实验四 具有中断处理的内核

18340052 何泽

一、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

二、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

三、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.
>
> 3、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!"。
>
> 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

四、实验方案

1. 相关基础原理

- 异步事件
  - 许多活动或事件可能并发进行，随时可能发生或结束，不可预测
  - 硬件系统的并发活动提高了计算机系统的效率，这些活动由操作系统进行有效的管理
  - 计算机硬件系统提供中断技术，支持CPU与外部设备的并发工作，也利用中断技术处理硬件错误、支持程序调试、实现软件保护和信息安全等
- 中断
  - 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）
  - PC采用32位的中断向量（中断处理程序的映射地址），可处理256种不同类型的中断
  - 两条外部中断请求线
    - `NMI` （ `Non Maskable Interrupt` ，不可屏蔽中断）和 `INTR` （ `Interrupt Request` ，中断请求[可屏蔽中断]）
    - `CPU` 是否响应在 `INTR` 线上出现的中断请求，取决于标志寄存器 `FLAGS` 中的 `IF` 标志位的状态值是否为 `1` 。可用机器指令 `STI/CLI` 置 `IF` 标志位为 `1/0` 来开/关中断
    - 在系统复位后，会置 `IF=0` （中断响应被关闭）。在任意一中断被响应后，也会置 `IF=0` （关中断）。若想允许中断嵌套，必须在中断处理程序中，用STI指令来打开中断
    - 在 `NMI` 线上的中断请求，不受标志位 `IF` 的影响。CPU在执行完当前指令后，会立即响应。不可屏蔽中断的优先级要高于可屏蔽中断的

- PC中断的处理过程
  - 保护断点的现场
    - 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
    - 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
- `x86` 处理器用两个级联的 `8259A` 芯片作为外设向CPU申请中断的代理接口，使一条 `INTR` 线扩展成 `15` 条中断请求线
- 中断向量
  - x86计算机在启动时会自动进入实模式状态
    - 系统的 `BIOS` 初始化 `8259A` 的各中断线的类型
    - 在内存的低位区（地址为 `0~1023[3FFH]` ， `1KB` ）创建含 `256` 个中断向量的表 `IVT` （每个向量[地址]占4个字节，格式为：16位段值:16位偏移值）
  - 保护模式
    - `IVT` （ `Interrupt Vector Table` ，中断向量表）会失效
    - 需改用 `IDT` （ `Interrupt Descriptor Table` ，中断描述表），必须自己编程来定义 `8259A` 的各个软中断类型号和对应的处理程序
  - 请求与类型，其中 `IRQ0-7` 为主 `8259A` ， `IRQ8-15` 为从 `8259A`

中断请求	中断类型
IRQ0	Intel 8253/8254可编程间隔计时器，即系统计时器
IRQ1	Intel 8042键盘控制器
IRQ2	级联从8259A
IRQ3	8250 UART串口COM2和COM4
IRQ4	8250 UART串口COM1和COM3
IRQ5	在PC/XT中为硬盘控制器，在PC/AT以后为Intel 8255并行端口LPT2
IRQ6	Intel 8272A软盘控制器
IRQ7	Intel 8255并行端口LPT1/伪中断
IRQ8	RTC（Real-Time Clock，实时时钟）
IRQ9	无公共的指派
IRQ10	无公共的指派
IRQ11	无公共的指派
IRQ12	Intel 8042 PS/2鼠标控制器
IRQ13	数学协处理器
IRQ14	硬盘控制器1
IRQ15	硬盘控制器2

    - `8259A` 的I/O端口
        - 主 `8259A` 所对应的端口地址为 `20h` 和 `21h`
        - 从 `8259A` 所对应的端口地址为 `A0h` 和 `A1h`
        - 通过 `in/out` 指令读写这些端口来操作这两个中断控制器

2.实验环境与工具版本

- 平台： `Windows + Ubuntu`

```
heze@ubuntu:~/os/os3$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04 LTS
Release:        20.04
Codename:       focal
heze@ubuntu:~/os/os3$ cat /proc/version
Linux version 5.4.0-29-generic (buildd@lgw01-amd64-035) (gcc version 9.3.0 (Ubu
ntu 9.3.0-10ubuntu2)) #33-Ubuntu SMP Wed Apr 29 14:32:27 UTC 2020
```

