

Report

18340161 田蕊 18级超算方向 tianr6@mail2.sysu.edu.cn

一、实验题目

具有中断处理的内核

二、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

三、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

四、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

五、相关知识

(一)、关于中断

实模式下的中断向量表

所谓中断处理，归根结底就是处理器要执行一段与该终端有关的程序（指令）。处理器可以识别256个中断，那么理论上就需要256段程序。这些程序的位置并不重要，重要的是，在实模式下，处理器要求将他们的入口点集中存放在内存中物理地址0x00000开始，到0x003ff结束，共1KB的空间内，这就是所谓的中断向量表。

8259芯片

在个人计算机中，用的最多的中断代理就是8259芯片，就是我们通常所说的中断控制器。Intel处理器允许的256个终断中，8259负责提供其中的15个，但中断号并不固定。8259只有8个终端输入引脚，而在个人计算机上使用它需要两块。第一块8259芯片的代理输出INT直接送到处理器的INTR引脚，这是主片；第二块8259芯片的INT输出送到第一块的引脚2上，是从片，连你赶快芯片之间形成级联关系。在8259芯片内部，有中断屏蔽寄存器，这是个8位寄存器，对应着该芯片的8个中断输入引脚，对应的位是0还是1，决定了从该引脚来的信号是否能够通过8259送往处理器（0表示允许，1表示阻断）。

中断是否被处理，除了要看8259芯片的脸色外，最终决定权还是在处理器手中。在处理器内部，标志寄存器有一个标志位IF，这就是中断标志。当IF为0时，所有从处理器INTR引脚来的中断信号都被忽略掉；当其为1时，处理器可以接受和响应中断。IF标志位可以通过两条指令cli和sti来改变。这两条指令都没有操作数，cli用于清除IF位，sti用于置位IF标志。

六、实验过程

（一）、无敌风火轮

代码讲解

无敌风火轮的实现非常简单，老师给出的代码已经实现了一个简单的版本，我们你所需要做的只是对显示的内容进行一定的修改。下面对程序中的重点代码进行一定的讲解。

```
xor ax,ax          ; AX = 0
mov es,ax          ; ES = 0
mov word [es:20h],Timer ; 设置时钟中断向量的偏移地址
mov ax,cs
mov word [es:22h],ax ; 设置时钟中断向量的段地址=CS
mov ds,ax          ; DS = CS
mov es,ax          ; ES = CS
```

这一部分看起来只是对程序的初始化，但其实是我们本次实验的核心，我们本次的核心操作，其实就是将我们所完成的中断处理程序加载到中断向量表中。而上述代码中设置段地址和偏移地址的语句就是这样的操作。

```
mov al,byte[chara]
cmp al,1
je outch1
cmp al,2
je outch2
cmp al,3
je outch3
cmp al,4
je outch4

outch1:
mov ah,0Fh        ; 0000: 黑底、1111: 亮白字（默认值为07h）
mov al,byte[char1] ; AL = 显示字符值（默认值为20h=空格符）
mov [gs:((80*24+79)*2)],ax
mov byte[count],delay ; 重置计数变量=初值delay
mov byte[chara],2
jmp end

outch2:
mov ah,0Fh        ; 0000: 黑底、1111: 亮白字（默认值为07h）
mov al,byte[char2] ; AL = 显示字符值（默认值为20h=空格符）
mov [gs:((80*24+79)*2)],ax
mov byte[count],delay ; 重置计数变量=初值delay
```

```

    mov byte[chara],3
    jmp end
outch3:
    mov ah,0Fh      ; 0000: 黑底、1111: 亮白字（默认值为07h）
    mov al,byte[char3]      ; AL = 显示字符值（默认值为20h=空格符）
    mov [gs:((80*24+79)*2)],ax
    mov byte[count],delay   ; 重置计数变量=初值delay
    mov byte[chara],4
    jmp end
outch4:
    mov ah,0Fh      ; 0000: 黑底、1111: 亮白字（默认值为07h）
    mov al,byte[char4]      ; AL = 显示字符值（默认值为20h=空格符）
    mov [gs:((80*24+79)*2)],ax
    mov byte[count],delay   ; 重置计数变量=初值delay
    mov byte[chara],1
    jmp end

```

这一部分是根据不同的时序输出无敌风火轮的部分，程序很简单，只是简单的switch-case语句，所以我们这里不再赘述。

最后我们还增加了一个返回DOS的语句，利用的是21号中断

```

exit:
    MOV AH, 4CH      ;返回DOS系统
    INT 21H

