

操作系统 实验报告

——实验二 13349112 王钦 计科二班

一、实验目的：

1. 掌握COM格式的一种可执行程序的结构和运行要求。
2. 实现操作系统执行用户程序这一任务，理解操作系统产生的原始动机之一：自动执行用户程序，提高计算机的利用效率
3. 了解IBM_PC计算机硬件系统的内存布局和磁盘结构，设计一种简单的内存逻辑组织方法和磁盘组织方法，实现用户程序的存储。
4. 建立具有简单控制命令的批处理原型操作系统。

二、实验内容：

在实验一的基础上，进化你的原型操作系统，增加下列操作系统功能：

1. 可以加载COM格式的用户，多个用户程序顺序存放在软盘的连续扇区中，大小不等，但最大不超过5个扇区。
2. 允许用控制台命令指定按某种顺序执行一组（不少于3个）要执行的用户程序。比如，磁盘上有3个用户程序，可以用命令指定按某顺序运行其中1个、2个或3个程序。命令格式由你指定，并在实验报告中加以说明。

【助教对于实验内容第2点的解释：相当于有三个用户的程序，放在连续扇区内，你们提供控制台的调度程序以及格式。比如我输入 3 2 1 就会顺序式执行程序3 而后执行程序2，再然后执行程序1，你要根据我的输入调度程序，而且我还可只输入一个，即例如1或1 3或1 3 2等调度都可以执行。这项内容要求调用程序而不是调用函数】

三. 基础知识

IBM_PC 计算机硬件系统的内存布局和磁盘结构，软盘结构。COM 文件基础知识，如何设计 COM 文件。x86的指令集和NASM汇编程序的若干附加指令。

四. 实验环境工具

操作系统：Mac os X 10.10（Linux）

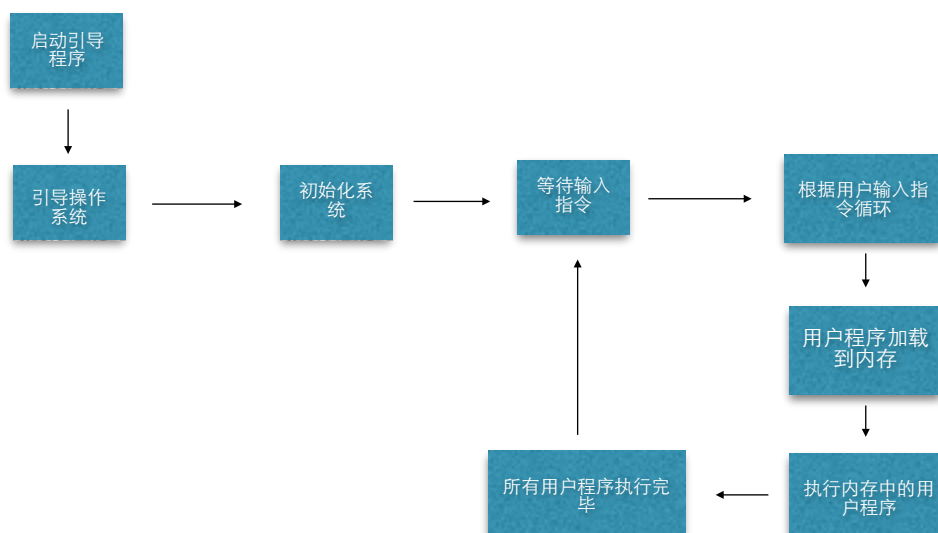
语言：NASM version 2.11.05

虚拟机平台：VMware Fusion pro 7.0.0

编辑器：Vim

所用命令行工具：xxd、dd

五. 程序流程图



程序流程图解释：

- 1 在软盘第一个扇区中加载mbr程序，加载到内存0x7c00中执行
- 2 mbr程序将第二个扇区中的操作系统程序加载到0x8100处执行
- 3 操作系统显示欢迎界面并等待用户输入指令序列
- 4 根据用户输入指令序列将后面三个扇区中先后加载到内存0xa700中并执行
- 5 用户程序执行完毕后返回操作系统，再次等待指令序列输入

