



《操作系统实验》

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 计算机类 4 班

学 生 姓 名 : 陈泓仰

学 号 : 15303009

时 间 : 2018 年 3 月 17 日

成绩：

实验二：加载用户程序的监控程序

一. 实验目的

1. 掌握监控程序加载用户程序的过程
2. 实现一个可以加载COM格式的用户程序的监控程序，并实现用户程序的返回
3. 学习如何调用BIOS中断向量
4. 了解并掌握FAT32软盘结构与扇区

二. 实验要求

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。

三. 实验方案

1. 实验工具与环境：

本次实验是在Mac OS系统上进行。所有的工具都是自己根据老师在Windows下所需要的软件功能自己在网上查阅资料收集而来。

- a) 汇编器：NASM
- b) 虚拟机：VmWare Fusion
- c) 十六进制编辑器：HexFined
- d) 汇编程序编辑器：Sublime
- e) 汇编调试工具（虚拟机）：Bochs

2. 工具链使用方法

在上次实验中，只是初步的提及MacOS系统上，我是使用什么软件去写操作系统的，并没有详细的说明该如何使用这一套工具链。现在我详细的说明一下我们该如何使用这些软件。

a) 汇编器NASM

NASM编译比较简单，打开终端，直接输入格式如下的命令即可：

```
nasm -f <format> <filename> [-o <output>]
```

这里需要注意的是：

-f: 提供输出的文件格式 <format> 里提供文件格式

<filename>: 源代码文件，这文件名的后缀不必是 .asm 或 .s

[-o <output>]: 提供输出的 object 文件名。当不提供输出文件名时，输出的文件名就是 filename (和源代码文件名一样)。

如果不提供输出文件格式，默认的输出文件格式是 bin

例子如下：

```
chen$ nasm -f bin a.asm -o a.com
```

b) 虚拟机VmWare Fusion:



选择安装方法



从光盘或映像中安装

将您的 ISO 文件拖到此处以开始安装



迁移您的 PC



从恢复分区中安装 macOS



导入现有虚拟机



从 Boot Camp 安装



创建自定义虚拟机



在远程服务器上创建虚拟机



取消

继续

选择创建自定义虚拟机。

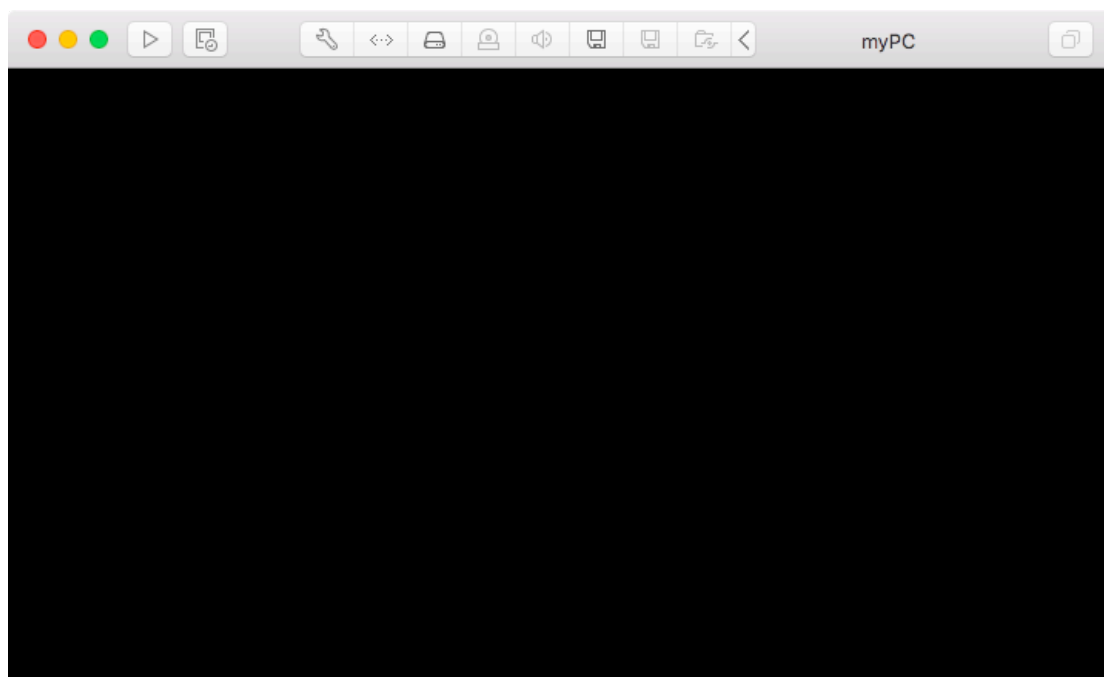



选择操作系统为：“其他”

一直默认选项，最后得到虚拟机。



最后界面如下：



在加载软盘的时候，只需要点击 ，在弹出框中选择你所要运行的flp文件。在这

里必须提及一点，就是nasm编译出来的文件是com格式，在这里我是手动将com格式改成flp格式的，能否直接生成flp文件我还没有进行尝试。

c) 十六进制编译器：HexFined

这一个软件使用方法与WinHex一样，也是复制粘贴。但是要注意的是，在修改文件内容时，需要将编辑模式由覆盖改为插入。正是由于这一点，在和WinHex相比又多了几分便利。因为WinHex在修改文件十六进制的时候需要先删除在粘贴，删多少加多少。操作不如HexFined便捷。

