实验二：加载执行COM格式用户程序的监控程序

18340052 何泽

一、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

二、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。

三、实验内容

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 。

四、实验方案

1. 相关基础原理

- `BIOS` 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。
  - 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
  - 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

- `10H` （显示字符串）调用
  - `AL` ：放置光标的方式
    - 0/2：光标在串头
    - 1/3：光标在串尾
    - 0/1：串中只有字符
    - 2/3：串中字符和属性字节交替存储
  - `BL`
    - 位 `7` ：为 `1` 则闪烁
    - 位 `6-4` ：背景色 `RGB`
    - 位 `3` ：为 `1` 则前景色高亮
    - 位 `2-0` ：前景色 `RGB`
- `13H` （读扇区）调用
  - `AL` ：扇区数（ `1-255` ）
  - `DL` ：驱动器号（0和1表示软盘，80H和81H表示硬盘或U盘）
  - `DH` ：磁头号（ `0-15` ）
  - `CH` ：柱面号的低八位
  - `CL` ： `0-5` 位为起始扇区号（ `1-63` ）， `6-7` 位为硬盘柱面号高2位（共 `10` 位柱面号，取值 `0-1023` ）
  - `ES：BX` 为读入数据在内存中的存储地址
  - 返回值：
    - 操作完成后 `ES：BX` 指向数据区的起始地址
    - 出错时置进位标志 `CF=1` ，错误代码存放在 `AH` 中
    - 成功时 `CF=0` ，``AL=0
- 监控程序
  - 获取计算机硬件系统的控制权
  - 提供计算机输入设备和输出的控制程序
  - 控制用户程序的执行
- COM格式程序
  - `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。

2.实验环境与工具

- 平台：`Windows + Ubuntu`
- 汇编工具：`nasm`
- 创建空白软盘文件：`Bochs Disk Image Creation Tool`
- 虚拟机：`VMware Workstation 15`

3. 实验流程与思路

五、实验过程与结果

（这一部分只写最终结果。完成过程中遇到的问题、错误以及一步步检查问题并不断改进的过程将在下一板块“六、问题及解决方案”中详细叙述）

1. 能加载用户程序的监控程序

- 首先把引导扇区加载到`7c00h`处，并开始执行

```
org  7c00h
```

- 因为需要从用户程序返回`DOS`监控程序，所以每次返回时都要清屏，将用户程序输出的内容清除。

```
clear:
      mov ah,0x06
      mov al,0
      mov ch,0                ;左上角行号
      mov cl,0                ;左上角列号
      mov dh,24               ;右下角行号
      mov dl,79               ;右下角行号
      mov bh,0x0              ;用黑色填充
      int 10h
```

- 下面显示开头的字符串信息，我设计的是显示姓名、学号，输入`1-4`代表四个程序以及四个方向的提示信息，这里便用到了前面写到的`BIOS`调用。

```
;显示学号姓名
mov    bp, Name             ; BP为当前串的偏移地址
mov    ax, ds               ; ES:BP = 串地址
mov    es, ax
mov    cx, NameLength       ; CX为字符串长度
mov    ax, 1301h            ; 调用BIOS的13h功能，AL为01h即光标置于串尾
mov    bx, 0007h            ; 页号为0，黑底白字(BL = 07h)
mov    dh, 0                ; 行号为0
mov    dl, 1                ; 列号为0
int    10h                  ; 调用BIOS的10h功能，显示一行字符

;显示信息1
mov    bp, Message1         ; BP为当前串的偏移地址
mov    ax, ds               ; ES:BP = 串地址
mov    es, ax
```

```
mov    cx, Message1Length    ; CX为字符串长度
mov    ax, 1301h             ; 调用BIOS的13h功能，AL为01h即光标置于串尾
mov    bx, 0007h             ; 页号为0，黑底白字(BL = 07h)
mov    dh, 1                 ; 行号=1
mov    dl, 1                 ; 列号=0
int    10h                   ; 调用BIOS的10h功能：显示一行字符

;显示信息2
mov    bp, Message2          ; BP为当前串的偏移地址
mov    ax, ds                ; ES:BP = 串地址
mov    es, ax
mov    cx, Message2Length    ; CX为字符串长度
mov    ax, 1301h             ; 调用BIOS的13h功能，AL为01h即光标置于串尾
mov    bx, 0007h             ; 页号为0，黑底白字(BL = 07h)
mov    dh, 2                 ; 行号=2
mov    dl, 1                 ; 列号=0
int    10h                   ; 调用BIOS的10h功能：显示一行字符


Message1:
      db 'This is the second OS of mine. Enter 1-4 to choose the program:'
      Message1Length  equ ($-Message1)
Message2:
      db '1:up-left  2:up-right  3:down-left  4:down-right  ESC:return dos'
      Message2Length  equ ($-Message2)
Name:
      db 'Name: HeZe  Student Number:18340052'
      NameLength  equ ($-Name)
```