    - c语言编译器： `tcc` ,运行环境： `DosBox 0.74`
    - 汇编工具： `nasm` + `tasm`

```
heze@ubuntu:~$ nasm --version
NASM version 2.14.02
```

- `CMake` ：

```
heze@ubuntu:~$ make --version
GNU Make 4.2.1
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

- 虚拟机：`VMware Workstation 15`

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板块“七、问题及解决方案”中详细叙述）

1. 操作系统功能

- 操作系统内核功能

  进入系统后，会在最右侧那一列显示“无敌风火轮”

输入命令	功能描述与参数解释
name	显示程序的名字
size	显示程序的名字与大小
clean	清屏，只留下开头的指引
time	获取当前的时间
author	显示姓名学号
cal	计算某个字母在某个单词的出现次数，命令格式举例：`cal a apple` 代表计算a在apple中出现多少次
lower	将输入字符全部转换为小写，命令格式举例：`lower ABcdE`
upper	将输入字符全部转换为大写，命令格式举例：`upper ABcdE`

- 执行用户程序

  用户程序功能为数字在屏幕反弹，第几个程序就是数字几在反弹，**执行过程中每按一次键盘就会在上、下、左、右、中出现“OUCH！OUCH！”**

输入命令	解释
run + 程序序号	可以执行单个程序，如 `run 1` ，也可以按顺序执行多个，如 `run 2341`

- 批处理命令

输入命令	功能
a.cmd	按顺序执行1-4用户程序
b.cmd	执行完1-4用户程序后获取当前时间
c.cmd	显示用户文件的名字、大小和所在扇区号

- 中断服务程序

  输入 `int 33h-36h` 便可以利用 `int 33` 、 `int 34` 、 `int 35` 和 `int 36` 产生中断调用4个服务程序，分别是在左上、右上、左下、右下显示数字33、34、35、36。

2. 引导程序

- 引导程序的作用是加载操作系统内核，同时输出字符，因为和之前的一样，不再详细叙述
- 因为引导成功后直接进入内核，而内核会先清屏后输出字符，所以在实际过程中引导程序的字符并不会被看见，因为太快了，只有在程序出错导致无法正确引导的时候才能看见这串字符

3.内核：汇编部分

这部分大多数与实验三相同，这里就不赘述了，只写这次实验新加入的代码

- “无敌风火轮”

  主要思路是利用时钟中断，对 8 号中断进行编程，显示一个字符后将字符修改为下一个。随后将 `0x08` 放入 `0x20` 的位置，处理时钟中断函数的入口放入 `0x22` 。最后需要告诉硬件端口已经处理完中断并返回。

  - 系统时钟中断，设置时钟中断向量（ `08h` ），初始化段寄存器

```
delay equ 3
count db delay
ch1 db '|'
ch2 db '/'
ch3 db '\'
remark db 3
Timer:
    push ax
    push bx
    push cx
    push dx
    push bp
    push es
    dec byte ptr [count]
    jnz int_end
    cmp byte ptr [remark],3
    jz first
    cmp byte ptr [remark],2
    jz second
    mov byte ptr [remark],4
    jmp third
    first:
        mov bp,offset ch1
        jmp show
    second:
        mov bp,offset ch2
        jmp show
    third:
        mov bp,offset ch3
        jmp show
```

  - 显示，功能号 `13h` ，颜色为 `0fh` ，亮白色，第 `0` 页，串长（cx）为 `1` ， `loop1` 调用 `10h` 中断

```
show:
        dec byte ptr [remark]
        mov ah,13h
        mov al,0
        mov bl,0fh
        mov bh,0
```

```
        mov cx,1
        mov dh,0
        mov dl,79
loop1:
        int 10h
        add dh,2
        cmp dh,24
        jne loop1
        mov byte ptr es:[count],delay
```

- end，将 `End Of Interrupt` 信号赋值给 `al` ，然后发送到主、从 `8529A`

```
int_end:
    mov al,20h
    out 20h,al
    out 0A0h,al
    pop es
    pop bp
    pop dx
    pop cx
    pop bx
    pop ax
    iret
```

- ouch！ouch！
  - 这个的中断和时钟中断接近，我设计的是在5个地方显示，上下左右中：