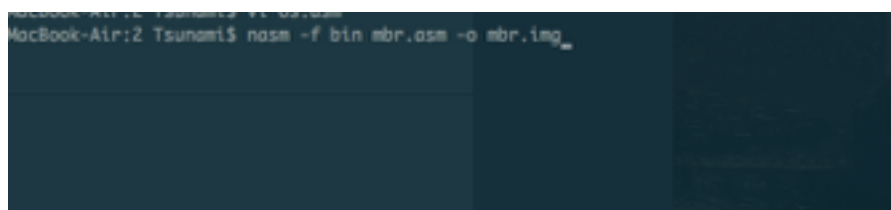
六 实验步骤

- 1 编写主引导扇区启动程序mbr.asm



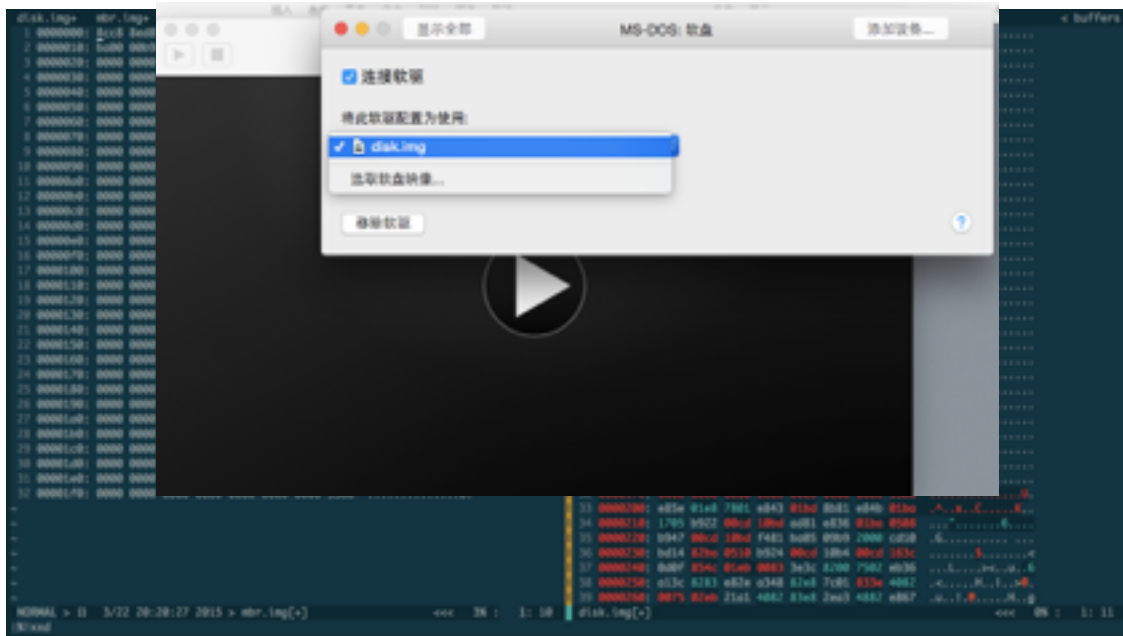
```
mbr.asm
1 org 7c00h
2 mov ax,cs ;代表用户输入第一个字
3 mov ds,ax ;1 为用户已经输入的
4 mov es,ax ;为所在扇区号。
5
6 ;load os to mem
7 xor ax,ax
8 mov es,ax ;made es zero,ex:bx is addr of mem
9 mov bx,8100h
10 mov ax,0201h
11 mov dx,0
12 mov cx,0002h;扇区号为2
13 int 13h
14
15 jmp 8100h
16
17 times 510-($-$$) db 0
18 db 0x55,0xaa
19
20
```

- 2 编译mbr程序生成img文件，并复制到软盘的第一个扇区中

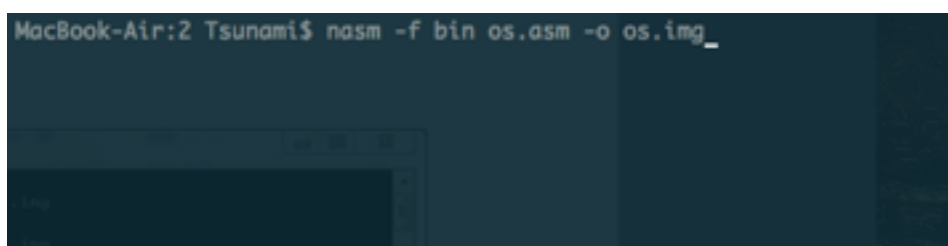


```
MacBook-Air:2 Tsunami$ nasm -f bin mbr.asm -o mbr.img_
```