- 显示完字符后，便需要调用 `IO` 输入，输入 `1-4` 从而执行不同程序，这里判断输入哪个字符是通过 `ASCII` 判断的。

  调用IO输入，便是应用BIOS 中所支持的 16h 中断，其 00 功能阻塞输入，可以用来等待用户输入一个 字符。

```
listen_keyboard:

      mov ah,0
      int 0x16              ;输入一个字符

      cmp al,49             ;比较ASCII码
      je LU
      cmp al,50
      je RU
      cmp al,51
      je LD
      cmp al,52
      je RD

      jmp listen_keyboard   ;循环执行等待输入
```

- 事先确定好四个用户程序在内存中加载的位置，并承接上面的字符使用 call 指令调用对应的目的地址。

```
OffSetOfLU equ 0xA100
OffSetOfRU equ 0xB100
OffSetOfLD equ 0xC100
OffSetOfRD equ 0xD100
```

```
5
6
7   LU:
8         mov word[offset],OffSetOfLU
9         mov byte[sectionNum],2
10        jmp LoadnEx
11  RU:
12        mov word[offset],OffSetOfRU
13        mov byte[sectionNum],3
14        jmp LoadnEx
15  LD:
16        mov word[offset],OffSetOfLD
17        mov byte[sectionNum],4
18        jmp LoadnEx
19  RD:
20        mov word[offset],OffSetOfRD
21        mov byte[sectionNum],5
22        jmp LoadnEx
```

- 下面调用 `BIOS` 的 `13h` 即读磁盘功能，**将软盘上的物理扇区读到内存的 `ES：BX` 处**

```
1   mov ax,cs                ;段地址即存放数据的内存基地址
2   mov es,ax                ;设置段地址
3   mov bx, word[offset]     ;偏移地址即存放数据的内存偏移地址
4   mov ah,2                 ;功能号
5   mov al,1                 ;扇区数
6   mov dl,0                 ;驱动器号
7   mov dh,0                 ;磁头号
8   mov ch,0                 ;柱面号
9   mov cl,[sectionNum]      ;起始扇区号
10  int 13H                  ;调用读磁盘BIOS的13h功能
11
12  jmp [offset]
```

2. 用户程序

> 我设计的四个用户程序是左上、右上、左下、右下分别弹出数字1、2、3、4，并且起始颜色也不一样，同时每碰到边界就变换一种颜色。同时在这个过程中只要按下 `ESC` 键便可以返回到刚才的 `DOS` 界面。

- 清屏

  每一种用户程序需要先将字符清除，这与监控程序的清屏相同

```
1   mov ah,0x06
2   mov al,0
3   mov ch,0                 ;左上角行号
4   mov cl,0                 ;左上角列号
5   mov dh,24                ;右下角行号
6   mov dl,79                ;右下角行号
7   mov bh,0x0               ;用黑色来填充
8   int 10h
```

