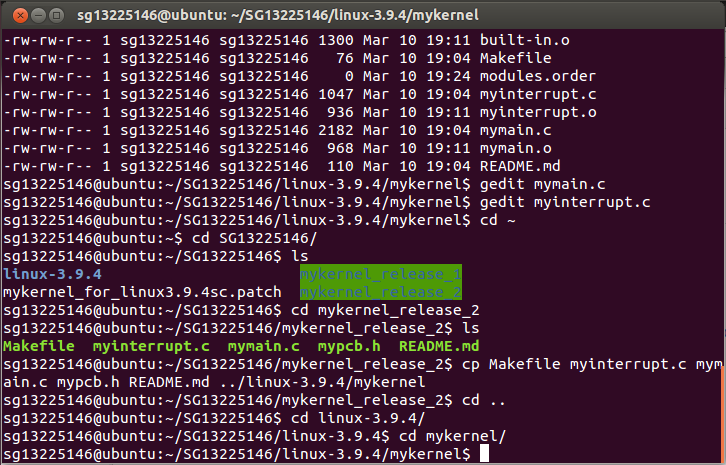
查看mykernel代码



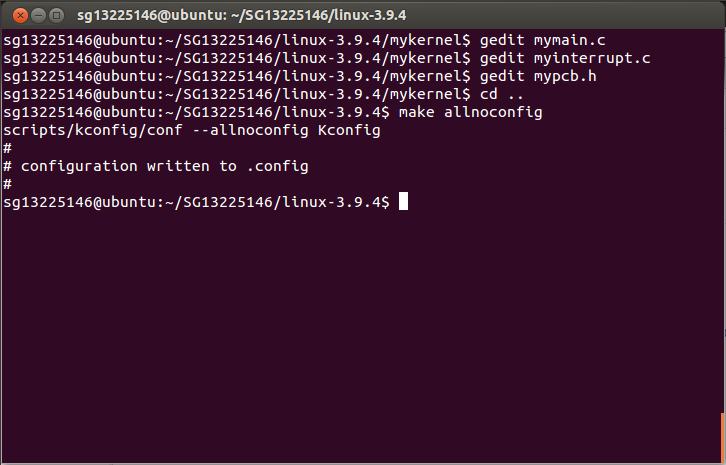
## 3.3多进程时间片轮转分析

### 3.3.1内核工作方式分析

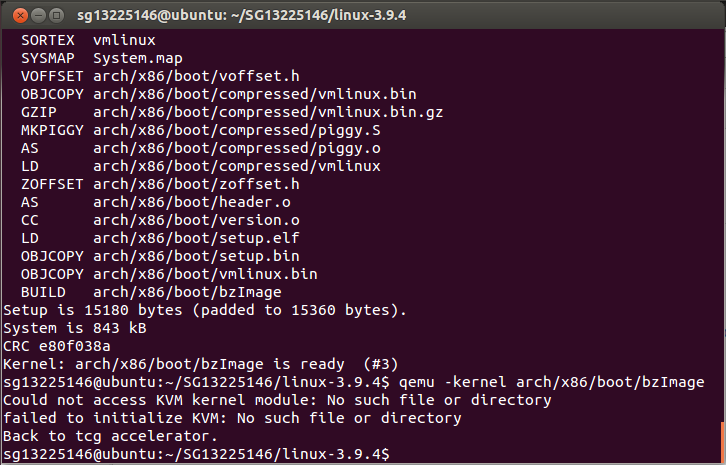
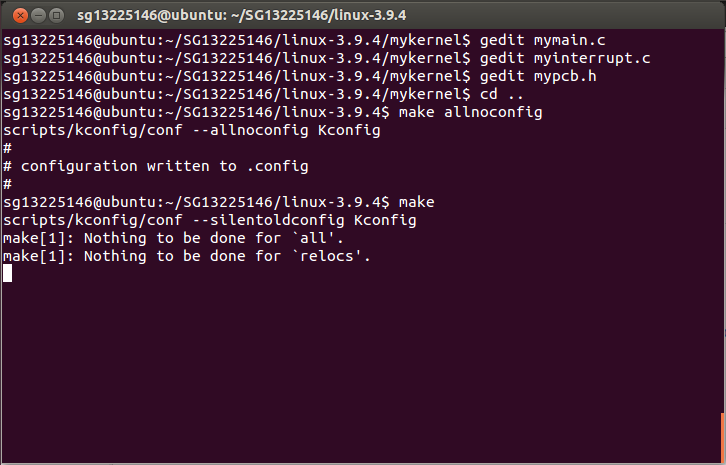
将多进程实践片轮转的mykernel放入linux3.9.4