```

我们规定程序显示3000次字符。这样程序执行完之后，就可以回到DOS操作系统。

参数调节

程序中有两个参数可供调节，分别是

```

delay equ 1      ; 计时器延迟计数
num    equ 3000

```

其中delay表示的是运行延迟，delay越小，无敌风火轮的旋转速度就越快。

其中num表示的是显示的次数，num越大，程序运行的时间就越长。

可以调节这两个参数使得程序有不同的运行时间和运行速度。

运行结果

用NASM编译得到.COM文件。

```
nasm
Microsoft Windows [版本 10.0.18362.836]
(c) 2019 Microsoft Corporation。保留所有权利。

D:\NASM>E:

E:\>cd E:\code\assembly\OSEx4\test1

E:\code\assembly\OSEx4\test1>nasm mytest1.asm -o mytest1.com

E:\code\assembly\OSEx4\test1>_
```

我们直接看无敌风火轮的运行结果。

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, P...
Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com


Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount A E:\code\assembly
Drive A is mounted as local directory E:\code\assembly\

Z:\>A:

A:\>cd OSEX4\test1

A:\OSEX4\TEST1>mytest1
_
```



DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, P...

```
Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com


Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount A E:\code\assembly
Drive A is mounted as local directory E:\code\assembly\

Z:\>A:

A:\>cd OSEX4\test1

A:\OSEX4\TEST1>mytest1.com
```



DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, P...

```
Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com


Z:\>SET BLASTER=A220 I7 D1 H5 T6

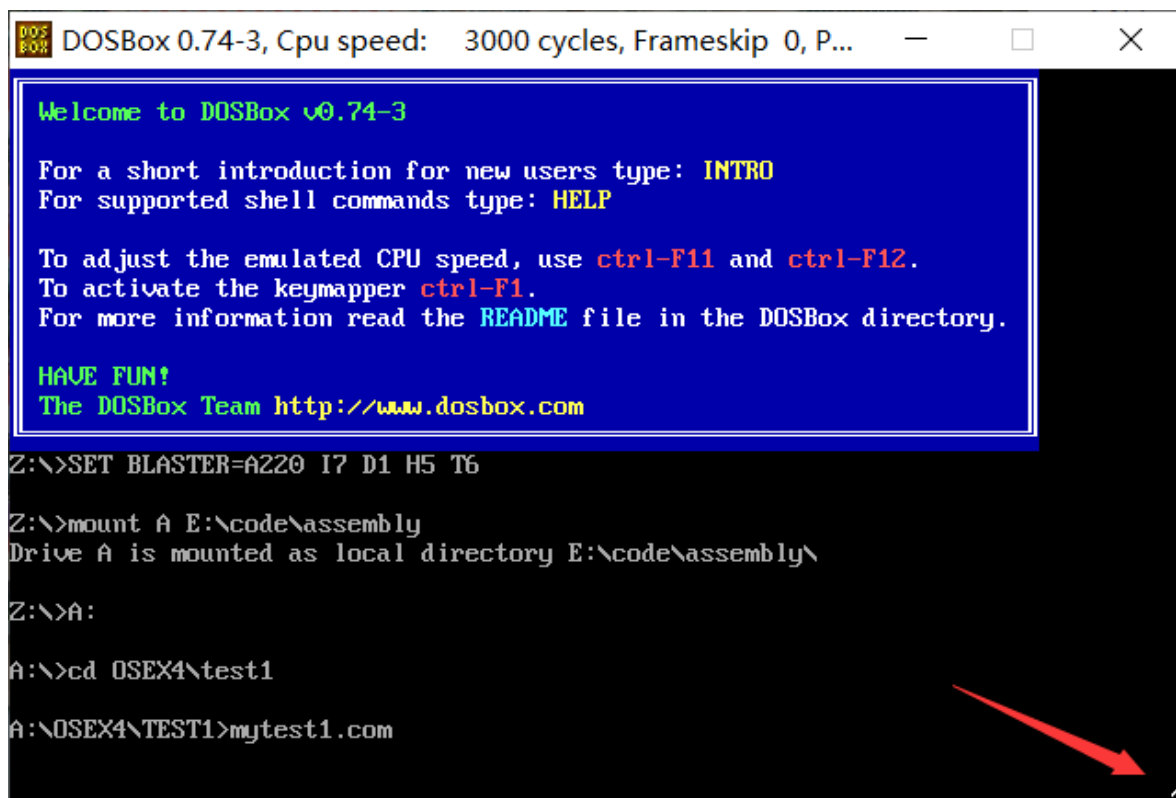
Z:\>mount A E:\code\assembly
Drive A is mounted as local directory E:\code\assembly\

Z:\>A:

A:\>cd OSEX4\test1

A:\OSEX4\TEST1>mytest1.com
```





```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, P...

Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount A E:\code\assembly
Drive A is mounted as local directory E:\code\assembly\

Z:\>A:

A:\>cd OSEX4\test1

A:\OSEX4\TEST1>mytest1.com
```

(二)、内核程序加无敌风火轮

代码讲解

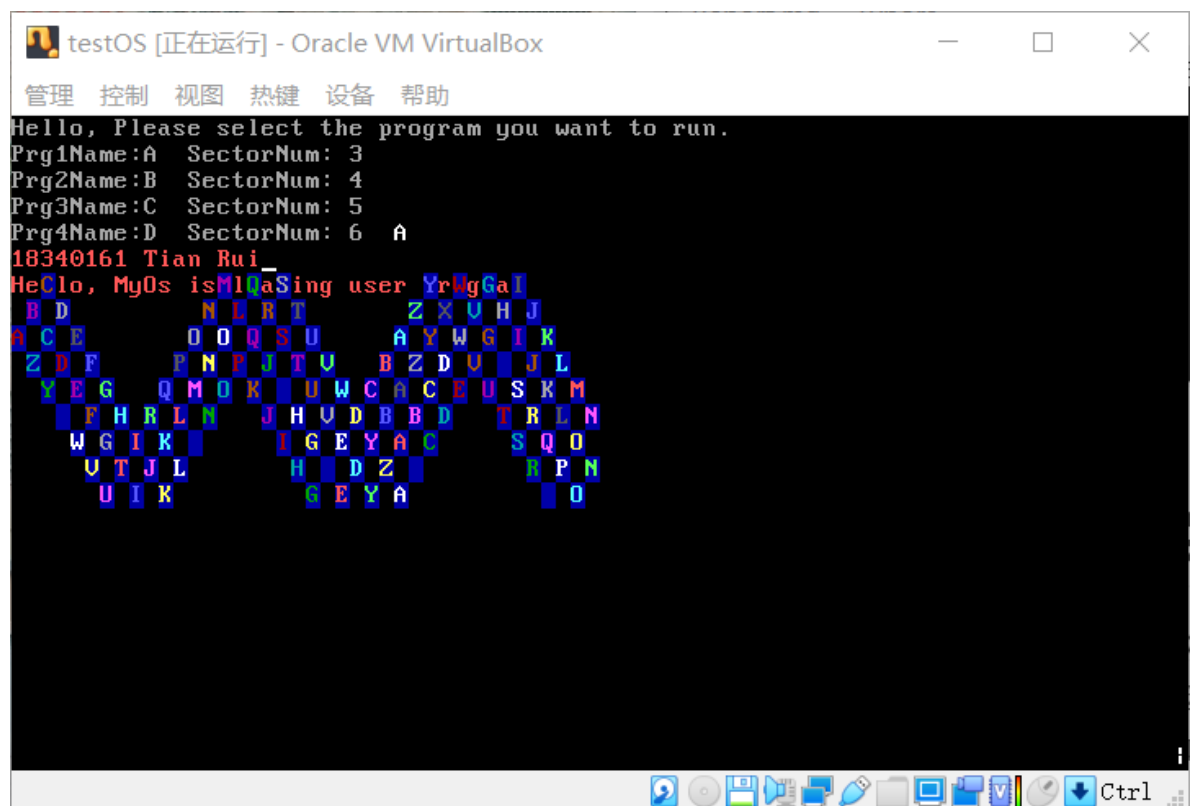
这一部分对程序的改动并不大，只要将实现的无敌风火轮中断处理程序在内核程序中加载到中断向量表就可以了，其余的部分都和上一个实验一样。在内核程序的开始部分增加这样一段代码

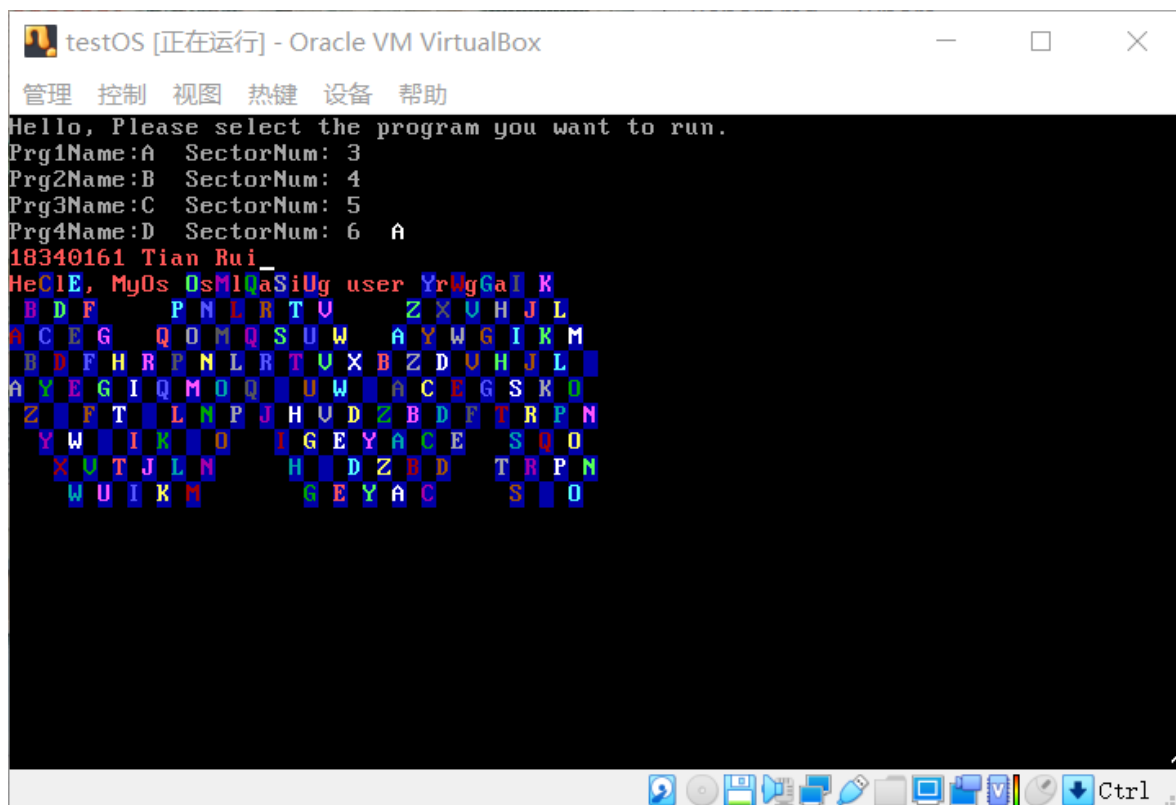
```
xor ax,ax          ; AX = 0
mov es,ax           ; ES = 0
