



# 《操作系统原理实验》 实验报告

(实验二)

学 院 名 称 : 数据科学与计算机学院

专业 (班级) : 16 计科 2 班

学 生 姓 名 : 朱志儒

学 号 : 16337341

时 间 : 2018 年 3 月 17 日

## 实 验 二 ： 加载用户程序的监控程序

### 一. 实验目的

- 1、 设计四个（或更多）有输出的用户可执行程序
- 2、 实现一个监控系统，能够运行不同的用户程序，并且能从用户程序返回监控系统
- 3、 自行组织映像盘的空间存放用户可执行程序

### 二. 实验要求

- 1、 设计四个（或更多）有输出的用户可执行程序  

设计四个有输出的用户可执行程序，分别在屏幕1/4区域动态输出字符，如将用字符‘A’从屏幕左边某行位置45度角下斜射出，保持一个可观察的适当速度直线运动，碰到屏幕相应1/4区域的边后产生反射，改变方向运动，如此类推，不断运动；在此基础上，增加你的个性扩展，如同时控制两个运动的轨迹，或炫酷动态变色，个性画面，如此等等，自由不限。还要在屏幕某个区域特别的方式显示你的学号姓名等个人信息。
- 2、 实现一个监控系统，能够运行不同的用户程序，并且能从用户程序返回监控系统  

修改参考原型代码，允许键盘输入，用于指定运行这四个有输出的用户可执行程序之一，要确保系统执行代码不超过512字节，以便放在引导扇区。
- 3、 自行组织映像盘的空间存放用户可执行程序

### 三. 实验方案

#### 1、 虚拟机配置

使用Vmware Workstation配置虚拟机，虚拟机的配置：核心数为1的处理器、4MB的内存、10MB的磁盘、1.44MB的软盘。

#### 2、 软件工具与作用

Notepad++：编写程序时使用的编辑器；

16位编辑器WinHex：可以以16进制的方式打开并编辑任意文件；

NAMS汇编工具：可以将汇编代码编译成对应的二进制代码；

WinImage：可以创建虚拟软盘。

#### 3、 方案思想

在DOS系统中，用户程序可以调用DOS的4ch子功能，从而返回DOS系统。从这我得到启发，我在监控程序中添加了20h中断程序，并将它们加入了中断向量表，这样只需在用户程序中加入指令int 20h，即可通过按键触发返回监控程序。

#### 4、 相关原理

##### (1) BOIS调用

BIOS是英文"Basic Input Output System"的缩略语，直译过来后中文名称就是"基本输入输出系统"。其实，它是一组固化到计算机内主板上一个ROM芯片上的程序，它保存着计算机最重要的基本输入输出的程序、系统设置信息、开机后自检程序和系统自启动程序。其主要功能是为计算机提供最底层的、最直接的硬件设置和控制。

##### (2) BIOS芯片中主要存放：

自诊断程序：通过读取CMOSRAM中的内容识别硬件配置，并对其进行自检和初始化；

CMOS设置程序：引导过程中，用特殊热键启动，进行设置后，存入CMOS RAM中；

系统自举装载程序：在自检成功后将磁盘相对0道0扇区上的引导程序装入内存，让其运行以装入DOS系统；

主要I/O设备的驱动程序和中断服务：由于BIOS直接和系统硬件资源打交道，因此总是针对某一类型的硬件系统，而各种硬件系统又各有不同，所以存在各种不同种类的BIOS，随着硬件技术的发展，同一种BIOS也先后出现了不同的版本，新版本的BIOS比起老版本来说，功能更强。

### (3) BIOS中中断例程

即BIOS中断服务程序是微机系统软、硬件之间的一个可编程接口，用于程序软件功能与微机硬件实现的衔接。DOS/Windows操作系统对软、硬盘、光驱与键盘、显示器等外围设备的管理即建立在系统BIOS的基础上。程序员也可以通过 对INT 5、INT 13等终端的访问直接调用BIOS终端例程。

### (4) 调用BIOS中断服务程序的方法

每个中断服务有特定的参数，一般使用指定的寄存器传递参数；

利用软中断指令调用；

BIOS中断调用的一般格式为：

`mov ah,功能号`

…… ；设置各种入口参数

int 中断号

(5) 常用BIOS调用

功能	中断号	功能号
插入空行上滚显示页窗口	10H	06H
以电传方式显示单个字符	10H	0EH
显示字符串	10H	13H
复位磁盘系统	13H	00H
读扇区	13H	02H
读下一个按键	16H	00H

