实验四 异步事件编程技术 实验报告

数据科学与计算机学院　2017 级计算机科学与技术　17341146　王程钥

1　实验题目

异步事件编程技术

2　实验目的

学习中断机制相关知识
掌握中断处理程序设计的方法
掌握设计时钟中断处理程序的方法
掌握设计键盘中断处理程序的方法
掌握设计系统调用的方法

3　实验要求

1) 操作系统工作期间，利用时钟中断，在屏幕 24 行 79 列位置轮流显示’|’、’/’和’\’，适当控制显示速度，以方便观察效果。
2) 编写键盘中断响应程序，原有的你设计的用户程序运行时，键盘事件会做出有事反应：当键盘有按键时，屏幕适当位置显示”OUCH! OUCH!”。
3) 在内核中，对 33 号、34 号、35 号和 36 号中断编写中断服务程序，分别在屏幕 1/4 区域内显示一些个性化信息。再编写一个汇编语言的程序，作为用户程序，利用 int 33、int 34、int 35 和 int 36 产生中断调用你这 4 个服务程序。

4　实验方案

4.1 实验环境

编程环境：Dosbox+TCC+TASM+Tlink,NASM
16 进制编辑器：Hex Editor
虚拟机：WMware Workstation
虚拟机环境

- 操作系统　MS-DOS
- 内存　1MB
- 硬盘　102MB
- 处理器　1 核心

编译命令

```
nasm -f bin guide.asm -o guide.com
tasm myos.asm myos.obj
tcc -mt -c –omain.obj main.c
tlink /3 /t myos.obj main.obj,test.com,
```

4.2 编写时钟中断

时钟中断每 1/18.2 秒会运行一次，本次实验需要利用时钟中断编写一些和时钟相关的功能，让它在视觉上持续运行。我首先编写了一个“风火轮”程序，在屏幕右下方显示一个不停旋转的“风火轮”。之后，我对时钟中断增加了显示系统时间的功能，使得屏幕右下方能够实时显示系统时间。

4.3 编写键盘中断

键盘中断在键盘按下或弹起时会运行。本次实验需要利用键盘中断实现在特定时间内按下键盘显示“OUCH!”的功能。

4.4 编写 33,34,35,36 号中断

我在 33,34,35,36 号中断程序中实现了一些个性化的功能，使屏幕上能够以打字机的形式显示一首诗。因为在裸机中显示汉字比较困难，所以这首诗以拼音的形式显示。

4.5 程序存储设计与地址存放

以下为存储扇区

功能	程序名	存储扇区
引导程序	guide.com	1
内核	test.com	2-12
文件夹	file.txt	13
用户程序 1	stone1.com	14
用户程序 2	Stone2.com	15
用户程序 3	stone3.com	16
用户程序 4	stone4.com	17
脚本程序 1	shell1.bat	18
脚本程序 2	shell2.bat	19
8 号中断处理程序	newint8.com	20
9 号中断处理程序	newint9.com	21
33 号中断处理程序	newint33.com	22
34 号中断处理程序	newint34.com	23
35 号中断处理程序	newint35.com	24
36 号中断处理程序	newint36.com	25

以下为内存地址

功能	地址
引导程序	7c00h
内核	A100h
用户程序	E100h
8 号中断处理程序	9500h
9 号中断处理程序	9900h
33 号中断处理程序	C500h
34 号中断处理程序	C700h

35 号中断处理程序	C900h
36 号中断处理程序	Cb00h
文件信息	8000h
文件大小	8400h
脚本程序加载	8800h

5 实验过程

本次实验以实验三中实现的内核为基础进行修改。本次实验对操作系统的功能进行了一些修改，并嵌入了几个中断程序。

我对命令行进行了修改，将 time 命令改为系统时间和“风火轮”显示的开关，并增加了 ouch 命令，作为 9 号中断修改的开关。开关在指令奇数次打开，偶数次关闭。

因为修改后的 9 号中断会影响键盘输入，所以 9 号中断只有在 ouch 模式开启且运行用户程序时才会被加载为输出“OUCH!”的模式。另外因为在 ouch 模式下用户程序无法按键退出，所以我修改了用户程序的退出机制。用户程序有两种退出机制，非 ouch 模式下的按键退出和等待运行 500 次（约 30 秒）后自行退出。

另外上次实验中的关机是假关机，在本次实验中我实现了真关机的功能。

5.1 “风火轮”的编写

首先我使用了老师给的 8 号中断例程，将其加载到扇区里，装入虚拟机后时钟中断成功运行。于是我将老师程序中的字母递增改成了“风火轮”显示，将其加载到扇区并装入虚拟机后，却发现它不能正确执行。“风火轮”需要等一段时间后才能开始旋转，且键盘输入崩溃，不能输入信息。