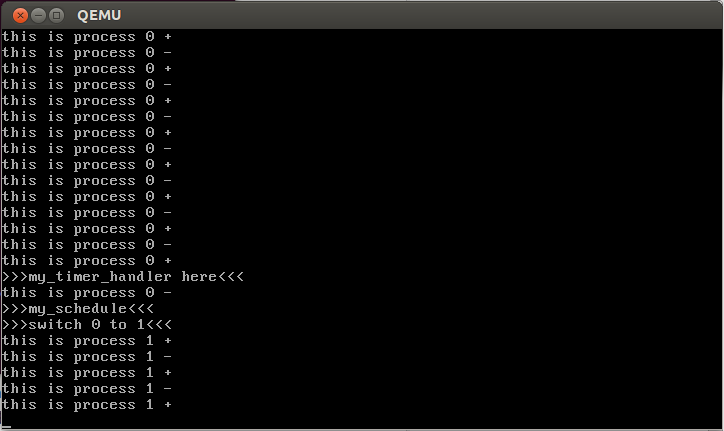
重新复位



编译



使用qemu



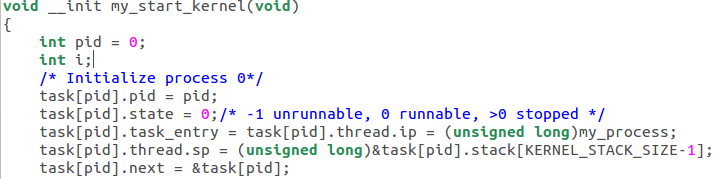
系统开始在process 0中运行，在出现my\_timer\_handler后跳转到process 1，之后在process 1中运行，在出现my\_timer\_handler后跳转到process 2，之后在process 2中运行，在出现my\_timer\_handler后跳转到process 3，之后在process 3中运行，在出现my\_timer\_handler后跳转到process 0，依次循环。