(6) BIOS的10H调用

BIOS的10H提供了显示字符串的调用

BIOS的10H号调用功能与参数

显示字符串	10H	13H	AL: 放置光标的方式 BL: 字符属性字节 BH: 显示页(0~3) DH: 行位置(0~24) DL: 列位置(0~79) CX: 字符串的字节数 ES:BP: 字符串的起始地址	AL=0/2 光标留在串头 AL=1/3 光标放到串尾 AL=0/1 串中只有字符 AL=2/3 串中字符和属性字节交替存储 BL: 位 7 为 1 闪烁 位 6~4 为背景色 RGB 位 3 为 1 前景色高亮 位 2~0 为前景色 RGB
-------	-----	-----	---------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------

(7) BIOS的13H调用

BIOS的13H提供了磁盘读写的调用

BIOS的13H号调用功能与参数

读扇区↵	13H↵	02H↵	AL: 扇区数(1~255)↵ DL: 驱动器号(0 和 1 表示软盘, 80H 和 81H 等表示硬盘或 U 盘) ↵ DH: 磁头号(0~15)↵ CH: 柱面号的低 8 位↵ CL: 0~5 位为起始扇区号(1~63), 6~7 位为硬盘柱面号的高 2 位(总共 10 位柱面号, 取值 0~1023)↵ ES:BX: 读入数据在内存中的存储地址↵	返回值: ↵ ■ 操作完成后 ES:BX 指向数据区域的起始地址↵ ■ 出错时置进位标志 CF=1, 错误代码存放在寄存器 AH 中↵ ■ 成功时 CF=0、AL=0↵
------	------	------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------

(8) 用户程序的格式

监控程序如何获取计算机硬件系统的控制权

如果程序不超过512字节，则可以引导扇区程序的方法实现

如果程序超过512字节，则要分解为引导扇区部分和其它功能部分

监控程序如何控制用户程序的执行

用户程序的格式：COM、BIN

加载用户程序：磁盘存储方法、磁盘读取控制编程

(9) COM格式

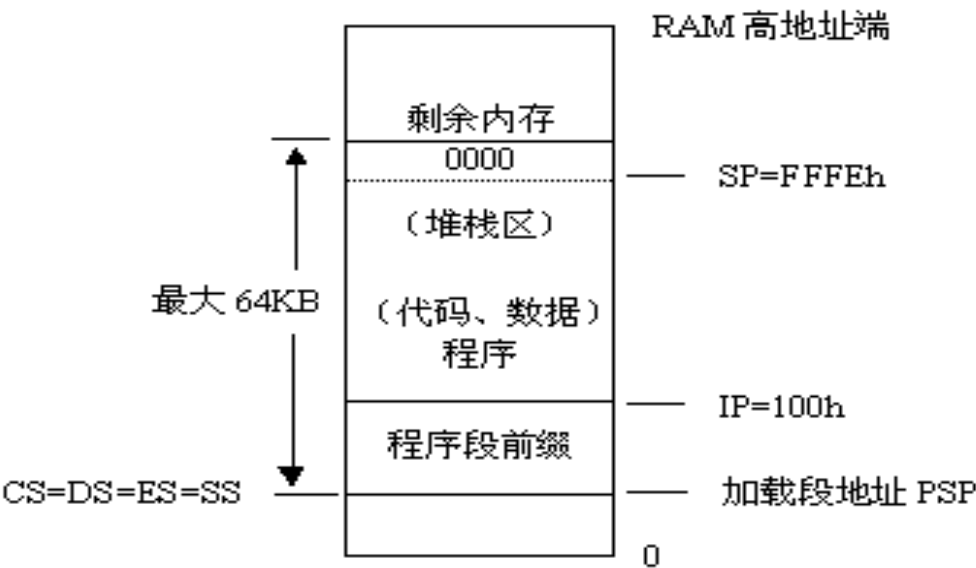
COM（command file，命令文件）是CP/M和DOS的一种原始二进制可执行格式，以.com为扩展名。COM文件非常简单，没有文件头、没有元数据，只有代码和数据。

