

# 操作系统-实验四

---

学号：18340020 姓名：陈贤彪 学院：数据科学与计算机学院

## 1.实验目的

---

- 1、PC系统的中断机制和原理
- 2、理解操作系统内核对异步事件的处理方法
- 3、掌握中断处理编程的方法
- 4、掌握内核中断处理代码组织的设计方法
- 5、了解查询式I/O控制方式的编程方法

## 2.实验要求

---

- 1、知道PC系统的中断硬件系统的原理
- 2、掌握 x86 汇编语言对时钟中断的响应处理编程方法
- 3、重写和扩展实验三的的内核程序，增加时钟中断的响应处理和键盘中断响应。
- 4、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

## 3.实验内容

---

- (1)编写 x86 汇编语言对时钟中断的响应处理程序：设计一个汇编程序，在一段时间内系统时钟中断发生时，屏幕变化显示信息。在屏幕24行79列位置轮流显示'|'、'/'和'\'(无敌风火轮)，适当控制显示速度，以方便观察效果，也可以屏幕上画框、反弹字符等，方便观察时钟中断多次发生。将程序生成COM格式程序，在DOS或虚拟环境运行。
- (2)重写和扩展实验三的的内核程序，增加时钟中断的响应处理和键盘中断响应。，在屏幕右下角显示一个转动的无敌风火轮，确保内核功能不比实验三的程序弱，展示原有功能或加强功能可以工作。
- (4)扩展实验三的的内核程序，但不修改原有的用户程序，实现在用户程序执行期间，若触碰键盘，屏幕某个位置会显示"OUCH!OUCH!"。
- (5)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

## 4.实验方案

---

### 1) 实验环境

a)系统：Linux Ubuntu18.04

### 2) 实验工具

a) VM VirtualBox

虚拟机软件，用于模拟虚拟不同的操作系统，也可以创建多个虚拟软盘

b) NASM-2.13.02

汇编语言编译器, 可以将写好的 .asm 文件编译成二进制文件.bin

c) gcc (Ubuntu 5.5.0-12ubuntu1) 5.5.0 20171010

c语言编译器

d) Visual Studio Code

代码编辑器, 用于编辑 .asm 代码

e) GNU bash version 4.4.20(1)-release (x86\_64-pc-linux-gnu)

系统跟计算机硬件交互时使用的中间介质, 用于简便对文件进行转换

f) GNU ld (GNU Binutils for Ubuntu) 2.30

链接器, 将汇编与c生成的.o文件链接在一起

f) github

源代码托管平台, 用于存储管理编写的代码

g) bochs 2.6.11

软盘调试工具

### 3) 实验原理

#### (1) 中断技术

中断(interrupt)是指对处理器正常处理过程的打断。中断与异常一样, 都是在程序执行过程中的强制性转移, 转移到相应的处理程序。

- 硬中断 (外部中断) —— 由外部 (主要是外设[即I/O设备]) 的请求引起的中断、

时钟中断 (计时器产生, 等间隔执行特定功能)

I/O中断 (I/O控制器产生, 通知操作完成或错误条件)

硬件故障中断 (故障产生, 如掉电或内存奇偶校验错误)

- 软中断 (内部中断) —— 由指令的执行引起的中断

中断指令 (软中断int n、溢出中断into、中断返回 iret、单步中断 TF=1)

- 异常/程序中断 (指令执行结果产生, 如溢出、除0、非法指令、越界)

#### (2) PC中断处理过程

- 硬件实现的保护断点现场

要将标志寄存器FLAGS压栈, 然后清除它的IF位和TF位

再将当前的代码段寄存器CS和指令指针寄存器IP压栈

- 执行中断处理程序

由于处理器已经拿到了中断号, 它将该号码乘以4 (毕竟每个中断在中断向量表中占4字节), 就得到了该中断入口点在中断向量表中的偏移地址

从表中依次取出中断程序的偏移地址和段地址, 并分别传送到 IP 和 CS, 自然地, 处理器就开始执行中断处理程序了。

注意, 由于IF标志被清除, 在中断处理过程中, 处理器将不再响应硬件中断。如果希望更高优先级的中断嵌套, 可以在编写中断处理程序时, 适时用 sti 指令开放中断。

- 返回到断点接着执行

所有中断处理程序的最后一条指令必须是中断返回指令 iret。这将导致处理器依次从堆栈中弹出 (恢复) IP、CS和FLAGS的原始内容, 于是转到主程序接着执行。

(3) 总体磁盘架构

磁头号	扇区偏移	占用扇区数	功能
0	1	1	引导扇区程序
0	2	17	操作系统内核
1	1	2	userpro1
1	3	2	userpro2
1	5	2	userpro3
1	7	2	userpro4
1	9	2	用户程序表 list

userpro1 对应左上角滚动字符， userpro2 对应又上角滚动字符， userpro3 对应左下角滚动字符， userpro4 对应右下角滚动字符。

在内核中，我创建了一个结构体存储用户程序的信息

(4) MY操作系统内核的设计

内核程序结构：

程序名	代码形式	作用
kernel.asm	ASM	内核的入口，调用c中的函数， 新增运行风火轮代码
kernel_a.asm	ASM	包含一些显示打印，IO借口，扇区加载的函数,新增运行 ouch 代码
stdio.h	C	包含一些对字符串处理的函数
kernel_c.asm	C	内核c代码，内核命令的主要部分
ouch.asm	ASM	新增文件：封装了关于 ouch 键盘中断的代码

以下是我已经实现的指令功能（新增了关机指令）

指令名	功能
clear	清楚屏幕
ls	调用 list 程序显示用户程序的信息，需要 按 esc 退出
help	显示帮助的信息
run	可以按照自定义顺序运行1234程序
time	显示当前系统时间
shutdown	关机（新增）