- 判断 `rdul` 为1、2、3、4则分别跳转到向右下，右上，左上，左下运动代码段

```
mov al,1
cmp al,byte[rdul]
jz  DnRt
mov al,2
cmp al,byte[rdul]
jz  UpRt
mov al,3
cmp al,byte[rdul]
jz  UpLt
mov al,4
cmp al,byte[rdul]
jz  DnLt
jmp $
```

- 向各个方向运动的代码段在不同的用户程序中参数是不同的
  - 在左上的程序中，x为 `0-13` ，y为 `0-40`
  - 在右上的程序中，x为 `0-13` ，y为 `40-80`
  - 在左下的程序中，x为 `13-25` ，y为 `0-40`
  - 在右下的程序中，x为 `13-25` ，y为 `40-80`

  下面的代码参数是右上程序的一个方向，其他方向和其他区域的代码只是参数不同，这里只放一个区域的一个方向作为示例，其他程序的参数详见源码，放在这也没什么必要~

  显卡按文本方式显示，最小可控制单位为字符

```
DnRt:
   ;x,y为屏幕矩阵横纵坐标
   inc word[x]
   inc word[y]
   mov bx,word[x]
   mov ax,13
   sub ax,bx
   ;down-right to up-right,x到达边缘,改变方向为右上
   jz  dr2ur

   mov bx,word[y]
   mov ax,80
   sub ax,bx
   ;down-right to down-left,y到达边缘,改变方向为左下
   jz  dr2dl
   jmp show
dr2ur:
   ;改变方向
    mov word[x],11
    mov byte[rdul],Up_Rt
   ;改变颜色
   jmp changeColor
dr2dl:
   mov word[y],78
   mov byte[rdul],Dn_Lt
   jmp changeColor
```

- 每碰撞一次就改变颜色

```
mov ax,word[color]
sub ax,1
mov byte[color],al
jnz show
mov byte[color],0Fh
```

- 显示

```
mov ax,word[x]                  ;计算显存地址
mov bx,80
mul bx                          ;(ax)=(ax)*80低位
add ax,word[y]                  ;(ax)=(ax)+y      80x+y
mov bx,2
mul bx                          ;(ax)=(ax)*2 1字节字符 1字节颜色
mov bp,ax                       ;bp为要显示字符的显存地址
mov ah,byte[color]              ;AH(ax高位)为颜色
mov al,byte[char]               ;AL(ax低位)显示字符值
mov word[gs:bp],ax              ;将字符值和颜色送到要显示字符的显存地址
```

- 在用户程序执行的过程中，使用非阻塞的输入，判断是否按下 `ESC` ，则检测缓冲区是否有字符，有的话则返回 `DOS` 程序

```
xor ax,ax
mov ah,1
int 0x16
cmp al,27
jne main
jmp 7c00h
```

- `datadef` 起始向右下运动，每个程序的起始颜色也都不一样

```
datadef:
    count dw delay
    dcount dw ddelay
    rdul db Dn_Rt             ; 开始默认向右下运动
    x    dw 7
    y    dw 40
    char db '2'
    color db 08h             ;颜色开始默认白色
    times 510-($-$$) db 0
    db 0x55,0xaa
```

3. 编译运行结果

- 先用 `nasm` 对5个程序编译成 `com` 格式程序

```
heze@ubuntu:~/Desktop/myos$ nasm -f bin boot.asm -o boot.com
heze@ubuntu:~/Desktop/myos$ nasm -f bin LU.asm -o LU.com
heze@ubuntu:~/Desktop/myos$ nasm -f bin LD.asm -o LD.com
heze@ubuntu:~/Desktop/myos$ nasm -f bin RD.asm -o RD.com
heze@ubuntu:~/Desktop/myos$ nasm -f bin RU.asm -o RU.com
```