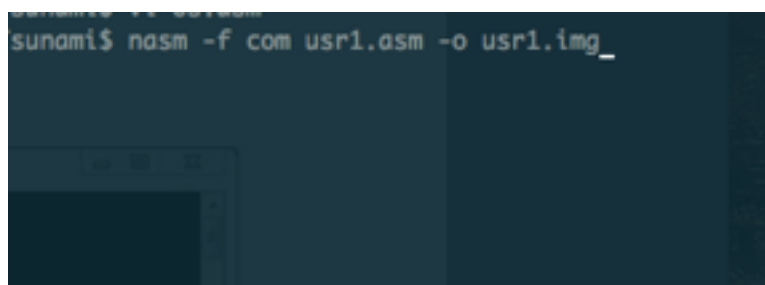
复制代码：



3 编写操作系统程序os.asm,编译后复制到软盘的第二个扇区



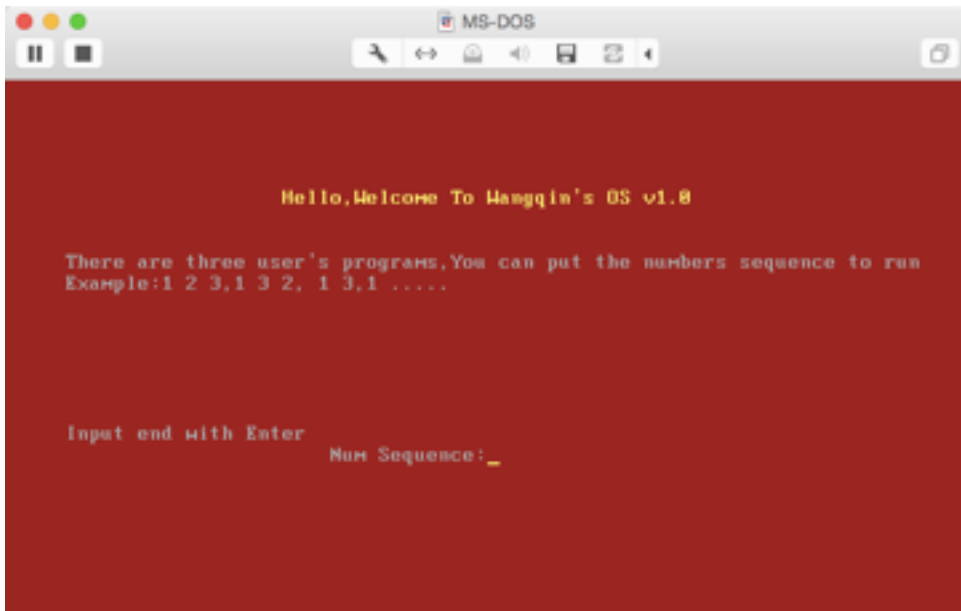
4 分别编写三个用户程序usr1.asm,usr2.asm,usr3.asm 并将代码复制到软盘的第三,四,五的扇区中



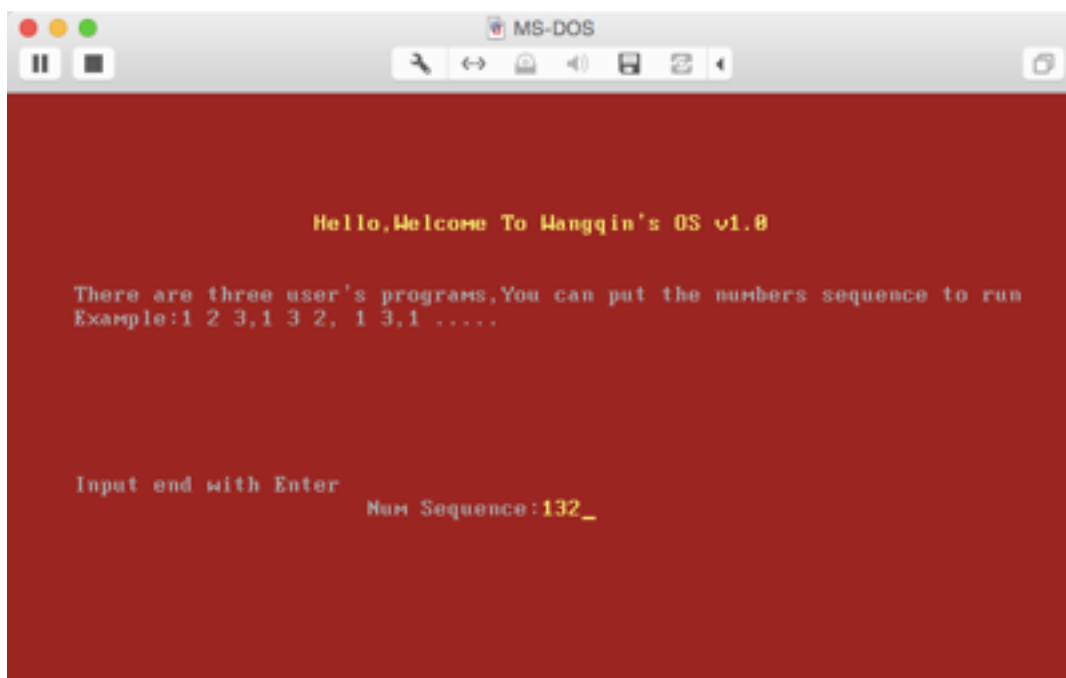
5 启动虚拟机，选择启动软盘为disk.img(即是前面被复制的软盘)，测试功能

效果图:

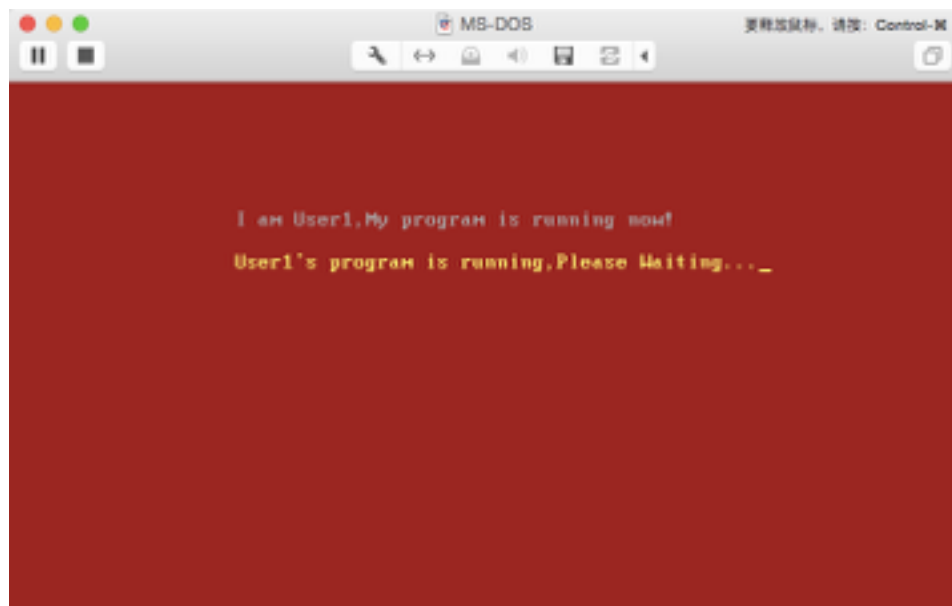
操作系统加载成功:



输入指令132，代表执行顺序执行用户程序132，键入回车符开始

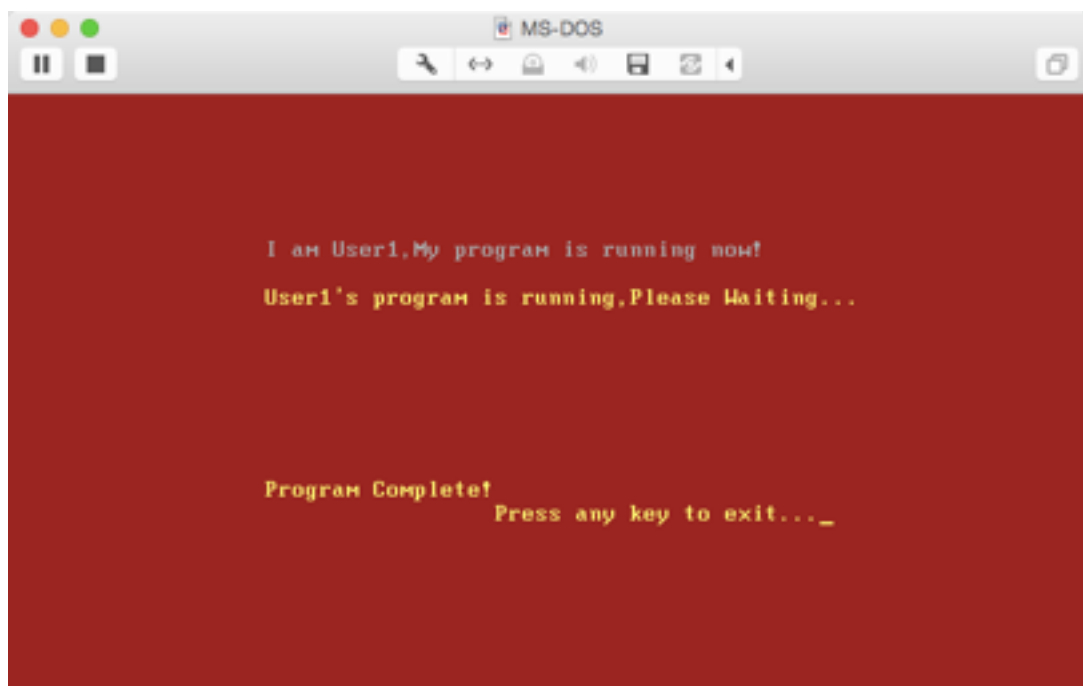


执行用户程序一，

