**实验二：**

**加载执行COM格式用户程序的监控程序**

**实验目的：**

1、了解监控程序执行用户程序的主要工作

2、了解一种用户程序的格式与运行要求

3、加深对监控程序概念的理解

4、掌握加载用户程序方法

5、掌握几个BIOS调用和简单的磁盘空间管理  
**实验要求：**

1、知道引导扇区程序实现用户程序加载的意义

2、掌握COM/BIN等一种可执行的用户程序格式与运行要求

3、将自己实验一的引导扇区程序修改为3-4个不同版本的COM格式程序，每个程序缩小显示区域，在屏幕特定区域显示，用以测试监控程序，在1.44MB软驱映像中存储这些程序。

4、重写1.44MB软驱引导程序，利用BIOS调用，实现一个能执行COM格式用户程序的监控程序。

5、设计一种简单命令，实现用命令交互执行在1.44MB软驱映像中存储几个用户程序

6、编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性  
**实验内容：**  
 (1) 将自己实验一的引导扇区程序修改为一个的COM格式程序，程序缩小显示区域，在屏幕第一个1/4区域显示，显示一些信息后，程序会结束退出，可以在DOS中运行。在1.44MB软驱映像中制定一个或多个扇区，存储这个用户程序a。  
相似地、将自己实验一的引导扇区程序修改为第二、第三、第四个的COM格式程序，程序缩小显示区域，在屏幕第二、第三、第四个1/4区域显示，在1.44MB软驱映像中制定一个或多个扇区，存储用户程序b、用户程序c、用户程序d。  
 (2) 重写1.44MB软驱引导程序，利用BIOS调用，实现一个能执行COM格式用户程序的监控程序。程序可以按操作选择，执行一个或几个用户程序。解决加载用户程序和返回监控程序的问题，执行完一个用户程序后，可以执行下一个。  
 (3)设计一种命令，可以在一个命令中指定某种顺序执行若干个用户程序。可以反复接受命令。  
 (4)在映像盘上，设计一个表格，记录盘上有几个用户程序，放在那个位置等等信息，如果可以，让监控程序显示出表格信息。  
 (5)拓展自己的软件项目管理目录，管理实验项目相关文档

**实验环境：**

* Windows 10-64bit
* Vmware WorkStation 15 pro 15.5.1 build-15018445：虚拟机软件
* NASM version 2.13.02：汇编程序的编译器，在linux下通过sudo apt-get install nasm下载
* Oracle VM VirtualBox：一款开源的虚拟机软件
* Ubuntu-18.04.4:安装在Vmware的虚拟机上
* 代码编辑器：Visual Studio Code 1.44.2

**实验思路：**

首先我们可以根据实验要求列出我们所需要实现的程序如下：

|  |  |
| --- | --- |
| 程序类型以及名称 | 功能 |
| 引导程序：bootloader.asm | 加载监控程序和用户程序，加载完后跳转至监控程序 |
| 监控程序：oskernel.asm | 接收用户命令，执行相应的用户程序 |
| 用户程序a：a.asm | 在**第二象限**实现字符串的跳动 |
| 用户程序b：b.asm | 在**第一象限**实现字符串的跳动 |
| 用户程序c：c.asm | 在**第三象限**实现字符串的跳动 |
| 用户程序d：d.asm | 在**第四象限**实现字符串的跳动 |

由上表可知，我们实际上只需要设计3种程序：引导、监控和用户程序，四种用户程序实现的功能大同小异，只需要修改相应的变量即可。因此，接下来我们对这3种程序进行设计。

**程序设计：**

**准备工作：**

由于在各个程序中都有着打印字符串的功能需要我们实现，因此我们将该功能独立于一个宏程序（head.asm）中进行实现，然后通过调用它以达到打印字符串的目的。具体代码如下：

%macro PRINT\_IN\_POS 4

    pusha            ; 压栈

    mov ax, cs       ; 置其他段寄存器值与CS相同

    mov ds, ax       ; 数据段

    mov bp, %1       ; BP=当前串的偏移地址

    mov ax, ds       ; ES:BP = 串地址

    mov es, ax       ; 置ES=DS

    mov cx, %2       ; CX = 串长（=9）

    mov ax, 1301h    ; AH = 13h（功能号）、AL = 01h（光标置于串尾）

    mov bx, 0007h    ; 页号为0(BH = 0) 黑底白字(BL = 07h)

    mov dh, %3       ; 行号=0

    mov dl, %4       ; 列号=0

    int 10h          ; BIOS的10h功能：显示一行字符

    popa             ; 出栈

%endmacro

当我们需要对它进行调用时，只需要在开头加上下面一行代码即可：

%include "head.asm"

