

中山大学数据科学与计算机学院本科生实验报告

(2020 学年春季学期)

课程名称：操作系统实验

任课教师：凌应标

助教：

年级&班级	2018 级 1 班	专业(方向)	计算机科学与技术(大数据方向)
学号	18308045	姓名	谷正阳
电话	13355426001	Email	Guzy0324@163.com
开始日期	2020.5.1	完成日期	2020.5.6

一、实验题目

操作系统 实验 2

二、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

三、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

四、实验内容：

1. 实验步骤

(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

2. 实验原理

1. 监控程序执行用户程序的主要工作：

运行监控程序→加载用户程序→运行用户程序→运行监控程序

2. COM/BIN/EXE 可执行的用户程序格式和运行要求：

<https://www.cnblogs.com/arkhe/articles/2615928.html>

3. 掌握加载用户程序方法：

使用 BIOS 调用 int 13 将映像上指定扇区的用户程序加载到指定位置的内存中

4. BIOS 调用和简单的磁盘空间管理：

https://blog.csdn.net/weixin_37656939/article/details/79684611

5. bochs 使用：

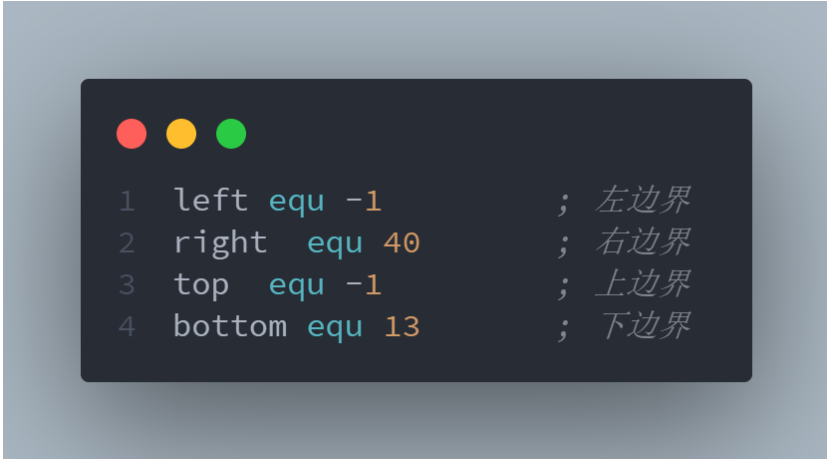
<https://www.cnblogs.com/jikebiancheng/p/6160337.html>

五、实验结果

- 1.

- a. stoneN.asm，边界未定义，扩展性不够强，因而增加边界定义（修改后：

stoneN00_not_used.asm



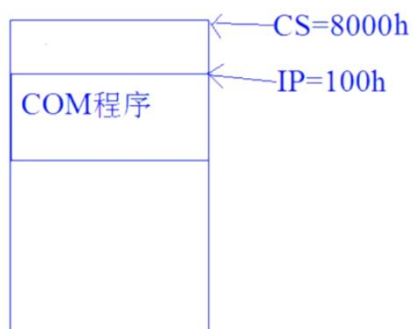
```
1 left equ -1          ; 左边界
2 right equ 40          ; 右边界
3 top equ -1           ; 上边界
4 bottom equ 13         ; 下边界
```

- b. stoneN00_not_used.asm如果打到角上,无法原路反弹,原因是老师给的stoneN.asm只考虑碰边界的情况,未考虑碰角的情况。因而重写运动轨迹模块为横向判断一次左右,纵向判断一次上下

```
1  drx:
2      cmp byte[dirx],Rt
3      jz  rt          ; if dirx == Rt goto rt
4  lt:
5      dec word[x]      ; else x--
6      cmp word[x],left
7      jnz dry          ; if x != left goto dry
8      add word[x],2     ; else x += 2
9      mov byte[dirx],Rt ; dirx = Rt
10     jmp dry          ; goto dry
11  rt:
12     inc word[x]       ; x++
13     cmp word[x],right
14     jnz dry          ; if x != right goto dry
15     sub word[x],2     ; else x -= 2
16     mov byte[dirx],Lt ; dirx = Lt
17  dry:
18     cmp byte[diry],Dn
19     jz  dn          ; if diry == Dn goto dn
20  up:
21     dec word[y]       ; else y--
22     cmp word[y],top
23     jnz show          ; if y != top goto show
24     add word[y],2     ; else y += 2
25     mov byte[diry],Dn ; diry = Dn
26     jmp show          ; goto show
27  dn:
28     inc word[y]       ; y++
29     cmp word[y],bottom
30     jnz show          ; if y != bottom goto show
31     mov byte[diry],Up ; diry = Up
32     sub word[y],2     ; else y -= 2
```