### 3.3.2代码分析

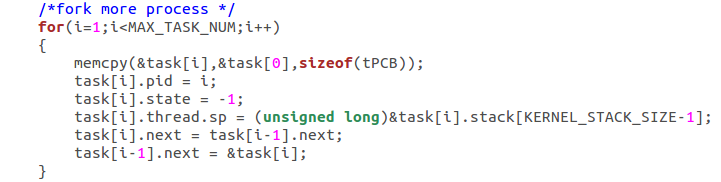
定义循环链表的循环结构体



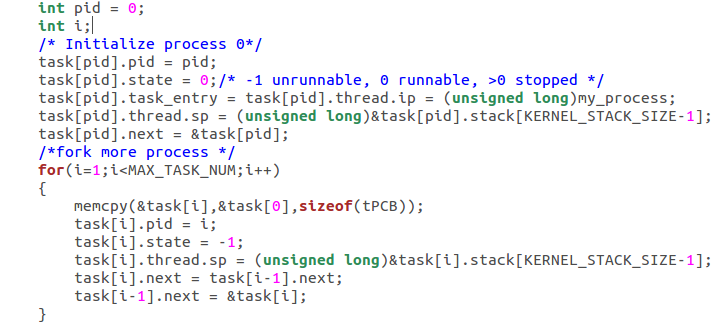
初始化循环体，初始一个单PCB循环链表



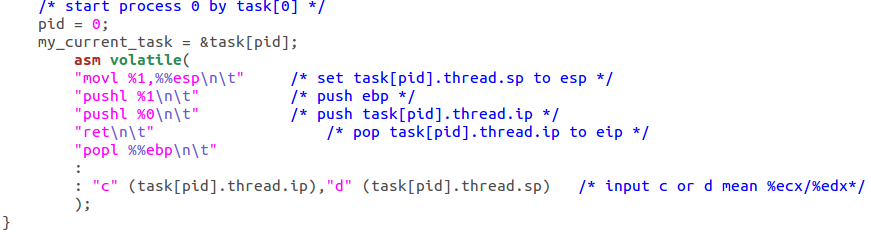
扩充循环链表，使用memcpy将task[0]初始状态复制到task[i]。



至此形成一个有4个PCB的循环链表

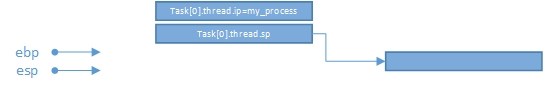


初始化堆栈esp、ebp、eip

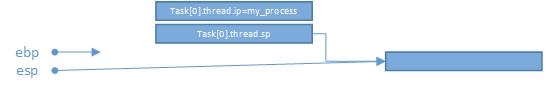


代码分析

初始堆栈状态



movl %1, %%esp：将esp指向task[0]的堆栈stack顶端。



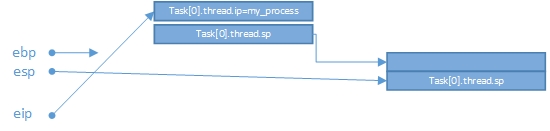
push1 %1：将task[0].thread.sp即堆栈顶端地址压栈，为以后的%ebp的复位使用。



pushl %0：将task[0].thread.ip即程序入口my\_process压栈。



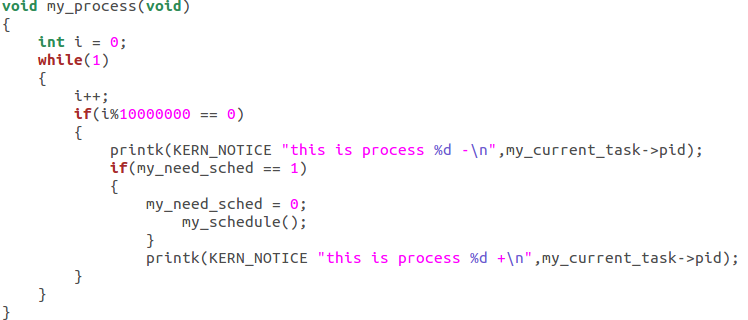
ret：将task[0].thread.ip即程序入口my\_process放到eip。



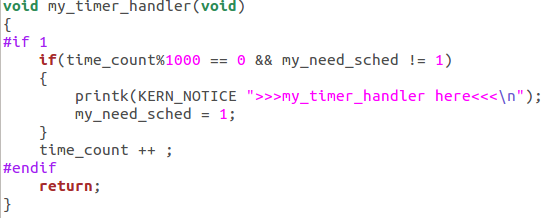
popl %%ebp：引文eip被更改，所以实际没有运行。



my\_process：循环执行因为my\_need\_sched初始为0，只用当时钟中断my\_timer\_handler将my\_need\_sched置为1时，才会调用my\_schedule()。