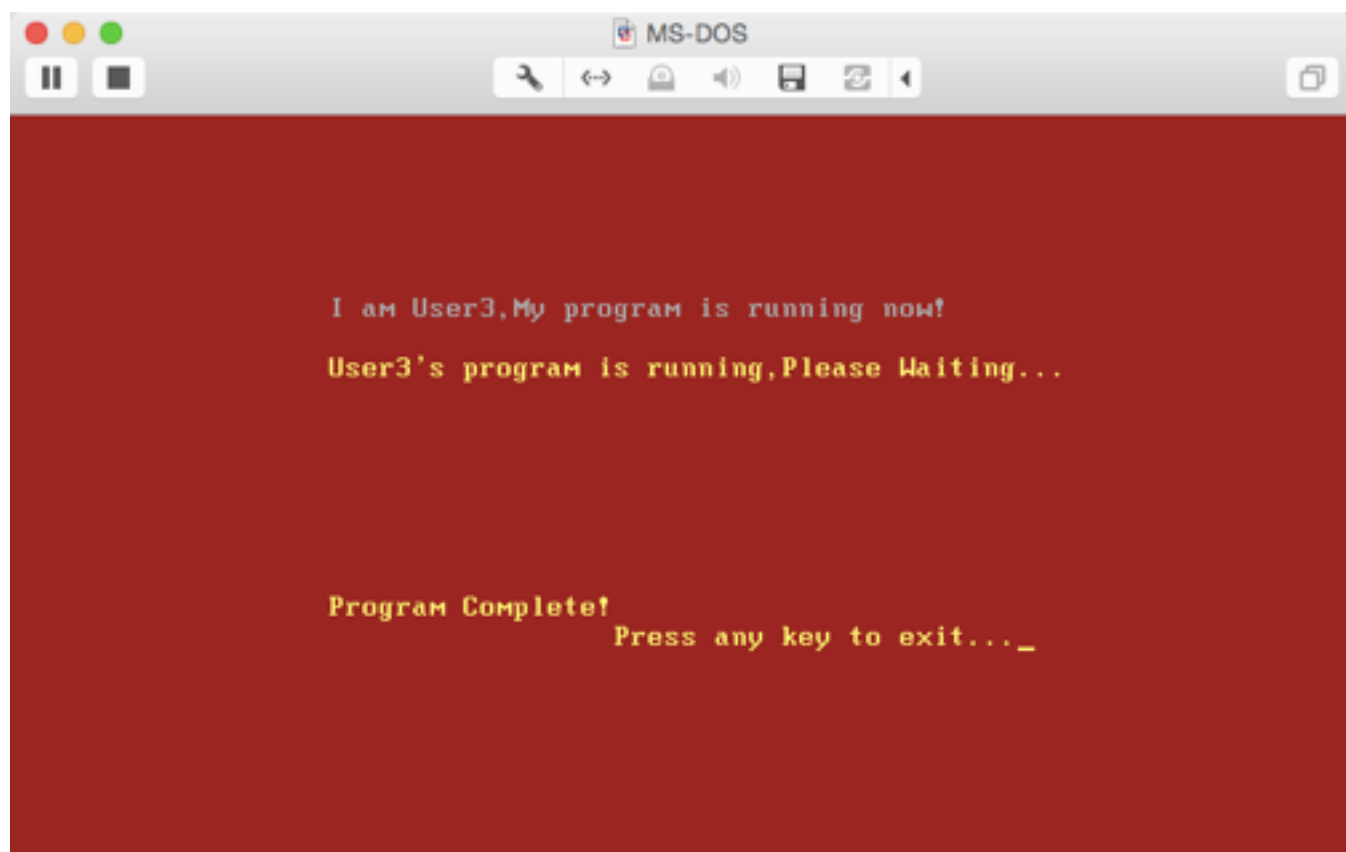
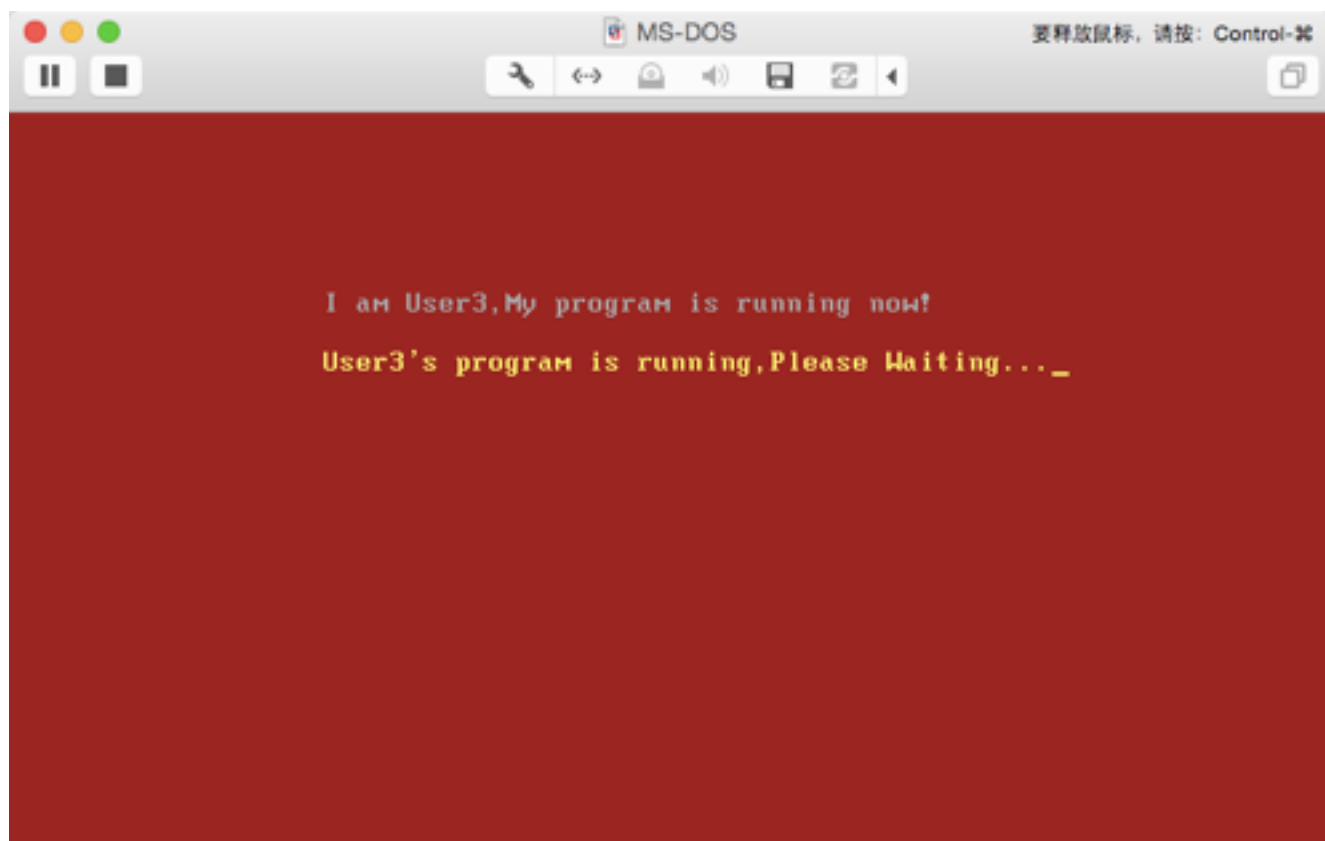


