实验二：加载执行COM格式用户程序的监控程序

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

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

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

实验过程：

1. 操作环境：
    - 操作系统：win10 + Linux
    - 虚拟机：VirtualBox
    - x86编译器：nasm

    实验一修改步骤：

    1. 将第一行代码 `org 7c00h` 修改为 `org 100h` ，使其满足 .com 程序的要求：这是因为 .com 载入内存后的起始偏址就是 100h 。前面的 100h 字节是该程序的 PSP 部分。所以，为了程序中对地址引用的正确，必需加上 `org 100h` 语句。

2. 添加如下代码

```
    mov ax,cs        ; 初始化ds和ss，原代码有初始化es
    mov ds,ax        ; 这样做是为了防止程序加载成功却出现乱码情况
    mov ss,ax
```

3. 修改反弹边沿的值，使得程序在 1/4 范围内弹射，同时特意修改了初始值和右边沿反弹方式，使得反弹轨道不会从四个角落飞出导致混乱。

   以其中一部分为例

```
DnRt:
    inc word[x]
    inc word[y]
    mov bx,word[x]
    mov ax,12                    ; 范围从 25 修改成 12
    sub ax,bx
    jz  dr2ur
    mov bx,word[y]
    mov ax,38                    ; 范围从 80 修改成 38
    sub ax,bx
    jz  dr2dl
    jmp show

dr2ur:
    mov word[x],10               ; 反弹值从 23 修改成 10
    mov byte[rdul],Up_Rt         ; 保持和范围的差值为 2
    jmp change

dr2dl:
    mov word[y],37               ; 反弹值从 78 修改成 37 ，把范围差值修改为 1
    mov byte[rdul],Dn_Lt         ; 这样做实是无心之举
    jmp change                   ; 却恰好使得反弹轨道不会从角落飞出

    ...

    x   dw  3                    ; 初始值从 3 开始，反弹轨道不会从角落飞出
    y   dw  0
```

4. 在显示字符之后，增加 `int 20h` 中断检测，实现可以从用户程序返回监控程序的功能（`int 20h` 的具体代码和技术细节将会在监控程序中讲解）
5. 添加计数变量 count2 = 100 ， 意义为显示 100 个字符后进入 `int 20h` 中断，实现显示 100 个字符后，用户程序自动停止并返回监控程序的功能

   关键代码：

```
    dec word[count2]             ; 计数变量--
    mov dx,word[count2]
    cmp dx,0                     ; 比较变量是否为 0
    jz end                       ; 为 0 则跳转到 end

    ...

end:
    int 20h                      ; 检测退出中断
```

6. 删除最后的 `dw 0xaa55` 代码，该代码为主引导扇区所特有的标识，同时将 `times 510-($-$$) db 0` 中的 510 修改为 512（尽管在实际测试中，如果不修改这些部分，实现的最终效果基本没有影响）
7. 保留之前有的显示学号姓名拼音，字母顺序变化，反弹遇到边缘变色，学号姓名闪烁效果，即完成了第一个用户程序
8. 将用户程序代码复制成 4 份，每份分别修改反弹范围和起始位置，具体表现为：
   - 第一个用户程序为左上角
   - 右上角的用户程序体现在 x 坐标 + 40
   - 左下角的用户程序体现在 y 坐标 + 13
   - 右下角的用户程序体现在 x 坐标 + 40，y 坐标 + 13

   至此，四份用户程序完成，分别命名为 1.asm / 2.asm / 3.asm / 4.asm
9. 将 asm 文件通过 Linux 下的 nasm 运行代码 `nasm 1.asm -o 1.com` 即可生成对应的com文件

   至此，得到了四份用户程序的com文件，分别命名为 1.com / 2.com / 3.com / 4.com

2. 在老师提供的 myos1.asm 基础上，我作出如下修改：
   1. 纠错：有两个错误地方
      - 第 2 行 `;org 7cooh` 中';'应删去，同时删掉后面的 `org 100h` ，使得程序成为引导扇区程序
      - 第 33 行 `jmp OffSetOfUserPrg1` 应改为 `jmp 800h:100h` ，因为执行结果前者 cs = 810h，IP = 0，后者 cs = 800h，IP = 100h，后者才能正确跳转至用户程序
   2. 添加选择功能：通过 `int 16h` 从键盘中读取一个字符，实现选择执行哪个用户程序的功能。同时具有异常处理功能，面对不规范的输入能不作反应