- 然后为了将这5个程序连接到一起，我使用了 `Ubuntu` 下的 `cat` 命令

```
heze@ubuntu:~/Desktop/myos$ cat LU.com>>boot.com
heze@ubuntu:~/Desktop/myos$ cat RU.com>>boot.com
heze@ubuntu:~/Desktop/myos$ cat LD.com>>boot.com
heze@ubuntu:~/Desktop/myos$ cat RD.com>>boot.com
```

- 这样，监控程序就和4个用户程序结合到了一起
- 利用安装 `Bochs` 时一起安装的 `Bochs Disk Image Creation Tool` 创建 `1.44MB` 空白软盘

Disk Image Creation Tool

```
1. Create new floppy or hard disk image
2. Convert hard disk image to other format (mode)
3. Resize hard disk image
4. Commit 'undoable' redolog to base image
5. Disk image info

0. Quit

Please choose one [0] 1

Create image

Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd] fd

Choose the size of floppy disk image to create.
Please type 160k, 180k, 320k, 360k, 720k, 1.2M, 1.44M, 1.68M, 1.72M, or 2.88M.
 [1.44M]

What should be the name of the image?
[a.img] hz.img

Creating floppy image 'hz.img' with 2880 sectors

The following line should appear in your bochsrc:
  floppya: image="hz.img", status=inserted
(The line is stored in your windows clipboard, use CTRL-V to paste)

Press any key to continue
```

- 使用 `dd` 命令将 `com` 程序写入扇区

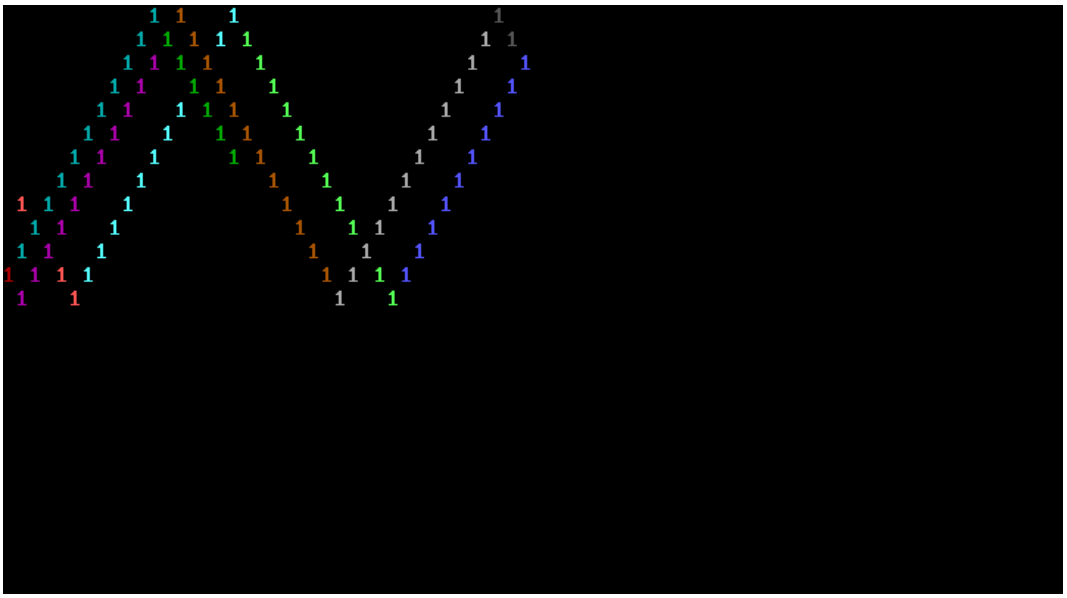
```
heze@ubuntu:~/Desktop/myos$ dd if=boot.com of=hz.img bs=1440 count=1 conv=notrunc
1+0 records in
1+0 records out
1440 bytes (1.4 kB, 1.4 KiB) copied, 0.00023756 s, 6.1 MB/s
```