- b. stoneN00.asm中org的问题：我最初是将其加上org 0x8100，虽然能正常运行，但是不符合COM用户程序的要求的，COM用户程序要求偏移100h。这个问题从老师的ppt中，我得到了解答：

例如，COM程序被加载到内存8000:100h开始的位置，那么CS=8000h,IP=100h,可用jmp 8000h:100h跳过去。



是段寄存器出了错（后面另一个bug对此有验证

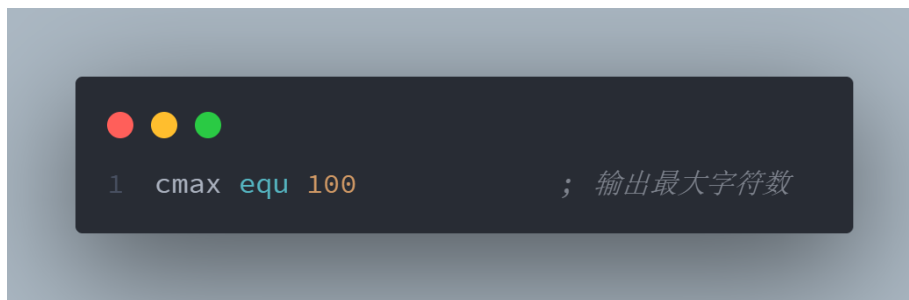
而jmp 8000h:100h是可以自动修改段寄存器为8000h的

然而另一方面，段地址16位偏移量4位，一共20位。如此8100h对应段地址应为810h，

