实验 4：异步事件编程技术

姓名：姜洋帆　　学号：17341068
院系：数据科学与计算机学院　专业：17 级计算机科学（大数据）

【实验题目】异步事件编程技术

【实验目的】

1. 学习中断机制知识，掌握中断处理程序设计的要求。
2. 设计一个汇编程序，实现中断处理程序。
3. 扩展 MyOS2，增加时钟中断服务，利用时钟中断实现与时间有关的操作。

【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。

【实验方案】

一、 硬件及虚拟机配置：Lenovo PC ； Oracle Virtual Box

二、 软件工具及作用：

Notepad++ ：写代码

Nasm：编译引导程序的 asm 文件到 bin 文件

Tasm：编译 16 位的内核代码

Tcc：编译 16 位 C 语言，实现 C 语言与 x86 联合编译

Sublime ：查看、编辑二进制文件，将编译完成的程序机器码写入软盘

C 程序：

1. 自己编写的一个 C 语言程序,用来将编译生成的二进制文件填充为 1.44M 大小，并将后缀改为.img，生成 1.44M 的软盘
2. 内核程序，实现一些逻辑性较强的功能，通过调用汇编实现的一些函数来实现各种基本操作，组合出复杂的功能

86 汇编：

1. Myos3.asm 实现引导程序，偏移量位 7c00h，将内核加载至内存并跳转进入内核程序
2. Kernel.asm、ckernal.c 为内核程序的主要部分，自定义了一些需要的中断，完成内核管理机器的一些基本功能
3. Kliba.asm 实现内核程序以及一些常用的函数，如打印字符，输入字符等，可以被 C 语言调用

三、 程序功能

Myos1 作为引导程序，设置好内核的偏移和段地址，加载内核程序，然后跳入内核程序，进行下一步操作。主要功能与实验 3 类似，通过命令行输入执行各种程序，增加了时钟中断和键盘中断，并通过中断的方式去调用四个用户程序。由于实验 4 设置了 int 9 键盘中断，需要一些方法来处理一下，使得键盘在需要的时候能够正常读入。

本次实验主要实现的功能有：

1. 编写时钟中断，在屏幕右下角的位置显示‘无敌风火轮’，引导程序将内核引导后就会一直存在，以稳定的速度旋转，直到关闭虚拟机。
2. 编写键盘中断，即自定义了一个 int9 中断，对键盘输入做出反应，并且敲击键盘时会根据次数显示出不同的字符串内容。
3. 将用户程序调用通过中断机制来实现，int 33，34，35，36 分别调用四个不同的用户程序。
4. 对 2 中键盘中断做了一些改动，在修改硬件中断前将原来的中断程序代码段以及入口地址保存，以便在需要的时候（读命令行的时候）恢复中断，否则会导致 int16 号中断失效，无法从键盘读入。

四、 程序设计

引导程序采用实验二的形式，修改内核的数据段地址和内存地址，跳转进内核，即可实现引导出内核程序，内核程序大概占用 5 个扇区，在引导程序里直接读了 10 个扇区。

中断程序的实现全部编写在 kliba.asm 这个库文件中，被内核程序调用，时钟中断在老师给的样例代码 ts.asm 的基础上实现，加入了一些计数器轮换着显示不同的内容，达到实验要求，由于是写在内核中实现，需要将代码改成 16 位的格式，还需要在前后压栈出栈增加对寄存器的保护。

键盘中断通过自定义 int9 来实现，刚开始没有注意太多，直接仿照时钟中断的实现方法来进行，结果发现需要额外增加一个 in al,60 指令，将键盘的输入读到 al 寄存器，其中 60h 应该代表的是 x86 的 port60h，键盘端口，否则之后又会遇到各种奇奇怪怪的问题。

其他几个中断的功能是实现调用用户程序，个人的做法是直接将调用 4 个程序的汇编代码封成一个汇编的方法（实现机制为读磁盘并加载到内存的指定位置），在 int 中断里直接 call 这些方法即可。需要注意的是，这里要对原来环境的上下文进行保护，只要将寄存器压栈出栈即可。另外还需要注意开关中断，即 CLI,STI 两条指令，否则可能会导致在用中断调用用户程序时，不能触发键盘和时钟中断（这里不知道没有设 STI 的话会怎么样，据室友表示，会导致时钟中断和键盘中断都不被 CPU 相应，不能出现转动的风火轮以及键盘中断效果）