COM文件会被装载到当前段的0x100（256）处，不能重新定位。由于不能分段，所以COM文件的大小必须≤64KB-256B，且不能有独立的数

据段和堆栈段，程序的所有代码和数据都必须位于一个段中。

另外,在Windows操作系统的64位版本中,不再支持COM程序的运行。

DOS加载COM程序的内存映象图



(10) PSP

DOS装载可执行文件 (.COM/.EXE) 自动构造, 返回DOS后自动释放该空间

偏移量	内容
00~01H (2B)	INT 20H指令
02~03H (2B)	存储器中段的大小
04~09H (6B)	保留
0A~0DH(4B)	中止 中断向量INT 22H
0E~11H(4B)	断点 中断向量INT 23H
12~15H(4B)	错误 中断向量INT 24H
16~2BH(22B)	DOS保留
2C~2DH(2B)	环境变量区段地址

2E~4FH(34B)          DOS运行区

50~51H(2B)          INT 21H指令

#### (11) 程序执行前存放的安排

一种简单的磁盘存储组织：

1.44MB的软盘共2880个扇区，布局如图



#### (12) 内存安排

目前我们使用IBM\_PC基本内存640k，实模式的段+段内偏移的方式访问内在单元。

BIOS装入引导扇区程序时，优先使用最低端的一个64k段作为引导扇区程序的存放区域，偏移量7c00h。

为了简单起见，我们也把用户程序a.com存放在同一个段中，就放在偏移量8100h开始的一个区域。

#### (13) 监控程序基本型

本程序按引导扇区程序要求设计

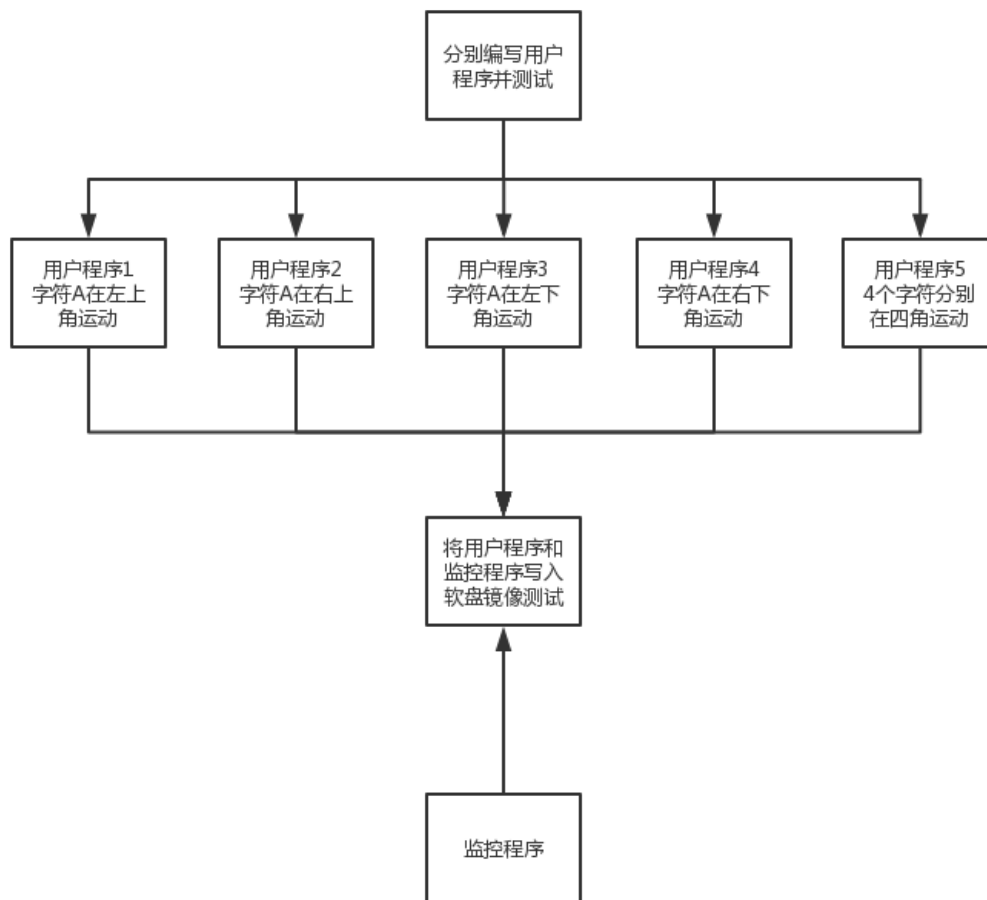
功能：显示必要的提示信息后，从引导盘的特定扇区加载一个他人开发的COM格式的可执行程序到指定的内存位置，然后启动这个程序，实现操作系统执行用户程序这一项基本功能。