而又有org 100h，所以段地址应为800h。综上，应该为



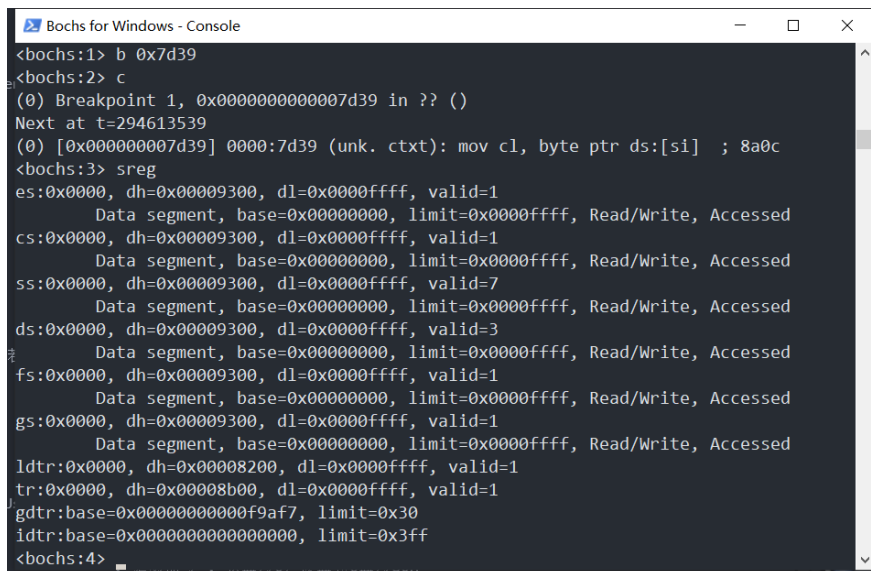
- c. 用户程序无法结束，定义一个输出字符数最大值，如下：



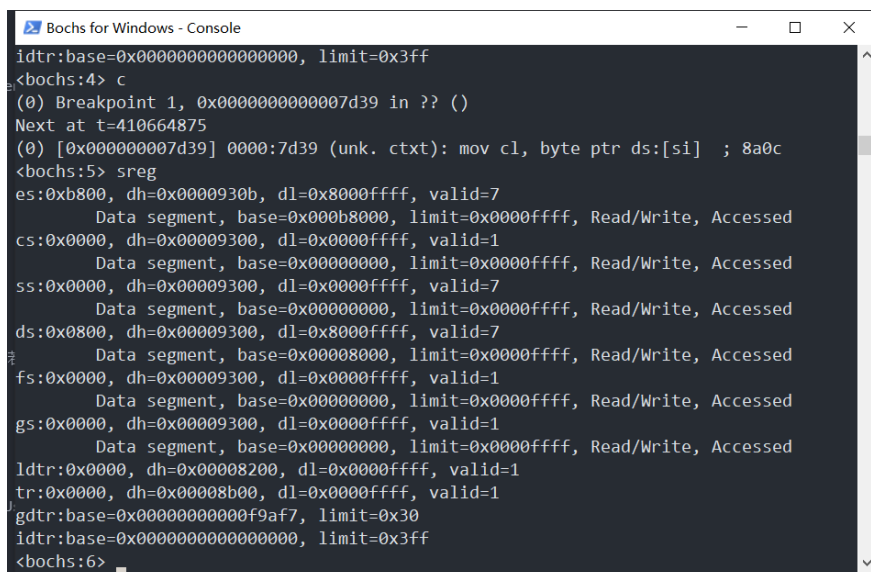
为了切换到下一个，将jmp改为call，加上retf

注意：retf不会改变DS和ES的值

call前：



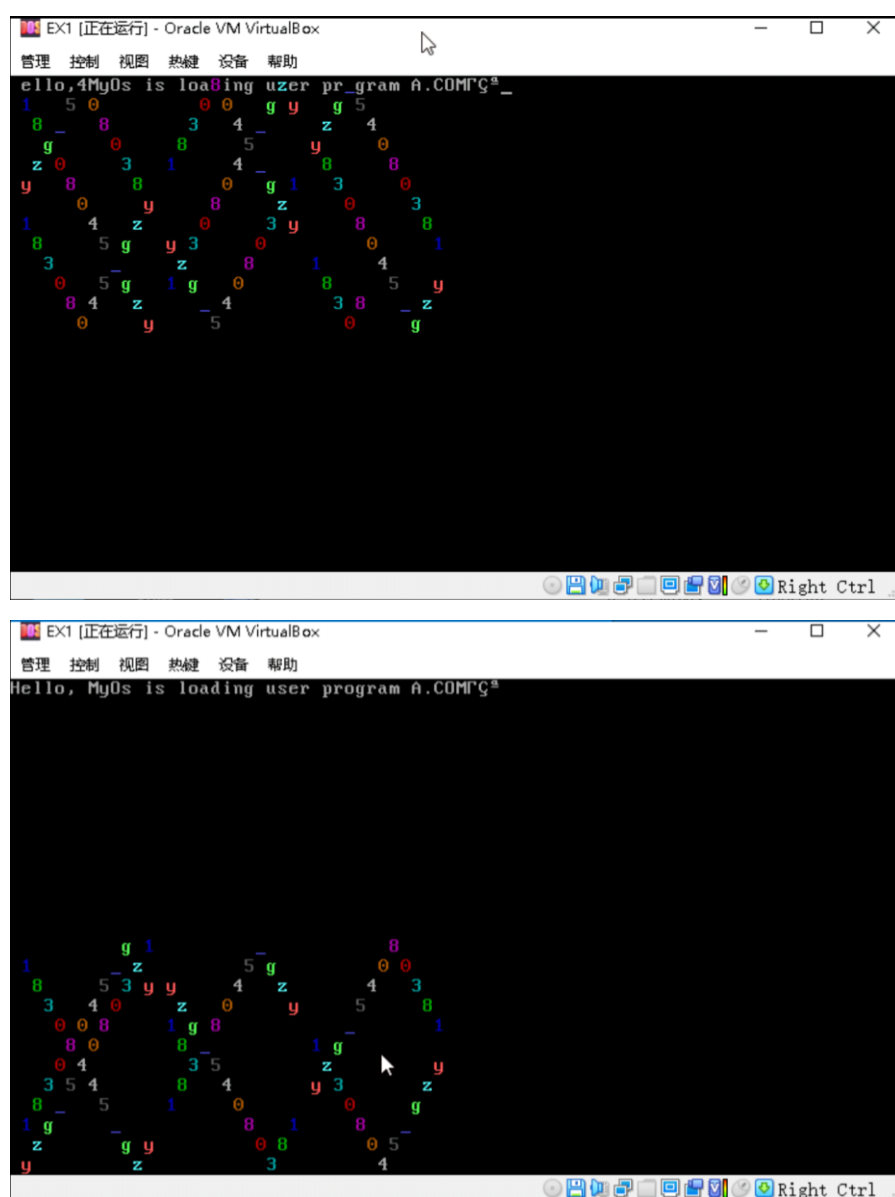
call后：



所以需要retf后手动修改:

```
1  call 800h:100h