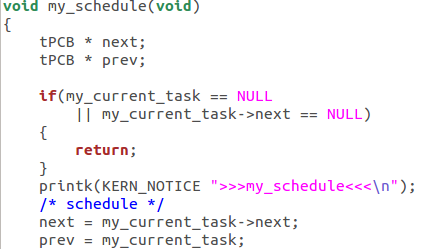


my\_timer\_handler：定期将my\_need\_sched置为1，以调用my\_schedule()。

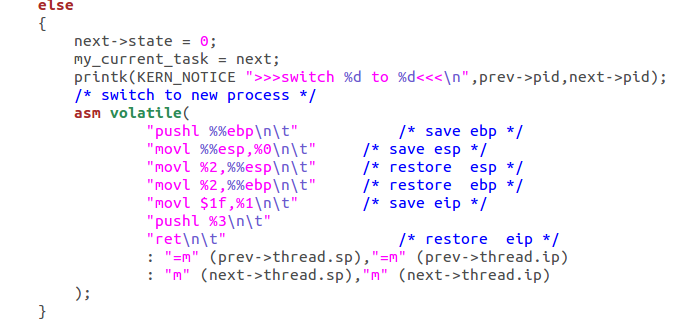


my\_schedule：时间片轮转。

初始化定义next与prev。



初始化循环体堆栈代码



代码分析

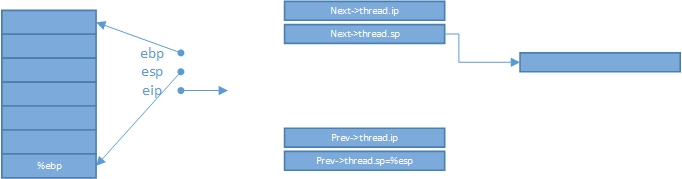
初始状态



pushl %%ebp

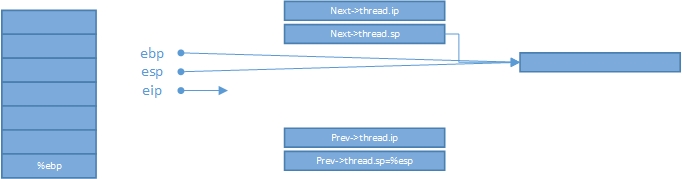


movl %%esp %0



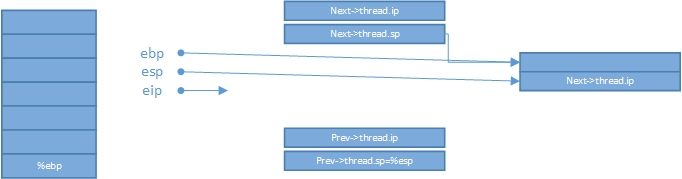
movl %2, %%esp

movl %2, %%ebp

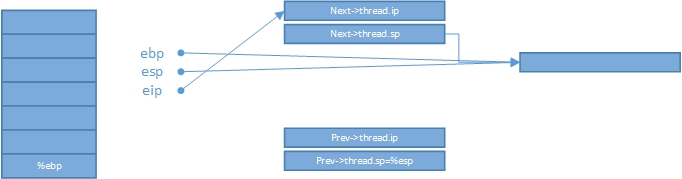


movl $1f, %1

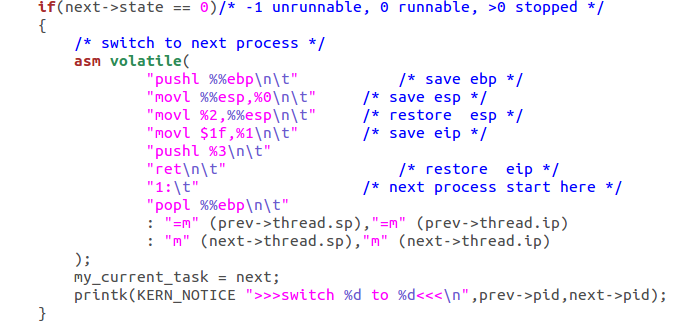
Pushl %3



ret

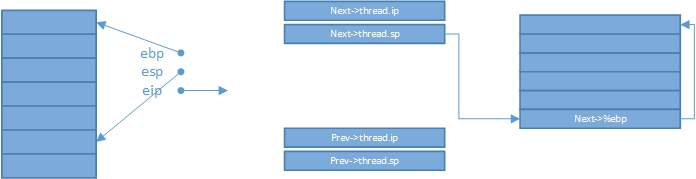


循环运行代码