程序利用BIOS调用显示字符串和读磁盘扇区加载用户程序

涉及BIOS的10H和13H号调用



## 5、 程序流程



## 6、 算法和数据结构

### 算法：

显示器显示模式为文本方式时，最小可控制单位为字符，VGA: 25X80。

字符A在左上角运动时，x的范围是（0，12），y的范围是（0，39）；

字符A在右上角运动时，x的范围是（0，12），y的范围是（40，79）；

字符A在左下角运动时，x的范围是（13，24），y的范围是（0，39）；

字符A在右下角运动时，x的范围是（13，24），y的范围是（40，79）。

字符出现的位置为（X，Y），则内存地址的偏移量为  $(X \times 80 + Y) \times 2$

用户程序5中，字符位置  $P = (X, Y)$ ，速度  $S = (1, 1)$ ，则新位置  $NP = P + S$

## 数据结构:

用户程序5中，pos表示字符的地址，其中pos的高位表示y，低位表示x，speed表示字符移动的速度，其中speed的高位表示y轴方向的速度，低位表示x轴方向的速度。

## 7、 程序关键模块

由于用户程序加载至内存的偏移地址不同，在不同用户程序中加入的org伪指令也不相同。例如，用户程序1中是org 8100h；用户程序2中是org 8300h；用户程序3中是org 8500h；用户程序4中是org 8700h；用户程序5中是org 8900h。

### 用户程序5中:

```
%macro rolling 8 ;(pos1, char2, color3, speed4, min_x5, min_y6, max_x7,
max_y8)
```

```
    mov ax, word[%1]
```

```
    mov word[pos], ax
```

```
    mov ax, word[%4]
```

```
    mov word[speed], ax
```

```
    mov al, byte[%2]
```

```
    mov byte[char], al
```

```
    mov al, byte[%3]
```

```
    mov byte[color], al
```

```
    mov al, %5
```

```
    mov byte[min_x], al
```

```
    mov al, %6
```

```

mov byte[min_y], al

mov al, %7

mov byte[max_x], al

mov al, %8

mov byte[max_y], al

call have_fun      ;调用显示字符运动的程序

mov ax, word[pos]

mov word[%1], ax

mov ax, word[speed]

mov word[%4], ax

%endmacro

rolling pos1, char, color, speed1, 0, 0, 12, 39

rolling pos2, char, color, speed2, 0, 40, 12, 79

rolling pos3, char, color, speed3, 13, 0, 24, 39

rolling pos4, char, color, speed4, 13, 40, 24, 79

```

为了成功实现在四个角分别显示4个字符的运动，我使用宏汇编，简化代码，使得用户程序5不超出512字节。

为了达到这个目的，我还重构了实验一的原型代码，其中改动最大的如下：

计算字符下个位置的坐标：

change\_position:

```
mov ax, word[pos]    ;当前字符位置(y, x)
```

```
mov bx, word[speed] ;当前字符速度(Vy, Vx)
```

```
add ah, bh
```

```
cmp ah, byte[min_y]
```

```
ja YA ;字符y坐标没有超出左边界则跳转
```

```
mov bh, 1 ;Vy = 1
```

YA:

```
cmp ah, byte[max_y]
```

```
jb YB ;字符y坐标没有超出右边界则跳转
```

```
mov bh, -1 ;Vy = -1
```

YB:

```
add al, bl
```

```
cmp al, byte[min_x]
```

```
ja XA ;字符x坐标没有超出上边界则跳转
```

```
mov bl, 1 ;Vx = 1
```

XA:

```
cmp al, byte[max_x]
```

```
jb XB ;字符x坐标没有超出下边界则跳转
```

```
mov bl, -1 ;Vx = -1
```