d) 编辑器IDE：Sublime

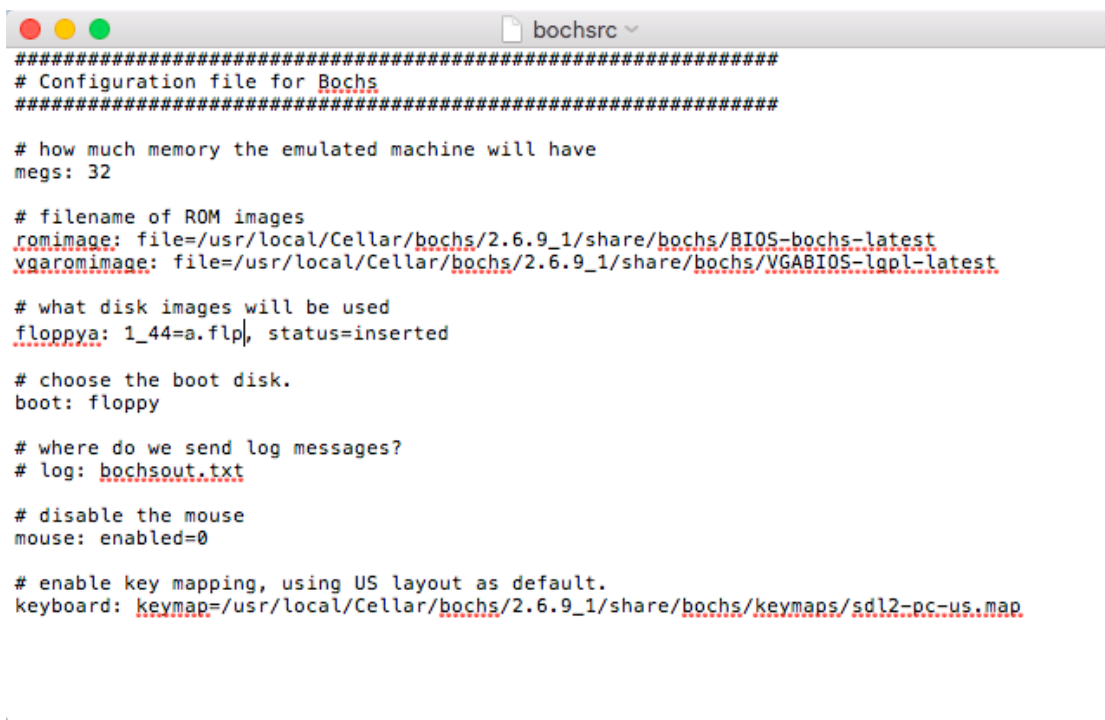
由于原生Sublime中并没有支持汇编代码的编辑，要想得到语法高亮的效果，需要自己安装package: x86 and x86_64 Assembly，最终效果如下：



```
1 ;程序源代码 (myos1.asm)
2 org 7c00h ; BIOS将把引导扇区加载到0:7C00h处, 并开始执行
3 Offset0fUserPrg1 equ 0A100h
4 %macro SetInt 2; 载入中断向量表的宏
5     mov ax,0
6     mov es,ax
7     mov ax,%1
8     mov bx,4
9     mul bx
10    mov di,ax
11    mov ax,%2
12    mov [es:di],ax
13    mov ax,cs
14    mov [es:di+2],ax
15 %endmacro
16
17 mov ax, cs ; 置其他段寄存器值与CS相同
18 mov ds, ax ; 数据段
19 SetInt 20h,SetINT20h
20
21 %macro Print 4
22     push es
23     push ds
24     push cs
25     push bp
26     push bx
27     push ax
28     push cx
29     push dx
30     mov bp,%1; bp为当前串偏移地址
31     mov ax,cs; es:bp = 串地址
32     mov es,ax; es = ds
33     mov cx,%2; cx = 串长 (=9)
34     mov ax,%3; AH = 13h (功能号)、AL = 01h (光标置于串尾)
35     mov bx,0007h;
36     mov dx,%4; 行号和列号
37     int 10h; BIOS的10h功能: 显示一行字符
```

e) 汇编调试工具及虚拟机: bochs

- 1) 配置和使用它是一个比较困难的工程。首先我们要确保我们在mac系统下安装了gcc工具, 然后使用homebrew直接在终端安装bochs。
- 2) 此时我们还需要有一个配置文件bochsrc, 可以到网上找。这里粘贴一份配置好的。



```
#####
# Configuration file for Bochs
#####

# how much memory the emulated machine will have
megs: 32

# filename of ROM images
romimage: file=/usr/local/Cellar/bochs/2.6.9_1/share/bochs/BIOS-bochs-latest
vgaromimage: file=/usr/local/Cellar/bochs/2.6.9_1/share/bochs/VGABIOS-lgpl-latest

# what disk images will be used
floppya: 1_44=a.flp, status=inserted

# choose the boot disk.
boot: floppy

# where do we send log messages?
# log: bochsout.txt

# disable the mouse
mouse: enabled=0

# enable key mapping, using US layout as default.
keyboard: keymap=/usr/local/Cellar/bochs/2.6.9_1/share/bochs/keymaps/sdl2-pc-us.map
```

i. romimage:file=/usr/local/share/bochs/BIOS-bochs-latest

#这个是BIOS-bochs-latest的路径,自己去慢慢找,应该都不同的

ii. vgaromimage:file=/usr/local/share/bochs/VGABIOS-lgpl-latest

#这个是VGABIOS-lgpl-latest的路径,自己去慢慢找,应该都不同的

iii. floppya: 1.44=a.flp, status=inserted

#这里是你所要调试的软盘文件

3) 命令行中输入: bochs -f bochsrc, 按6进入虚拟机。

3. 相关基础原理:

a) BIOS调用

首先,我们来看一下BIOS调用的相关知识。BIOS,是“基本输入输出系统”。它是一组固化到计算机内主板上的一個ROM芯片上的程序,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。这次实验我们需要BIOS所为我们提供的功能：其一就是BIOS在自检成功过后会将磁盘相对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，其二就是利用BIOS给我们提供的中断服务。