最后对原 int9 中断进行了一个保存，将原来的中断程序入口和 cs 段保存在 200h 和 202h 这个位置（这个想法是参考 https://blog.csdn.net/csdn_blog_lcl/article/details/54926726 这篇博客的，本来自己的想法是直接保存在某几个寄存器里，但是好像不可行，会出问题，可能是寄存器不容易保护，所以就放弃自己的想法了），在执行用户程序前将 int9 设为自定义的中断程序，在回到主菜单前将原 int9 中断恢复。这样可以实现，在用户程序中，敲击键盘是会在屏幕上显示出一些有趣的信息，而在程序运行结束，返回主菜单后，可以通过命令行（与实验 3 一样），继续执行想要执行的功能。

五、主要代码

引导程序与实验 3 完全一致

内核程序（汇编部分）

有了实验 3 的基础，这次实验变得容易很多了，在内核中主要实现几个中断程序即可，由于之前调用用户程序就是采用内核代码封装成函数来给 C 调用的方式来是实现的，所以这次改成用中断来实现程序调用功能变得很容易，只要在程序前后保存好寄存器，然后直接调用即可。

时钟中断：

这个主要是根据老师给的 ts.asm 来修改实现，处理要增加计数器来轮流显示不同的字符以外，还需要将程序改成 16 位风格的语法，另外还需要在程序前后对重要的寄存器进行保护，以免发生意外。

```
    ; 时钟中断处理程序
    Timer:
        push dx
        push ax
        push es

        mov ax,0B800h       ; 文本窗口显存起始地址
        mov es,ax       ; GS = B800h

        ;mov ah,0Fh     ; 0000：黑底、1111：亮白字（默认值为 07h）
        ;mov al,'|'         ; AL = 显示字符值（默认值为 20h=空格符）
        ;mov [es:((80*12+39)*2)],ax    ; 屏幕第 24 行，第 79 列

        dec byte ptr [tcount]      ; 递减计数变量
        jnz endTimer           ; >0：跳转

        dec byte ptr [icount]

        cmp byte ptr [icount],0
        jz t0
        cmp byte ptr [icount],1
        jz t1
        cmp byte ptr [icount],2
        jz t2
        cmp byte ptr [icount],3
        jz t3

        t0:
            mov byte ptr es:[((80*24+78)*2)],'|'
            mov byte ptr [icount],icnt
            jmp t4
        t1:
            mov byte ptr es:[((80*24+78)*2)],92
            jmp t4
        t2:
            mov byte ptr [es:((80*24+78)*2)],'-'   ; 屏幕第 24 行，第 79 列
            jmp t4
        t3:
            mov byte ptr [es:((80*24+78)*2)],'/'   ; 屏幕第 24 行，第 79 列
            jmp t4
        t4:
            mov byte ptr [tcount],tdelay       ; 重置计数变量=初值 delay
    endTimer:
        mov al,20h          ; AL = EOI
        out 20h,al          ; 发送 EOI 到主 8529A
        out 0A0h,al         ; 发送 EOI 到从 8529A

        pop es
        pop ax
        pop dx
        iret            ; 从中断返回
```