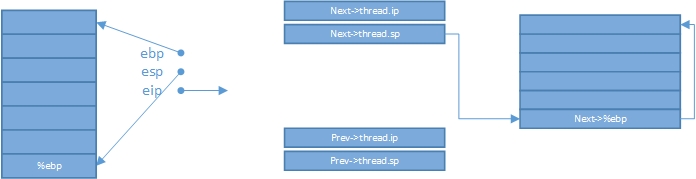


代码分析

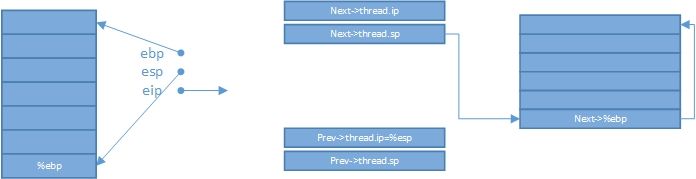
初始状态



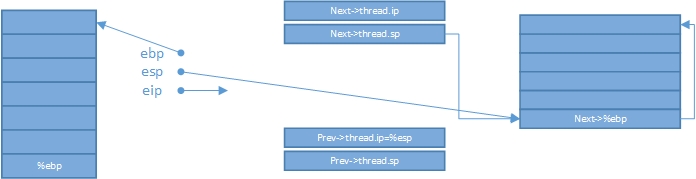
pushl %%ebp



movl %%esp %0

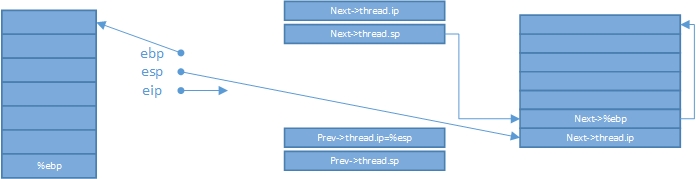


movl %2, %%esp

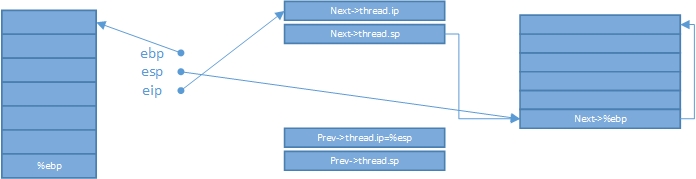


movl $1f, %1

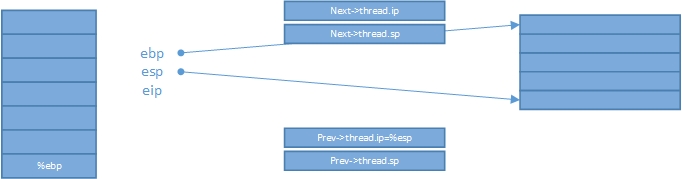
Pushl %3



Ret



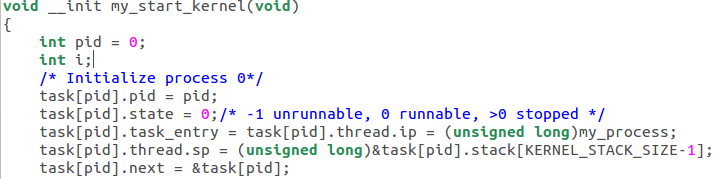
popl %%ebp



### 3.3.3过程分析

#### 3.3.3.1创建循环链表

通过mymain.c中的代码初始化一个单PCB的循环链表



如图

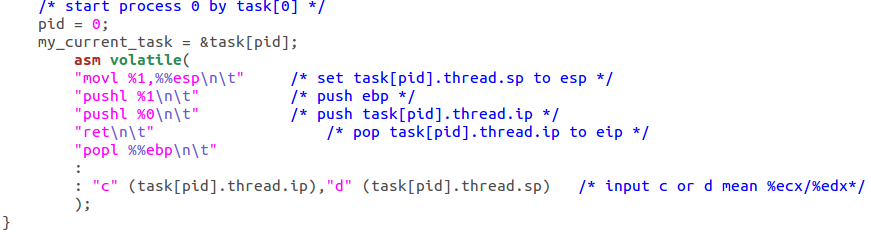


扩充循环链表，使用memcpy将task[0]初始状态复制到task[i]。

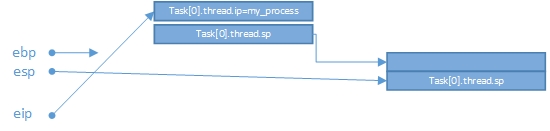


