哈尔滨工业大学计算机科学与技术学院
2016年秋季学期《操作系统》

Lab1：操作系统引导

姓名	学号	联系方式
匡盟盟	1143220116	kuangmeng@msn.com
樊晨霄	15S008199	18513534698

目　录

一、实验目的

- 熟悉hit-oslab实验环境；
- 建立对操作系统引导过程的深入认识；
- 掌握操作系统的基本开发过程；
- 能对操作系统代码进行简单的控制，揭开操作系统的神秘面纱。

二、实验内容

实验基本内容

1. 阅读《Linux内核完全注释》的第6章，对计算机和Linux 0.11的引导过程进行初步的了解；
2. 按照下面的要求改写0.11的引导程序bootsect.s；
3. 有兴趣同学可以做做进入保护模式前的设置程序setup.s。

改写bootsect.s

主要完成如下功能：

1. bootsect.s能在屏幕上打印一段提示信息“XXX is booting...”，其中XXX是你给自己的操作系统起的名字，例如LZJos、Sunix等（可以上论坛上秀秀谁的OS名字最帅，也可以显示一个特色logo，以表示自己操作系统的与众不同。）我们的操作系统名字：“MIC”，所以我们需要在启动时打印这样的一段信息：“MIC is booting…”。

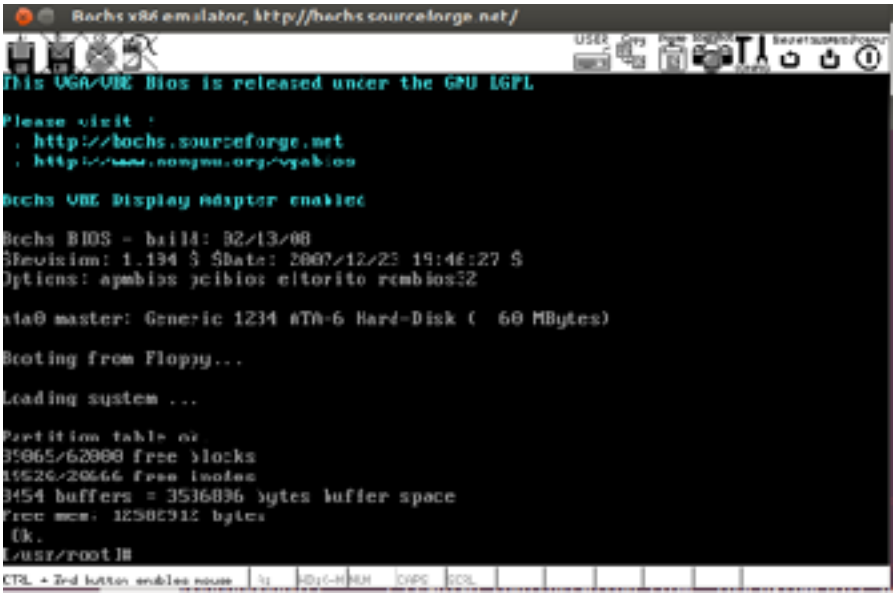
改写setup.s

主要完成如下功能：

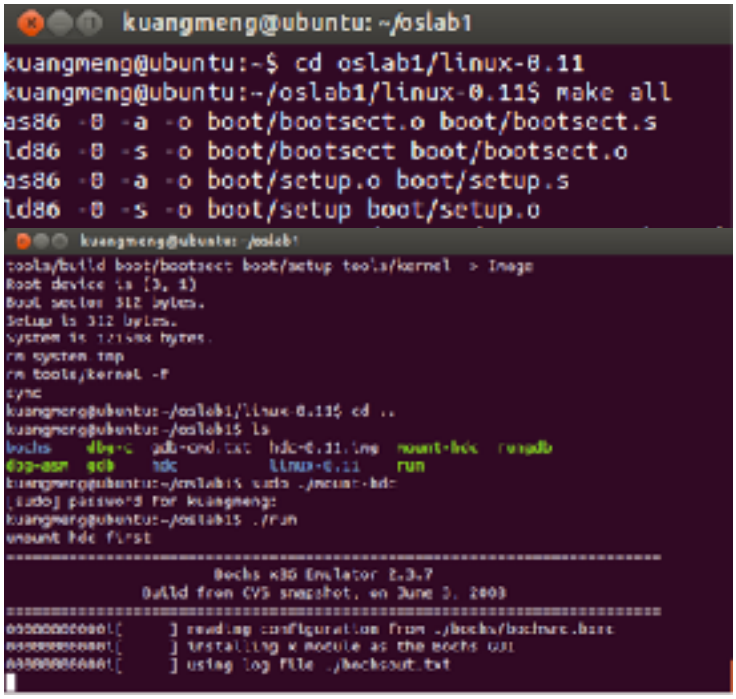
1. bootsect.s能完成setup.s的载入，并跳转到setup.s开始地址执行。而setup.s向屏幕输出一行"Now we are in SETUP"。
2. setup.s能获取至少一个基本的硬件参数（如内存参数、显卡参数、硬盘参数等），将其存放在内存的特定地址，并输出到屏幕上。
3. setup.s不再加载Linux内核，保持上述信息显示在屏幕上即可。