```
up_pos:
    mov word ptr[row],4
    mov word ptr[col],35
    jmp printOUCH
mid_pos:
    mov word ptr[row],12
    mov word ptr[col],35
    jmp printOUCH
down_pos:
    mov word ptr[row],20
    mov word ptr[col],35
    jmp printOUCH
left_pos:
    mov word ptr[row],12
    mov word ptr[col],10
    jmp printOUCH
right_pos:
    mov word ptr[row],12
    mov word ptr[col],60
    jmp printOUCH
```

  - 键盘中断,跟前面的一样，将 `End Of Interrupt` 信号赋值给 `al` ，然后发送到主、从 `8529A` ，再从中断返回

```
keyin:
    in al,60h
    mov al,20h
    out 20h,al
    out 0A0h,al
    pop es
    pop bp
    pop dx
    pop cx
    pop bx
    pop ax
    iret
```

- 每按一次就改变位置：

```
change_pos:
    cmp word ptr[pos],1
    je up_pos
    cmp word ptr[pos],2
    je mid_pos
    cmp word ptr[pos],3
    je down_pos
    cmp word ptr[pos],4
    je left_pos
    cmp word ptr[pos],5
    je right_pos
```

- 打印ouch，功能号 `13h` ，颜色为绿色，最后调用 `10h` 中断

```
printOUCH:
    mov ah,13h
    mov al,0
    mov bl,0ah
    mov bh,0
    mov dh,byte ptr[row]
    mov dl,byte ptr[col]
    mov bp, offset string
    mov cx,10
    int 10h
```

- 调用33h、34h、35h、36h中断并打印数字

  这里四个都差不多，就拿33h进行说明

  - 首先中断设置

```
setINT:
    push ax
    push es

    xor ax,ax
    mov es,ax
    mov word ptr es:[51*4],offset int_33h       ;33h
    mov ax, cs
    mov word ptr es:[51*4+2],ax

    xor ax,ax
    mov es,ax
    mov word ptr es:[52*4],offset int_34h       ; 34h
```

```
    mov ax, cs
    mov word ptr es:[52*4+2],ax

    xor ax,ax
    mov es,ax
    mov word ptr es:[53*4],offset int_35h       ; 35h
    mov ax, cs
    mov word ptr es:[53*4+2],ax

    xor ax,ax
    mov es,ax
    mov word ptr es:[54*4],offset int_36h       ; 36h
    mov ax, cs
    mov word ptr es:[54*4+2],ax

    pop es
    pop ax
ret
```

- C程序调用的 `33h` 中断函数：

```
public _run33
_run33 proc
    push ax
    push bx
    push cx
    push dx
    push es

    call _cls

    int 33h
call DelaySome
    pop ax
    mov es,ax
    pop dx
    pop cx
    pop bx
    pop ax
    ret
_run33 endp
```

- `33h` 功能，在左上角打印33这个数字，通过“ `*` ”叠加拼出一个数字的33

```
int_33h:
    push ax
    push bx
    push cx
    push dx
    push bp

    mov ah,13h
    mov al,0
    mov bl,05h
    mov bh,0
    mov dh,0
    mov dl,0
    mov bp,offset message33
```

```
    mov cx,356
    int 10h

    pop bp
    pop dx
    pop cx
    pop bx
    pop ax

    mov al,33h
    out 33h,al
    out 0A0h,al
    iret

message33:
    db "  ************     ************",0ah,0dh
    db "  ************     ************",0ah,0dh
    db "            ***              ***",0ah,0dh
    db "            ***              ***",0ah,0dh
    db "  ************     ************",0ah,0dh
    db "  ************     ************",0ah,0dh
    db "            ***              ***",0ah,0dh
    db "            ***              ***",0ah,0dh
    db "  ************     ************",0ah,0dh
    db "  ************     ************",0ah,0dh
    db 0ah,0dh
    db "        This is INT 33H!",'$'
```

4. 内核：C程序部分 & 用户程序

除了C程序多了判断33h-36h中断命令的语句之外，其余的功能性的函数和上个实验一模一样，没有变化，这里就不再叙述了。

5.编译

- 首先在 `DosBox` 中使用 `TCC` 、 `TASM` 以及 `TLINK` 编译内核，并生成 `.com` 程序
  - 启动 `DosBox` ，将目录挂载到 `DosBox` 的D盘并进入