mov word [es:20h],Timer ; 设置时钟中断向量的偏移地址
mov ax,cs
mov word [es:22h],ax ; 设置时钟中断向量的段地址=CS
mov ds,ax           ; DS = CS
mov es,ax           ; ES = CS
```

这一部分内容就是我们所说的将我们的程序加载到中断向量表中。由于我们的加载操作是在内核程序完成的，所以当程序处于引导扇区部分的时候，我们并不能看到无敌风火轮。

运行结果

用户程序运行，无敌风火轮转动。





(三)、键盘中断

代码讲解

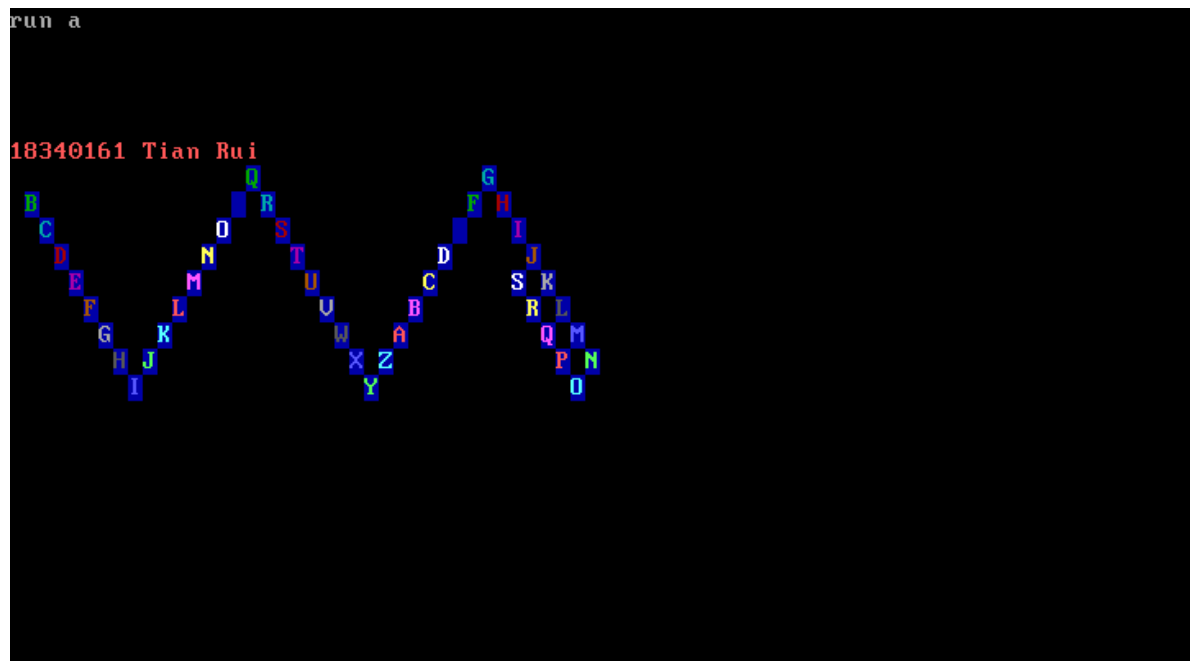
经过资料的查找发现实现键盘的中断有两种方法，一种是调用九号中断实现我们期望的要求。另一种方法更为巧妙，是使用时钟中断的方式，每一次时钟中断的时候都扫描键盘缓冲区，如果没有扫描到内容就继续执行，如果扫描到了内容就显示“OUCH”。我们这里采用了第二种方法（因为第一种没有成功）