b) 监控程序

在理论课上，老师谈到了关于操作系统实现的一个种类——批处理系统，批处理系统是指用户将一批作业交给操作系统之后不再干预，有操作系统自己自动运行。我们要实现的监控程序也是类似这样，我们将监控程序作为引导程序，然后又监控程序来调用各种不同的用户程序，同时还可以由用户程序调回监控程序。

c) 如何加载引导程序与用户程序

我们知道，BIOS会自动帮我们把引导程序装入内存，但是仔细一想，我们会发现，如果我们的引导程序大于512个字节，或者我们需要实现一个批处理系统，实现一个监控程序。这个监控程序要帮我们做的就是，加载用户程序。加载的方法，就是利用BIOS中断中存取磁盘的功能，将用户程序由磁盘加载到内存中，然后跳转到该内存地址运行。

d) 磁盘的读取与存储

BIOS的13h中断为我们提供了磁盘读写的调用。

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
-----	-----	-----	---	--

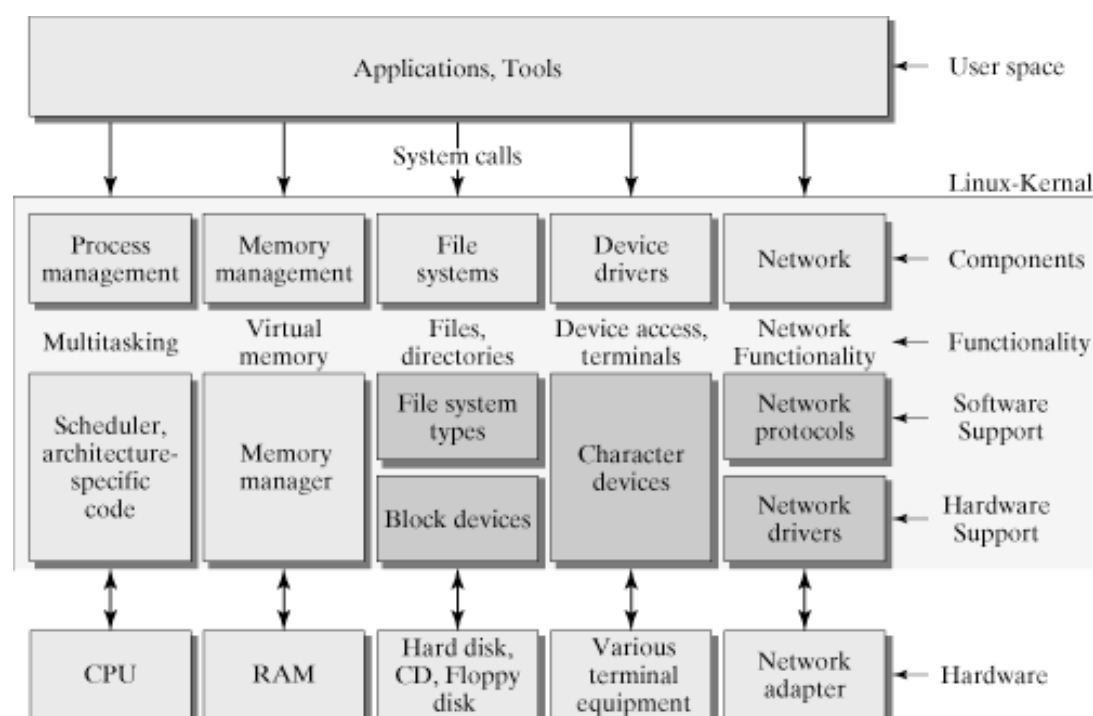
然后我们编译好了用户程序之后就直接粘贴在磁盘中监控程序之后即可(大小为512字节)。

e) 磁盘结构:

1.44MB 软盘的格式: 2 个磁头(head) / 盘面(side)、每磁头 80 个柱面(cylinder) / 磁道(track)、每个柱面有 18 个扇区(sector)、每个扇区有 512 个字节(byte), 所以软盘的容量为:

$$2\text{磁头} \times 80\text{柱面} \times 18\text{扇区} \times 512\text{B} = 2880\text{扇区} \times 512\text{B} = 1474560\text{B} = 1440\text{KB} = 1.44\text{MB}$$

f) 我们必须说明用户态与内核态之间的关系:



有此图，内核所要做的事主要表现为：向下控制硬件资源，向内管理操作系统资源：包括进程的调度和管理、内存的管理、文件系统的管理、设备驱动程序的管理以及网络资源的管理，向上则向应用程序提供系统调用的接口。从整体上来看，整个操作系统分为两层：用户态和内核态，这种分层的架构极大地提高了资源管理的可扩展性和灵活性，而且方便用户对资源的调用和集中式的管理，带来一定的安全性。

由此，我们可以看到，用户态可以通过系统调用的方法来访问内核态的资源，这保证了整个系统的安全性。这也引出了下一个话题。

g) 由用户程序与监控程序转换（内核态与用户态的切换）

这个实验中，最难的俩个部分就是实现由监控程序跳转到用户程序，以及由用户程序返回监控程序，前者我们可以直接在监控程序中跳转到用户程序的地址即可。而后者好像很难实现。一开始，我想了一个办法就是直接跳转到监控程序在内存中的地址，也就是0:7c00h，

但是后来仔细一想，这方法确实不妥当。原因是，用户程序又不知道你监控程序加载的地方，直接那么一跳，有可能监控程序并非加载在07c00h处，不仅很冒险，而且就相当于直接让内核态的地址暴露在用户态眼中，只要用户对内核态这块地址的内存重新进行读写，那么整个操作系统（监控程序）就会崩溃。这方法不仅有点蠢而且还很不安全！