时钟中断程序是几乎实时运行的，运行时会改变寄存器的值。因此中断程序需要加上寄存器保护的措施。我在中断程序中加上了如下的寄存器保护的代码，并再装入虚拟机尝试运行。

```
push es
push ds
<8 号中断程序>
pop ds
pop es
```

结果键盘输入的问题得到了解决，但“风火轮”还是要等一段时间后才能开始转动。且“风火轮”在运行用户程序的时候会停转，在返回内核后恢复转动。

我通过输出调试的方法，发现时钟中断中 count 变量的初值和我的预设值不同。Count 变量的值很大，它需要不断地减小，直到 0，之后风火轮才会开始旋转。时钟中断一秒运行 18 次，count 值很大的话就需要等很长的一段时间后才能减到 0。

我加入了以下代码，将 count 变量的初值进行了修正，再装入虚拟机运行后，“风火轮”运行的问题得到了解决。

```
cmp byte[count],delay
jbe go
mov byte[count],delay
```

但强行修正并不是解决问题的根本方法。我继续思考可能出现的问题。变量初值出错

一般是数据段设置的问题，但我在安装中断部分已经将 ds 寄存器的值赋成了 cs。我尝试在中断程序的运行部分强行将 ds 寄存器修正为 cs，并将之前的特判代码注释掉。装入虚拟机进行测试后，发现“风火轮”成功运行，不在出现上述问题。

以下为“风火轮”部分的代码。

```
mov ax,cs
    mov ds,ax
    mov es,ax
    mov ax,0B800h
    mov gs,ax

    dec byte[count]
    jnz end
    mov ah,0Fh
    dec byte[num]

    cmp byte[num],0
      jz show0
    cmp byte[num],1
      jz show1
    cmp byte[num],2
      jz show2
    cmp byte[num],3
      jz show3
show0:
    mov byte[gs:((80*24+79)*2)],'-'
    mov byte[num],4
    jmp reset
show1:
    mov byte[gs:((80*24+79)*2)],'/'
    jmp reset
show2:
    mov byte[gs:((80*24+79)*2)],'|'
    jmp reset
show3:
    mov byte[gs:((80*24+79)*2)],'\'
reset:
    mov byte[count],4
```

5.2 时间显示程序的编写

我希望在界面右下角，即风火轮的旁边显示实时更新的系统时间。这个功能的实现需要用到每 1/18.2 秒运行一次的时钟中断。

获取系统时间需要使用 BIOS 的 0Ah 号中断的 02h 和 04h 号功能。这两个中断取出来的时间为 BCD 码格式，我需要将其转化为 ASCII 码并显示到屏幕上。关于显示时间部分的代码的实现，我参考了李忠老师的《X86 汇编语言-从实模式到保护模式》一书第九章的一个例程。这段例程有一段简短且优美的 BCD 码转 ASCII 码的代码。

在实现了这个功能后，实验三中的 time 命令显示系统时间的功能由于功能重叠，没有意义，因此被删除。

以下为显示时间部分的代码。

```
showtime:
    mov bx,(80*24+59)*2

    mov ah,04h
    int 1ah
    mov al,ch
    call bcd_to_ascii
    mov al,cl
    call bcd_to_ascii
    mov byte[gs:bx],'/'
    add bx,2
    ;年

    mov al,dh
    call bcd_to_ascii
    mov byte[gs:bx],'/'
    add bx,2
    ;月

    mov al,dl
    call bcd_to_ascii
    mov byte[gs:bx],' '
    add bx,2
    ;日

    mov ah,02h
    int 1ah

    mov al,ch
    call bcd_to_ascii
    mov byte[gs:bx],':'
    add bx,2
    ;时
```

```
    mov al,cl
    call bcd_to_ascii
    mov byte[gs:bx],':'
    add bx,2
    ;分

    mov al,dh
    call bcd_to_ascii
    ;秒

    jmp end

bcd_to_ascii:
    mov ah,al
    and al,0x0f
    add al,0x30
    shr ah,4
    and ah,0x0f
    add ah,0x30
    mov byte[gs:bx],ah
    add bx,2
    mov byte[gs:bx],al
    add bx,2
    ret
    ;BCD 码转 ASCII 码
```

5.3 键盘中断处理程序的编写

我希望键盘中断在 OUCH 模式开启并进入用户程序后，能够在屏幕上的随机位置显示“OUCH”字符。

我借助了系统时钟构造了一种伪随机数的生成方法。系统时钟的使用需要用到 5.2 中提到的 BIOS 的 0Ah 中断的 02h 号功能。我的伪随机数生成方式如下：将初始坐标乘上一个质数，加上一个随机数，再对另一个质数取模数。

```
x=(x*17+rand()) mod 23
y=(y*17+rand()) mod 71
```

键盘中断的处理是有坑的。在写好键盘中断处理程序后，我将其写入虚拟机并尝试运行。我发现键盘中断在第一次按下后可以在对位位置显示“OUCH!”，但之后再按键盘屏幕却不再有反应。