```
int9:
    cli
    ;push ax
    dec byte [count]          ; 递减计数变量
    jnz end9                  ; >0: 跳转
    mov byte[count],delay     ; 重置计数变量=初值delay
    mov ah, 01h               ; 缓冲区检测
    int 16h
    jz end9                   ; 缓冲区无按键
    print ouch,OuchLength,3,40
    ;pop ax
    sti
end9:
    mov al,20h                ; AL = EOI
    out 20h,al                ; 发送EOI到主8529A
    out 0A0h,al               ; 发送EOI到从8529A
    iret
```

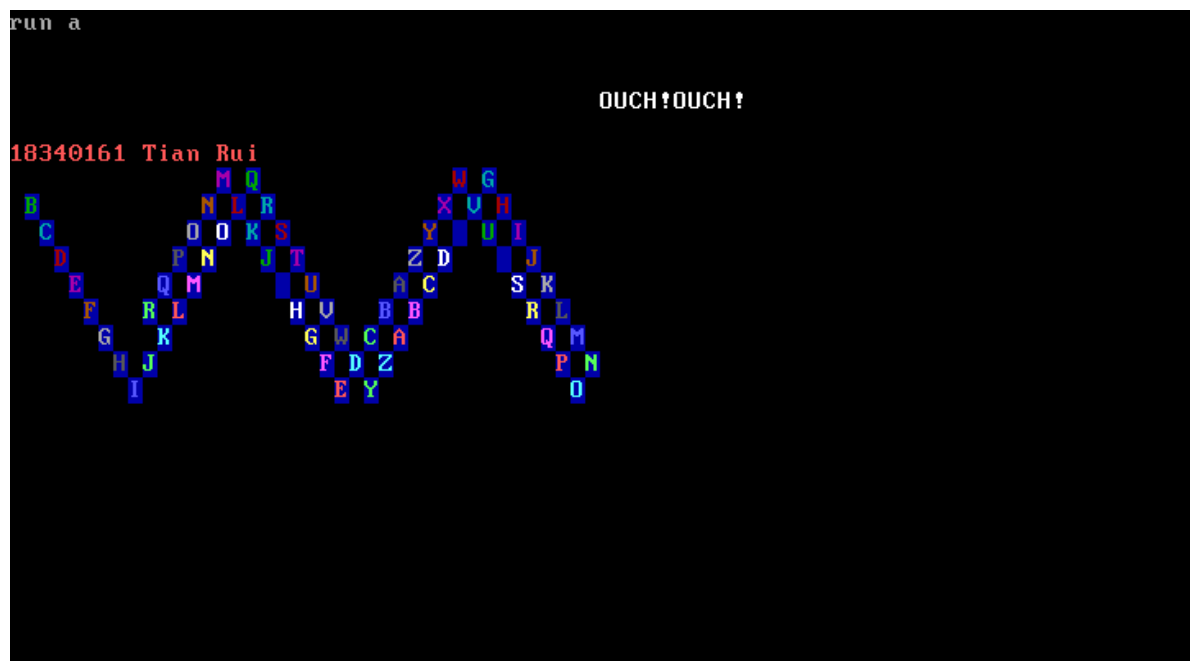
这里使用了一个print的宏定义函数。开始并没有使用宏，但是不知道原由地，如果不这样使用宏的话，程序就会卡在引导扇区的地方不再向下执行了。所以无可奈何使用了宏。这一段宏是借鉴的网上的代码，具体出处我们在参考文献中给出。

运行结果

开始运行时没有输出字符



在按下键盘之后



屏幕上出现了“Ouch! Ouch! ”

同理其他的三个程序也都运行成功

run b

18340161 Tian Rui_



run b

OUCH!OUCH!