XB:

```
mov word[pos], ax ;字符下个位置的坐标(y, x)
```

```
mov word[speed], bx ;字符在下一个位置的速度(Vy, Vx)
```

```
ret
```

## 监控程序：

写入中断向量表模块：

```
%macro write_inerrupt_vector 2

    pusha          ;将寄存器压栈，保护现场

    mov ax, 0h

    mov es, ax

    mov ax, %1

    mov bx, 4

    mul bx

    mov bp, ax

    mov ax, %2

    mov word[es:bp], ax ;在中断向量表中存入中断程序的地址

    add bp, 2

    mov ax, cs

    mov word[es:bp], ax

    popa          ;寄存器出栈，恢复现场

%endmacro
```

定义20h中断向量：

```
write_interrupt_vector 20h, myinterrupt20h
```

myinterrupt20h:

```
    pusha          ;将寄存器压栈，保护现场

    print_message message1, 24, 24, 3
```

```
mov ah, 01h
```

```
int 16h
```

```
jz no_input      ;没有按键，则跳转至no_input
```

```
cmp al, 'q'
```

```
jne no_input ;若没按q，跳转至no_input
```

```
jmp 0000h:Start ;返回监控程序
```

```
no_input:
```

```
popa      ;寄存器出栈，恢复现场
```

```
iret
```

显示信息模块：

```
%macro print_message 4 ;(message, message_length, x, y)
```

```
pusha
```

```
mov ax, cs
```

```
mov es, ax
```

```
mov bp, %1      ;es:bp: 字符串首地址
```

```
mov cx, %2      ;字符串长度
```

```
mov bx, 0007h
```

```
mov ax, 01301h  ;调用显示字符串的功能
```

```
mov dh, %3      ;行
```

```
mov dl, %4      ;列
```

```
int 10h
```

```
popa
```

```
%endmacro
```

由于显示信息的操作在监控程序中执行多次，为了简洁代码，我使用了宏汇编。

读取用户的输入：

```
mov ah, 00h
```

```
int 16h
```

```
cmp al, '1'
```

```
je condition1
```

```
cmp al, '2'
```

```
je condition2
```

```
cmp al, '3'
```

```
je condition3
```

```
cmp al, '4'
```

```
je condition4
```

```
cmp al, '5'
```

```
je condition5
```

```
jmp Start
```

根据用户的选择决定存放数据的内存地址：

condition1:

```
mov word[choose_adress], OffSetOfUserPrg1
```

```
jmp next
```

condition2:

```
mov word[choose_adress], OffSetOfUserPrg2
```

```
jmp next
```

condition3:

```
mov word[choose_adress], OffSetOfUserPrg3
```

```
jmp next
```

condition4:

```
mov word[choose_adress], OffSetOfUserPrg4
```

```
jmp next
```

condition5:

```
mov word[choose_adress], OffSetOfUserPrg5
```

```
jmp next
```

根据用户的选择决定读取的扇区号：

next:

```
sub al, '0'
```

```
inc al
```

```
mov byte[choose_number], al
```



读软盘或硬盘上的若干物理扇区到内存的ES:BX处：

```
call cls

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,word[choose_adress] ;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,byte[choose_number] ;起始扇区号 ; 起始编号为1

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

; 用户程序a.com已加载到指定内存区域中

jmp word[choose_adress]
```

## 四. 实验过程和结果

- 1、 分别编写用户程序1, 2, 3, 4和5并测试, 测试结果如图所示。

```
16337341 ZhuZhiru's first program
```

```
A
```

```
Please input q to return_
```

```
16337341 ZhuZhiru's second program
```

```
A
```

```
Please input q to return_
```