(5) 无敌风火轮的实现

由于中断的编写在进入内核的开始就要使用，因此该部分代码我直接加在 kernel/kenerl.asm 里面。

首先，无敌风火轮的具体实现是在屏幕右下角（第24行，第79列）的位置循环依次显示' -\|/' 这四个字符。我的实现方式就是有一个' -\|/'的字符串，还有一个记录字符偏移的数字，每次进入时钟中断便在屏幕右下角，右上角，左下角显示字符串中对应字符偏移的字符，之后字符偏移+1，字符偏移到4则回到1，如此下去。

实现代码如下：

```
1  wudifenghuolun:
2      pusha
3      push ds
4      push gs;压栈保护寄存器
5      xor ax,ax
6      mov ax,cs
7      mov ds,ax      ; DS = CS
8      mov ax,0B800h   ; 文本窗口显存起始地址
9      mov gs,ax      ; GS = B800h
10
11     dec byte[count];增加延迟使效果更好
12     jnz end
13     mov byte[count],delay
14     mov ah,0Fh
15     mov si,fenghuolun;取字符串首地址
16     add si,[fenghuoluncount];然后加上偏移量
17     inc byte[fenghuoluncount];偏移量+1
18     mov al,[si]
19     mov [gs:((80*24+79)*2)],ax;右下角
20     mov [gs:((80*0+79)*2)],ax;右上角
21     mov [gs:((80*24+0)*2)],ax;左下角
22     cmp byte[fenghuoluncount],4
23     jne end
24     mov byte[fenghuoluncount],0;若偏移量达到4，则变回0
25 end:
26     mov al,20h      ; AL = EOI
27     out 20h,al      ; 发送EOI到主8529A
28     out 0A0h,al     ; 发送EOI到从8529A
29     pop gs
30     pop ds;出栈
31     popa
32     iret
33
34 Data:
35     delay equ 1
36     count db delay
37     fenghuolun db '\|/-'
38     fenghuoluncount dw 0
```

经过将delay调整过后，发现delay为1的时候，转动看起来比较清晰

程序写完后，便是向量表的写入，时钟中断的中断号位置是8号，每个中断号占用4个字节地址，因此时钟中断的字节地址是 ip=20h cs=22h，因为我写了一个 write\_vector\_timer 函数来写入，具体代码如下：

```

1  write_vector_timer:
2      pusha
3      xor ax,ax      ; AX = 0
4      mov es,ax      ; ES = 0
5      mov word [es:20h],wudifenghuolun ;设置时钟中断向量的无敌风火轮偏移地址
6      mov ax,cs
7      mov word [es:22h],ax ;设置时钟中断向量的段地址=CS
8      mov ds,ax      ; DS = CS
9      mov es,ax      ; ES = CS
10     popa
11     ret

```

然后在进入我的内核开始的时候，调用该函数完成了

```

1  call dword write_vector_timer

```

## (6) ouch 的实现

实现在用户程序执行期间，若触碰键盘，屏幕某个位置会显示"OUCH!OUCH!"

首先我需要编写一个在屏幕中间显示"OUCH!OUCH!"的代码，实现代码在 kernel/ouch.asm 的后部分，实现如下：

```

1  BITS 16
2  global ouch
3  ouch:
4      pusha
5      push es
6      push ds
7
8      xor ax,ax
9      mov ax,cs
10     mov ds,ax
11     mov bp, ouch_mess ; BP=当前串的偏移地址
12     mov ax, ds        ; ES:BP = 串地址
13     mov es, ax        ; 置ES=DS
14     mov cx, ouch_mess_len ; CX = 串长
15     mov ax, 1300h     ; AH = 13h (功能号)、AL = 00h (光标不动)
16     mov bx, 0007h     ; 页号为0(BH = 0) 黑底白字(BL = 07h)
17     mov dh, 13        ; 行号=0
18     mov dl, 31        ; 列号=0
19     int 10h           ; BIOS的10h功能：显示一行字符
20
21     loopdelay:
22         dec word[count]
23         jnz loopdelay
24         mov word[count],delay
25         dec word[dcount]
26         jnz loopdelay
27         mov word[count],delay
28         mov word[dcount],ddelay
29
30     mov ax, cs        ; 置其他段寄存器值与CS相同
31     mov ds, ax        ; 数据段
32     mov bp, kongbai ; BP=当前串的偏移地址
33     mov ax, ds        ; ES:BP = 串地址

```

```

34     mov es, ax      ;置ES=DS
35     mov cx,ouch_mess_len; CX = 串长
36     mov ax, 1300h   ; AH = 13h (功能号)、AL = 00h (光标不动)
37     mov bx, 0007h   ; 页号为0(BH = 0) 黑底白字(BL = 07h)
38     mov dh, 13      ; 行号=0
39     mov dl, 31      ; 列号=0
40     int 10h         ; BIOS的10h功能: 显示一行字符
41
42     int 37h         ; 调用原来的键盘中断
43
44     mov al,20h       ; AL = EOI
45     out 20h,al       ; 发送EOI到主8529A
46     out 0A0h,al      ; 发送EOI到从8529A
47     pop ds
48     pop es
49     popa
50     iret
51
52 data:
53     delay equ 55000
54     ddelay equ 580
55     count dw delay
56     dcount dw ddelay
57     ouch_mess db 'Ouch!Ouch!'
58     ouch_mess_len equ ($-ouch_mess)
59     kongbai db '      '

```

该部分代码实现比较简单，过程分成三部分，第一部分就是调用 10h 中断在13行31列显示'Ouch!Ouch!'，然后经过一个delay保留该字符，然后调用 10h 中断，在13行31列显示空格（抹去原来的字符），这样就可以产生每次按键盘就显示'Ouch!Ouch!'。

但是键盘中断是原本是有代码的，所以我不能够直接将该 ouch.asm 的偏移地址直接放上9号中断，因此我需要保留原来的9号中断，将其放到没有使用的 37h 号中断上，然后在我调用9号中断的代码上在调用 37h 中断，这样看起来就是都调用了。下面是该中断地址的代码

```

1  moveto:
2  push ax
3  push es
4  push si
5  mov ax, 0
6  mov es, ax
7  mov si, [es:09h*4];将9h中断放到37h中断上
8  mov [es:37h*4], si
9  mov si, [es:09h*4+2]
10 mov [es:37h*4+2], si
11 pop si
12 pop es
13 pop ax
14 ret