1. **用户程序的设计**

因为我们在实验一中已经完成了类似的字符跳动的程序，所以我们优先对该程序进行设计。从实验一中的代码中，我们可以得到一个有用的信息：

WIDTH equ 80    ;宽度

HEIGHT equ 25   ;高度

因为字符在触及边界时会反弹，从界面的高度和宽度，以及我们要设计4个象限的字符串跳动这几个因素来看，我们可以将界面均分为4个部分来代表4个程序的4个象限。具体数据如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 象限 | 上边界 | 左边界 | 下边界 | 右边界 |
| 第一象限 | -1 | 39 | 13 | 80 |
| 第二象限 | -1 | -1 | 13 | 40 |
| 第三象限 | 11 | -1 | 25 | 40 |
| 第四象限 | 11 | 39 | 25 | 80 |

借助上表，我们便可以通过改变实验一触碰边界的反弹条件来分别实现4个类似功能的用户程序了。如下给出用户程序a的一些数据：

   os\_left equ -1     ; 左边界

    os\_top equ -1      ; 上边界

    os\_right equ 40    ; 右边界

    os\_bottom equ 13   ; 下边界

    pos\_y equ 1       ; 起点所在列

    pos\_x equ 6       ; 起点所在行

因此，我们只需要修改四个边界，同时将字符的初始位置设置在合理的位置即可。

1. **引导程序的设计**

在实验课上，我们了解到引导程序主要的功能有三个：

1. 提示系统正在启动
2. 加载监控程序和用户程序
3. 将控制权交到监控程序去执行

课堂上，老师提供了引导程序的模板代码（myos.asm），里面包含了除加载监控程序和加载用户程序外的代码，部分关键代码如下：

Start:

    mov ax, cs                 ; 置其他段寄存器值与CS相同

    mov ds, ax                 ; 数据段

    mov bp, Message            ; BP=当前串的偏移地址

    mov ax, ds                 ; ES:BP = 串地址

    mov es, ax                 ; 置ES=DS

    mov cx, msglen             ; CX = 串长（=9）

    mov ax, 1301h              ; AH = 13h（功能号）、AL = 01h（光标置于串尾）

    mov bx, 0007h              ; 页号为0(BH = 0) 黑底白字(BL = 07h)

    mov dh, 0                  ; 行号=0

    mov dl, 0                  ; 列号=0

    int 10h                    ; BIOS的10h功能：显示一行字符

Message即是启动操作系统时的提示信息。通过调用BIOS的10h功能将其显示在屏幕上。

因此，我们需要补充加载监控程序和用户程序到内存中的代码，其中，加载监控程序的代码如下：

LoadOsKernel:                  ; 读软盘或硬盘上的若干物理扇区到内存的ES:BX处：

    mov ax,cs                  ; 段地址 ; 存放数据的内存基地址

    mov es,ax                  ; 设置段地址（不能直接mov es,段地址）

    mov bx, offset\_oskernel    ; 偏移地址; 存放数据的内存偏移地址

    mov ah,2                   ; 功能号

    mov al,1                   ; 扇区数

    mov dl,0                   ; 驱动器号 ; 软盘为0，硬盘和U盘为80H

    mov dh,0                   ; 磁头号 ; 起始编号为0

    mov ch,0                   ; 柱面号 ; 起始编号为0

    mov cl,2                   ; 起始扇区号 ; 起始编号为1

    int 13H                    ; 调用读磁盘BIOS的13h功能

具体的功能是：从扇区号（cl）为2的位置读取1个扇区（al）的大小，并放入offset\_oskernel当中（即加载监控程序）。类似地，加载用户程序的部分代码如下：

LoadUsrProg2:

    mov ax,cs                  ; 段地址 ; 存放数据的内存基地址

    mov es,ax                  ; 设置段地址（不能直接mov es,段地址）

    mov bx, offset\_usrprog2    ; 偏移地址; 存放数据的内存偏移地址

    mov ah,2                   ; 功能号

    mov al,2                   ; 扇区数

    mov dl,0                   ; 驱动器号 ; 软盘为0，硬盘和U盘为80H

    mov dh,0                   ; 磁头号 ; 起始编号为0

    mov ch,0                   ; 柱面号 ; 起始编号为0

    mov cl,5                   ; 起始扇区号 ; 起始编号为1

    int 13H                    ; 调用读磁盘BIOS的13h功能

需要修改的地方只是起始扇区号，读取扇区大小和存放数据的偏移地址。

1. **监控程序的设计**

根据键盘I/O中断调用的相关资料，我们得知程序读取至int 16h时中断，且当功能号AH=0时，从键盘读入字符送AL寄存器。执行时，等待键盘输入，一旦输入，字符的ASCII码放入AL中。若AL＝0，则AH为输入的扩展码。同时，将键盘的输入局限于1/2/3/4，分别执行第二、一、三、四象限的字符跳跃；若输入其他按键，则返回循环头部重新输入。具体代码如下：