很幸运的是，周五上课的时候老师答疑的过程中提到了这个问题，并给出了解决方案。键盘输入的值没有被提取出来，阻塞在里面，键盘再按下后字符不会被写入，自然也就不会再调用键盘中断。因此，要在 09 号中断程序中，把写入的字符提取出来。在加上以下代码后，我的键盘中断终于能正常工作了。

```
in al,60h
mov byte[cc],al
```

以下为键盘中断部分的相关代码。

```
get_pos:
    mov ax,word[X]
    mov bx,80
    mul bx
    add ax,word[Y]
    mov bx,2
    mul bx
    mov bx,ax
    ;获取显示位置

show:
    mov ah,0Fh
    mov byte[gs:bx],'O'
    mov byte[gs:bx+2],'U'
    mov byte[gs:bx+4],'C'
    mov byte[gs:bx+6],'H'
    mov byte[gs:bx+8],'!'
    ;显示“OUCH!”

get_x:
    mov ax,word[X]
    mov bx,17
    mul bx
    call random
    add ax,bx
    xor dx,dx
    mov bx,23
    div bx
    mov word[X],dx
    ;修改 x 坐标

get_y:
    mov ax,word[Y]
    mov bx,17
    mul bx
    call random
    add ax,bx
    xor dx,dx
    mov bx,73
    div bx
    mov word[Y],dx
    ;修改 y 坐标
```

```
    jmp end

random:
    push ax
    mov ah,02h
    int 1ah
    mov bx,dx
    pop ax
    ret
    ;获取随机数
```

5.4 中断处理程序的加载与卸载

5.4.1 装载中断

我将中断当作用户程序存储在了扇区里，以避免内核过大，代码太乱，不方便调试。装载中断的代码从老师给的代码中进行修改，从 NASM 格式改成 C 内嵌汇编的格式即可。装载中断和运行用户程序基本相似，使用 BIOS 的 int13h 中断将段地址和偏移量调整到对应扇区，并将相应的段地址和偏移量修改到中断向量对应的段地址和偏移量即可。

以下为相关代码。

```
void load_int(char pos,int Intvec,int offset)
{
  int i,pre=Intvec;
  char p=(pos-1)/18;
  pos%=18;if(!pos)pos=18;
  asm push es
  asm mov ax,0h
  asm mov es,ax
  asm mov bx,offset
  asm mov cl,pos
  asm mov ah,2
  asm mov al,1
  asm mov dl,0
  asm mov dh,p
  asm mov ch,0
  asm int 13h
  asm pop es

  asm push ds
  asm mov ax,0
  asm mov ds,ax
  Intvec*=4;
```

```
  asm mov bx,Intvec

  asm mov ax,offset
  asm mov [bx],ax
  Intvec+=2;
  asm mov bx,Intvec

  asm mov ax,0
  asm mov [bx],ax
  asm pop ds
}
```

5.4.2 卸载中断

8 号中断和 9 号中断只有才特定模式才会被加载，所以需要编写一段程序，在特定时间加载这两个中断，并将这两个中断原来的地址提取并存储出来。在其他时间则利用提取的原中断向量段地址和偏移量，将这两个中断恢复为原来的中断。

我在内核中用 C 语言内嵌汇编实现了这一段代码，结果也遇到了一些问题。一开始在恢复回原中断后，我发现操作系统死机了，键盘无法输入。我尝试将提取到的中断向量的段地址和偏移地址输出，结果发现是两个 0。我是将这两个变量设置为全局变量的，因此我怀疑是因为在取这两个地址时数据段被更改，导致其获得的值并没有储在我想要存储的全局变量里面。因此我将这两个值先存储在局部变量中，并在 ds 寄存器退栈后再将其存入全局变量中。结果提取到的值就神奇地变正常了。其实我并不能说太清楚为什么。

以下为相关代码。

存储 8,9 号中断地址。

```
int pre=Intvec,cs_8,ip_8,cs_9,ip_9;
asm push ds
asm mov ax,0
asm mov ds,ax
Intvec*=4;
asm mov bx,Intvec
if(pre==8){
  asm mov ax,[bx]
  asm mov ip_8,ax
}
if(pre==9){
  asm mov ax,[bx]
  asm mov ip_9,ax
}
(装中断，略)
Intvec+=2;
asm mov bx,Intvec
if(pre==8){
```

```
 asm mov ax,[bx]
 asm mov cs_8,ax
}
if(pre==9){
  asm mov ax,[bx]
  asm mov cs_9,ax
}
(装中断，略)
asm pop ds
if(pre==8)cs8=cs_8,ip8=ip_8;
if(pre==9)cs9=cs_9,ip9=ip_9;
```

卸载 8,9 号中断

```
void reload_int(int vec,int Intcs,int Intip)
{
  asm push ds
  asm mov ax,0
  asm mov ds,ax
  vec*=4;
  asm mov bx,vec
  asm mov ax,Intip
  asm mov [bx],ax
  vec+=2;
  asm mov bx,vec
  asm mov ax,Intcs
  asm mov [bx],ax
  asm pop ds
}
```