正在执行程序1

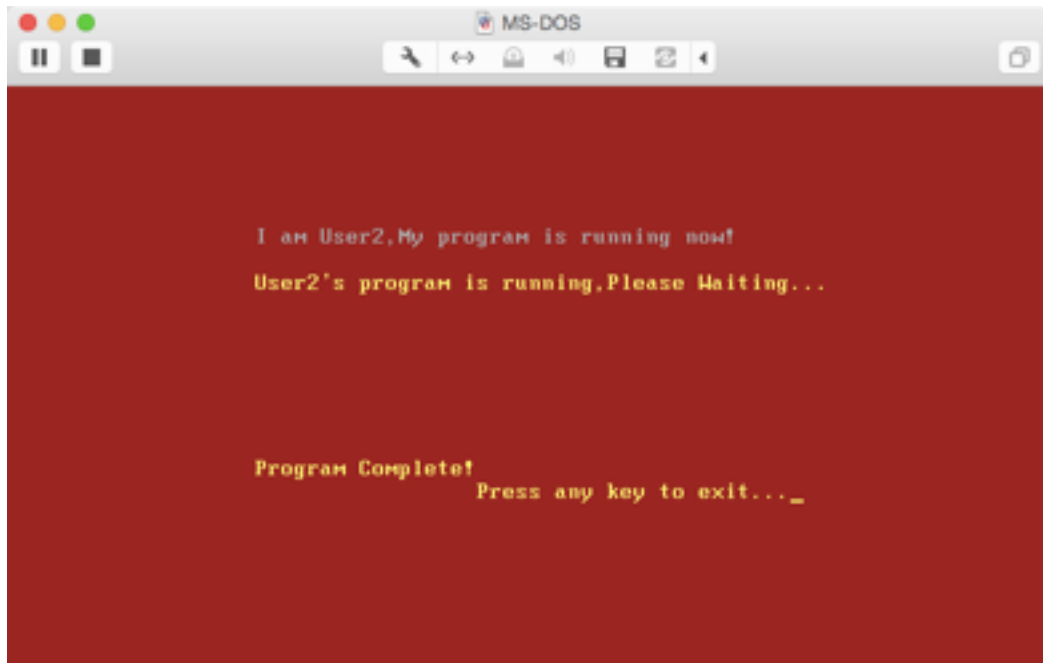
程序1执行完毕，输入任意键退出，开始执行下一个程序



开始执行用户程序3:



执行用户程序2



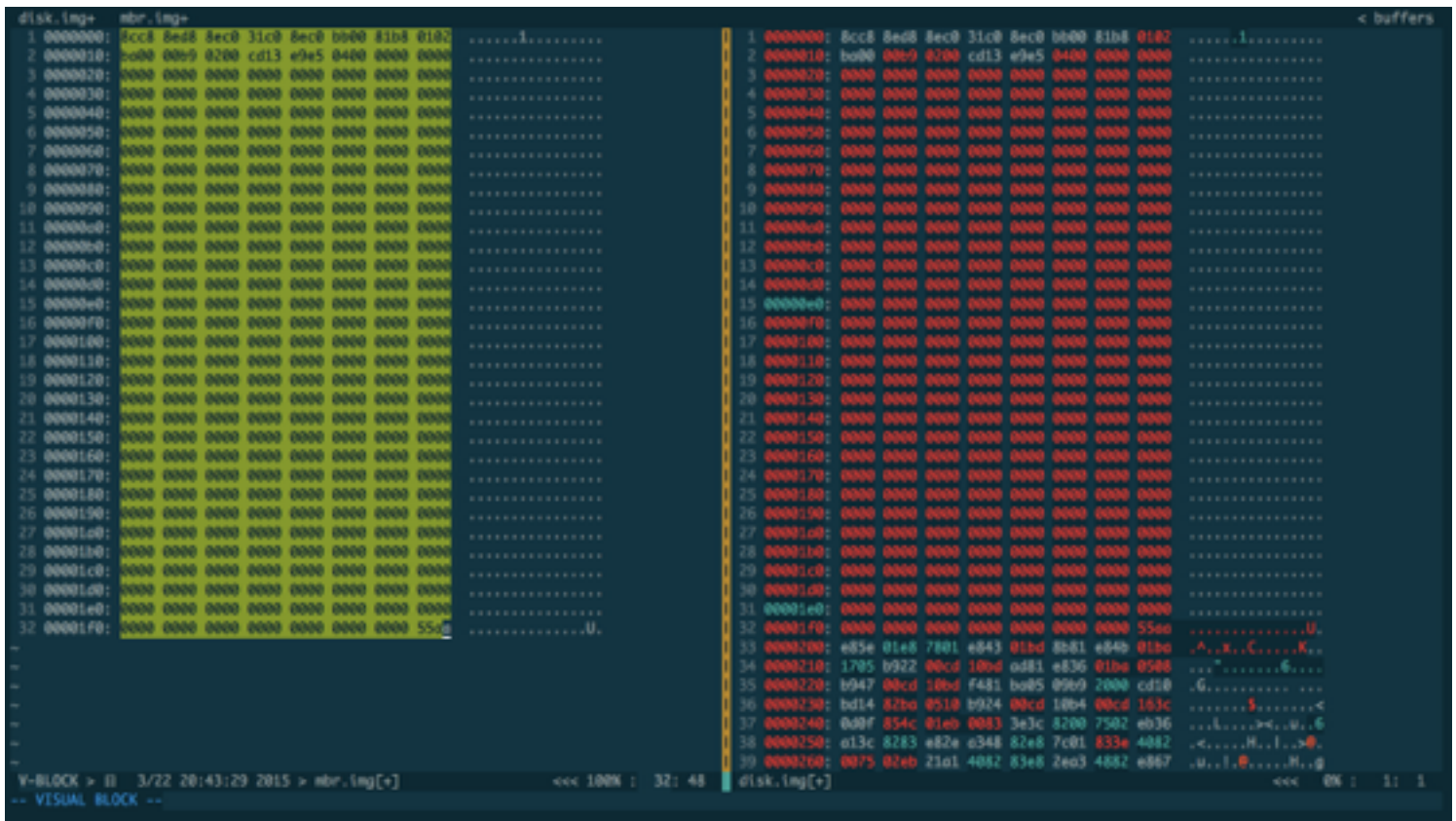
序列132执行完毕，回到操作系统界面



类似的还可以执行序列132, 13, 21, 2等……
二进制代码软盘粘贴说明:

- 1 在Linux下使用 vim -b disk.img 打开软盘
- 2 在vim的普通模式下输入指令%!xxd,显示软盘的十六进制代码
- 3 普通模式下输入vsp mbr.img 打开引导扇区程序并执行2的步骤, 显示出十六进制代码
- 4 光标定位到mbr要被复制的十六进制代码的右上角, 按下ctrl+v,开始按列进行选择
选择完毕后按下y
- 5 ctrl+w + -> 转移到软盘的界面中同样定位到要被粘贴区域的右上角, 按下ctrl+v进行选择, 选择完毕后输入大写P, 完成粘贴

图:
黄色区域表示按列进行选择,左边为mbr引导程序, 右边为软盘。



七 程序主要模块和变量

第一部分引导程序:

mbr.asm:

```
1 org 7c00h      ;加载到7c00初执行
2 mov ax,cs
3 mov ds,ax
4 mov es,ax
5
6 ;load os to mem 加载操作系统
7 xor ax,ax
8 mov es,ax ;made es zero,ex:bx is addr of mem
9 mov bx,8100h   ;加载到8100h
10 mov ax,0201h
11 mov dx,0
12 mov cx,0002h;扇区号为2
13 int 13h
14
15 jmp 8100h
16
17 times 510-($-$$) db 0 ;剩余填充
18 db 0x55,0xaa
```

第二部分 操作系统:

os.asm

初始化模块: 屏幕背景变为红色, 显示欢迎信息和一些基本的字符

```
1 OS_Start:
2
3 org 0x8100;加载到0x8100执行
4 ;org 0x7c00
5 ;style BEGIN=====
6
7  call init;操作系统初始化
8  call initvar;变量初始化
9
10  call print_init
11  mov bp,menu_msg ;str's offset
12  call print_style_header
```

```

13  mov dx,0517H; setting char position
14  mov cx,menu_msg_1;setting string'slength
15  int 10h ;display
16
17  mov bp,menu_msg2  ;str's offset
18  call print_style
19  mov dx,0805H
20  mov cx,menu_msg_l2
21  int 10h
22
23  mov bp,msg3 ;str's offset
24  mov dx,0905H
25  mov cx,msg_l3
26  int 10h
27
28  mov bp,msg5 ;str's offset
29  mov dx,1005H
30  mov cx,msg_l5
31  int 10h
32
33 ;style END=====

```

监听键盘输入模块：使用16号终端来监听键盘的输入，并判断是不是回车，如果是就开始执行输入的序列，不是则继续监听并将其显示在屏幕上

```

;style END=====
34 LISTEN:
35  mov ah,0x00; listen keyboard
36  int 0x16
37  cmp al,0x0d
38
39
40  jne display
41
42  jmp RUN_USER