16337341 ZhuZhiru's third program

A

Please input q to return\_

16337341 ZhuZhiru's fourth program

Please input q to return\_

A

D

D

D

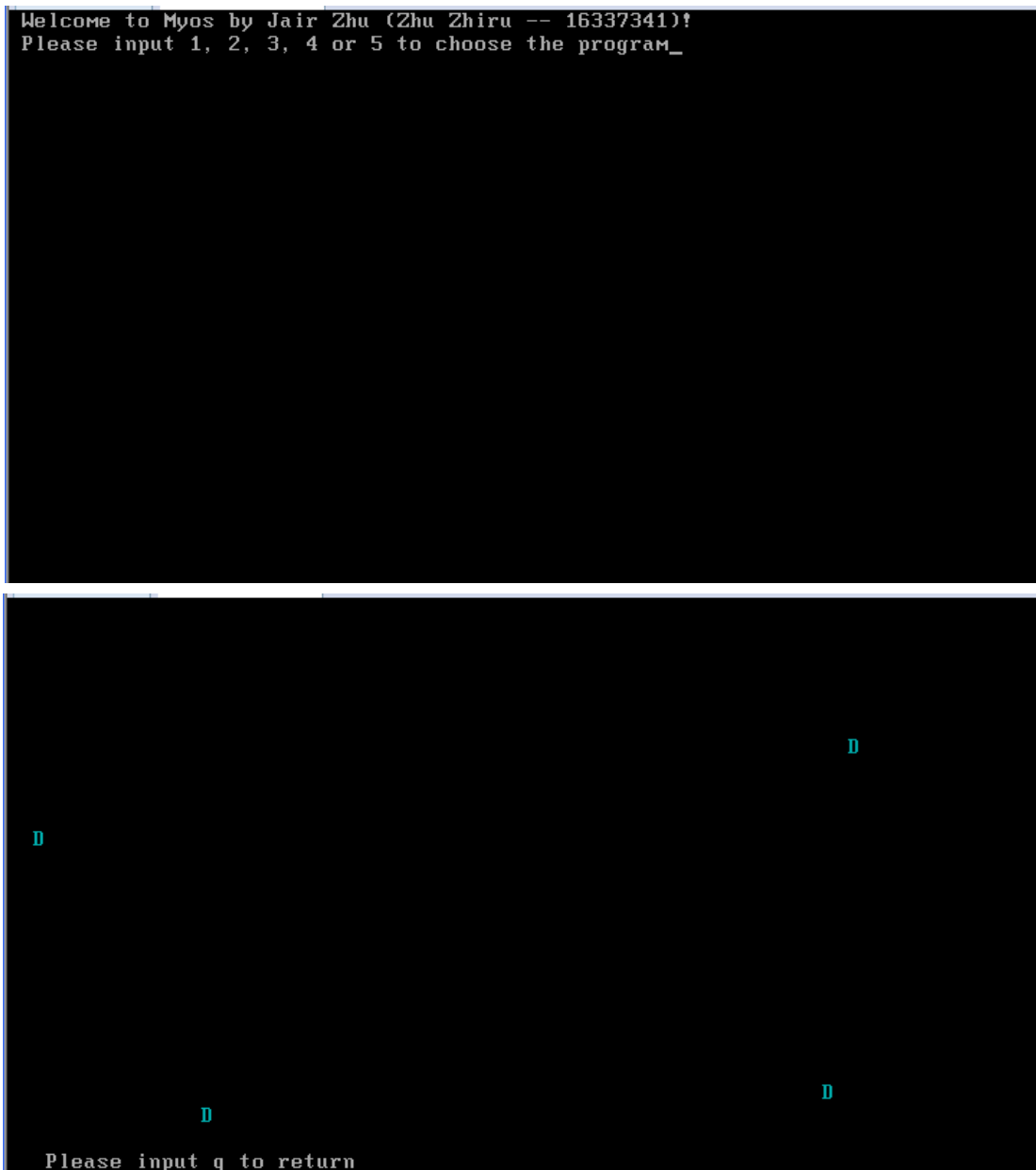
D

Please input q to return

## 2、 编写监控程序并将其和用户程序一并写入软盘镜像，如图所示

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	60	B8	00	00	8E	C0	B8	20	00	BB	04	00	F7	E3	89	C5	`?. 幫?.?. 麼壘. 腔
00000010	B8	57	7C	26	89	46	00	83	C5	02	8C	C8	26	89	46	00	寔 &垠. 廼. 肩&垠.
00000020	61	60	B8	00	00	8E	C0	B8	21	00	BB	04	00	F7	E3	89	a`?. 幫?.?. 麼??..
00000030	C5	B8	82	7C	26	89	46	00	83	C5	02	8C	C8	26	89	46	鸥俅&垠. 廼. 肩&垠
00000040	00	61	EB	44	60	B4	06	B0	00	B7	07	B5	00	B1	00	B6	.a陸`???????解?)
00000050	18	B2	4F	CD	10	61	C3	60	60	8C	C8	8E	C0	BD	29	7D	.暉?a胎`肩幫?) 徑
00000060	B9	18	00	BB	07	00	B8	01	13	B6	18	B2	03	CD	10	61	?.?.?. ???a漳. 樺?.
00000070	B4	01	CD	16	74	0A	B4	00	CD	16	3C	71	75	02	CD	21	??t. ??<qu. ?戛. 當
00000080	61	CF	EA	88	7C	00	00	CF	8C	C8	8E	D8	E8	B5	FF	60	a详垠.. 蟬蒞罔?)
00000090	8C	C8	8E	C0	BD	43	7D	B9	34	00	BB	07	00	B8	01	13	肩幫紺) ?.?.?. €
000000A0	B6	00	B2	01	CD	10	61	60	8C	C8	8E	C0	BD	79	7D	B9	???a`肩幫統) 漳..
000000B0	32	00	BB	07	00	B8	01	13	B6	01	B2	01	CD	10	61	B4	2.?.?. ???a? .解
000000C0	00	CD	16	3C	31	74	12	3C	32	74	16	3C	33	74	1A	3C	.?<1t.<2t.<3t.<?
000000D0	34	74	1E	3C	35	74	22	EB	AF	C7	06	AE	7D	00	81	EB	4t.<5t“氙?富..?)
000000E0	20	C7	06	AE	7D	00	83	EB	18	C7	06	AE	7D	00	85	EB	?富. 泮. ?富. 吗..
000000F0	10	C7	06	AE	7D	00	87	EB	08	C7	06	AE	7D	00	89	EB	.?富. 国. ?富. 嚶??
00000100	00	2C	30	FE	C0	A2	AD	7D	E8	39	FF	8C	C8	8E	C0	8B	.,0 }? 肩幫?
00000110	1E	AE	7D	B4	02	B0	01	B2	00	B6	00	B5	00	8A	0E	AD	.富???????臄. 繫?
00000120	7D	CD	13	FF	26	AE	7D	EB	FE	50	6C	65	61	73	65	20	}? &富臄Please
00000130	69	6E	70	75	74	20	71	20	74	6F	20	72	65	74	75	72	input q to retur