三、实验过程

1. 正常启动页面（为了体验操作系统linux-0.11的启动过程）：



相关过程如下截图（先进入oslab1/linux-0.11目录，编译“make all”，然后将bochs挂载，使用“./run”命令就可以得到上图所示的启动界面）：

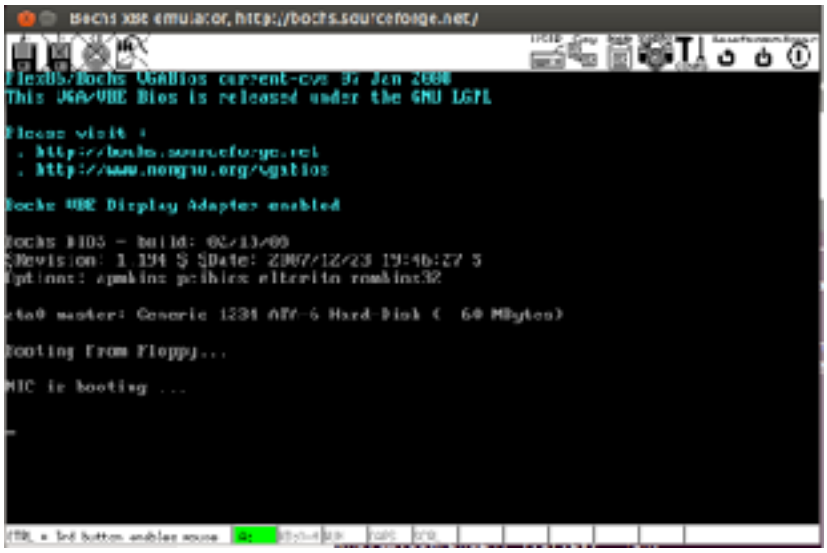


2. 在bootsect.s中加入死循环，并修改启动信息：

```
inf_loop:
   jmp  inf_loop                ! 后面不是正经代码了，得往回跳

msg1:
        .byte 13,10
        .ascii "MIC is booting ..."
        .byte 13,10,13,10
```

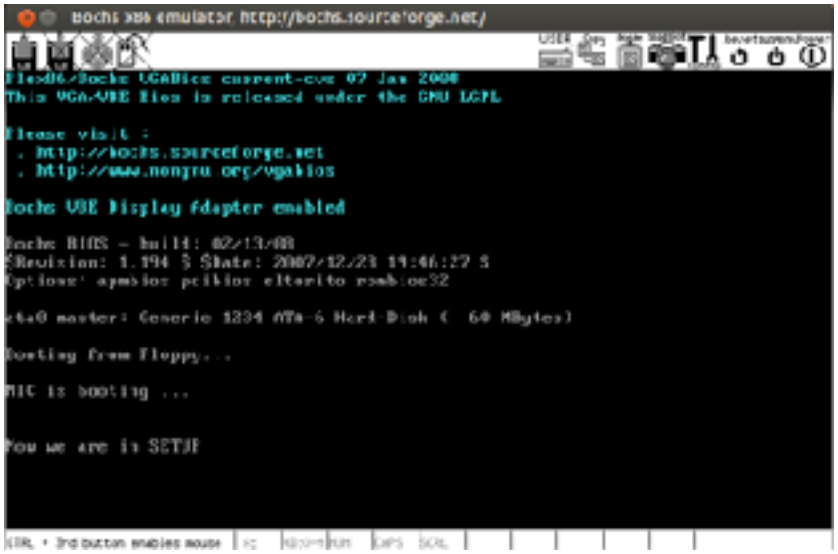
此时，再次按照“1”中的步骤运行，可以得到如下界面，至此完成屏幕输出功能：



3. 将bootsect.s中的死循环移除，加入到setup.s打印第一句话的位置，同时修改setup.s中的打印信息：

```
!此处添加setup.s向屏幕输入的代码
mov ah,#0x03
xor bh,bh
int 0x10          !先读光标位置，返回值在cx中
mov ax,#SETUPSEG
mov es,ax
xor bh,bh
int 0x10
mov cx,#25
mov bx,#0x0007
mov bp,#msg2
mov ax,#0x1301
int 0x10
inf_loop:
  jmp  inf_loop                ! 后面不是正经代码了，得杜回跳
msg2:
    .byte 13,10
    .ascii "Now we are in SETUP"
    .byte 13,10,13,10
```