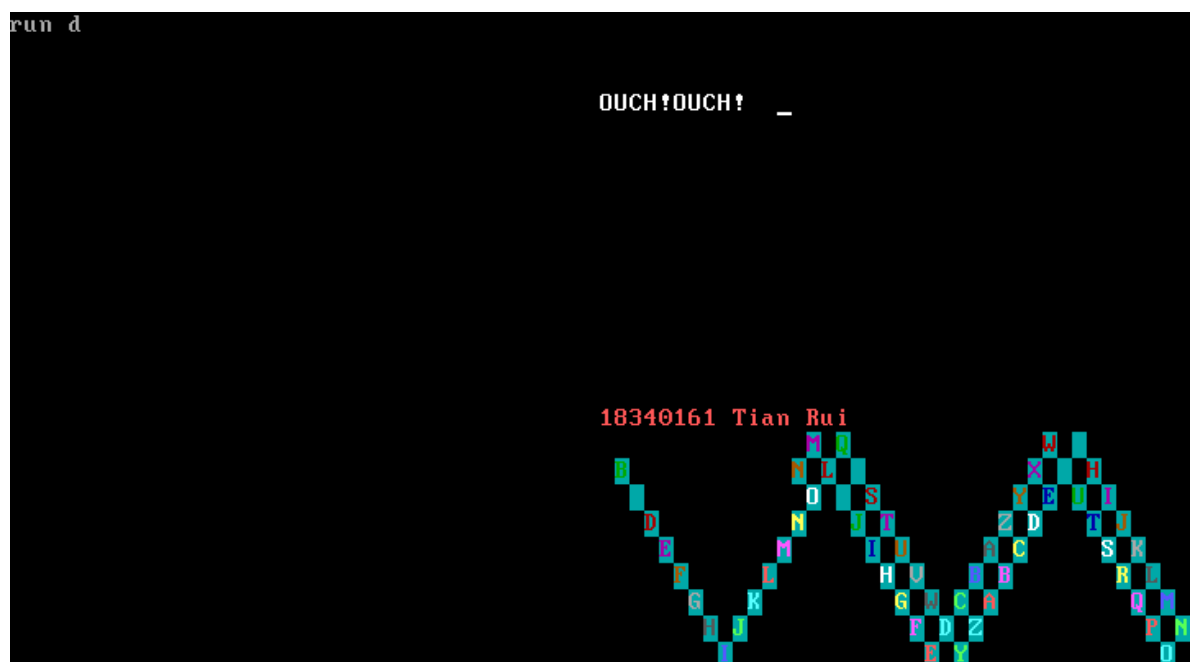
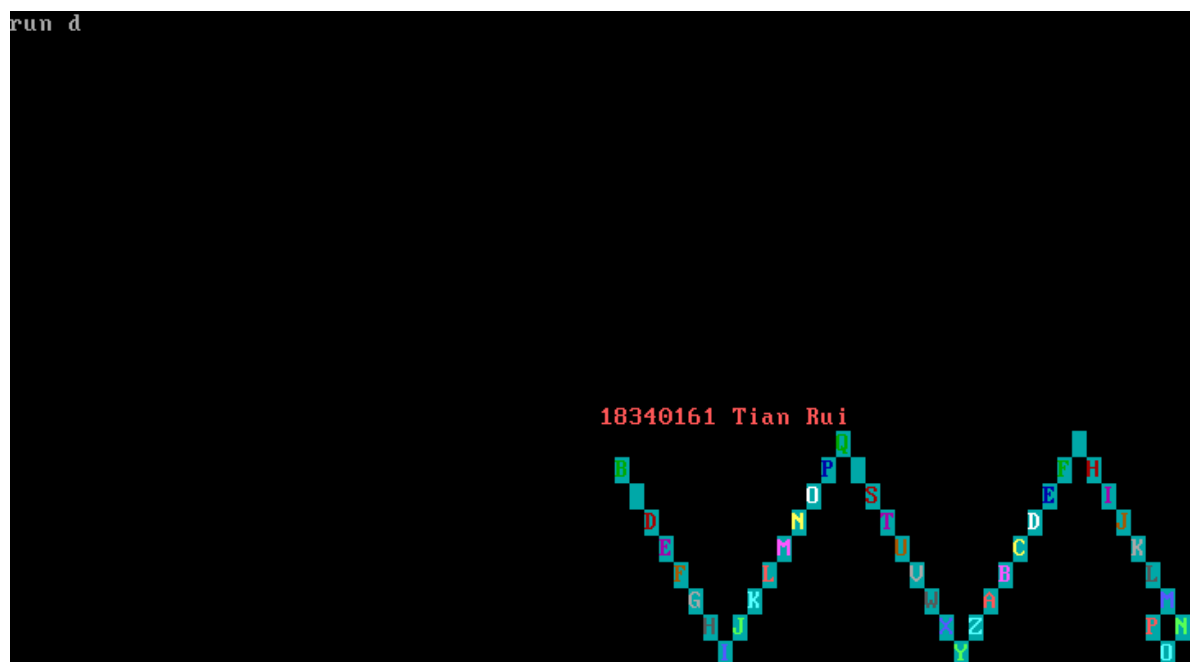
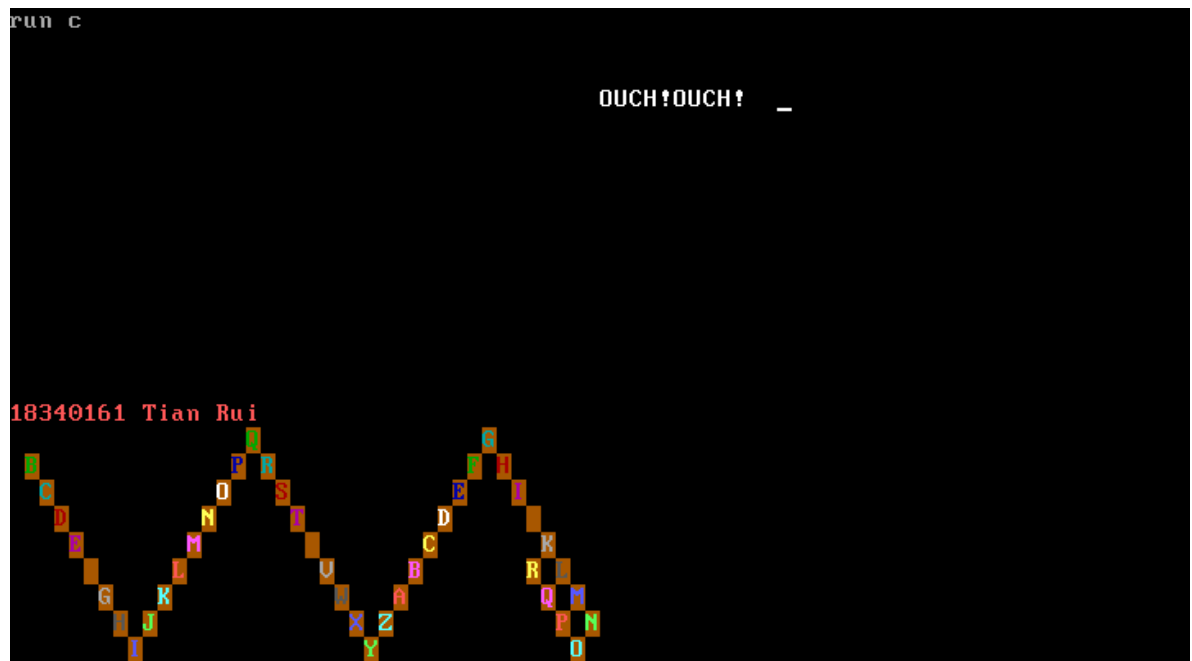
18340161 Tian Rui



run c

18340161 Tian Rui_





七、实验总结

这次实验想说的真的太多了。因为上一次的实验就没有做好，导致这次做实验的时候全面雪崩。所以这个故事告诉我们没一个实验都要认认真真的，千万不能投机取巧。这次试验大部分时间都在填上一个实验的坑，其实这次实验的工作量没有很大，所以这次实验做到自闭只能怪自己之前不够努力，不过好在终于把该填的坑都填上了。操作系统还有很多不完善的地方以后还要慢慢改进。另外还要特别谢谢傅禹泽同学，我真的特别感谢他。没有他我真的写不出这次的代码，大家的时间都很紧张，他能特意抽出一个上午的时间来教我写代码一点一点指导我我真的非常感谢，所以把他写在实验报告里表达我对他的感激之情。

1. 其实这次返回DOS的方式自己是有一些困惑的。因为在中断处理程序中调用中断，这样是不是就算是中断嵌套了呢？如果是这样的话，这个返回DOS的方式是否正确呢？是不是还有别的方法返回DOS呢？这些问题都还没有解决，希望以后可以再探讨一下这个问题。
2. 开始gets()函数是以阻塞模式实现的，导致在用户输入指令的时候，无敌风火轮不能转动，只有在程序运行的时候才能转动。后来尝试实现了不阻塞的输入，就是不断扫描键盘缓冲区，如果有输入了才向下进行，这样就实现了非阻塞输入。
3. 为了程序的精简性，开始将下面这一段代码放在了中断执行程序部分