```

因为这个ouch的实现只需要在调用用户程序的时候才调用，因此还需要一个写回9h中断的代码

```

1  moveback:
2  push ax
3  push es
4  push si

```

```

5    mov ax, 0
6    mov es, ax
7    mov si, [es:37h*4]
8    mov [es:09h*4], si
9    mov si, [es:37h*4+2]
10   mov [es:09h*4+2], si
11   pop si
12   pop es
13   pop ax
14   ret

```

最后在调用用户程序之前，移动向量偏移，然后将ouch的偏移写进9h向量上，改写原来的loadrun（运行用户程序函数）如下：

```

1    loadrun:
2    pusha
3    mov bp, sp
4    add bp, 20; pusha的栈跳过
5    call moveto ;将9h中断写到37h上
6    call write_vector_ouch;将ouch代码放进9h中断
7    LOADPRO offset_program1, 2, [bp], [bp+4]
8    call dword offset_program1
9    call moveback;写回向量
10   popa
11   retf

```

## （7）新增shutdown关机指令

做着做着操作系统发现自己每次关闭都是手动的关机，然后就想着是否可以通过一些调用来自动关闭操作系统。于是上网查了一下关于汇编关机指令的资料，然后就写了这个指令

实现代码位于 kernel\_a.asm，

```

1    global shutdown
2    shutdown:
3    pusha
4    mov ax, 5307h
5    mov bx, 0001h
6    mov cx, 0003h
7    int 15h
8    popa
9    retf

```

然后在c模块，使用指令来调用

```

1    extern void shutdown();
2    else if(strcmp(first, commands[5])==0)//shutdown
3    {
4        shutdown();
5    }

```

## （8）程序的编译与整合

由于程序的编译以及整合是一个大量重复工作，因此我使用bash脚本来快速进行编译与整合，本次实验加上的文件只有 `ouch.asm`，因此只需要增加编译它并汇总便可

`combine.sh`

```
1  #!/bin/bash
2  rm -rf temp
3  mkdir temp
4  rm *.img
5
6  nasm booter.asm -o ./temp/booter.bin
7
8  cd usrprog
9  nasm topleft.asm -o ../temp/topleft.com
10 nasm topright.asm -o ../temp/topright.bin
11 nasm bottomleft.asm -o ../temp/bottomleft.bin
12 nasm bottomright.asm -o ../temp/bottomright.bin
13 nasm list.asm -o ../temp/list.bin
14 cd ..
15
16 cd kernel
17 nasm -f elf32 kernel.asm -o ../temp/kernel.o
18 nasm -f elf32 kernel_a.asm -o ../temp/kernel_a.o
19 nasm -f elf32 ouch.asm -o ../temp/ouch.o
20 gcc -c -m16 -march=i386 -masm=intel -nostdlib -ffreestanding -mpreferred-stack-boundary=2 -lgcc
   -shared kernel_c.c -fno-pic -o ../temp/kernel_c.o
21 ld -m elf_i386 -N -Ttext 0x7e00 --oformat binary ../temp/kernel.o ../temp/kernel_a.o
   ../temp/kernel_c.o ../temp/ouch.o -o ../temp/kernel.bin
22 cd ..
23 rm ./temp/*.o
24
25 dd if=./temp/booter.bin of=myosv4.img bs=512 count=1 2>/dev/null
26 dd if=./temp/kernel.bin of=myosv4.img bs=512 seek=1 count=17 2>/dev/null
27 dd if=./temp/topleft.com of=myosv4.img bs=512 seek=18 count=2 2>/dev/null
28 dd if=./temp/topright.bin of=myosv4.img bs=512 seek=20 count=2 2>/dev/null
29 dd if=./temp/bottomleft.bin of=myosv4.img bs=512 seek=22 count=2 2>/dev/null
30 dd if=./temp/bottomright.bin of=myosv4.img bs=512 seek=24 count=2 2>/dev/null
31 dd if=./temp/list.bin of=myosv4.img bs=512 seek=26 count=2 2>/dev/null
32 echo "[+] Done."
```