经过查阅书籍与资料，我知道了什么情况下会发生由用户态到内核态的切换。包括以下三种情况：系统调用，异常事件（当CPU正在执行运行在用户态的程序时，突然发生某些预先不可知的异常事件，这个时候就会触发从当前用户态执行的进程转向内核态执行相关的异常事件，典型的如缺页异常。），外围设备的中断。在这里，我们暂时不会出现什么异常啊，外围设备的中断，我的操作系统还没有达到那种境界。但是，我们可以利用系统调用来做文章。

系统调用本质上其实也是一种中断，但相对于外围设备的硬中断，这种中断方式被称为软中断，这是操作系统为用户开放的中断。这一中断可以由操作系统自己指定，也就是说我们可以自己载入中断向量表，同时还可以实现用户主动请求切换而无需访问监控程序内存。

h) 软中断与BIOS中断的区别

BIOS和DOS都是存在于实模式下的程序，他们建立的中断调用都是建立在中断向量表上的，通过软中断指令int调用。计算机启动之初，中断向量表中的中断例程是由BIOS建立的，它从物理内存地址0x0000处初始化并在中断向量表中添加各种处理例程。BIOS中断调用的主要功能是提供了硬件访问的方法，该方法使对硬件的操作变得简单易行。BIOS中断这方法，有点类似于我们编程时构造的函数，提高了代码的复用性，而且中断会固定存在内存的中断向量表（有1024个字节，可以容纳256个中断向量处理程序）中，使得用户程序、引导程序都能去使用。不过在中断向量表中第0H~1FH是BIOS中断，而这也意味着，

后面的中断向量表可以我们自己载入。载入方法我们后面介绍。

4. 程序实现的功能描述：

- a) 监控程序加载用户程序
- b) 利用宏汇编实现多粒子轨迹变色运动
- c) 名字变色显示
- d) 用户程序键盘输入处理
- e) 利用宏汇编来减少大量代码量，增加代码复用性。

5. 程序流程：

- a) 监控程序：设置中断处理由于跳出用户程序，读取键盘输入，读取磁盘跳到用户程序，用户输入情况的处理。利用软中断，实现用户程序的退出。
- b) 用户程序：使用宏汇编来定义粒子运动轨迹，从而实现多粒子运动。代码多模块化，分为粒子显示模块，消除粒子轨迹模块，粒子运动位置更新模块，姓名显示模块。
- c) 具体的算法实现，会在下列模块介绍中一起出现。

四. 实验过程与结果

说明：根据需要书写相关内容，如：

主要工具安装使用过程及截图结果、程序过程中的操作步骤、测试数据、输入及输出说明、遇到的问题及解决情况、关键功能或操作的截图结果。

1. 算法说明、模块介绍：

- a) 监控程序：
 - i. 定义一个设置中断向量的宏：

```

%macro SetInt 2;载入中断向量的宏
    mov ax,0
    mov es,ax
    mov ax,%1
    mov bx,4
    mul bx
    mov di,ax
    mov ax,%2
    mov [es:di],ax
    mov ax,cs
    mov [es:di+2],ax
%endmacro

```

在DOS下，设置中断向量的方法有很多，甚至你可以直接用DOS下的软中断指令来实现中断向量的读取和载入。但是我们现在是在自己写的操作系统中实现，因此无法使用DOS下的软中断指令，所以只能直接写。方法就是将中断向量函数的段与偏移量送到4*向量号~4*向量号+3的地方去。

ii. 退出用户程序的中断：

```

SetINT20h:
    ;Print Help,HelpLength,13
    call print2
    mov ah,01h
    int 16h
    jnz back
    iret
back:
    cmp al,'q'
    jnz h
    call 0:clean
    mov ah,0
    int 16h
    jmp 0:e_of_p
h:
    cmp al,'b'
    jz back_to_Select
    jmp 0:EnterX
back_to_Select:
    mov ah,0
    int 16h
    jmp 0:EnterX
    iret

```

这里定义了20h的中断，用户只要调用int 20h即可退出用户程序，但是这里使用中断号为16h，ah为01h的中断来检测键盘状态，如果有输入就会对键盘输入检测，分为一下几种情况：

1, 2, 3, 4即实现扇区选择，选择屏幕加载哪个程序。

q, 调用退出程序，并选择是否重新开机。

iii. 定义一个可以用来显示字符串的宏

```
%macro Print 4
    push es
    push ds
    push cs
    push bp
    push bx
    push ax
    push cx
    push dx
    mov bp,%1;bp为当前串偏移地址
    mov ax,cs;es:bp = 串地址
    mov es,ax;es = ds
    mov cx,%2;cx = 串长 (=9)
    mov ax,%3;AH = 13h (功能号)、AL
    mov bx,0007h;
    mov dx,%4;行号和列号
    int 10h;BIOS的10h功能: 显示一行字符
    pop dx
    pop cx
    pop ax
    pop bx
    pop bp
    pop cs
    pop ds
    pop es
%endmacro
```

调用了BIOS中断号的10h功能，来显示一行字符串。

这里需要注意的是，程序在调用这个宏来显示字符串的时候，段寄存器的值需要我们合理保存，尤其是从用户程序退回监控程序调用的时候更是如此。但是这里我有一个很严重的问题，就是如果在自己写的中断中调用这个宏汇编显示字符串，我的程序就会崩。