      关键代码：

```
input:
    mov ah,00h              ; 功能号：代表读入键盘输入的ascii码
    int 16h                 ; BIOS的16h功能：从键盘读入一个字符，读入到 AL 中
    cmp al,'1'              ; 判断越界( 是否 < 1 )
    jl input                ; 重新输入
    cmp al,'4'              ; 判断越界( 是否 > 4 )
    jg input                ; 重新输入
    sub al,'0'-1            ; 把 AL 转为读取的扇区号（从扇区 2 开始）
    mov byte[index],al

    ...

Message:
    db "Welcome to syj's os, please press 1-4 to run a program"
                            ; 修改提示信息，非最终版本
MessageLength  equ ($-Message)
```

      之后将老师代码 30 行 `mov cl,2` 改成 `mov cl,byte[index]` ，即可实现选择哪个用户程序来执行的功能
   3. 添加 `int 20h` 中断，实现从用户程序返回监控程序的功能：由于前面讲到用户程序显示 100 个字符后会自动停止，因此 `int 20h` 只需直接返回监控程序顶部即可

      关键代码：

```
    mov ax, 0000h               ; 内存前 64k 放置中断向量表，将段寄存器指向该处
```

```
    mov es, ax
    mov ax, 20h                 ; 定义 20 号中断：返回监控程序
    mov bx, 4
    mul bx
    mov si, ax
    mov ax, int20h              ; 定位自定义的 20 号中断位置
    mov [es:si], ax
    add si, 2
    mov ax, cs
    mov [es:si], ax             ; 存储 20 号中断进中断向量表

    ...

int20h:
    jmp start                   ; 这只是第一步，并不是最终的代码
```

4. 添加清屏函数，实现美观效果

   关键代码：

```
cls:
    mov bx,0b800h               ; 显存地址 0xb800
    mov es,bx
    mov bx,0
    mov cx,4000                 ; 设置循环次数：长*宽*2：80*25*2 = 4000字节
s:
    mov dx,0000h                ; 黑底无字
    mov [es:bx],dx
    add bx,2
    loop s                      ; 循环 4000 次
    ret                         ; 返回函数调用点
```

   至此，目前的监控程序已经可以实现：从键盘输入一个字符，选择一个用户程序进行调用，用户程序自动停止并返回监控程序，之后重复上述循环的功能

3. 实现创造命令，指定某种顺序运行用户程序的步骤如下：

   1. 在 input 后的从键盘读入字符的代码中，增加以下代码：

```
    cmp al,'5'          ; 新建功能：输入 5 跳转到"输入运行顺序"模块
    jz getstr
```

   2. 新建 " getstr " 代码块，实现输入顺序功能。关键代码如下：

      输出提示：

```
getstr:
    call cls
    mov ax, cs                  ; 置其他段寄存器值与CS相同
    mov ds, ax                  ; 数据段
    mov bp, Message2            ; BP = 当前串的偏移地址
    mov ax, ds                  ; ES:BP = 串地址
    mov es, ax                  ; 置ES = DS
    mov cx, MessageLength2      ; CX = 串长
    mov ax, 1301h               ; AH = 13h（功能号）、AL = 01h（光标置于串
尾）
    mov bx, 0007h               ; 页号为 0 (BH = 0) 黑底白字(BL = 07h)
    mov dh, 0                   ; 行号 = 0
```

```
    mov dl, 0                   ; 列号 = 0
    int 10h                     ; BIOS的10h功能：显示一行字符
    ; 提示信息为 "Please input 4 number, from 1 to 4:"
```

输入4个顺序并储存：

```
    mov ch,0
incre:
    inc ch
input2:
    mov ah,00h                  ; 功能号：代表读入键盘输入的ascii码
    int 16h                     ; BIOS的16h功能：从键盘读入一个字符
    cmp al,'1'                  ; 判断越界( 是否 < 1 )
    jl input2                   ; 重新输入
    cmp al,'4'                  ; 判断越界( 是否 > 4 )
    jg input2                   ; 重新输入
    sub al,'0'-1                ; 转换成扇区号( 从 2 开始 )
    cmp ch,1
    jz save1
    cmp ch,2
    jz save2
    cmp ch,3
    jz save3
    cmp ch,4
    jz save4
save1:
    mov ah,0
    mov word[c1],ax             ; 第 1 个顺序存在 c1 中
    jmp incre
save2:
    mov ah,0
    mov word[c2],ax             ; 第 2 个顺序存在 c2 中
    jmp incre
save3:
    mov ah,0
    mov word[c3],ax             ; 第 3 个顺序存在 c3 中
    jmp incre
save4:
    mov ah,0
    mov word[c4],ax             ; 第 4 个顺序存在 c4 中
```