5.5 33,34,35,36 号中断处理程序的编写

这四个中断的编写比较常规，但也是有一些坑的。我对这四个中断做了一些简单的个性化拓展，在显示中断编号信息之外，我还在这四个中断中各放了一句古诗，并采用类似打字机的方式将其显示在屏幕上。

实现方式很简单，我先调用 int10h 中断的 13h 号功能，在屏幕相应位置用白色显示相应的诗词。然后我写了一个循环，人工在同一个位置再将这句诗以不同颜色输出，从而达到效果。第二次输出每两个字之间有延迟，这可以通过循环空转实现。

我写完后将这四个中断装入软盘尝试在虚拟机上运行，结果发现在运行 13 号中断的时候时钟中断和键盘中断不再响应，“风火轮”和系统时间都不再显示。我并不知道是什么原因，我尝试在这四个中断的中断程序中加上 sti 指令，强行开中断，再装入虚拟机后时钟中断和键盘中断居然就能正常中断了。我感到十分震惊，并不知道为什么，之前调用 10h 号中断输出字符串也是能成功运行的，或许是哪里写得不规范导致莫名其妙被关中断了？

以下是相关代码，以 33 号中断为例。

```
org 0c500h

start:
```

```
    push ax
    push bx
    push cx
    push dx
    push ds
    push es
    push bp
    push si
    push di

    mov ax,cs
    mov ds,ax
    mov bp,Message1
    mov ax,ds
    mov es,ax
    mov cx,Message1Length
    mov ax,1301h
    mov bx,003Fh
    mov dh,4
    mov dl,42
    int 10h
    mov ax,cs
    mov ds,ax
    mov bp,Message2
    mov ax,ds
    mov es,ax
    mov cx,Message2Length
    mov ax,1301h
    mov bx,0007h
    mov dh,5
    mov dl,40
    int 10h

    sti                  ; I don't know why need this qaq

    mov ax,cs
    mov ds,ax
    mov es,ax
    mov ax,0B800h
    mov gs,ax

    mov si,Message2
    mov word[len],Message2Length
    mov bx,880
```

```
print_str:
    mov byte al,[si]
    mov ah,0eh
    cmp al,'Z'
    jg putchar
    mov ah,0ch
putchar:
    mov [gs:bx],ax
nxt:
    inc si
    add bx,2
    mov word[count1],50
    mov word[count2],10000
    call sleep
    dec word[len]
    jnz print_str
    jmp end

sleep:
    dec word[count2]
    jnz sleep
    mov word[count2],10000
    dec word[count1]
    jnz sleep
    ret

end:
    pop di
    pop si
    pop bp
    pop es
    pop ds
    pop dx
    pop cx
    pop bx
    pop ax
    mov al,20h    ; AL = EOI
    out 20h,al    ; 发送 EOI 到主 8529A
    out 0A0h,al   ; 发送 EOI 到从 8529A
    iret          ; 从中断返回

datadef:
    len dw 0
    count1 dw 0
```

```
    count2 dw 0
    Message1 db 'This is int 33h'
    Message1Length equ ($-Message1)
    Message2 db 'Bai Ri Yi Shan Jin'
    Message2Length equ ($-Message2)
```

以下为中断测试程序，先输出古诗名字《登鹤雀楼》，后依次调用四个中断。

```
org 0e100h
start:
  mov ax,cs
  mov ds,ax
  mov es,ax

  mov ah,06h
  mov al,0
  mov ch,0
  mov cl,40
  mov dh,12
  mov dl,79
  mov bh,07h ;黑底白字
  int 10h

  mov bp,Message        ; 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,MessageLength ; CX = 串长（=9）
  mov ax,1301h       ; AH = 13h（功能号）、AL = 01h（光标置于串尾）
  mov bx,006Fh       ; 页号为 0(BH = 0) 黑底白字(BL = 07h)
  mov dh,2           ; 行号=0
  mov dl,50          ; 列号=0
  int 10h               ; BIOS 的 10h 功能：显示一行字符



  int 33h
  int 34h
  int 35h
  int 36h
  ret

datadef:
    Message db 'Deng He Que Lou'
    MessageLength equ ($-Message)
    times 510-($-$$) db 0
    db 0x55,0xaa
```

5.6 关机程序

实验 3 的关机功能，我是通过退出 main 函数实现的，其实这只是退出了操作系统，并没有关机。本次实验，我在网上查找资料的时候意外发现了一段关机代码[3]，贴下来运行了一下，结果真的能够关机，惊了。