最后还是得自己在重写一个循环显示字符串的函数才能解决。如下：


```

print2:;显示字符串
    push es
    push si
    mov ax,0B800H
    mov es,ax
    mov si,Help
    mov di,0
    mov cx,HelpLength
S:
    mov al,byte [ds:si]
    mov byte [es:di],al
    mov byte [es:di+1],7
    add di,2
    inc si
    dec cx
    cmp cx,0
    jnz S
    pop si
    pop es
    ret
GetINT32h:

```

iv. 程序的主体部分:

```

Start:
    call clean
    Print Message,MessageLength,130
    ;Print Help,HelpLength,1301h,05h
EnterX:
    mov ah,0 ; 功能号
    int 16H ; 调用中断
    mov ah,0eh
    mov bl,0
    int 10h
    sub al,'0'
    add al,1
    mov byte [save],al; 非扇区有的是不
LoadnEx:
    ;读软盘或硬盘上的若干物理扇区到内存的
    call clean
    mov ax,cs ;段地址
    mov es,ax ;设置段
    mov bx, OffSet0fUserPrg1 ;偏移地
    mov ah,2 | ; 功能号
    mov al,1 ;扇区数
    mov dl,0 ;驱动器
    mov dh,0 ;磁头号
    mov ch,0 ;柱面号
    mov cl,byte [save]
    int 13H ; 调用读磁
    ; 用户程序a.com已加载到指定内存区
    jmp OffSet0fUserPrg1

```

利用BIOS中断来实现键盘输入状态检测(int 16h,ah = 1), 键盘输入(int 16h,ah = 0), 读取磁盘(int 10h,ah = 0eh)的效果。

b) 用户程序

i. 定义了一个粒子的宏

```
%macro POINT 6
    mov ax, [%1]
    mov [x],ax
    mov ax, [%2]
    mov [y],ax
    mov al, [%3]
    mov [xdul],al
    mov al, [%4]
    mov [ydu1],al
    mov al, [%5]
    mov [color],al
    mov al, [%6]
    mov [char],al

    call move

    inc byte [color]

    mov ax, [x]
    mov [%1], ax
    mov ax, [y]
    mov [%2], ax
    mov al, [xdul]
    mov [%3], al
    mov al, [ydu1]
    mov [%4], al
    mov al, [color]
    mov [%5],al
%endmacro
```

ii. mov函数负责粒子每次的运动, 其中包括: 消除轨迹, 更新位置, 显示字符, 以及显示名字。

```
move:
call Setpoint
call kill
call update
call Setpoint
call show
;call Enter
ret
```

iii. 函数主逻辑:

```
start:
    call loop1
    POINT x1,y1,xdul1,ydul1,color1,char1
    POINT x2,y2,xdul2,ydul2,color2,char2
    call showname
    int 20h
    jmp start
```

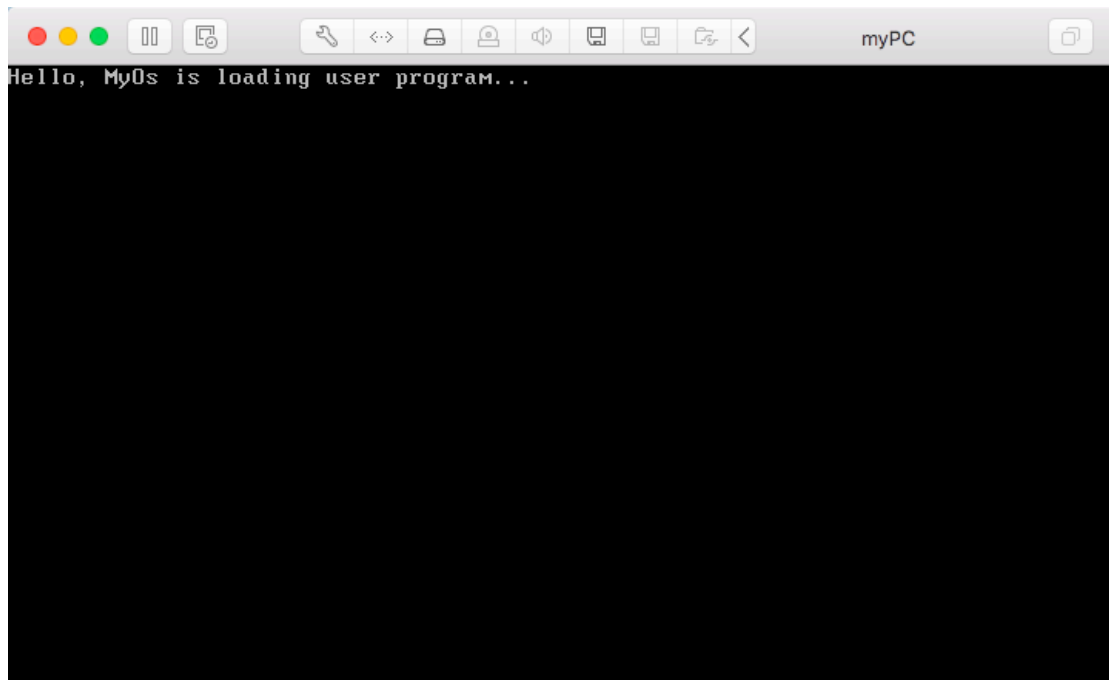
首先先延时操作，然后定义两个粒子轨迹，再者显示名字，**最后调用int 20h来检测是否退出程序**，并最终循环该主逻辑函数。

iv. 运动轨迹算法:

这次的运动过轨迹算法是经过进一步改良的，我把粒子的运动分为2步进行处理，先考虑纵向，再考虑横向。每次击墙后，会改变xdul, ydul，这俩个量会在下次运动时改变粒子的运动趋势。这样就只需要考虑4种情况，大大的减少了代码量。甚至还便于我们将代码模块化。