- 然后就可以将这个 `img` 文件当做自己操作系统的启动软盘运行
- 之后便可以成功运行
- 刚运行时显示的 `DOS` 界面

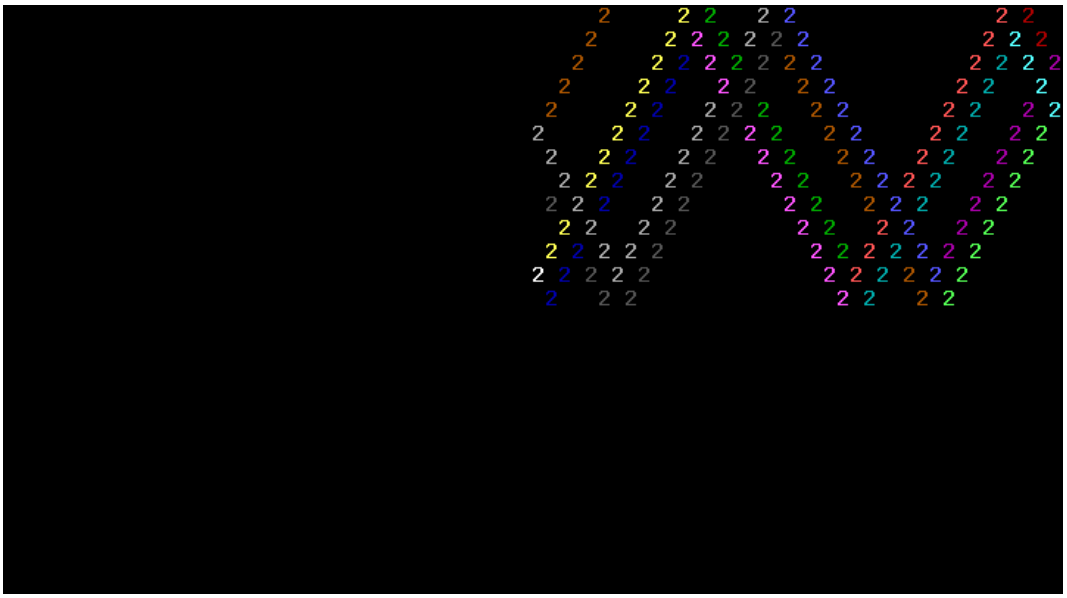
```
Name: HeZe  Student Number:18340052
This is the second OS of mine. Enter 1-4 to choose the program:
1:up-left  2:up-right  3:down-left  4:down-right  ESC:return dos
```

成功显示了应有的信息

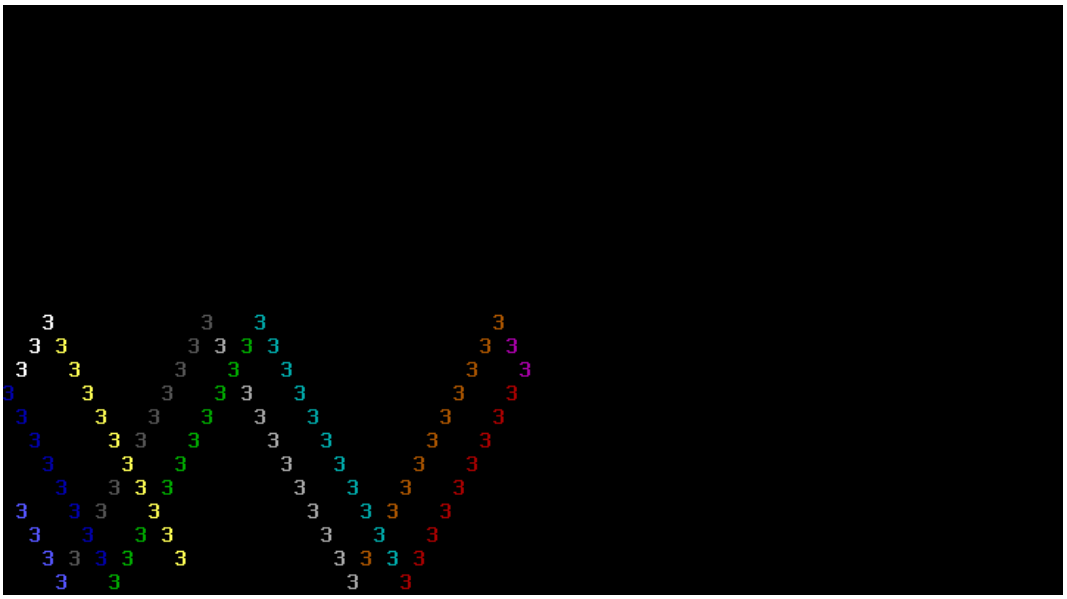
- 然后按下1-4便可分别进入4个用户程序，进入用户程序之后按下 `ESC` 便可返回 `DOS` 界面，返回后再次按下1-4可再次进入用户程序
    - 按下1