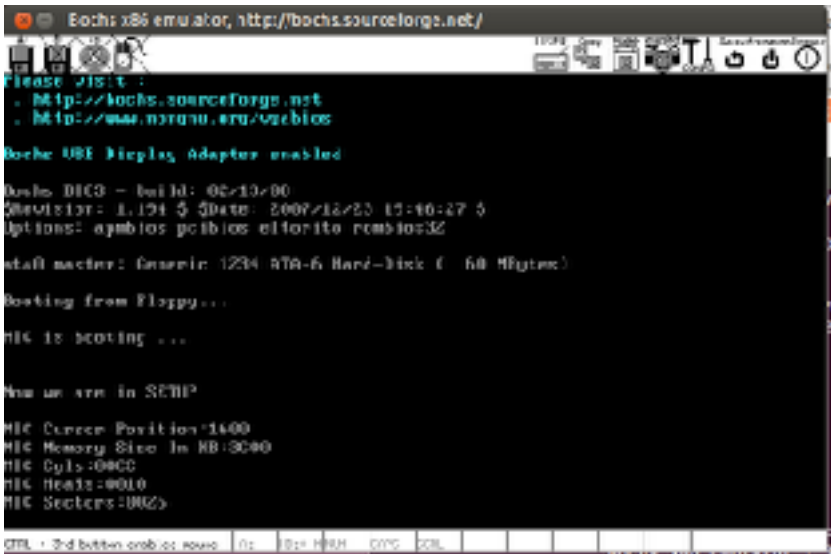
此时再次运行，可以得到如下截图所示的启动界面，至此完成bootsect.s读入setup.s功能：



4. 将死循环向下移（在setup.s打印出系统硬件信息之后）或直接移除，可以通过bootsect.s调用setup.s打印系统硬件信息（下图为部分代码截图）：

```
!以16进制方式打印机器参数
!先打印光标位置
print_cur:
  mov ah,#0x03
  xor bh,bh
  int 0x10          !先读光标位置，返回值在dx中
  mov ax,#SETUPSEG
  mov es,ax
  mov cx,#20
  mov bx,#0x0007
  mov bp,#msg3
  mov ax,#0x1301
  int 0x10
```

完成上述过程之后的启动界面如下图：



至此本实验完毕！

四、回答问题

1. **有时，继承传统意味着别手蹩脚。x86计算机为了向下兼容，导致启动过程比较复杂。请找出x86计算机启动过程中，被硬件强制，软件必须遵守的两个“多此一举”的步骤（多找几个也无妨），说说它们为什么多此一举，并设计更简洁的替代方案。**

答：几处被硬件强制，软件必须遵守的两个“多此一举”的步骤如下：

1. 计算机上电，BIOS初始化中断向量表后，会将启动设备的第一个扇区（即引导扇区）读入内存地址0x7c00（31KB)处，并跳转到此处开始执行。而为了方便加载主模块，引导程序首先会将自己移动到内存相对靠后的位置，如linux0.11的bootsect程序先将自己移动到0x90000(576KB)处。这样先移动是多此一举的。

   解决方案：在保证可靠性的前提下尽量扩大实地址模式下BIOS可访问的内存的范围，如引导扇区加载到0x90000等内存高地址处而不是0x7c00；或者不修改BIOS，在BIOS将引导扇区代码加载到0x07C00处并执行后，bootsect不复制自身，而是直接加载setup模块至0x90200处。不过在bootsect的执行过程中，需要在某些地方使用seg cs指令以指出下一语句的操作数在cs所指的段中。此时cs与ds、es的值不同。0x90000~0x90200留待setup模块存储参数使用。
2. 计算机上电后，ROM BIOS会在物理内存0处初始化中断向量表，其中有256个中断向量，每个中断向量占用4字节，共1KB，在物理内存地址0x000 - ox3ff处，这些中断向量供BIOS中断使用。这就导致了一个问题，如果操作系统的引导程序在加载操作系统时使用了BIOS中断来获取或者显示一些信息时，这1KB地址不能被覆盖。然而操作系统的主模块为了让其中代码地址等于实际的物理地址，需要将其加载到内存0x0000处。所以操作系统在加载时需要先将主模块加载到内存中不与BIOS中断向量表冲突的地方，之后可以覆盖中断向量表时才将其移动到内存起始处，如Linux0.11的System模块就是在bootsect程序中先加载到0x10000,之后在setup程序中移到0x0000处。 这样先加载到另外地方之后再移动到内存起始位置是多此一举的。

   解决方案：可以将BIOS中断向量表放到实模式下能寻址内存的其他地方，操作系统引导程序直接将操作系统的主模块读到内存的起始处。
3. bootsect模块将system模块加载到内存0x10000处，之后setup模块又将system模块向下移至内存0x00000处这一操作是多此一举的。

解决方案：在内核设计中，system模块的最大值设计为0x3000节，即3000*16=192KB,即使内核达到8*64KB=512KB的极限大小，0x10000~0x8FFFF的内存空间也足以存放下system模块。因此可以不必再将system模块向下移动。

实验心得

操作系统是控制其他程序运行，管理系统资源并为用户提供操作界面的系统软件的集合。操作系统身负诸如管理与配置内存、决定系统资源供需的优先次序、控制输入与输出设备、操作网络与管理文件系统等基本事务。

通过这次小实验，使我们更加了解Linux一些常用指令的操作以及其作用，了解了系统booting的基本过程，对于一个刚开始接触lniux操作系统的初学者来说非常有用，助于以后能够更进一步学习Linux操作系统。