```
Z:\>mount D D:\os
Drive D is mounted as local directory D:\os\

Z:\>d:\
```

  - 使用 `TCC`

```
D:\>tcc -mt -c -oos.obj myos.c >ccmsg.txt
```

  - 使用 `TASM`

```
D:\>tasm myos.asm myos.obj >amsg,txt
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996 Borland International

Assembling file:   myos.asm
*Warning* myos.asm(128) ":" operator ignored
Error messages:    None
Warning messages:  1
Passes:            1
Remaining memory:  466k
```

  - 使用 `TLINK` 链接

```
D:\>tlink /3 /t myos.OBJ os.obj,myos.com
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International
```

- 剩下的汇编我使用 `NASM` 编译，并在 `Ubuntu` 下使用 `dd` 命令写入软盘

  这里我使用 `CMake` 自动完成创建空白软盘、 `nasm` 编译引导程序、将各个程序写入扇区的工作

  下面是我的 `MakeFile`

```
BIN = boot.bin prog1.bin prog2.bin prog3.bin prog4.bin
IMG = heze.img
all: clear $(BIN) $(IMG)
clear:
    rm -f $(BIN) $(IMG)
%.bin: %.asm
    nasm -fbin $< -o $@
%.img:
    /sbin/mkfs.msdos -C $@ 1440
    dd if=boot.bin of=$@ conv=notrunc
    dd if=MYOS.COM of=$@ seek=1 conv=notrunc
    dd if=prog1.bin of=$@ seek=10 conv=notrunc
    dd if=prog2.bin of=$@ seek=11 conv=notrunc
    dd if=prog3.bin of=$@ seek=12 conv=notrunc
    dd if=prog4.bin of=$@ seek=13 conv=notrunc
clean:
    rm *.bin
```

  其中，先将以前生成的文件都删除，然后1.44MB软盘是用 `/sbin/mkfs.msdos -C $@ 1440` 这一句完成创建的， `nasm` 命令将所有 `.asm` 文件汇编为 `.bin` 文件，然后将所有程序都写入软盘。

  编译：