这段代码大概是调用了 BIOS 的 15h 号中断的 53h 号功能，具体我不是很懂，也没有更多的参考资料。不过我还是把这段代码贴到了内核里面。

以下为相关代码

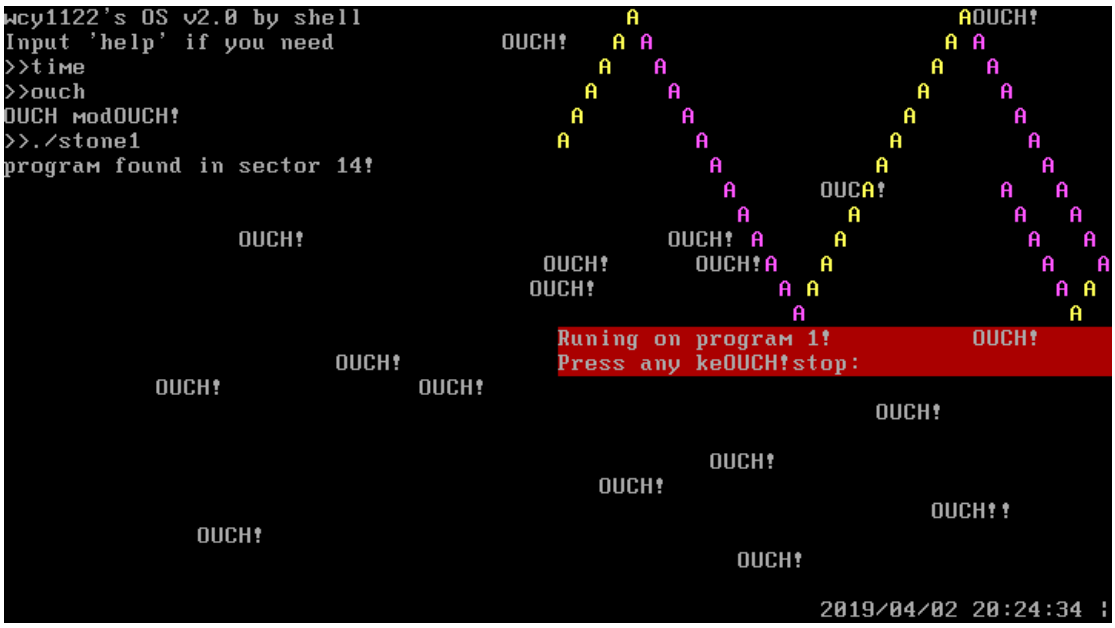
```
void sys_exit()
{
  asm mov ax,5301h
  asm xor ax,bx
  asm int 15h
  asm mov ax,530Eh
  asm xor bx,bx
  asm mov cx,102h
  asm int 15h
  asm mov ax,5307h
  asm mov bx,1
  asm mov cx,3
  asm int 15h
}
```