按顺序调用程序：

```
    mov word[c5],0              ; 计数变量
    call cls                    ; 调用清屏函数
run:
    inc word[c5]
    mov dx, word[c5]
    cmp dx,1
    jz change1
    cmp dx,2
    jz change2
    cmp dx,3
    jz change3
    cmp dx,4
    jz change4
```

```
run2:
    mov byte[index],al
    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, OffSetOfUserPrg1    ; 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[index]          ; 起始扇区号：起始编号为1
    int 13H                     ; BIOS的13h功能：读软盘或硬盘上的若干物理扇区
到内存的ES:BX处
    jmp 800h:100h               ; 跳转至用户程序

change1:
    mov ax,word[c1]             ; 取出第 1 个顺序
    jmp run2
change2:
    mov ax,word[c2]             ; 取出第 2 个顺序
    jmp run2
change3:
    mov ax,word[c3]             ; 取出第 3 个顺序
    jmp run2
change4:
    mov ax,word[c4]             ; 取出第 4 个顺序
    jmp run2
```

修改中断代码：

```
judge:
    mov ax,cs
    mov ds,ax                   ; 重置数据段
    mov ss,ax
    mov dx, word[c5]
    cmp dx,5
    jz start
 cmp dx,4
    jz start                    ; 回到监控程序
    jmp run                     ; 继续顺序执行

    ...

int20h:
    jmp judge
```

修改提示信息：

```
Message:
    db "Welcome to syj's os, please press 1-4 to run a program, press 5
to input running sequence:"
MessageLength  equ ($-Message)
Message2:
    db "Please input 4 number, from 1 to 4:"
MessageLength2  equ ($-Message2)
```

至此，监控程序可以实现：输入 5 进入输入顺序模式，从键盘中读取 4 个字符（1-4范围），按照顺序依次执行相应的用户程序（允许重复）

4. 映像盘上共存有 4 个用户程序，分别存在除引导扇区之外的第 1-4 个扇区

   以表格形式表达用户程序存储位置如下图：

引导扇区	1扇区	2扇区	3扇区	4扇区	…
监控程序	用户程序1	用户程序2	用户程序3	用户程序4	

5. 自己的软件项目管理目录如下图：

6. 将所有com文件整合成img文件步骤如下：

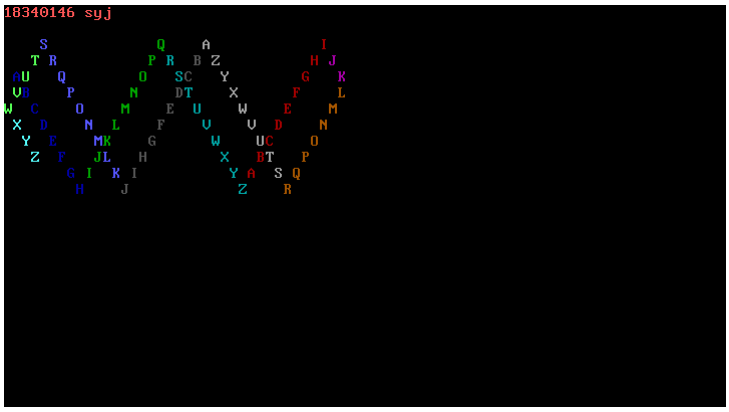
   1. Linux 环境下通过代码 `/sbin/mkfs.msdos -C syjos.img 1440` 新建一个名为 syjos.img ，大小为 1440kb 即 1.44mb 的 img 文件
   2. 通过代码 `dd if=syjos.com of=syjos.img conv=notrunc` 将监控程序 syjos.com 文件复制进 syjos.img 文件的引导扇区
   3. 通过代码 `dd if=1.com of=syjos.img seek=1 conv=notrunc` 将用户程序 1.com 文件复制进 syjos.img 文件的 1 扇区，其它用户程序同理，只需修改 if 后的文件名和 seek 后的扇区位置即可

```
syj@ubuntu:~/Desktop/chao$ nasm 1.asm -o 1.com
syj@ubuntu:~/Desktop/chao$ nasm 2.asm -o 2.com
syj@ubuntu:~/Desktop/chao$ nasm 3.asm -o 3.com
syj@ubuntu:~/Desktop/chao$ nasm 4.asm -o 4.com
syj@ubuntu:~/Desktop/chao$ /sbin/mkfs.msdos -C syjos.img 1440
mkfs.fat 3.0.28 (2015-05-16)
syj@ubuntu:~/Desktop/chao$ dd if=syjos.com of=syjos.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000193223 s, 2.6 MB/s
syj@ubuntu:~/Desktop/chao$ dd if=1.com of=syjos.img seek=1 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000163911 s, 3.1 MB/s
syj@ubuntu:~/Desktop/chao$ dd if=2.com of=syjos.img seek=2 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000195764 s, 2.6 MB/s
syj@ubuntu:~/Desktop/chao$ dd if=3.com of=syjos.img seek=3 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000179636 s, 2.9 MB/s
syj@ubuntu:~/Desktop/chao$ dd if=4.com of=syjos.img seek=4 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000166237 s, 3.1 MB/s
syj@ubuntu:~/Desktop/chao$
```