键盘中断：

按照时钟中断的思路来写就可以，其实和普通的汇编程序也差不多，不过最后需要向相应的端口读取、发送一些信息，并用 iret 命令从中断返回即可。用来计数器来实现根据输入的次数显示不同的字符串内容，同样需要记得对寄存器进行保护。这里由于每次显示的信息长度不同，如果在显示了长的字符串后再显示短的字符串，会有残留，就用了一段空格 MessageCLS 来覆盖旧的字符，达到局部清屏的效果，代码如下：

```
;*******************************************************
;键盘中断                                               *
;*******************************************************
Message: db 'OUCH!OUCH!'
Message2: db 'My name is Van'
```

```
Message3: db 'I am an performance artist'
MessageCLS: db '                          '

maxCnt equ 6
keycnt db maxCnt

KeyInt:
    ; 一些上下文的保存以及初始化
    push dx
    push ax
    push es
    push bx
    push cx
    push bp
    push ds

    mov ax,cs
    mov ds,ax
    mov es,ax

    dec byte ptr[keycnt]
    cmp  byte ptr[keycnt],4
    jz DisplayStr1
    cmp byte ptr[keycnt],2
    jz DisplayStr2
    cmp keycnt,0
    jz DisplayStr3

    ; 清屏
    mov bp, offset MessageCLS
    ;mov bp,ax  ;ES:BP = 串地址
    mov cx,26   ;CX=串长度
    mov ax,01301h   ;AH = 13, AL = 01h
    mov bx,000ch
    mov dl,15
    mov dh,20
    int 10h

    jmp show      ;如果不需要改显示的内容，就直接退出中断

    DisplayStr1:
    mov bp, offset Message
    ;mov bp,ax  ;ES:BP = 串地址
    mov cx,10   ;CX=串长度
    jmp show

    DisplayStr2:
    mov bp, offset Message2
    mov cx,14
    jmp show

    DisplayStr3:
    mov bp, offset Message3
    mov cx,26
    mov byte ptr[keycnt],6
    jmp show

    show:
```

```
    mov ax,01301h  ;AH = 13, AL = 01h
    mov bx,000ch
    mov dl,15
    mov dh,20
    int 10h
    ;ret
    endKey:
    in al,60h           ; 从 60h 端口当中读取出扫描码
    mov al,20h          ; AL = EOI
    out 20h,al          ; 发送 EOI 到主 8529A
    out 0A0h,al         ; 发送 EOI 到从 8529A

    pop ds
    pop bp
    pop cx
    pop bx
    pop es
    pop ax
    pop dx

    iret            ; 从中断返回
```

对 int9 的设置、保存以及恢复：

由于自定义了 int9 中断后，会导致 int16 中断失效，从而导致之前的 getChar 函数无效，这样会使不能在命令行输入，为了避免这个问题，需要对原来的 int9 进行一些保存和恢复的操作，具体的思路是在修改之前，将原 int9 中断程序的入口以及代码段保存在内存的固定位置，在需要的时候从那个位置读取数据恢复即可。代码如下：

```
;*******************************************************
;尝试保存、恢复 int9 硬件中断
;*******************************************************

save_int9:
    push ax
    push es
    xor ax,ax
    mov es,ax
    cli
    push es:[24h]
    pop es:[200h]
    push es:[26h]
    pop es:[202h]
    sti
    pop es
    pop ax
    ret

set_old_int9:
    push ax
    push es
    xor ax,ax
    mov es,ax
    cli
    mov ax,[es:200h]
    mov word ptr es:[24h],ax

    mov ax,[es:202h]
    mov word ptr es:[26h],ax
    sti
    pop es
    pop ax
    ret

public _set_my_int9
    _set_my_int9 proc
```

```
        push ax
        push es
        xor ax,ax
        mov es,ax
        cli

        mov word ptr [es:24h],offset KeyInt    ; 设置时钟中断向量的偏移地址
        mov word ptr [es:26h],cs       ; 设置时钟中断向量的段地址=CS

        sti
        pop es
        pop ax
        ret
_set_my_int9 endp
```

通过中断来调用用户程序（以调用 pro1 为例）：

这个功能实现比较简单，不过需要 sti，cli 两条指令的使用，要让用户程序在运行时开中断，让键盘和时钟中断可以中断运行程序的中断。

```
myInt33:
    ;保存上下文，初始化寄存器信息

    sti
    push dx
    push ax
    push es
    push bx
    push cx
    push bp
    push ds

    call _RunProm1

    mov al,20h          ; AL = EOI
    out 20h,al          ; 发送 EOI 到主 8529A
    out 0A0h,al         ; 发送 EOI 到从 8529A

    pop ds
    pop bp
    pop cx
    pop bx
    pop es
    pop ax
    pop dx

    iret

; 给 C 提供调用的函数，通过 33 号中断来调用用户程序
public _runPro1ByInt
    _runPro1ByInt proc

        push ds
        push es

        int 33

        pop es
        pop ds
        ret
_runPro1ByInt endp
```