```

变量定义初始化：a表示用户键入的第一个程序，b表示第二个，c表示第三个
now_shanqu表示当前要被加载的扇区编号，msg定义了一些系统初始化的字符串，

count表示用户键入的序列长度，判断超过三个就不再显示，以及用来保存键入的字符到a b c变量中

```
;=====data=====
=====
81 menu_msg:
82  db `Hello,Welcome To Wangqin's OS v1.0`
83 menu_msg_l equ ($-menu_msg)
84
85 menu_msg2:
86  db `There are three user's programs,You can put the numbers sequence to
run`
87 menu_msg_l2 equ ($-menu_msg2)
88
89 msg3:
90  db 'Example:1 2 3,1 3 2, 1 3,1 ....'
91 msg_l3 equ ($-msg3)
92
93 msg5:
94  db `Input end with Enter\n Num Sequence:`
95 msg_l5 equ ($-msg5)
96
97 var:    ;程序所用到的一些临时变量
98  count dd 0
99  a dd 0
100  b dd 0
101  c dd 0
102  now_shanqu dd 0
103
```

用户程序运行模块：首先判断第一个程序是否要求执行，然后加载到内存执行，结束后，在判断第二个。。。第三个

```
44 RUN_USER:
45
46  cmp word [a],0 ;第一个程序
47  jne exea
48  jmp RUN_OVER  ;没有则结束
49  exea:
50  mov ax,[a]
```

```

51  sub ax,46D ;扇区偏移, 前两个扇区中分别放着引导和操作系统 [a]-48 +2 = [a]-
46 [a] is char not int
52  mov [now_shanqu],ax ;ax is temp
53  call run
54
55  cmp word [b],0 ;第二个程序
56  jne exeb
57  jmp RUN_OVER
58  exeb:
59  mov ax,[b]
60  sub ax,46D ;扇区偏移, 前两个扇区中分别放着引导和操作系统
61  mov [now_shanqu],ax ;ax is temp
62  call run
63
64  cmp word [c],0 ;第三个程序
65  jne exec
66  jmp RUN_OVER
67  exec:
68  mov ax,[c]
69  sub ax,46D ;扇区偏移, 前两个扇区中分别放着引导和操作系统
70  mov word [now_shanqu],ax
71  call run
72

RUN_OVER:
74
75  jmp OS_Start;所有程序执行完毕返回操作系统
76

```

执行某个程序：首先清屏，然后加载到内存，跳转到0xa700中执行

```

run:                ;执行某个用户的程序,具体扇区在[now_shanqu] 中
192  call init ;清屏仅仅清屏而已
193 ;-----加载-----
194  mov ax,0
195  mov es,ax
196  mov bx,0xa700;加载到内存e000
197  mov ax,0201H;
198  mov dx,0
199  mov ch,0
200  mov cl,[now_shanqu];要被加载的扇区号
201  int 13h;加载
202 ;-----执行-----

```

```
203 call word 0xa700 ;not dword
204 ret
```

第三部分：用户程序

三个用户程序基本都类似，只介绍第一个用户程序

usr1.asm

显示字符串，并且调用延时函数表示正在执行

```
1 org 0xa700 ;加载到 e0000内存中执行
2 ;org 0x7c00 ;加载到 e0000内存中执行
3
4 mov ax,cs
5 mov es,ax
6 mov ds,ax ;es ,ds = cs
7
8 mov bp,msg
9 mov cx,msg_l
10 mov ax,1301h ;01 只有字符串
11 mov bx,71D ;Bh is font color
12 mov dx,0613h ;position
13 int 10h
14 ;delay time function
15 mov bp,msg2
16 mov cx,msg2_l
17 mov ax,1301h ;01 只有字符串
18 mov bx,78D ;Bh is font color
19 mov dx,0813h ;position
20 int 10h
21
22
23 call delay
```

延时函数： 模拟执行时间

```

57 delay:
58     mov dx,00
59     timer2:
60         mov cx,00
61         timer:
62             inc cx
63             cmp cx,60000D
64             jne timer
65             inc dx
66             cmp dx,60000D
67             jne timer2
68     ret

```

```

71
72 times 512-($-$$) db 0 ;填充剩余扇区0

```

结束，键入任意键返回操作系统，执行下一个任务

```

31
32 ;LISTEN_EXIT----
33 mov ah,0x00
34 int 0x16
35
36 ;-----
37
38 ret ;手动添加ret 结束后返回操作系统
39
40

```

八 实验心得

通过本次我了解了计算机操作系统的基本原理和开机流程，通过手动编写从磁盘加载到内存的程序了自己动手设计了内存管理和磁盘内部组织管理。加深了我对操作系统这门课的认识，同时也加强了自己编写NASM汇编语言的能力和调试的能力，成功装上了BOCHS调试工具，但是bochs在linux仍有一些问题现在正在解决。本次实验中我第一次调用了BIOS中断来实现监控用户键盘输入和显示字符串等功能，对NASM内部内存分段的机制了解的更加透彻同时掌握了一些NASM语言的DEBUG方法。学会了如何使用VIM进行磁盘映像文件的修改和复制粘贴，学会了如何使用13中断来加载代码到内存，基本清楚了磁盘中扇区的存储方式。

可改进之处：听说在linux下可以mount一个软盘镜像，然后将其他文件复制到这里来完成复制粘贴，而不需要复制16进制的代码，这种方式可以研究一下。本次实验的原型系统中并没有文件系统，可加入文件系统的磁盘组织方法。操作系统代码差不多快到了一个扇区大小，要考虑超过一个扇区的情况。另外对于用户输入的序列来说，没有考虑输入其他的字符除了123的情况