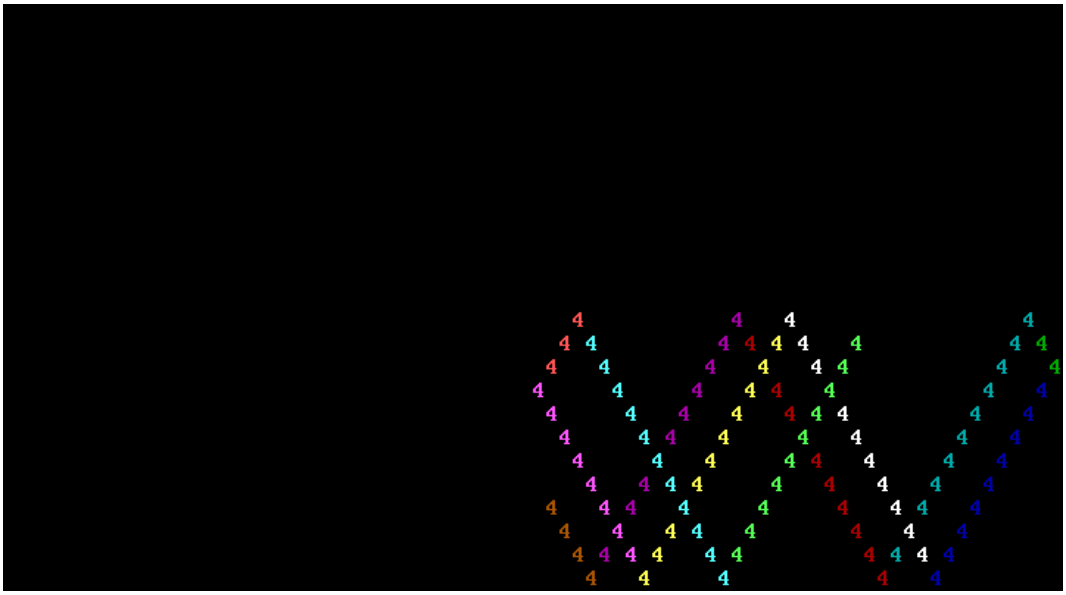
    - 按下2



    - 按下3



- 按下4



六、问题及解决方案

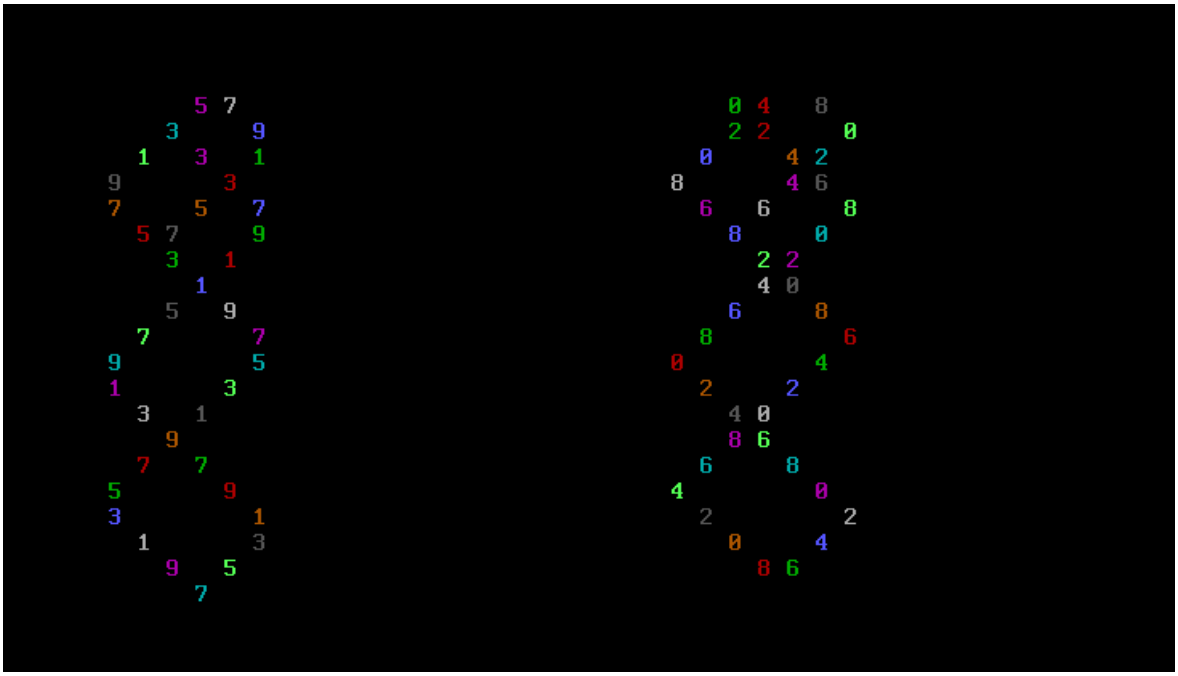
1.关于监控程序和用户程序的跳转