2  mov ax, 0
3  mov ds,ax
4  mov es,ax ; 设置段地址 (不能直接mov es,段地址)
5  inc si
```

e. 结果:



3. 使用int 10调整光标，打印字符，int 13接受字符的服务模拟输入的感觉：

说明：

↑：回到句首

↓：到句尾

←：左移一位

→：右移一位

退格：删除光标后的一个字符

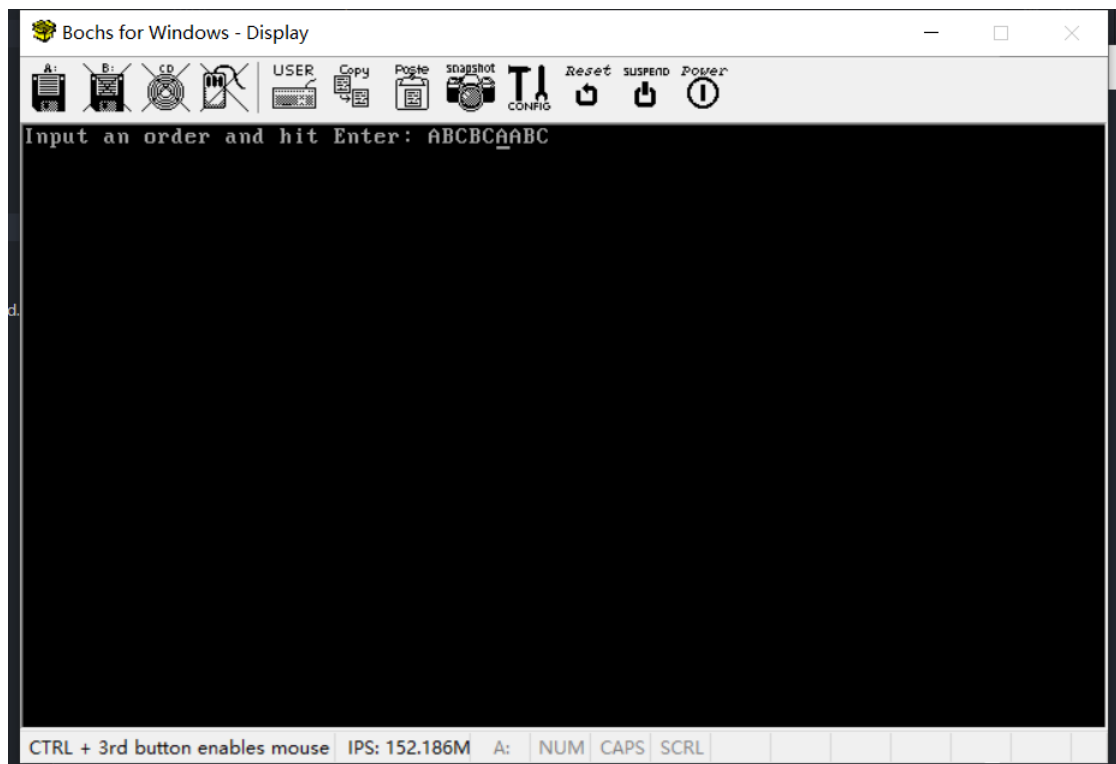
回车：执行字符串对应的用户程序序列

指定字母（ABCD）：显示，并作为序列

有效其他字符无效

执行完毕后，会记忆上次执行的序列


效果：



反思:


其中字符串用连续存储, 因为要频繁插入删除, 链接存储可能会更高效

4. a. 构建用户程序和位置的映射表



```
1 ProgramInfo:
2     db 'G'
3     db '2'
4     db 'U'
5     db '3'
6     db 'Z'
7     db '4'
8     db 'Y'
9     db '5'
10 TotalNum:
11     db ($-ProgramInfo) / 2
```

b. 更改判断输入是否有效 (如是否是ABCD左右上下退格)

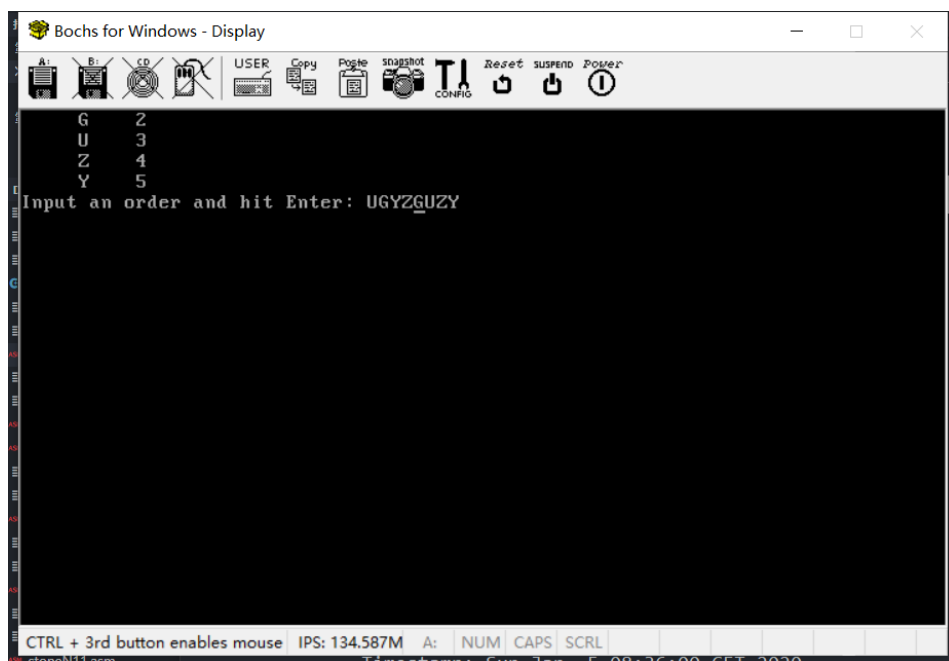


```
1     mov di,ProgramInfo
2 Defau:
3     cmp di,TotalNum
4     jz Input
5     cmp al,byte[di]
6     jz ValidProgram
7     add di,2
8     jmp Defau
```

- c. 更改映射算法为遍历整个表

```
1      mov cl,byte[si]           ; 起始扇区号 ; 起始编号为1
2      mov bx,ProgramInfo
3  MapLoop:
4      cmp cl,byte[bx]
5      jz SetSec
6      add bx,2
7      ; 前面有效输入的原则, 已经保证了一定可以找到
8      jmp MapLoop
9  SetSec:
10     mov cl,byte[bx+1]
11     sub cl,'0'
```

- c. 并在开头显示表,



五、实验感想

深刻理解了引导扇区程序实现用户程序加载 COM, 且更加深入地了解 nasm 的部分语法 (如 BIOS 中断, jmp, call) 和 bochs 的使用, 且通过 nasm 一定程度上实现了简单的 IO 操作。

附录 (流程图, 注释过的代码):

代码见

https://github.com/guzhegnyang/operating_system_experiment.git