5.7 运行结果

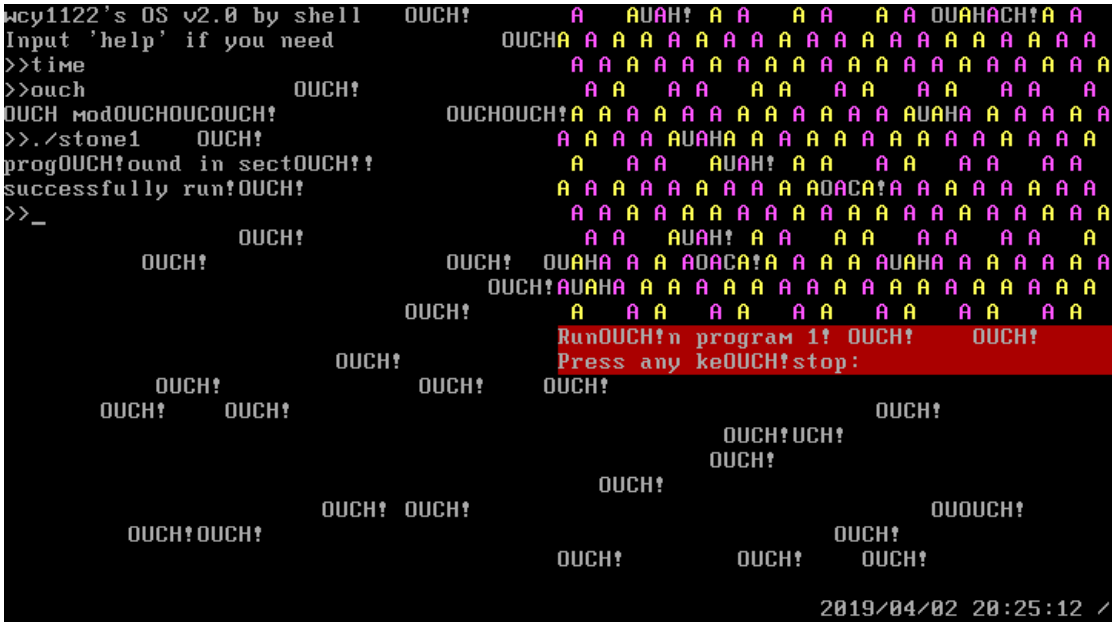
进入操作系统

装载 8 号中断，显示系统时间和风火轮

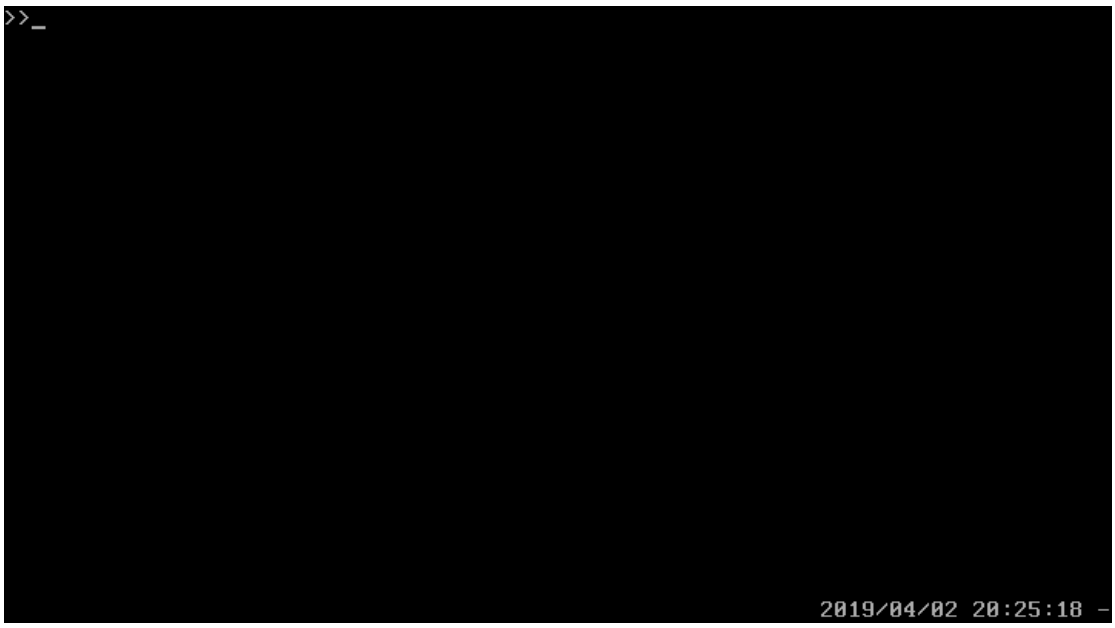
装载 9 号中断，运行用户程序时按键盘会显示“OUCH!”



运行用户程序，敲击键盘，屏幕上随机位置显示“OUCH!”