该脚本需要严格对应磁盘的放置，譬如 `dd` 时的扇区号，以及 `ld` 中 `-Ttext 0x7E00` 需要严格对照内存放置情况，不然会导致错误。

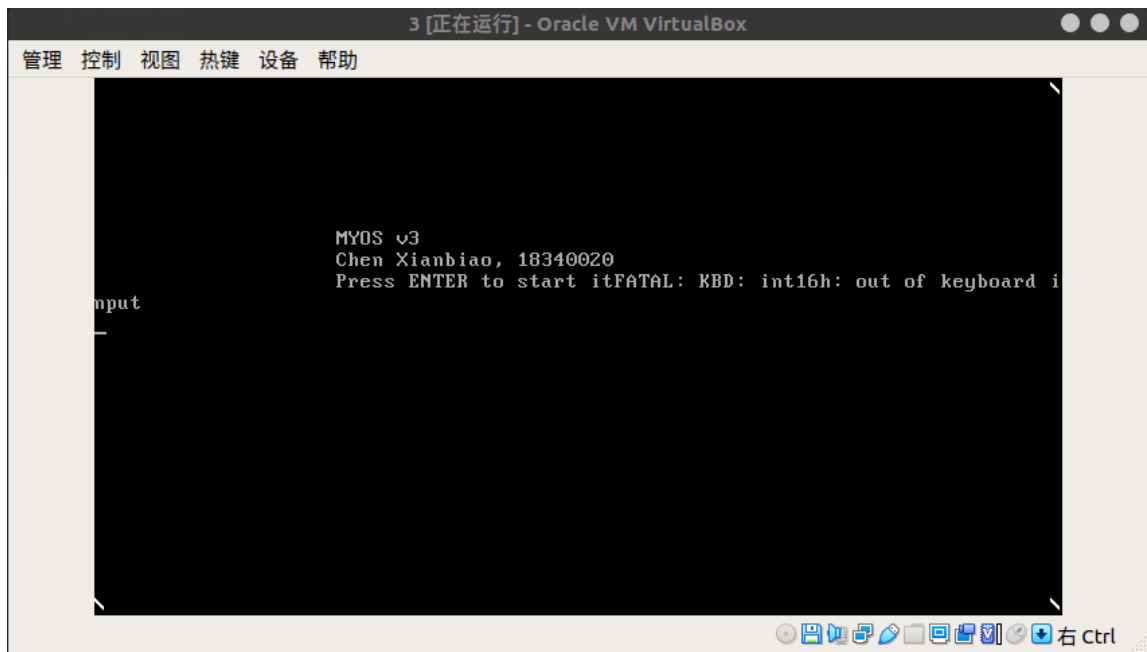
## 5. 实验过程

### 1) 踩坑过程

- 寄存器的保护

做了几次的操作系统的汇编实验，我其实已经知道寄存器是在进入不同函数的过程中要进行进栈保护，不然会出现bug，我一开始写无敌风火轮的代码的时候只 `popa` 只保护了一部分的代码，而忽略了后面改变了 `ds` 寄存器的值，导致我调用 `wudifenghuolun` 的时候没有达到我想要的效果而是出现乱字符的错误，显示了 `out of keyboard input`，并且风火轮没有转动，如下图





后来调试才发现需要保护 `ds` 寄存器

- 键盘中断位置改写问题

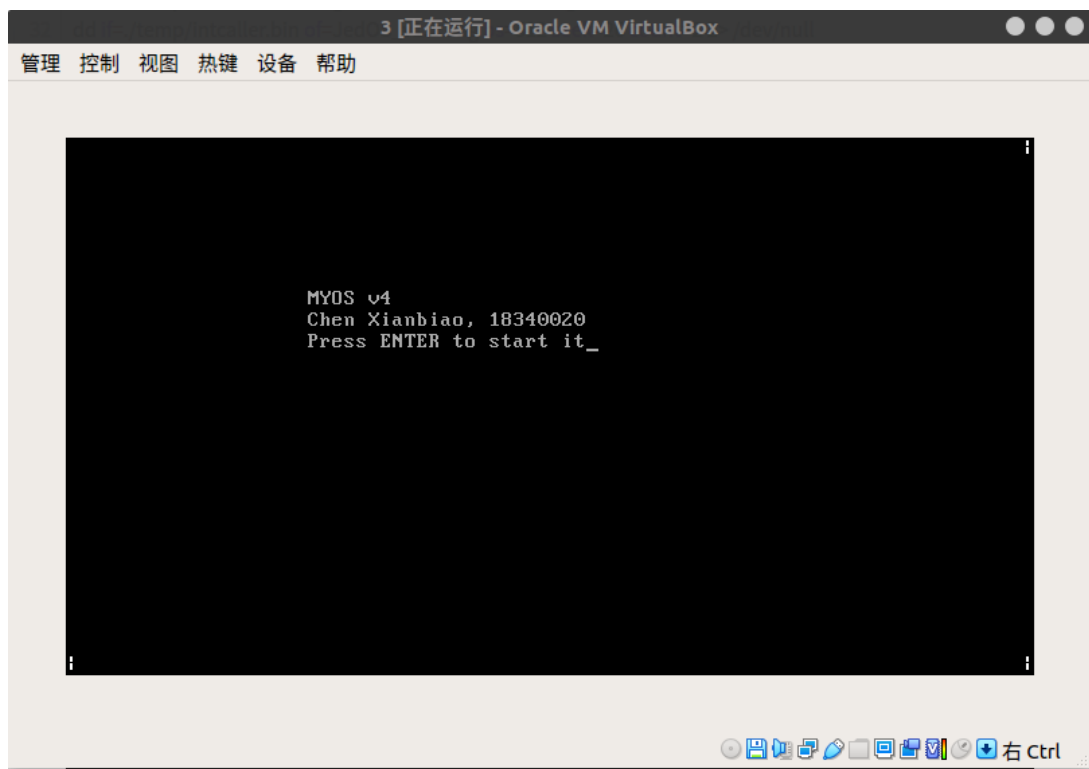
因为键盘9号中断本来就有东西，但是在我一开始写程序的时候没有注意带这一点，所以一开始的时候，我本来还以为已经成功显示出ouch了，但是发现只能显示一次，并且显示完之后，键盘的响应没有了。这时候我上网去搜才发现到这个问题，本来是想保存09号中断原来的偏移值，然后call，但是发现直接将中断写到 `37h` 的位置上更加的方便，直接 `int 37h` 就可以了

- 一开始写 `ouch.asm` 的时候我自己注意到了想要达到闪烁的效果，就需要显示后，再把显示的效果删除，但是我编写完发现，什么效果都没有了，然后我把删除显示效果的代码删除，发现我是能够显示出 `ouchouch` 的。这时候我就发现了由于代码运行得太快，在显示的一瞬间，操作系统就把显示删除了，我们的肉眼观察不到。因此为了解决这个问题，我想到一开始实验一在写滚动字符的时候就使用过延迟的方式，所以我就在显示和删除的中间加了个延迟的代码，这样ouch就成功的显示出来了