```
heze@ubuntu:~/os/os4$ make
rm -f boot.bin prog1.bin prog2.bin prog3.bin prog4.bin heze.img
nasm -fbin boot.asm -o boot.bin
nasm -fbin prog1.asm -o prog1.bin
nasm -fbin prog2.asm -o prog2.bin
nasm -fbin prog3.asm -o prog3.bin
nasm -fbin prog4.asm -o prog4.bin
/sbin/mkfs.msdos -C heze.img 1440
mkfs.fat 4.1 (2017-01-24)
dd if=boot.bin of=heze.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000112879 s, 4.5 MB/s
dd if=MYOS.COM of=heze.img seek=1 conv=notrunc
13+1 records in
13+1 records out
6739 bytes (6.7 kB, 6.6 KiB) copied, 0.000150235 s, 44.9 MB/s
dd if=prog1.bin of=heze.img seek=14 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000113745 s, 4.5 MB/s
dd if=prog2.bin of=heze.img seek=15 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000133363 s, 3.8 MB/s
dd if=prog3.bin of=heze.img seek=16 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000134178 s, 3.8 MB/s
dd if=prog4.bin of=heze.img seek=17 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000174762 s, 2.9 MB/s
```

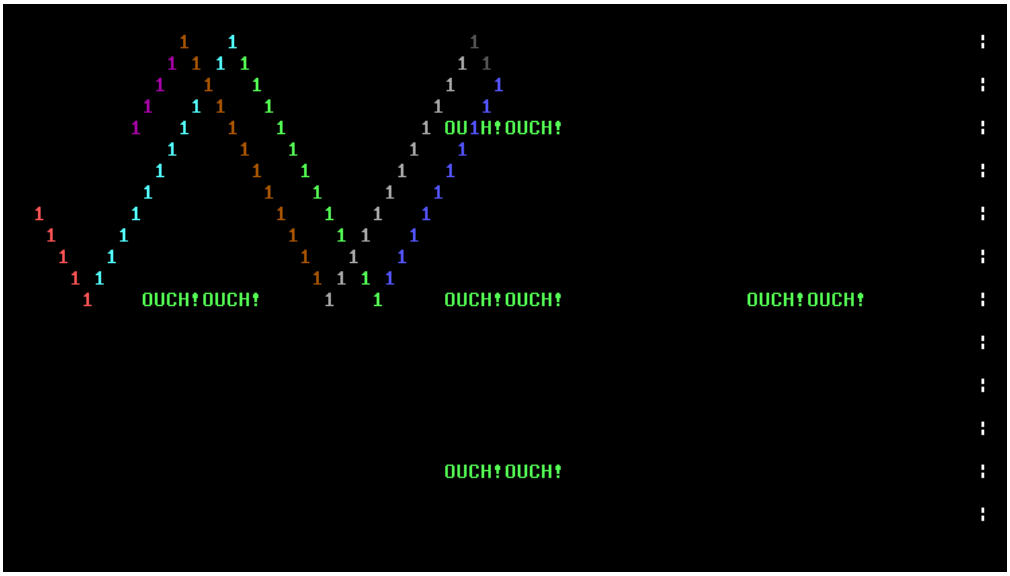
  可见所有的命令都自动执行了。

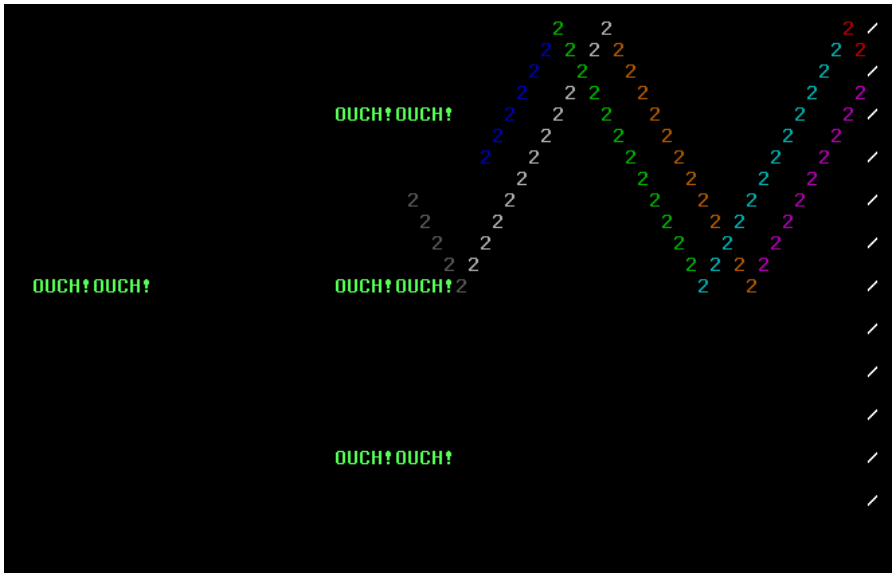
6.运行

- 开始界面，可以看到右面一列的风火轮（转动过程详见演示视频）

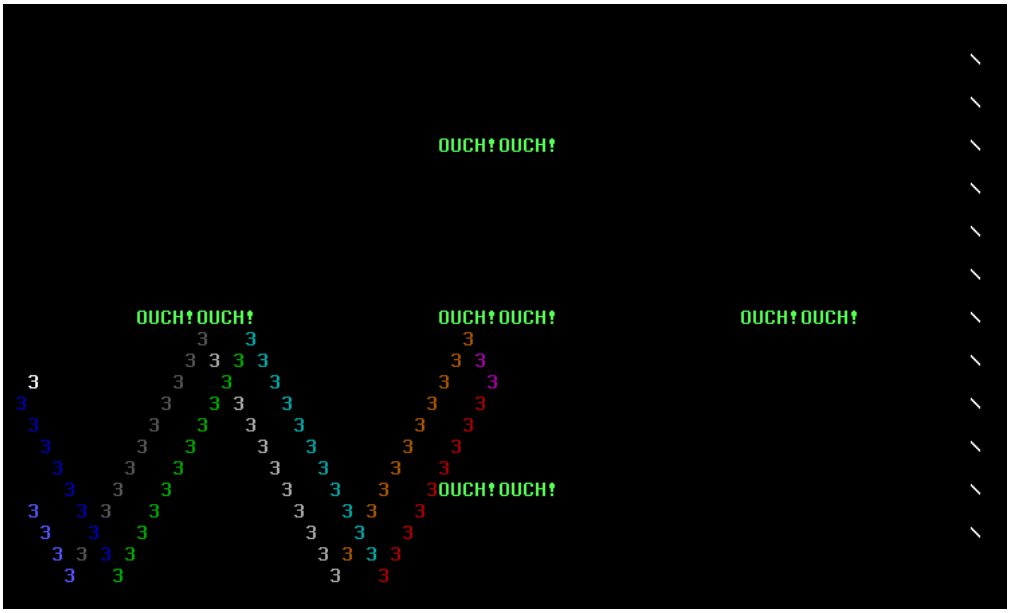
```
Welcome to HeZe's operating system.
Just enter the name of the instructions.