00000140	6E	18	00	57	65	6C	63	6F	6D	65	20	74	6F	20	4D	79	n..Welcome to My
00000150	6F	73	20	62	79	20	4A	61	69	72	20	5A	68	75	20	28	os by Jair Zhu (
00000160	5A	68	75	20	5A	68	69	72	75	20	2D	2D	20	31	36	33	Zhu Zhiru -- 163
00000170	33	37	33	34	31	29	21	34	00	50	6C	65	61	73	65	20	37341)!4.Please
00000180	69	6E	70	75	74	20	31	2C	20	32	2C	20	33	2C	20	34	input 1, 2, 3, 4
00000190	20	6F	72	20	35	20	74	6F	20	63	68	6F	6F	73	65	20	or 5 to choose
000001A0	74	68	65	20	70	72	6F	67	72	61	6D	32	00	00	00	00	the program2....
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA	.....U?
00000200	8C	C8	8E	D8	8E	C0	B8	00	B8	8E	E8	C6	06	C9	82	41	肩庝幫?穿杵. 彥姪
00000210	FF	0E	C0	82	75	FA	C7	06	C0	82	50	C3	FF	0E	C2	82	.纒u .纒P?. 腔
00000220	75	EE	C7	06	C0	82	50	C3	C7	06	C2	82	44	02	B0	01	u卦. 纒P们. 腔D. ?.
00000230	3A	06	C4	82	74	1C	B0	02	3A	06	C4	82	74	51	B0	03	..臄t. ?:. 臄tQ???
00000240	3A	06	C4	82	0F	84	83	00	B0	04	3A	06	C4	82	0F	84	..臄. 剥. ?:. 臄. ?.
00000250	B3	00	FF	06	C5	82	FF	06	C7	82	8B	1E	C5	82	B8	0D	? .解 . 菴?解??
00000260	00	29	D8	74	0E	8B	1E	C7	82	B8	29	00	29	D8	74	11	.)豸. ?菴?.)豸. ??
00000270	E9	CC	00	C7	06	C5	82	0B	00	C6	06	C4	82	02	E9	BE	槿. ?解..?臄. 樺 畚
00000280	00	C7	06	C7	82	27	00	C6	06	C4	82	04	E9	B0	00	FF	.?菴'. ?臄. 渠. ?
00000290	0E	C5	82	FF	06	C7	82	8B	1E	C7	82	B8	29	00	29	D8	.解 . 菴?菴?.)??
000002A0	74	0E	8B	1E	C5	82	B8	FF	FF	29	D8	74	11	E9	8F	00	t. ?解? )豸. ?. 至.
000002B0	C7	06	C7	82	27	00	C6	06	C4	82	03	E9	81	00	C7	06	?菴'. ?臄. ??. 某
000002C0	C5	82	01	00	C6	06	C4	82	01	EB	74	FF	0E	C5	82	FF	解..?臄. 雖 . 解
000002D0	0E	C7	82	8B	1E	C5	82	B8	FF	FF	29	D8	74	0D	8B	1E	. 菴?解? )豸. ?s
000002E0	C7	82	B8	FF	FF	29	D8	74	0F	EB	54	C7	06	C5	82	01	菴? )豸. 陳?解..
000002F0	00	C6	06	C4	82	04	EB	47	C7	06	C7	82	01	00	C6	06	■?臄. 陸?菴.. ?菴??