```
mov ds,ax          ; DS = CS
mov es,ax          ; ES = CS
; 在屏幕右下角显示字符
mov ax,0B800h      ; 文本窗口显存起始地址
mov gs,ax          ; GS = B800h
mov ah,0Fh         ; 0000: 黑底、1111: 亮白字（默认值为07h）
mov al,byte[char1] ; AL = 显示字符值（默认值为20h=空格符）
mov [gs:((80*24+79)*2)],ax ; 屏幕第 24 行，第 79 列
```

但是只要换了位置运行结果就不正确，最后无可奈何还是换了回来。

4. 这次程序也不出意料地发生了一些玄学问题。一个很重要的一点呢就是如果我是用自己写的输出“Ouch”的程序，就连引导扇区部分都不能正确执行，这个是很奇怪的，因为将中断程序加载到中断向量表是在内核程序完成的，按理来说不应该影响到引导扇区，但是不知道为什么就是不行，这个问题还有待解决。
5. 期间遇到的很难解决的一个问题是，将中断控制程序加载到内存之后，执行用户程序就不返回内核程序了，但是只要不加载就可以返回。
6. 令我很困惑的一点是，如果加载了自己的中断就不能用bochs来debug，所以后期基本上是靠猜来debug。这到底是为什么呀！！
7. OUCH部分这里使用时钟中断实现键盘中断还有一个问题就是在键盘输入一个字符到缓冲区之后，运行结束会把缓冲区的字符读出来显示到屏幕上，所以屏幕上会出现上一次输入的字符。这是我们就需要单独清空一下缓冲区才可以。
8. 这次实验在大佬的指导下学会了运用脚本文件编译运行程序并将数据写入磁盘。这真的比自己以前的方法快了太多了。实现方法也非常简单，只要将我们需要的命令写在一个文件里并保存为.sh格式