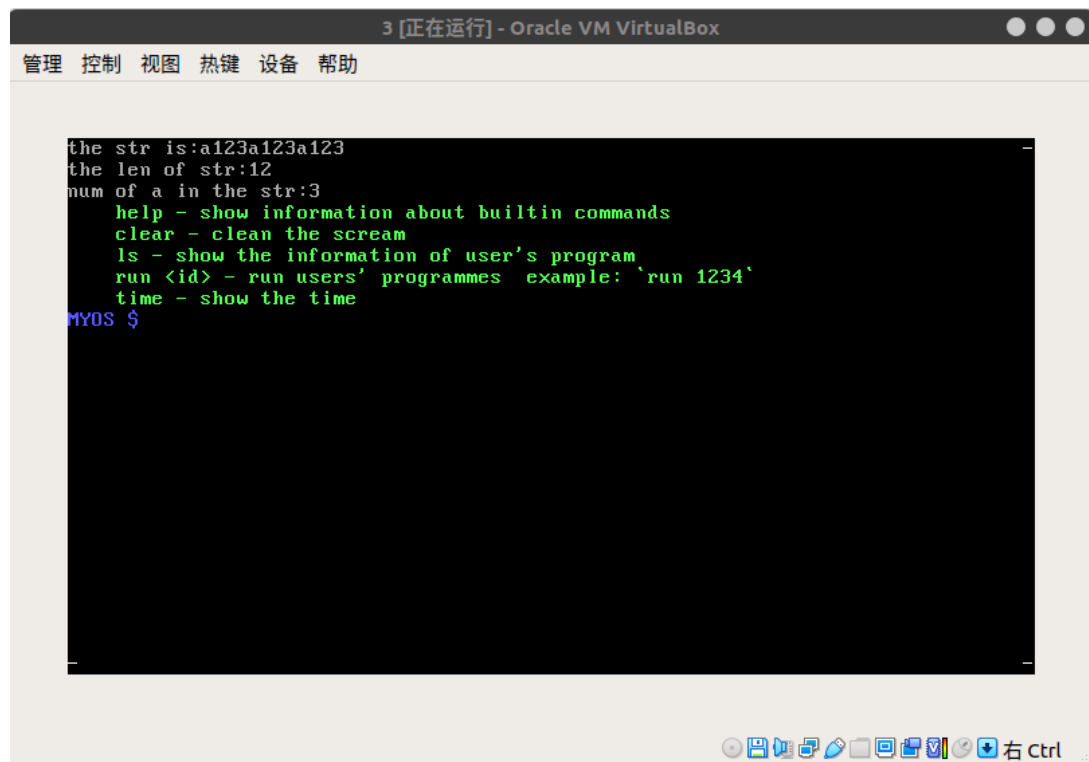
```
1 loopdelay:
2     dec word[count]
3     jnz loopdelay
4     mov word[count],delay
5     dec word[dcount]
6     jnz loopdelay
7     mov word[count],delay
8     mov word[dcount],ddelay
```

## 2) 实验结果展示

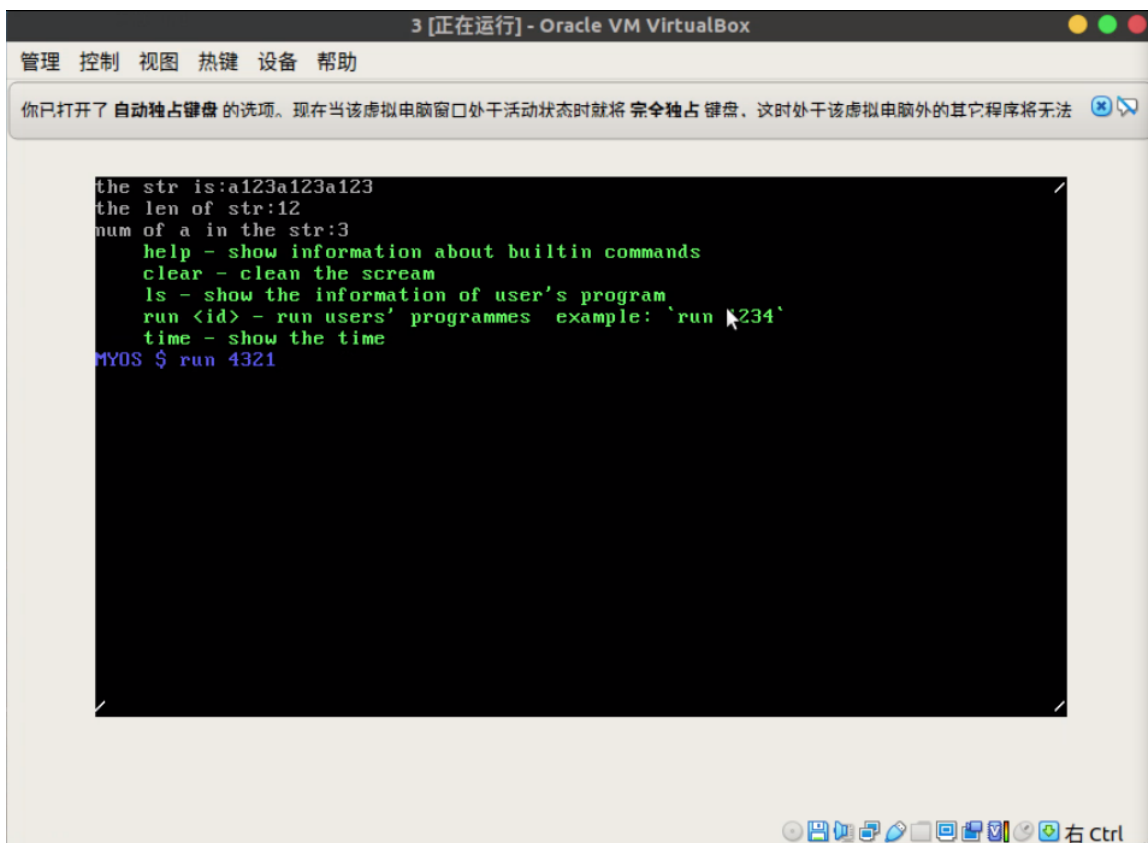
- 启动虚拟机，进入开机的页面，可以看到，屏幕左下角，右上角，右下角存在转动的无敌风火轮