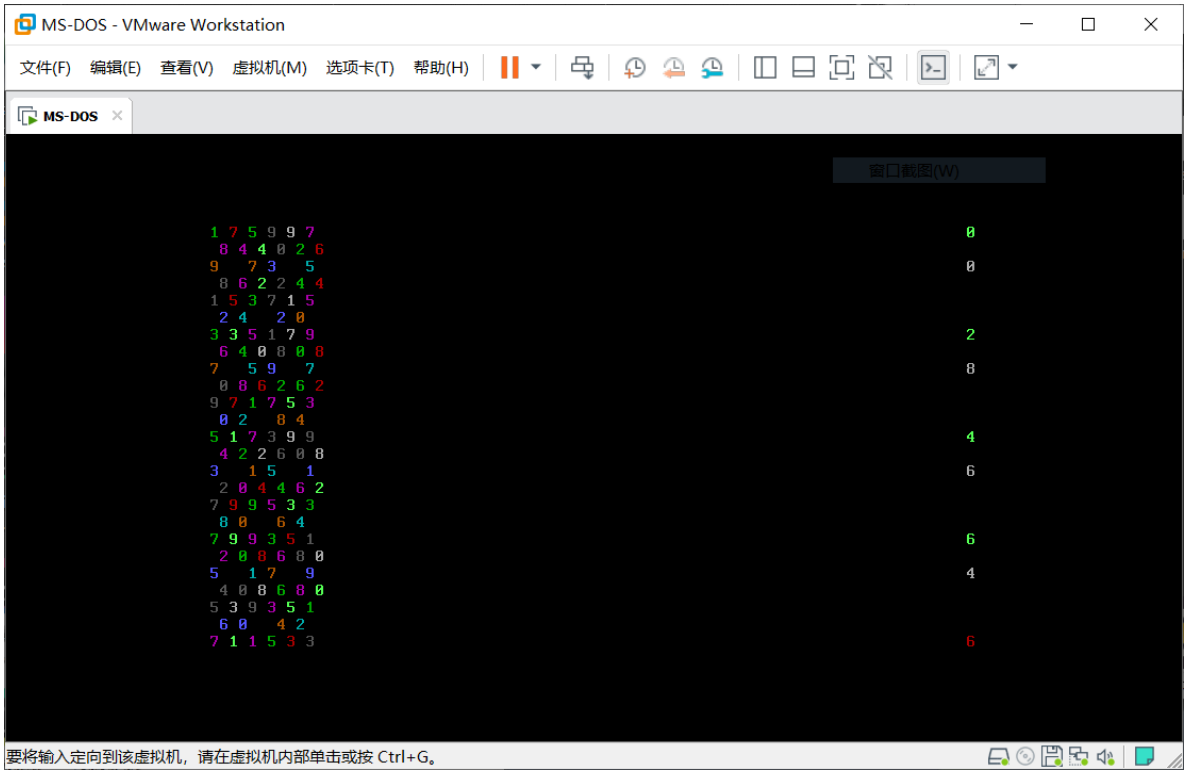
最开始，我以为从监控程序跳转到用户程序只需要确定加载位置后跳转即可，可试了之后发现不行。后来想了好久也和其他同学讨论后才明白是 `es` 寄存器的问题。因为 `ES：BX` 为缓冲区地址， `es` 寄存器会在运行过程中改变，所以需要特意设置它的值，之后便可正常跳转。