name: Show the files's name    size: Show the files's size and name
clean: Clear the screen        time: Get the time
author: Show the author of this operating system
cal: Calculate a char's appeared time  e.g: cal a apple (This mean there are ho
 many 'a' in 'apple'
lower: Upper to lower  e.g: lower ABcdE
upper: Lower to upper  e.g: upper ABcdE
run: Run any number of program  e.g: run 2 or run 2431
BE CAREFUL! WHEN YOU EXECUTE THE PROGRAM, YOU CAN PRESS ESC TO RETURN TO THE DOS

Quick CMD: a.cmd: Execute program 1-4 sequentially.
           b.cmd: Execute a.cmd then get the time
           c.cmd: Show the file name & size & disk
Others: int 33h , int 34h , int 35h , int 36h

Please enter your instruction:
```

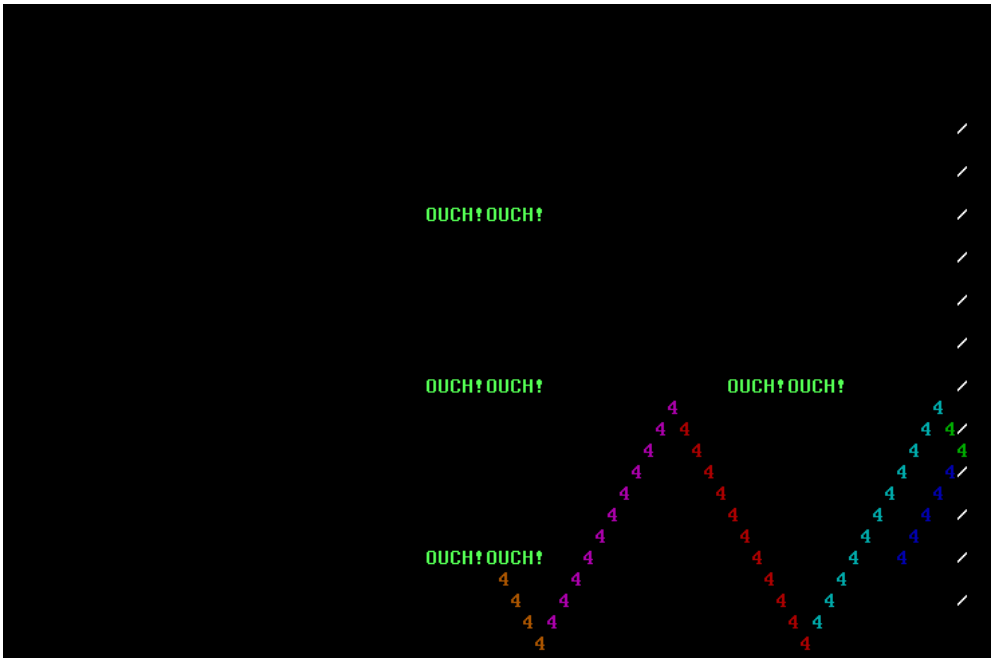
至于操作系统内核的功能上次实验已经展示过了，这里就只展示ouch和四个中断

- 

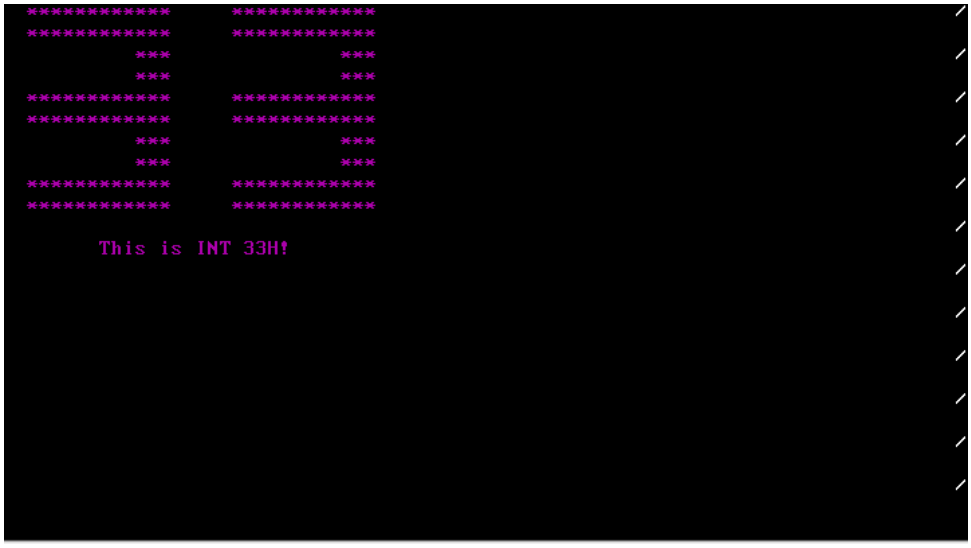
- 

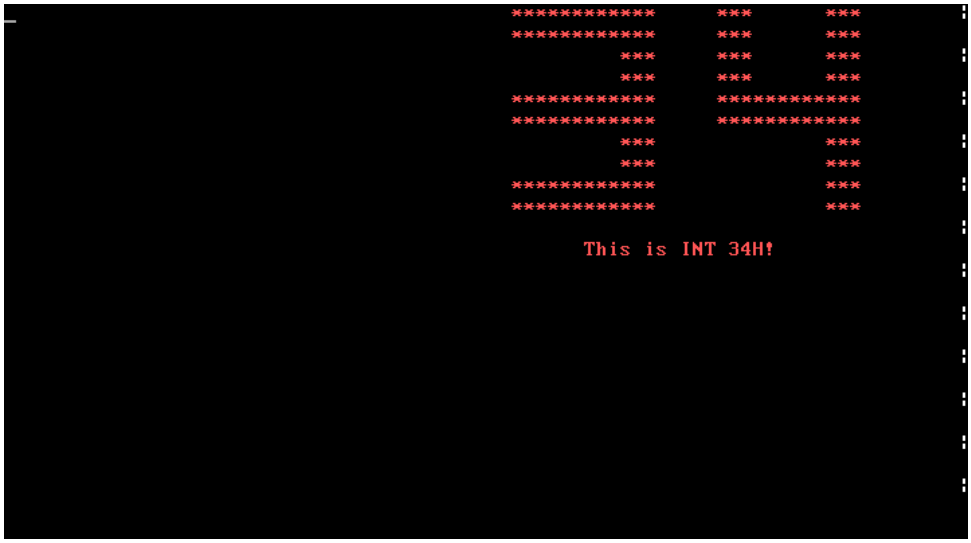
-

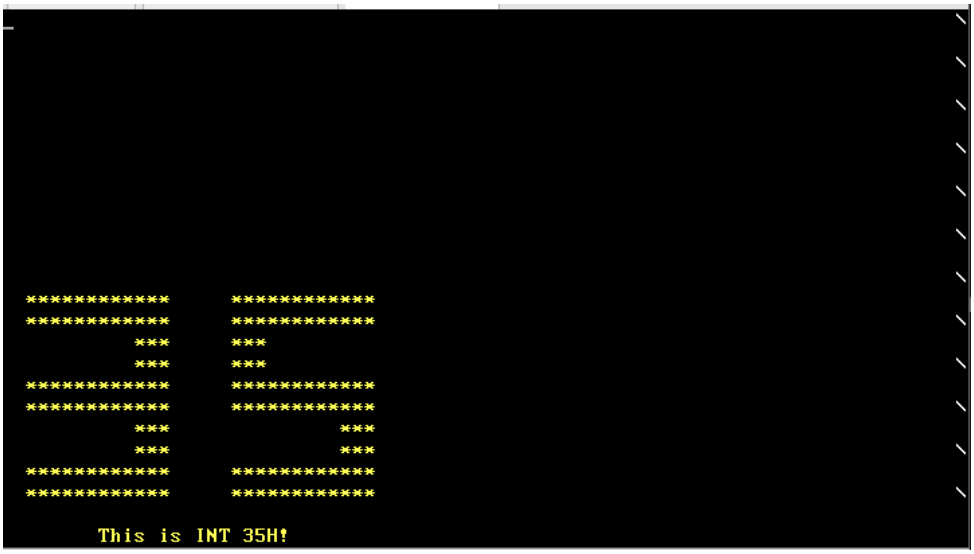

OUCH!OUCH!
OUCH!OUCH!
OUCH!OUCH!
OUCH!OUCH!
OUCH!OUCH!

- 
OUCH!OUCH!
OUCH!OUCH!
OUCH!OUCH!
OUCH!OUCH!

- 四个中断:

- 
This is INT 33H!



- 

- 

六、创新工作

1. 操作系统内核在老师的要求上添加了以下功能
    - 显示作者、文件大小等信息
    - 获取当前时间

    - 批处理命令
    - 转化大小写
    - 老师的要求是统计单词中特定字符的出现次数，我更进一步，统计哪个字母由用户输入，并非特定
2. 对于PPT中写的33h-36h中断老师说下一个实验再做，我在这个实验已实现
3. 使用 `CMake` ，各种命令自动执行

七、问题及解决方案

- 第一个问题是C程序的写法，一开始不知道一点，就是 `char[]` 读的是一个 `byte` ， `char*` 读取的是一个 `word`
- `TCC` 关于局部变量数组初始化有 `bug` ，我也是询问了其他同学之后在汇编模块加入了一个补丁程序，就是下面这段代码：