```

nasm a.asm -o a.bin
nasm b.asm -o b.bin
nasm c.asm -o c.bin
nasm d.asm -o d.bin
nasm boot1.asm -o boot1.bin
gcc -march=i386 -m16 -mpreferred-stack-boundary=2 -ffreestanding -fno-PIE -masm=intel -c mykernel.c -o kernelc.o
nasm -f elf32 mykernel.asm -o kernelasm.o
ld -m elf_i386 -N kernelasm.o kernelc.o -Ttext 0x100 --oformat binary -o kernel.bin

dd if=/dev/zero of=os2.img bs=512 count=2880
dd if=boot1.bin of=os2.img bs=512 count=1 seek=0 conv=notrunc
dd if=kernel.bin of=os2.img bs=512 count=10 seek=1 conv=notrunc
dd if=a.bin of=os2.img bs=512 count=1 seek=11 conv=notrunc
dd if=b.bin of=os2.img bs=512 count=1 seek=12 conv=notrunc
dd if=c.bin of=os2.img bs=512 count=1 seek=13 conv=notrunc
dd if=d.bin of=os2.img bs=512 count=1 seek=14 conv=notrunc

```

之后每一次编译只要调用这个文件就可以了，真的太方便了！！！果然还是要多和大佬交流！

```

gardenia@gardenia-virtual-machine:~/OS/EX4/test3$ sh ins.sh
记录了2880+0 的读入
记录了2880+0 的写出
1474560 bytes (1.5 MB, 1.4 MiB) copied, 0.00664961 s, 222 MB/s
记录了1+0 的读入
记录了1+0 的写出
512 bytes copied, 0.000137779 s, 3.7 MB/s
记录了3+1 的读入
记录了3+1 的写出
2041 bytes (2.0 kB, 2.0 KiB) copied, 0.000144847 s, 14.1 MB/s
记录了0+1 的读入
记录了0+1 的写出
465 bytes copied, 0.000138456 s, 3.4 MB/s
记录了0+1 的读入
记录了0+1 的写出
465 bytes copied, 0.000262398 s, 1.8 MB/s
记录了0+1 的读入
记录了0+1 的写出
465 bytes copied, 0.000134471 s, 3.5 MB/s
记录了0+1 的读入
记录了0+1 的写出
465 bytes copied, 0.000131637 s, 3.5 MB/s

```

八、参考文献

-
- [1]wu-kan.中断控制[EB/OL].<https://wu-kan.cn/posts/2019-03-31-%E4%B8%AD%E6%96%AD%E6%8E%A7%E5%88%B6/,2019-3-31>.
 - [2]李忠,王晓波.X86汇编语言 从实模式到保护模式[M].电子工业出版社:北京,2013:149.
 - [3]Halcyon.具有中断处理的内核[EB/OL].https://blog.csdn.net/qg_41858347/article/details/106187918,2020-05-18.