2. 最终效果展示:

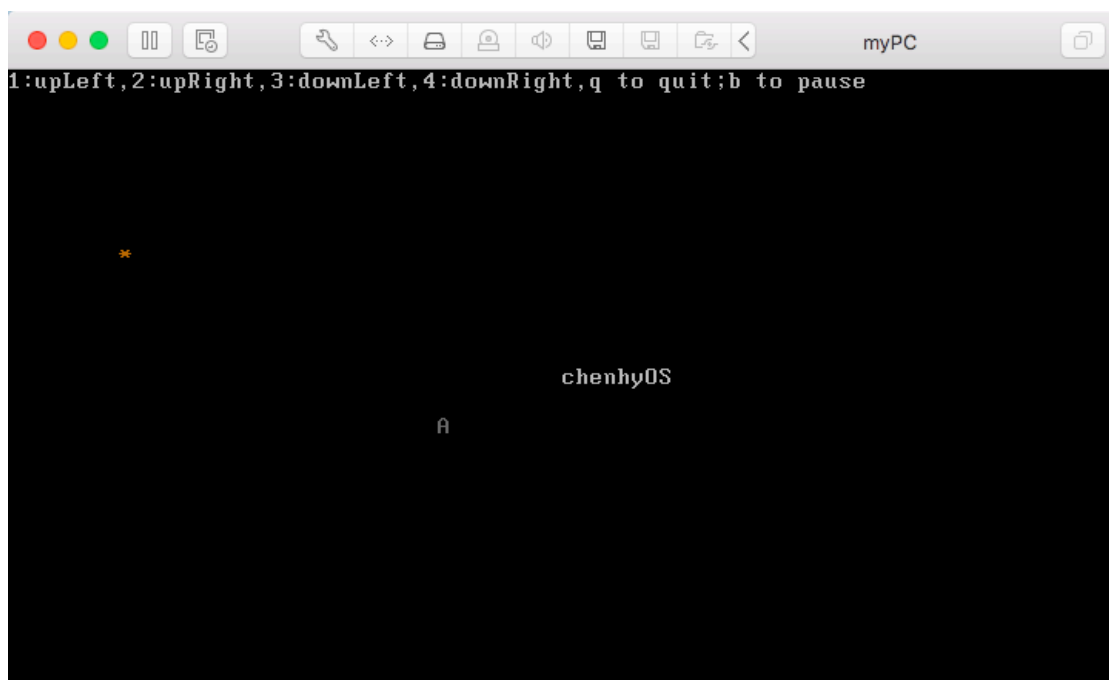
开机监控程序:



通过1, 2, 3, 4来选择加载上左, 上右, 下左, 下右四个方向的程序。

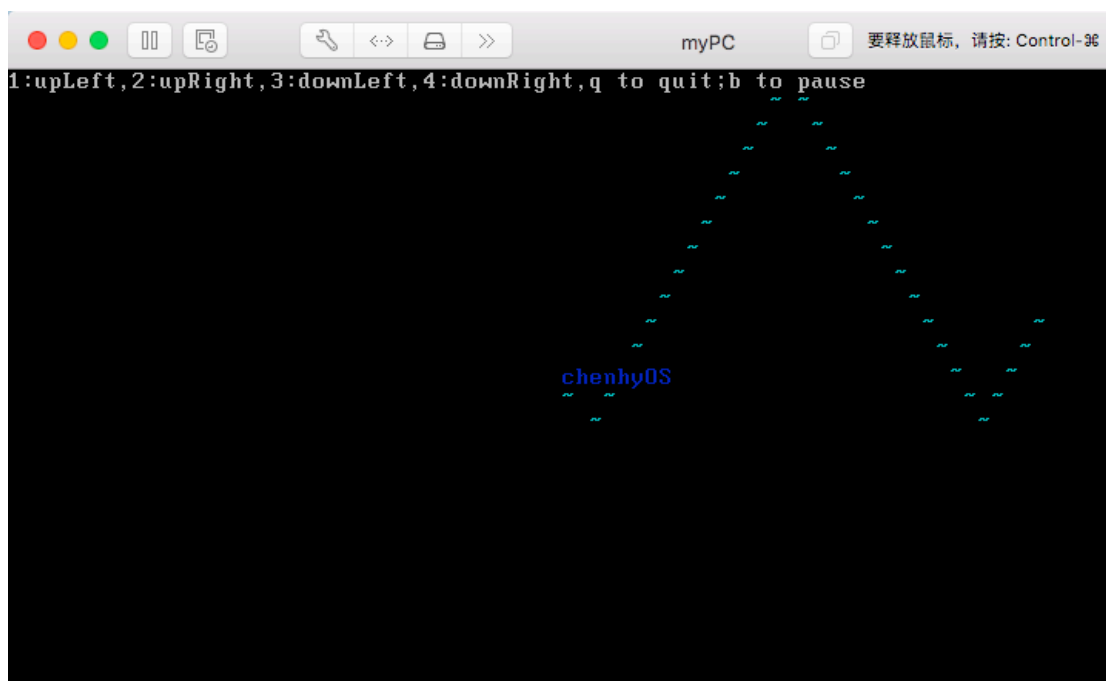
按1:

双变色粒子及变色名字:



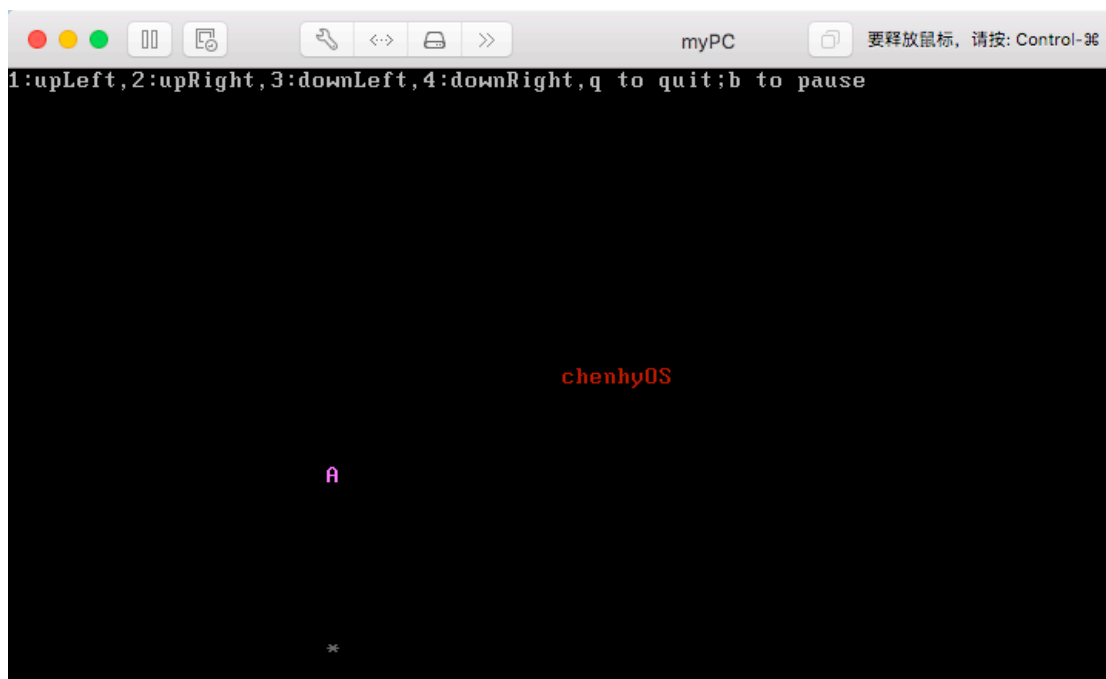
按2:

单色字符串:



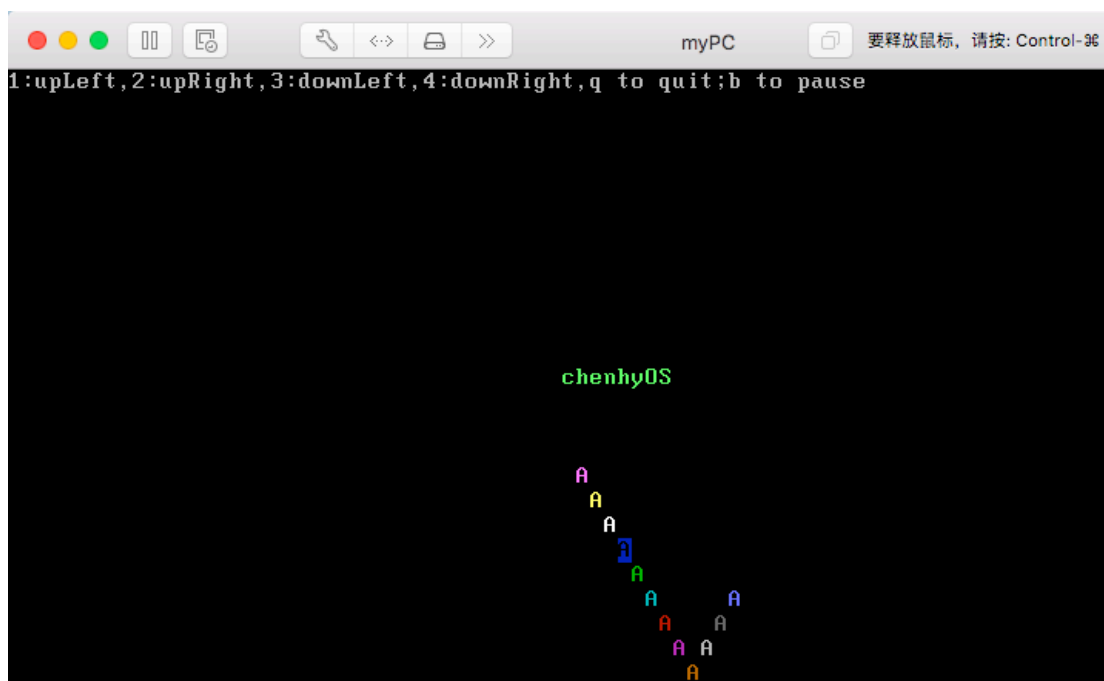
按3:

双单色粒子及名字显示



按4:

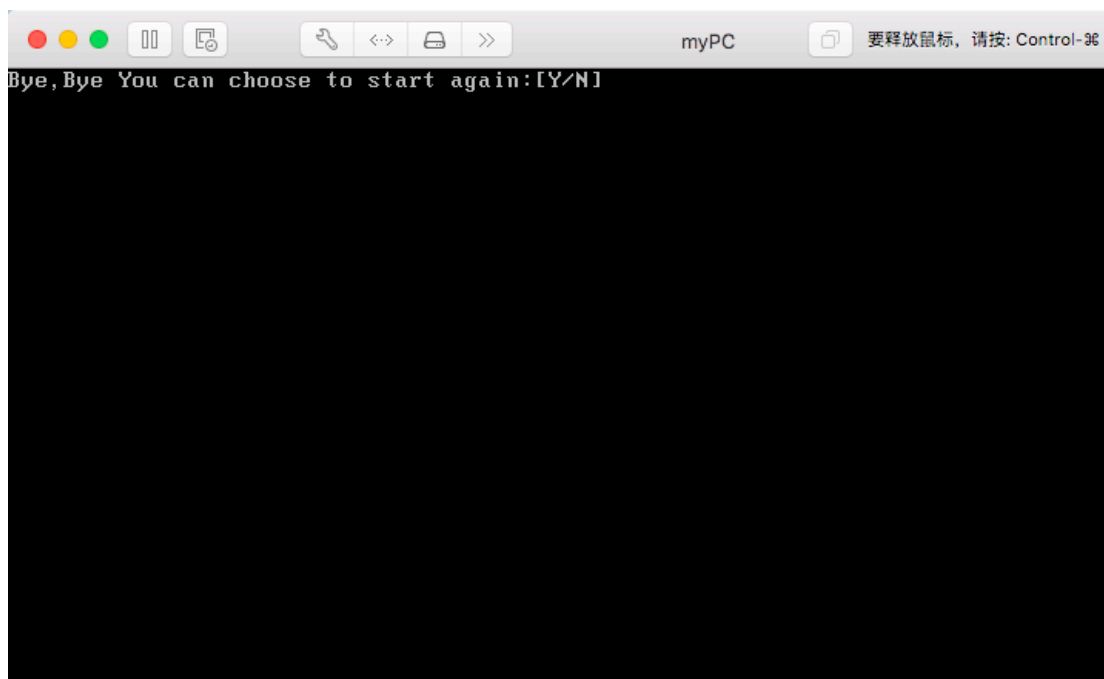
单变色字符串及变色名字显示



在进入用户程序之后，会在左上角显示不会被覆盖的提示信息。

这里做了个暂停机制，按b会暂停，然后按任意键会继续当前程序。详见视频演示。

又做了一个退出机制，按q会弹出一个选项



选y类似于会重新开机进入监控程序。而选n的话程序就会关机。

五. 实验心得

在写这次实验的过程中,经历了俩个过程。一个是对我的字符运动算法进行重构的过程,一个不断的查阅各种资料,思考如何实现由用户程序返回监控程序的过程。

由于之前的一次实验中,使用了老师的算法,这算法还是有一点小小的漏洞的,就是字符在运动的过程中有可能会跳出整个屏幕,而且这个算法比较大,无法增加我的新的功能。因此在学了几天汇编语言中的宏汇编之后,我开始着手用宏汇编的知识和新的算法来优化我的代码,由于采用了宏汇编,所以我可以减少我的代码量,甚至可以实现多粒子运动。这也是前一次实现所希望实现的。

但是,在这次实验中,重构粒子运动程序的代码这个过程并不是一个最关键的地方,毕竟我们这个监控程序目的是为了来加载用户程序的,无论用户程序是怎么样,只要他是正确的,只要他能够运行,只要他不超过512个字节,监控程序就需要可以把它加载出来。所以这个程序最关键的,同时也是让我学到最多的,就是思考并查阅如何由用户程序退回到监控程序的这一过程。

正如我前面所说的,用户程序中直接写个代码跳到监控程序这一过程是非常危险的,现实生活中,我也没讲过操作系统把自己的内核直接暴露在用户面前的行为。因此,我们必须思考一个类似于函数调用的功能,当用户想要退出程序了,就调用一下这个功能即可。但是一般的函数必须得要放在同一个文件下,才能编译的了,跨文件编译根本行不通。因此,我们必须借助于其他手段,来实现。

之前看书的查资料的时候,在讲述进程的时候有说到内核态与用户态之间的关系,但是彼时年少的我还不知道这个玩意儿有什么用。在这次查资料的时候,我又重新翻看了那一章,在看到了关于内核态与用户态是什么时候进行切换的时候,突然茅塞顿开。操作系统会在以下三种情况实现用户态到内核态的切换,包括系统调用,异常事件,外围设备的中断。后两者明显不会发生,但是系统调用是个好东西呀。这个时候我又想到了上学期计算机组成原理

这门课里面，关于软中断的知识。

也就是说，我们可以通过在中断向量表中新建一个退出中断，把中断程序存放在中断向量表的相应位置，这样，用户程序只要调用中断就能实现跳回了。但是关于这个中断的实现方式，我还有一些想法。在DOS系统中，int 21h，ah = 4ch，这个中断向量所要实现的就是退出程序返回DOS，那么DOS系统是如何实现的呢？在和舍友一起研究了free dos的源码之后，我知道：每一次调用用户程序之前，dos系统会在用户程序的前100h（PSP）的某一处地方，存下监控程序调用用户程序时的位置，当用户程序返回时，该程序会帮助用户在PSP中相应位置找到存起来的CS，IP，并赋给CS和IP，从而实现跳转。如果有可能，我也希望在我自己的操作系统中加入一个这样的机制：每次调用用户程序的时候，都会自动给程序开个PSP，这个PSP可以为用户提供一些进程切换所需要的空间。

我在写程序的时候，还有个这样的设想，就是在进程与进程之间切换的时候，能够保存进程在切换时的状态，当回到该进程时，还能从此状态继续执行。一开始，我的想法是，用栈来不断的存下切换时的状态，但是这样压栈的时候无法将进程与状态联系起来。有了PSP这个机制，我想，进程之间的切换，应该是可以实现的

参考资料：

1. BIOS中断、DOS中断、Linux中断的区别

<http://book.51cto.com/art/201604/509539.htm>

2. 中断向量表

<https://www.jianshu.com/p/9db2ab6a268f>

3. Linux探秘之用户态与内核态

<http://www.cnblogs.com/bakari/p/5520860.html>

4. 配置和使用bochs

<http://www.cnblogs.com/ccode/p/4273134.html>

5. ORANGE' S:一个操作系统的实现（于渊）