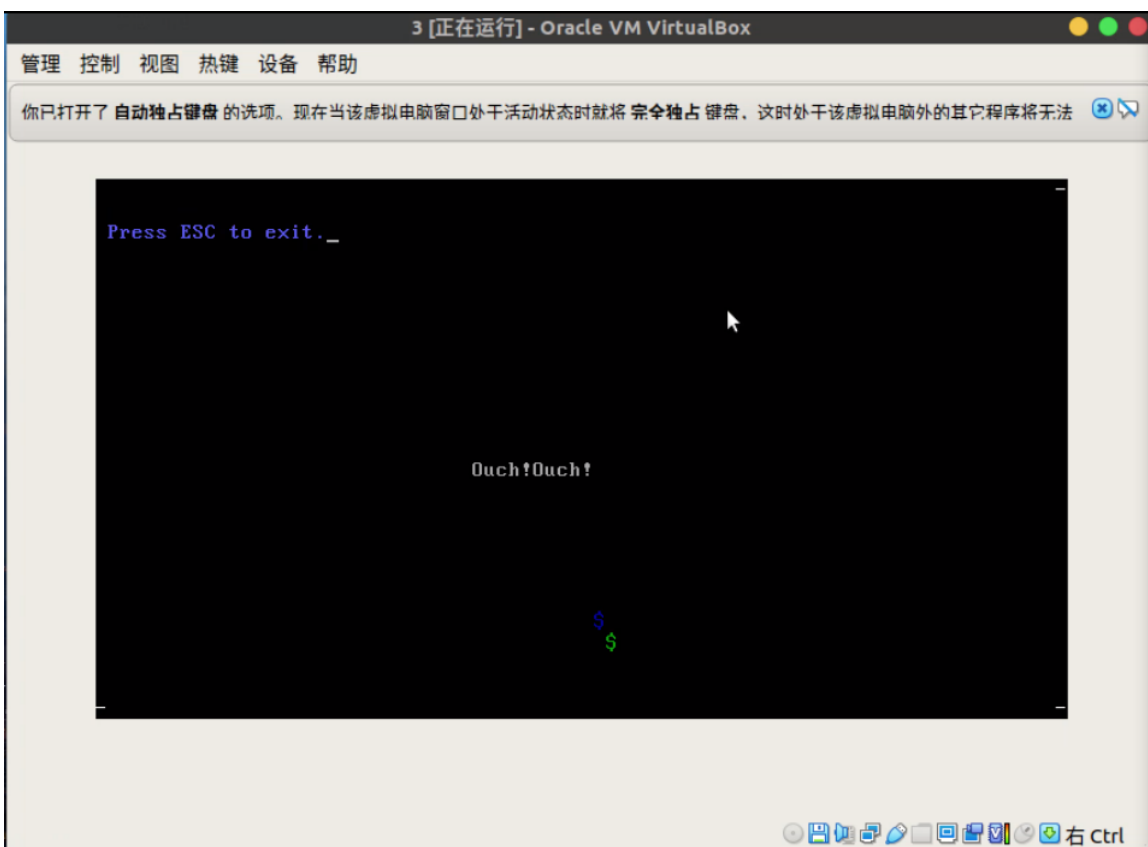
- 点击 `enter`，进入 `cmd`，可以看到基本和上一次实验一致，但文字的颜色更加多彩化，风火轮还在转动