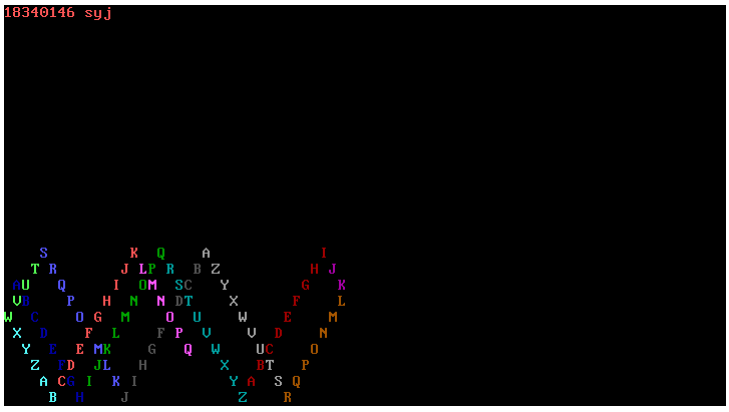
7. 测试效果如下

1. virtualbox虚拟机打开 syjos.img 文件：

2. 键盘按下 1， 监控程序将用户程序 1 读入内存并运行，运行一段时间后返回监控程序



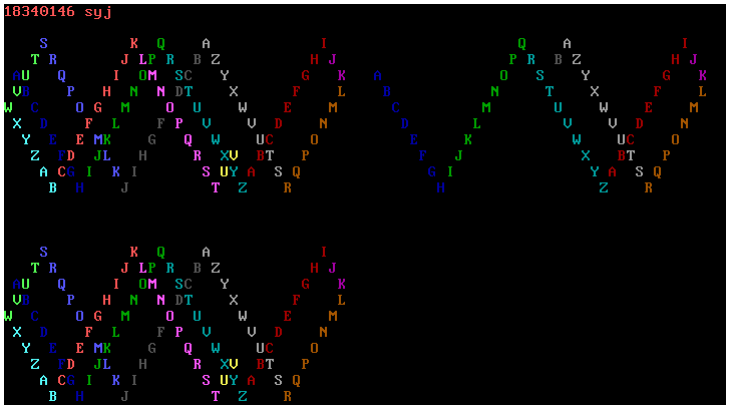
3. 键盘按下 3， 监控程序将用户程序 3 读入内存并运行，运行一段时间后返回监控程序



4. 键盘按下 5， 监控程序进入输入顺序阶段

5. 按下 1324， 监控程序按照 1324 的顺序显示，显示完后返回监控程序

(图为到用户程序 2 时的截图)



实验心得：

本次实验的主要流程其实走了不少弯路，也遇到不少困难：

1. 一开始选择修改用户程序，实现键盘输入字符即停止，后来发现该情况很难实现自动顺序执行（因为需要手动停止），因此改成了输出 100 个字符后自动停止
2. 在自己的网络搜索范围内似乎没有找到 BIOS 直接输入字符串的中断码，只能重复调用 int 16h，x86申请数组的方式网上也基本找不到资料，因此限定了顺序长度为 4
3. 老师源反弹程序如果从角落射出将会有 bug，导致轨道射出边框，多次实验调整了范围大小（甚至碰巧修改了右边框反弹形式后），实现轨道不会经过角落
4. 通过判断用户程序执行完后是按照顺序执行下一个，还是直接返回监控程序，也需要很多时间思考如何实现，最后使用一个“全局”变量来标识是顺序执行阶段还是单一用户程序阶段
5. 上一步引来“全局”变量数据段问题，每次中断之后，数据段需要定义回监控程序的数据段，同时也需要把全局变量的值再 mov 给通用寄存器进行值的判断
6. 再引来通用寄存器问题：百度得知 SI、SP、BP、DI也属于通用寄存器，但是不可随意用来存储数据（否则虚拟机直接死机）

但是解决完所有问题并成功写出最后的 img 文件时，还是有不少知识点的收获的，比如说 x86 的通用寄存器知识，中断向量表处理知识，linux 环境下生成 img 文件并将 com 格式文件拷贝进相应扇区的知识等等。总的来说很累，很迷茫，最后还是有不少收获。