# 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

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:
      jmp dry
      cmp word[x],right
      sub word[x],2
      cmp word[y],top
      jmp show
      mov byte[diry],Up
```

c. 之前写的fillout.cpp只能填充首扇区,扩展性不够强,因而增加两个输入分别是首字节位置和占用字节数

#### 操作如下:

```
PS C:\Users\guzy0\desktop\dbg> ./fillout2
512 512 stoneN00.com EX2.img
PS C:\Users\guzy0\desktop\dbg> ./fillout2
1024 512 stoneN01.com EX2.img
PS C:\Users\guzy0\desktop\dbg>
PS C:\Users\guzy0\desktop\dbg> ./fillout2
1536 512 stoneN10.com EX2.img
PS C:\Users\guzy0\desktop\dbg> ./fillout2
2048 512 stoneN11.com EX2.img
```

#### 效果如下:

```
00000200 b1 00 ba d5 7c 8c c8 8e c0 8e d8 8e c0 b8 00 b8 8e c0 ff 0e cb 7c 75 fa c7 06 cb 7c 50 c3 ff 0e
00000220 cd 7c 75 ee c7 06 cb 7c 50 c3 c7 06 cd 7c 44 02 80 3e cf 7c 01 74 17 ff 0e d1 7c 83 3e d1 7c 27
00000240 75 21 83 06 d1 7c 02 c6 06 cf 7c 01 eb 15 ff 06 d1 7c 83 3e d1 7c 50 75 0a 83 2e d1 7c 02 c6 06
00000260 cf 7c 00 80 3e d0 7c 01 74 17 ff 0e d3 7c 83 3e d3 7c 0c 75 21 83 06 d3 7c 02 c6 06 d0 7c 01 eb
00000280 15 ff 06 d3 7c 83 3e d3 7c 19 75 0a c6 06 d0 7c 00 83 2e d3 7c 02 31 c0 a1 d3 7c bb 50 00 f7 e3
000002a0 03 06 d1 7c bb 02 00 f7 e3 89 c3 88 cc 89 da 89 cb 8a 87 d5 7c 89 d3 80 c1 01 80 f9 0d 74 06 26
000002c0 89 07 e9 4d ff b1 00 eb f6 eb fe 50 c3 44 02 01 01 27 00 0c 00 20 31 38 33 30 38 30 34 35 5f 67
```

2.

a. myos1.asm中org的问题: org100h填到首扇区无法正常显示Message。使用bochsdbg 断点检查内存,发现org是用来调整虚拟地址的偏移

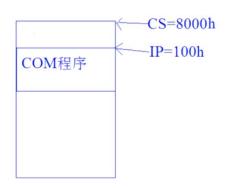
如下是偏移100h,发现虚拟地址被替换成了100h+36h,而物理地址为0x7c36,int 10输出的是物理地址0x0136的字符串

如下是偏移7c00h,输出正常

```
(0) [0x00000007c06] 0000:7c06 (unk. ctxt): mov bp, 0x7c36 ; bd367c <br/>
<br/>
<br/>
<br/>
<br/>
<br/>
(0) [0x000000007c36] 0000:7c06 (unk. ctxt): mov bp, 0x7c36 ; bd367c ; b
```

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:

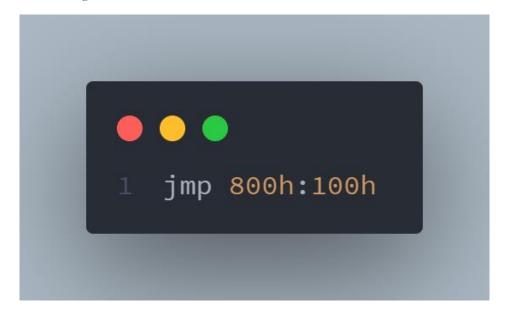
例如, COM程序被加载到内存8000:100h开始的位置, 那么 CS=8000h,IP=100h, 可用jmp 8000h:100h跳过去。



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:

```
Bochs for Windows - Console
                                                                                              П
<books:1> b 0x7d39
<books:2> c
(0) Breakpoint 1, 0x0000000000007d39 in ?? ()
Next at t=294613539
(0) [0x000000007d39] 0000:7d39 (unk. ctxt): mov cl, byte ptr ds:[si] ; 8a0c
<bochs:3> sreg
es:0x00000, dh=0x00009300, dl=0x0000ffff, valid=1
Data segment, base=0x000000000, limit=0x0000ffff, Read/Write, Accessed
cs:0x0000, dh=0x00009300, dl=0x00000ffff, valid=1
Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
ss:0x0000, dh=0x00009300, dl=0x0000ffff, valid=7
         Data segment, base=0x000000000, limit=0x00000ffff, Read/Write, Accessed
ds:0x0000, dh=0x00009300, dl=0x0000ffff, valid=3
Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
fs:0x0000, dh=0x00009300, dl=0x0000ffff, valid=1
         Data segment, base=0x000000000, limit=0x00000ffff, Read/Write, Accessed
gs:0x0000, dh=0x00009300, dl=0x00000000, limit=0x00000ffff, Read/Write, Accessed

Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
ldtr:0x0000, dh=0x00008200, dl=0x0000ffff, valid=1
tr:0x0000, dh=0x00008b00, dl=0x0000ffff, valid=1
gdtr:base=0x000000000000f9af7, limit=0x30
idtr:base=0x00000000000000000000, limit=0x3ff
<bochs:4>
```

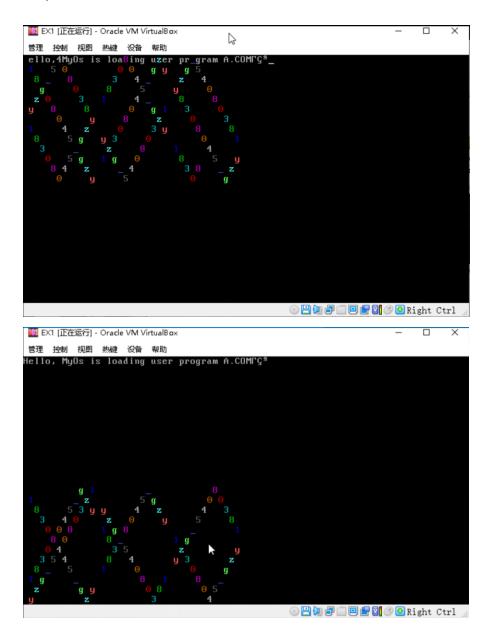
#### call后:

```
Bochs for Windows - Console
                                                                                                      П
idtr:base=0x0000000000000000, limit=0x3ff
<books:4> c
(0) Breakpoint 1, 0x0000000000007d39 in ?? ()
Next at t=410664875
(0) [0x000000007d39] 0000:7d39 (unk. ctxt): mov cl, byte ptr ds:[si] ; 8a0c
<bochs:5> sreg
es:0xb800, dh=0x0000930b, dl=0x8000ffff, valid=7
          Data segment, base=0x000b8000, limit=0x0000ffff, Read/Write, Accessed
cs:0x0000, dh=0x00009300, dl=0x00000ffff, valid=1
Data segment, base=0x00000000, limit=0x00000ffff, Read/Write, Accessed
ss:0x0000, dh=0x00009300, dl=0x0000ffff, valid=7
Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed
ds:0x0800, dh=0x00009300, dl=0x8000ffff, valid=7
Data segment, base=0x00008000, limit=0x0000ffff, Read/Write, Accessed
Data segment, base=0x000000000, limit=0x00001111, Redd, life, fs:0x0000, dh=0x00009300, dl=0x000000000, limit=0x00000ffff, Read/Write, Accessed gs:0x0000, dh=0x00009300, dl=0x00000ffff, valid=1
Data segment, base=0x00000000, limit=0x0000ffff, Read/Write, Accessed ldtr:0x0000, dh=0x00008200, dl=0x0000ffff, valid=1
tr:0x0000, dh=0x00008b00, dl=0x0000ffff, valid=1
gdtr:base=0x00000000000f9af7, limit=0x30
idtr:base=0x0000000000000000, limit=0x3ff
<bochs:6>
```

## 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

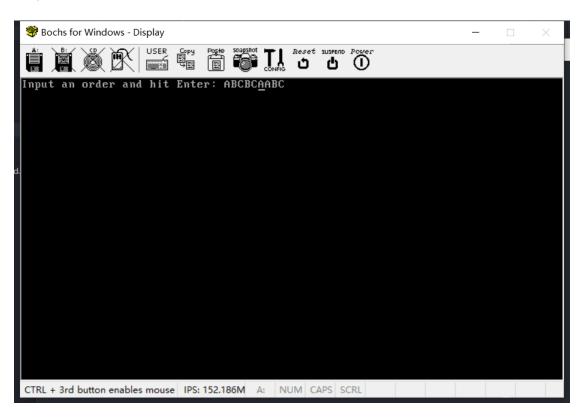
回车: 执行字符串对应的用户程序序列

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)

```
mov di,ProgramInfo
Defau:
cmp di,TotalNum
jz Input
cmp al,byte[di]
jz ValidProgram
add di,2
jmp Defau
```

c. 更改映射算法为遍历整个表

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, 且更加深入地了解了 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