- 输入 `run 4321`，依次执行第四个，第三个，第二个，第一个程序，并按 `esc` 进行切换



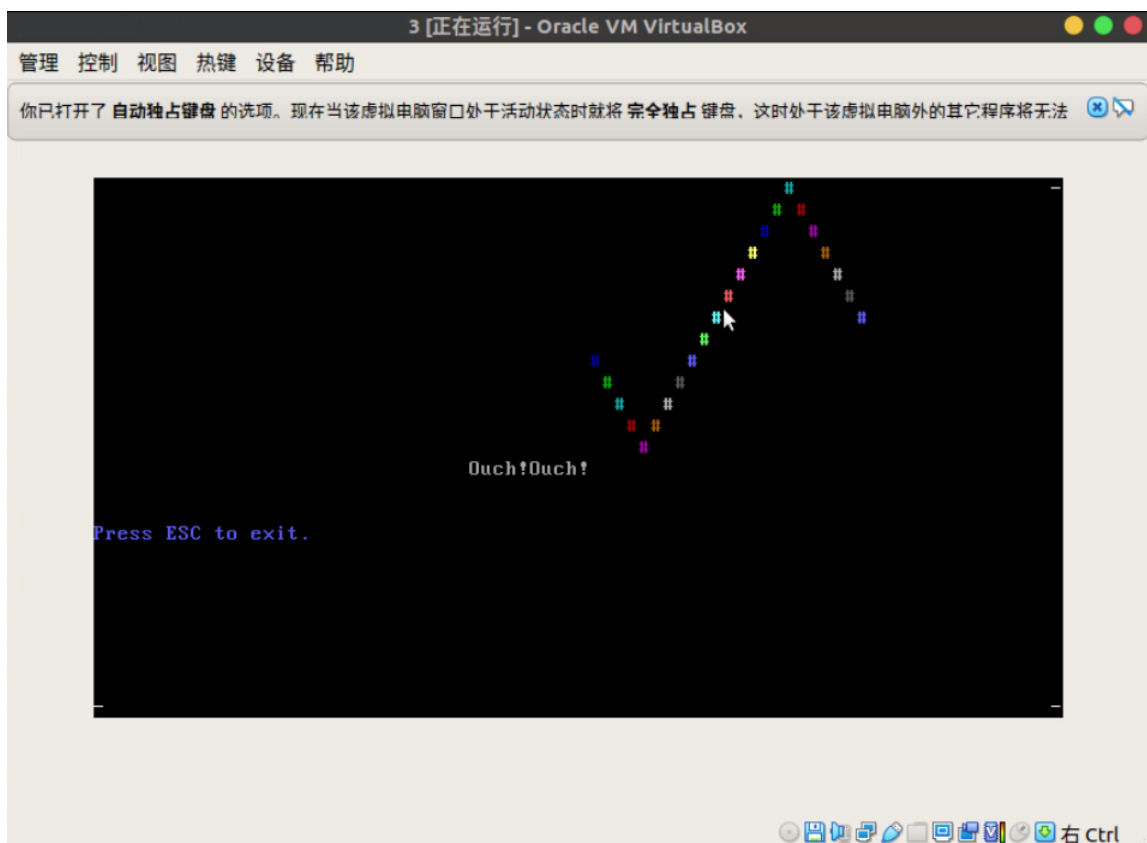
第四个用户程序，右下角，ouch出现



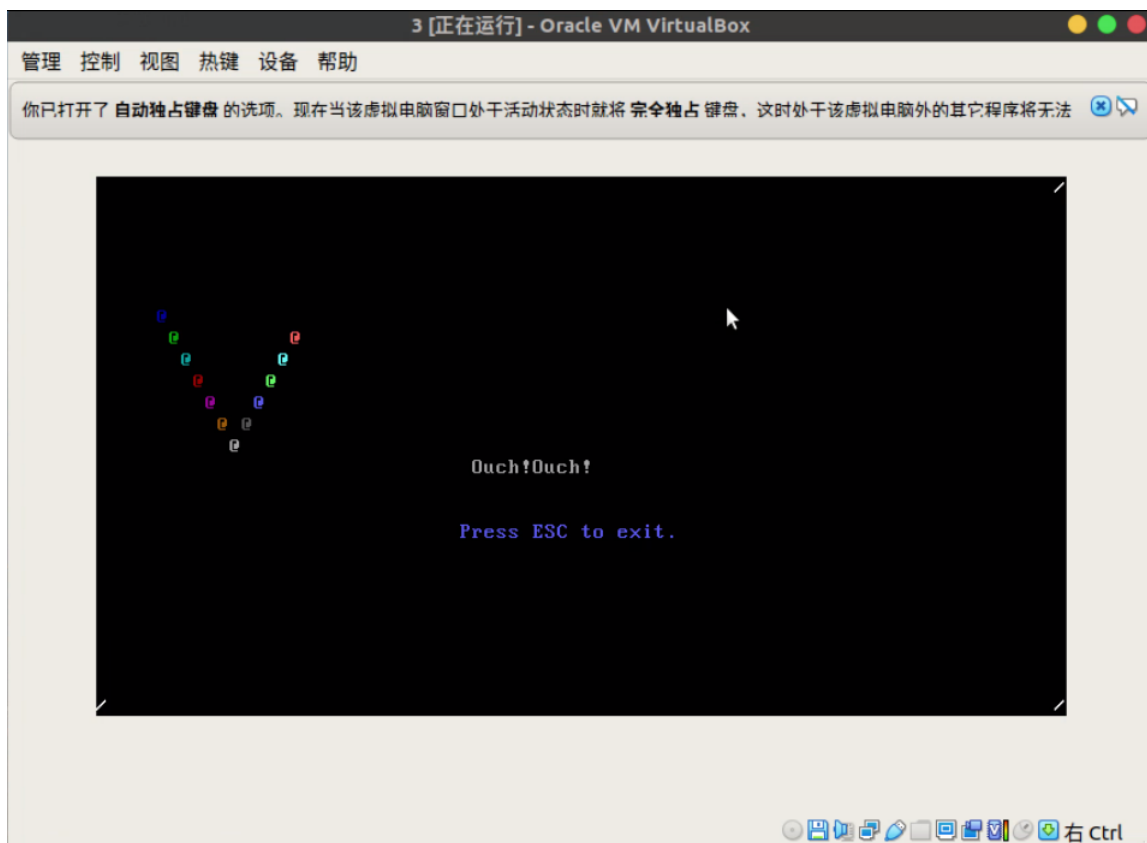
第三个用户程序，左下角，ouch出现



第二个用户程序，右上角，ouch出现



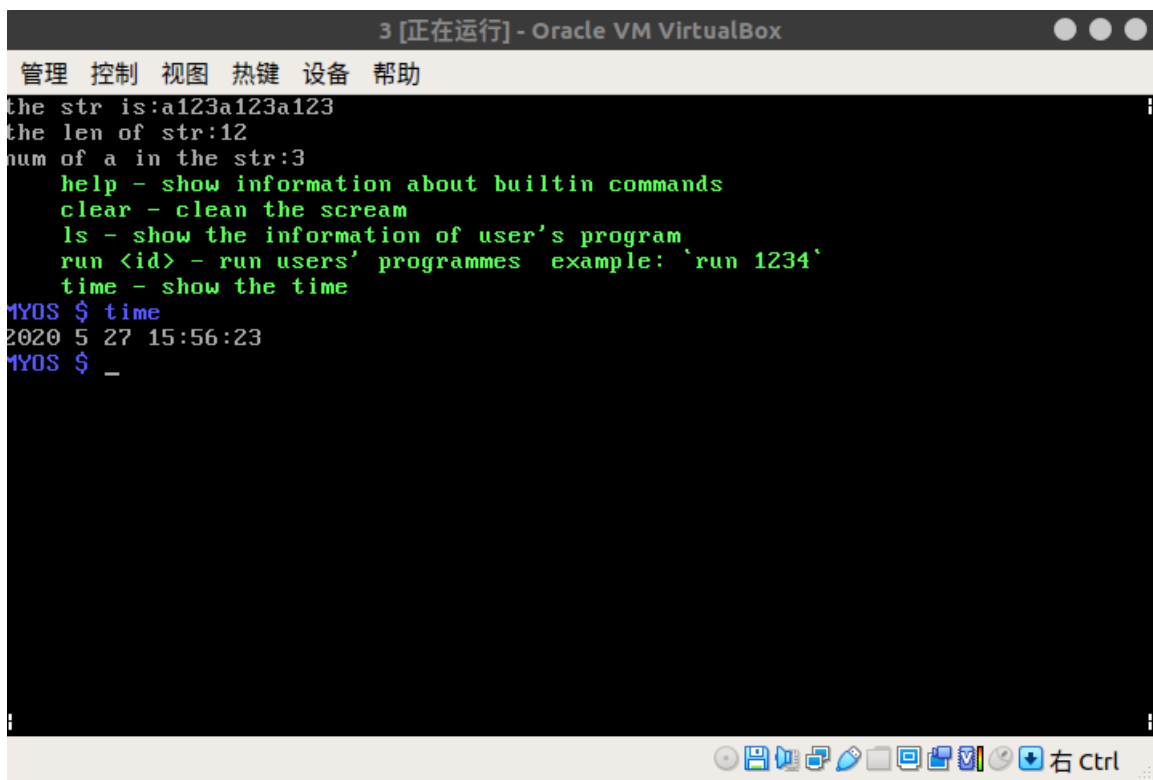
第一个用户程序，左上角，ouch出现



- 输入 `ls`，进入用户程序列表显示程序，ouch也能在键盘输入时出现



- 输入 `time` 显示当前时间



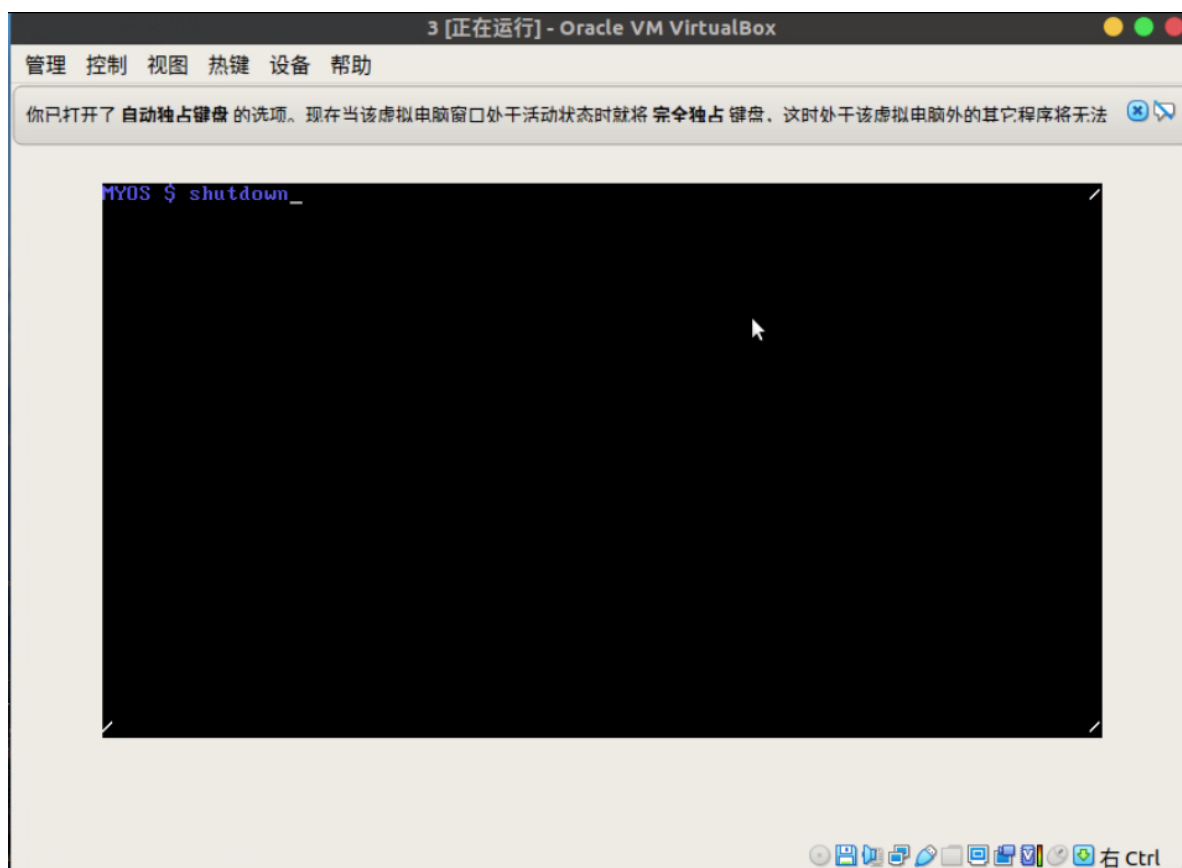
3 [正在运行] - Oracle VM VirtualBox

管理 控制 视图 热键 设备 帮助

```
the str is:a123a123a123
the len of str:12
num of a in the str:3
help - show information about builtin commands
clear - clean the screen
ls - show the information of user's program
run <id> - run users' programmes example: `run 1234`
time - show the time
MYOS $ time
2020 5 27 15:56:23
MYOS $ _
```

右 Ctrl

- 输入 shutdown 关机（新增功能）