Keyboard:

    mov ah, 0; Bochs: 0000:a173

    int 16h

    cmp al, '1'; 按下1

    je offset\_usrprog1   ; 执行用户程序b

    cmp al, '2'; 按下2

    je offset\_usrprog2   ; 执行用户程序a

    cmp al, '3'; 按下3

    je offset\_usrprog3   ; 执行用户程序c

    cmp al, '4'; 按下4

    je offset\_usrprog4   ; 执行用户程序d

    jmp Keyboard; 无效按键，用户需重新按键

至此，所有程序都已经准备好了。

**制作虚拟软盘：**

我们可以对制作好的7个程序分别nasm编译，然后用dd命令将他们一个个整合到一个软盘当中，但是我在查阅资料的时候发现了更好用的方法：使用Shell脚本进行整合，具体代码如下：

#!/bin/bash

output\_file="Condor\_os.img"

asm\_files=("bootloader" "oskernel" "b" "a" "c" "d")

rm -f ${output\_file}

for asm\_file in ${asm\_files[@]}

do

    nasm ${asm\_file}.asm -o ${asm\_file}.img

    cat ${asm\_file}.img >> "${output\_file}"

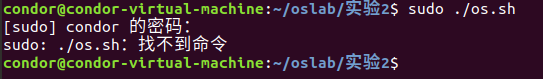
    rm -f ${asm\_file}.img

    echo "${asm\_file} finished"

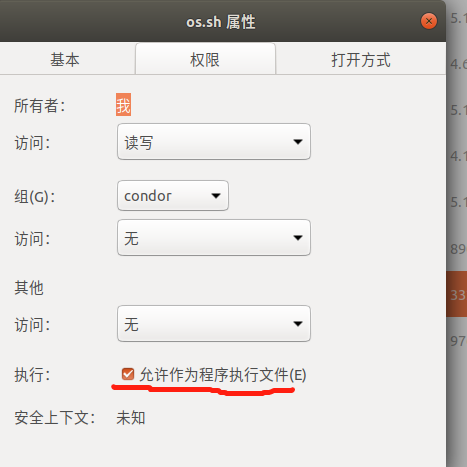
done

echo "${output\_file} finished."