```
public SCOPY@
SCOPY@ proc
        arg_0 = dword ptr 6
        arg_4 = dword ptr 0ah
        push bp
        mov bp,sp
        push si
        push di
        push ds
        lds si,[bp+arg_0]
        les di,[bp+arg_4]
        cld
        shr cx,1
        rep movsw
        adc cx,cx
        rep movsb
        pop ds
        pop di
        pop si
        pop bp
        retf 8
SCOPY@ endp
```

- 一开始放到虚拟机运行的时候会出现乱码并发出蜂鸣声，起初听了别的同学的建议将 `VMWare` 换成了 `VirtualBox` ，但还是一样的效果，这个bug困扰了我好久，后来才发现是在使用`dd``命令将二进制文件写入扇区的时候扇区号弄错了，整体往后了一个扇区，从而导致的错误，将扇区号改正之后就正常运行了。

八、实验总结

这次实验相对上次试验难度小了一些，主要是整体的框架在上次试验已经写好，这次也只是增加了中断。

这次实验我认为主要是要理解中断的概念和硬件是如何处理中断的，知道了流程和基础原理之后再完成此次试验并不是很困难，中断响应后，先到内存指定位置找到中断向量表，然后跳转到中断服务程序。中断服务程序需先保存寄存器。中断服务程序完成后，需先还原寄存器，然后调用中断返回指令。所以在做实验的时候，就需要修改操作系统内核，使操作系统内核修改中断向量表，才能实现自定义中

断服务程序。

理解了之后上面这套流程之后实现便会简单很多，在写的过程中遇到的问题也没有上次多，并且实现了老师PPT中的下次试验要完成的内容。

这次实验我理解了中断的概念，明白了什么是中断向量表以及怎么样调用自己设计的中断程序，以及在编程中要时刻注意段地址和各种寄存器，细心注意这些会减少很多之后debug的时间。