## 3、 在虚拟机上测试监控程序，如图所示



```
Welcome to Myos by Jair Zhu (Zhu Zhiru -- 16337341)!
Please input 1, 2, 3, 4 or 5 to choose the program_

D

D

D

Please input q to return
```

## 五. 实验总结

### 总结:

我觉得这次实验的难点在于从用户程序返回监控程序,最初我准备在监控程序使用call指令跳转至用户程序,然后在用户程序中编写按键触发返回监控程序的代码。虽说这样做是可以实现的,但在DOS系统中,用户程序可以调用中断程序返回DOS系统,受这个启发,我觉得可以在监控程序中编写一个中断程序,然后将这个程序地址加入中断向量表,这样就可以在用户程序中调用该中断程序返回监控程序,免去了在用户程序编写按键触发返回监控程序代码的繁碎。

### 问题和解决方法:

- (1) 在监控程序中的关于读软盘扇区的数据到内存的那段代码,我原本是用寄存器来存储数据存放内存的偏移地址和读取软盘上的起始扇区号,然而这样做的话,在程序运行时将无法获得准确的偏移地址和起始扇区号,所以我定义了两个变量来分别存储偏移地址和起始扇区号,这样做成功解决了无法获得准确的偏移地址和起始扇区号的问题。
- (2) 在编写20h号中断程序的时候调用16h号BIOS中断的0号功能,发现在用户程序在运行到int 20h指令时停止了,要等到有按键输入时才会运行,这是因为16h号BIOS中断的0号功能等待用户的按键输入,这就导致用户程序需要一直有按键输入才会正常运行。经过查阅资料后,我发现16h号BIOS中断的1号功能可以检测用户是否有按键输入,这样程序就不用等待用户的按键输入而正常的运行了,如果有按键输入也会做出响应。
- (3) 我觉得这次实验最大的困难在于如何实现4个字符在屏幕的四个角上运动互不干扰且字符不断变化、颜色也不断变化。由于之前编写的4个用户程

序是字符A分别在4个角上运动互不干扰,我的想法是并行运行这4个用户程序。又因为程序不能超过512字节,我不得不使用宏汇编来简化代码。在使用宏汇编时,我将字符的位置、速度、颜色作为参数传入调用的计算位置并显示字符的函数,显示第一个字符过后,再保存第一个字符的位置、速度、颜色等信息,接着分别显示第二个字符、第三个字符和第四个字符并一一保存各个字符的信息,再调用20h号中断程序响应用户的按键,最后再循环以计算并显示各个字符的下个位置。

我觉得并行运行这4个用户程序算是多任务系统的最初始的一种吧。

## 六.参考文献

- 1、 BIOS知识
- 2、 《x86 PC汇编语言,设计与接口》
- 3、 《x86 汇编语言-从实模式到保护模式》