#### 3.3.3.2初始化堆栈进入my\_process

通过

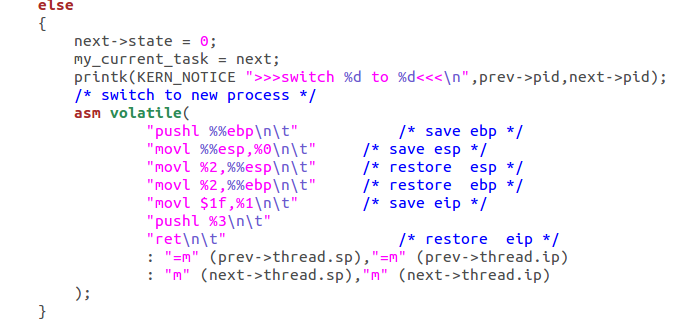


初始化一个堆栈

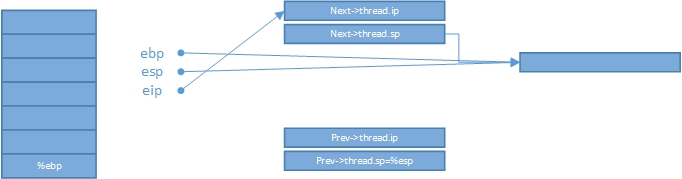


#### 3.3.3.3初始化循环体堆栈

通过代码



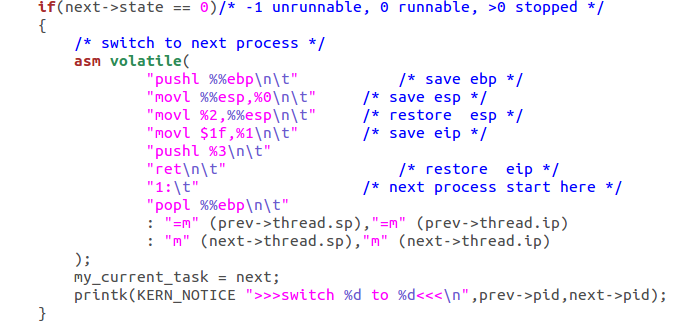
对于一个state = -1的task进行了堆栈的初始化并对原task的ebp、esp进行保存。



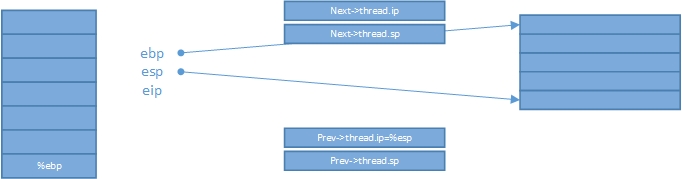
其中原ebp保存在原堆栈顶端，原esp保存在prev->thread.sp中，而新的ebp、esp指向next.task的堆栈底端即最高地址。

#### 3.3.3.4运行时复原堆栈

通过代码



对于一个state = 0的task进行了堆栈的复原并对原task的ebp、esp进行保存。



其中原ebp保存在原堆栈顶端，原esp保存在prev->thread.sp中，而新的ebp指向之前压入栈中的ebp即next.task的堆栈底端即最高地址，而新的ebp指向之前保存在next->thread.sp即next.task的堆栈顶端即最低地址。