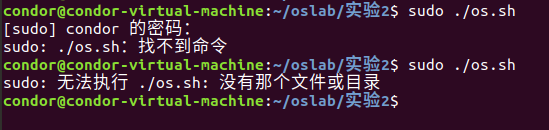
将上述代码另存为os.sh文件，通过以下命令运行，得到结果如下：



随之将错误敲在搜索引擎上，发现需要打开os.sh的属性，将其设置为允许作为程序执行文件即可。

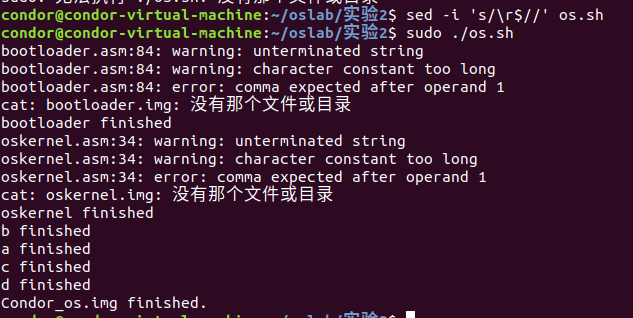


随即我再一次执行该文件，却依然发生了错误，如下：

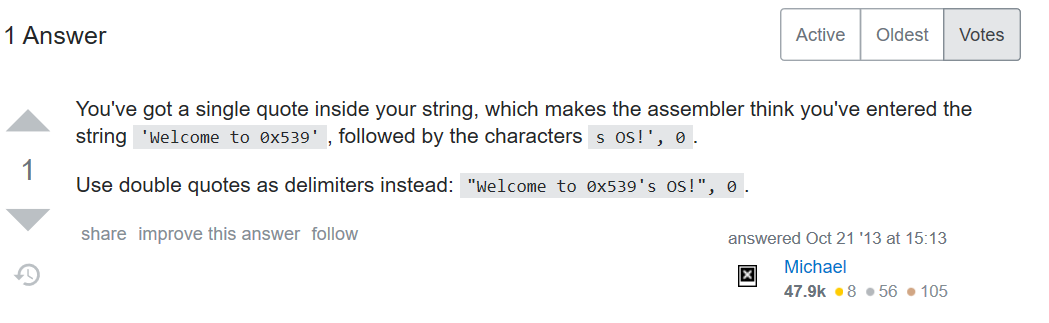


同样地，经过查阅资料得知，文件在Windows 下编辑过，在Windows下每一行结尾是\n\r，而Linux下则是\n，因此会有多出来的\r导致该错误。解决方法如下：

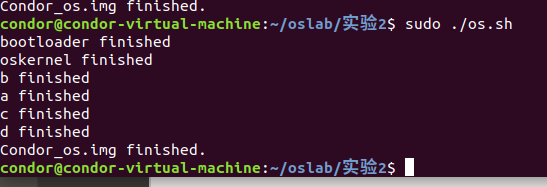
使用指令sed -i 's/\r$//' xxxxxxx.sh，上面的指令会把 [xxxxxxx.sh](http://xxxxxxx.sh/) 中的\r 替换成空白。执行完毕后我再次执行该文件，这次轮到要编译的文件出错了：



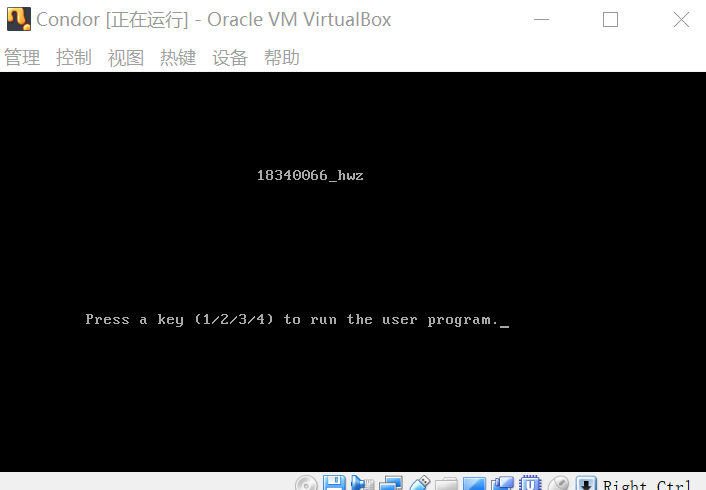
显然问题出在了bootloader.asm和oskernel.asm上，将错误信息查找一下，最终在StackOverflow上找到了答案：

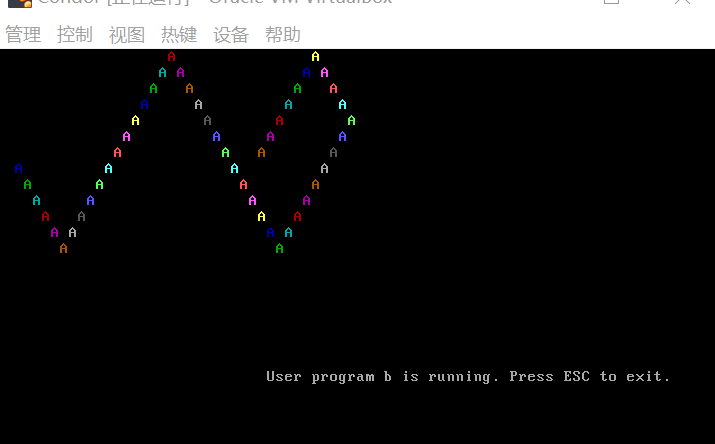


想不到……竟然是因为要打印的字符串里面多了一个单引号这种错误，将其改正后终于编译正确了。结果如下：



将导出的虚拟软盘放到裸机上跑一下得到结果如下：(仅展示部分，详情可查看附件中的os.mp4)







**实验心得：**

本次实验我最大的收获就是对引导程序、监控程序有了更深的了解，其功能、结构等，我都在本次实验中有深刻体会，对操作系统的感觉也没有原来那么抽象了，虽然还是有点晦涩难懂。其次就是对汇编语言的掌握更上一层楼吧（虽然感觉没什么很大的进步，但总归还是有一点点的）。