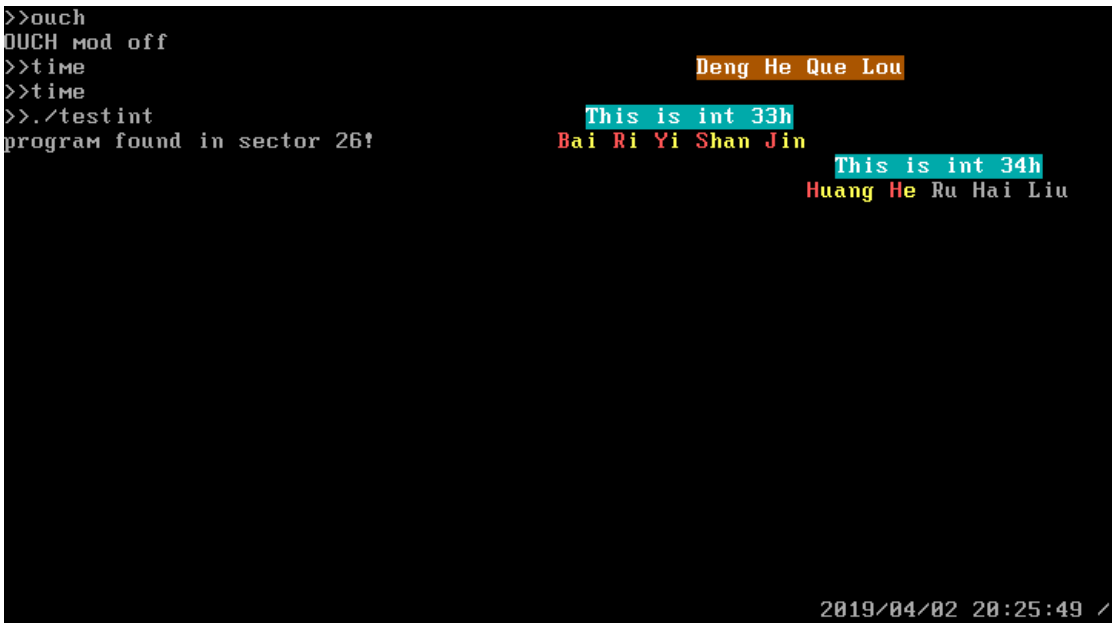
用户程序运行结束，返回内核



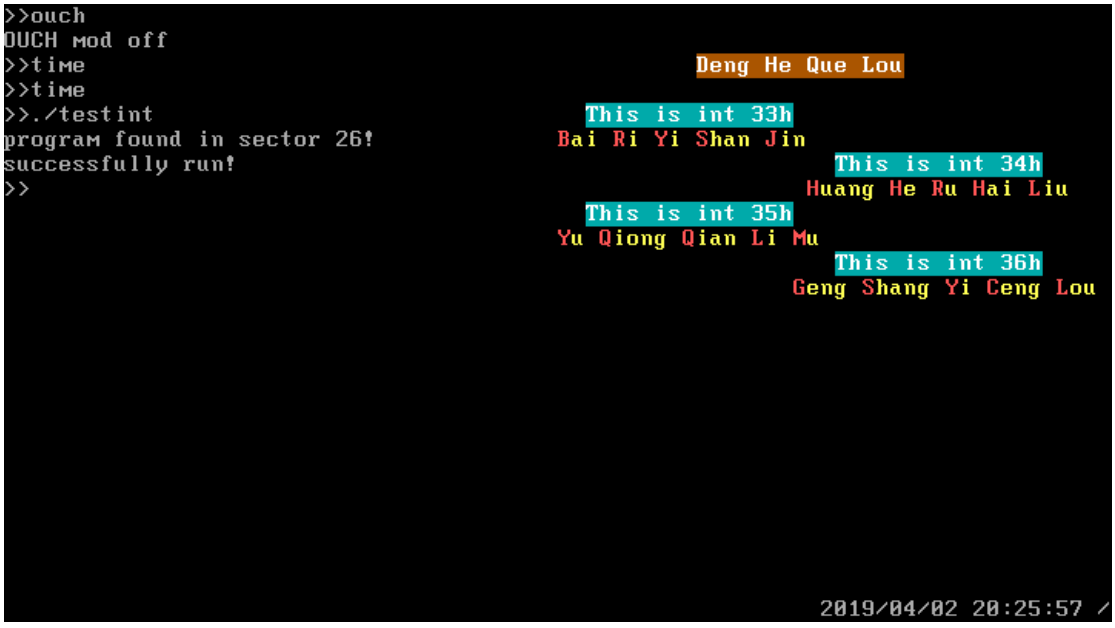
cls 命令，清屏

卸载 8 号中断，恢复为系统默认，并关闭 ouch 模式。

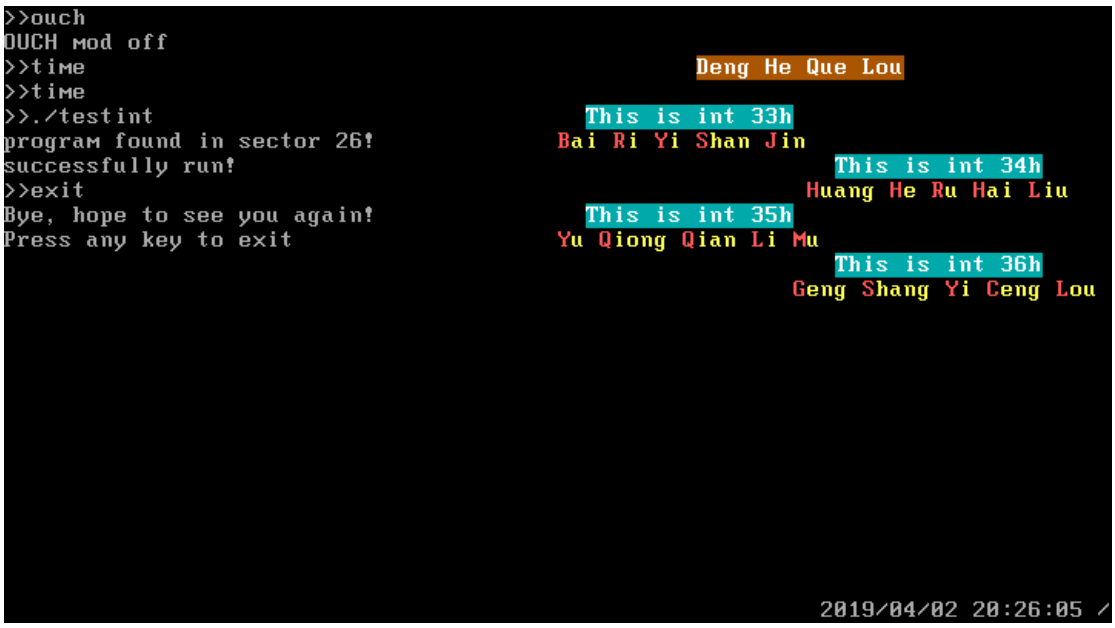
运行用户程序时敲击键盘不再显示 ouch



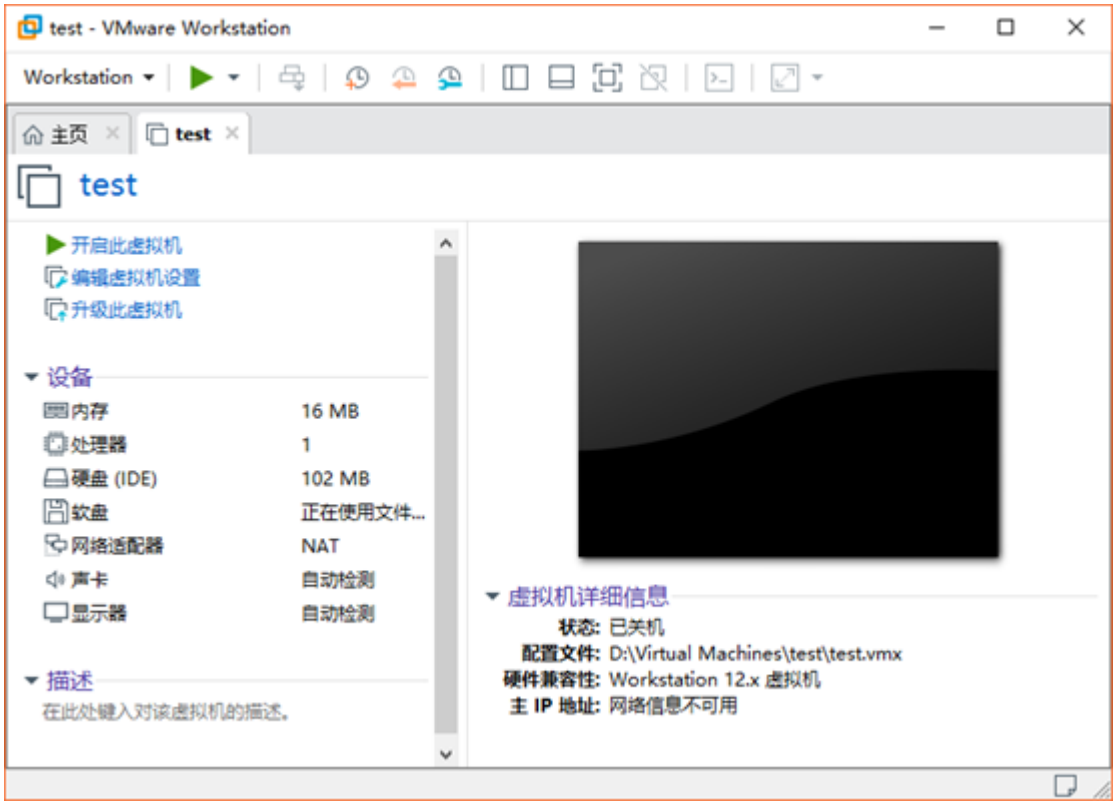
重新打开系统时间并运行中断测试程序测试 int33h~int36h 四个中断程序。



运行结束



输入 exit 命令，等待键盘按键以关机



成功关机

6 实验总结

本次实验要求在实验 3 的基础上编写中断处理程序，修改系统中断。在实验 3 的基础下，本次实验的代码量和难度都不算太大。不过本次实验有一些坑，尤其是段寄存器保护和赋值的问题，坑了我很久。代码不规范，debug 两行泪。

本次实验遇到了一些目前我还不能解释，或是不太清楚的问题，比如在运行 int33h~int36h 的测试程序过程中需要手动开中断以让 8 号中断和 9 号中断运行，我并不知道为什么需要这么做。还有获取 8 号中断和 9 号中断的时候需要开一个临时变量先存下这两个中断的偏移量和段地址，我也不太清楚为什么。如果 TA 知道为什么要这么做，可以私戳我告诉我原因，感谢。

虽然还有一些未能解决的疑问，加了一些不知道为什么要加的特判，不过这次实验也算是顺利地完成了，基本上实现了我一开始想要的功能。而且我还找到了一段关机的代码，虽然我不是很懂，也没有找到相关的其他参考资料，但至少我实现了真正关机，这算是一个意外的惊喜了吧。

通过本次实验，我了解了 DOS 中断向量和中断程序存储的机制，了解了写中断的方法，了解了保护寄存器的重要性。在实现基础功能后，我还实现了一些额外的个性化功能，比如系统时间的实时显示，比如 int33h~int36h 中断四个程序的个性化设计，比如中断的自由装载和卸载。我对操作系统的中断机制有了更深的了解。总体来说，对于本次实验，我还是非常满意了。

参考文献

[1] BIOS 中断大全 https://www.cnblogs.com/coderCaoyu/p/3638713.html

[2]《X86 汇编语言-从实模式到保护模式》，李忠

[3] 关机 https://zhidao.baidu.com/question/3099245.html