# 四、实验总结

1、通过实验加深了对Linux内核的理解，并对时间片轮转的实现的细节更深入的学习。

2、在分析代码的过程中，培养了动态分析代码的能力与习惯，习惯决定命运这，对以后的学习与工作的帮助非常巨大。

3、在实验过程中，培养了对相关资料查询与自学的能力。

4、在分析过程中，与孟宁老师和王建民同学的讨论，对我自身的理解起到了决定与点睛的作用，让我体会到了与人合作、团队协同的重要性。

5、制作PPT与书写报告的过程中对Microsoft Office Visio的使用进行了学习。

# 五、实验中遇到的问题

## 5.1qemu的使用

对于Ubuntu 12.04LTS没有qemu命令，可以使用qemu-system-i386（32位）或qemu-system-x86\_64（64位）替换，可以建立一条连接sudo ln –s /usr/bin/qemu-system-i386 /usr/bin/qemu3.2，再使用qemu查看内核工作方式

## 5.2代码中的不易理解的问题

### 5.2.1$1f

$1f为cpu的eip的一个指令，在它没有被eip执行时，只是一个立即数值1f，而当它被直行至指的是label1即代码中的“1:”

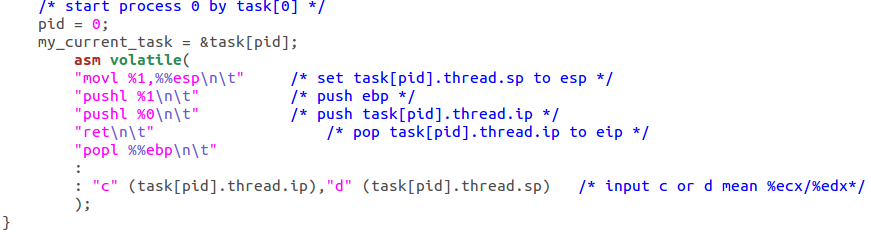
在初始化堆栈的函数中即next->state = -1时，将1f保存在prev->thread.ip中，其中只是一个立即数1f。

而在运行函数中即next->state = 0时，将1f保存在了prev->thread.ip，并将next->thread.ip压栈，因为state = 0，所以其next->thread.ip为if，在ret弹出时，把eip指向if，即“1:”，然后顺序执行，即执行popl %%ebp，将保存在栈中的%ebp还原给ebp寄存器。

### 5.2.2ret

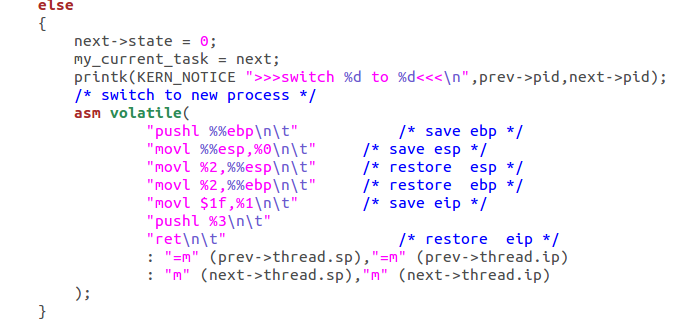
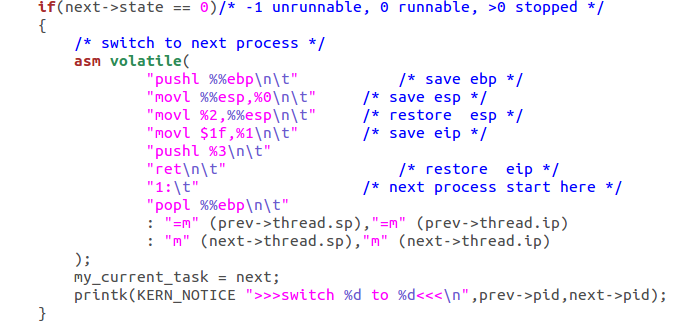
实验中，通过pushl与ret的组合实现了对eip执行指令的定向修改。

在my\_start\_kernel中的汇编代码中



将eip指向了task[0].entry即my\_process，这导致了下一句的popl %%ebp无法执行，但是为了书写规范，还是加上了这一句。

在其他两端汇编代码中



都是将eip指向了next->thread.ip,其中初始化堆栈代码指向的是next.entry即my\_process，而运行代码指向的是1f即“1：”。

# 六、参考文献

Linux操作系统分析课程主页

Homework1,mykernel.