## 5.实验体会

这次实验四总的来说比起实验三要轻松一点，由于实验三已经解决好了大部分的代码规范，整合规则的问题，也解决了许多基本函数的编写，因此这次异步事件的实现，我就只需要实现中断调用的代码，而不需要去实现过多其他的东西。

但是实验过程也还是比较艰辛的，虽然一开始根据老师的指引，实验思路（先编写程序，然后写入向量表）是比较清晰的。但是中途也是遇到不少的坑。在操作系统的一次次实验中，我最为有感触的就是细节问题，因为汇编语言的严格性，导致我们需要特别重视细节问题。譬如上面踩坑过程的寄存器的问題。pusha 指令只会按照 ax、cx、dx、bx、sp、bp、si、di 的顺序入栈进行保护，但是不会对 cs, ds, es, ss 这些段寄存器进行保护，因此在函数中修改了这些段寄存器我们就需要手动压栈出栈。

懂得了如何使用中断来进行异步编程，本次实验项目中的 int 09h 键盘中断，BIOS 中已经有对于此中断的处理程序，所以要将原本存在的中断处理程序的地址移动到中断向量表中的一个空闲的中断位置。

这次实验也加深了我对模块化的理解，因为上一次实验我已经在c语言中封装好字符串，函数调用的函数，因此这一次实验加上一点功能，只需要在原有的功能上加上具体实现的代码便可以成功。

还有一个体会就是，现在随着操作系统功能的增强，代码文件数量也越来越多，因此要使用不同文件夹来存储对应的文件，来进行管理，不然很容易就找不到所需的文件

## 6.参考资料

---

- 1.x86汇编如何关闭电源：<https://zhidao.baidu.com/question/1176459904219515299.html>
- 2.深入理解 x86/x64 的中断体系：<https://blog.csdn.net/fivedoumi/article/details/50451228>