2. 关于用户程序的加载

用户程序的加载过程我出现了很多错误，比如起初从DOS按下数字键后会直接黑屏，按键都没有反应，这是程序加载的问题，寄存器赋值不对。

此外，还出现了乱码、反弹区域不对的错误，如下图：





上面两个都是我出现过的错误，按下1之后是以上的情况，不仅反弹区域不对，左上变成了左侧$\frac{1}{4}$，而且右侧还出现了诡异的数字，同时按 `ESC` 也没有反应。后来我发现是寄存器参数值不对，而且有的地方由于不细心有时把ax、bx寄存器搞混了，于是就导致显示范围出现了不可描述的错误。每个区域的参数都要调整，我也是费了九牛二虎之力最后才把参数调对，这期间也出了很多错误。，但还好最后成功运行了。

七、实验总结

这是操作系统的第二次实验，理论课上讲了进程的概念，而实验课要实现可加载用户程序的的监控程序。

起初我以为这很简单，只需要写一个简单的监控程序然后跳转到4个程序即可，用户程序还在实验一完成了，可真当我写的时候就发现问题超多。最开始就是用户程序加载不进来，按下1就死机；然后能加载了，我以为只不过是实验一的代码改个范围就好，可数字竟然在各种不正确的区域反弹，其实需要改很多的参数，同时对于各种寄存器的值也要额外关心。

在代码写好、编译完之后，怎么把这5个程序结合到一起也是个问题，我想过直接将这5个程序写入空白软盘的不同扇区，可后来还是选择了一种很“简单粗暴”的方式，就是用 `cat` 命令直接将程序拼到一起，然后写入扇区，甚至其实直接将 `.com` 后缀改成 `.img` 后缀在一些不太严格的环境下也是可以运行的也没什么大问题。

通过这次实验，我首先是加深了对监控程序概念的理解，掌握了 `BIOS` 的简单功能调用，包括显示字符，磁盘读写等等。此外，还了解了 `com` 格式程序。最重要的是，我明白了写操作系统还是要细心，各种参数都要慢慢、认真地去调，否则变会出现各种各样的错误。

总之，通过这次实验我的收获很多，目前由于只是汇编实现的功能还是很简单，之后的实验应该是要将汇编和C联合编译运行，这样可玩性和可以实现的功能就会复杂很多也会变得有趣很多，我会继续认真完成的。