其他内容：

在内核程序中设置相关的中断。其中在设置 int9 之前要调用 save_int9 保存原来的键盘中断

```
    mov word ptr [es:20h],offset Timer ; 设置时钟中断向量的偏移地址
    mov word ptr [es:22h],cs       ; 设置时钟中断向量的段地址=CS

    call save_int9
```

```
mov word ptr [es:84h],offset myInt33 ; 33*4 = 132 = 84h
mov word ptr [es:86h],cs            ; 33 号中断
mov word ptr [es:88h],offset myInt34 ; 34*4 = 136 = 88h
mov word ptr [es:8ah],cs
mov word ptr [es:8ch],offset myInt35
mov word ptr [es:8eh],cs
mov word ptr [es:90h],offset myInt36
mov word ptr [es:92h],cs
```

在调用用户程序的汇编程序中(RunPro#)，需要在前后设置自定义 int9 中断和恢复原 int9 中断通过 call _set_my_int9、call set_old_int9 这两条指令来实现即可。

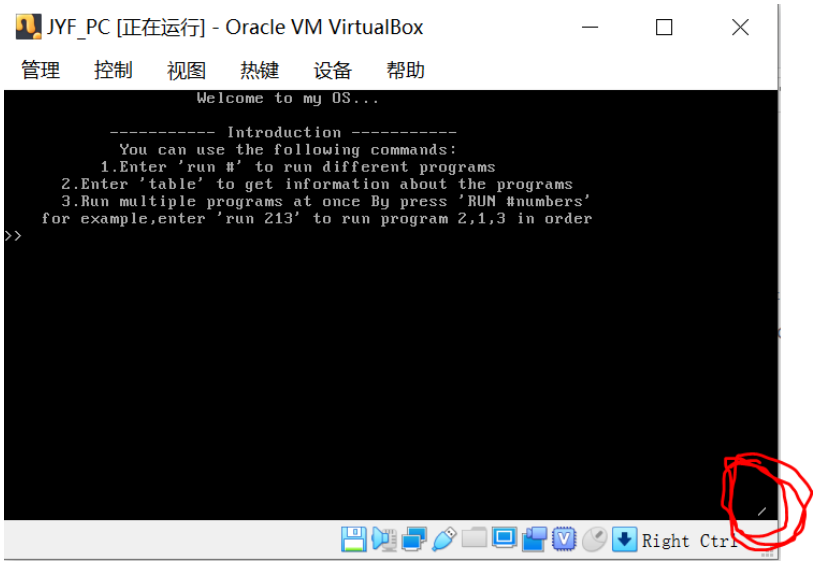
内核程序（C 语言部分）

这里基本上没有什么改动，实验三要求的是运行 3 个用户程序，而实验 4 要求运行 4 个，所以把命令行匹配的内容修改一下，增加一条 run 4 命令即可。

【实验过程】

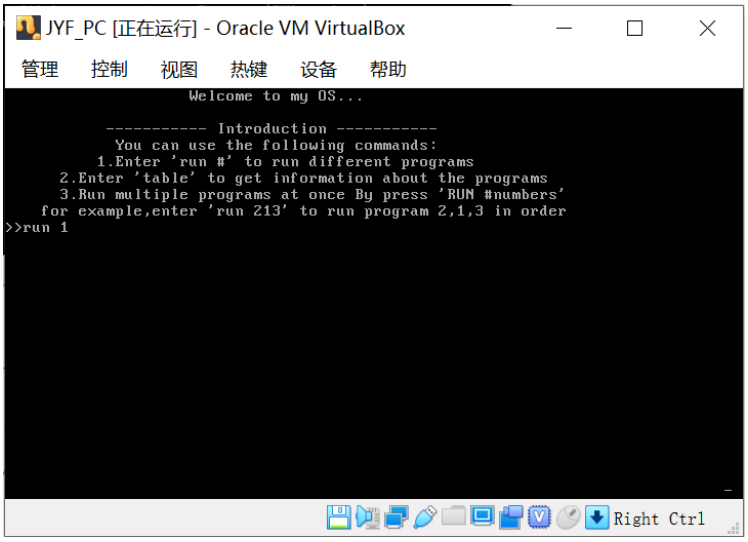
功能展示

运行虚拟机，界面与实验 3 一致，右下角多了一个无敌风火轮（由于不是很明显，用红圈标出来了）：



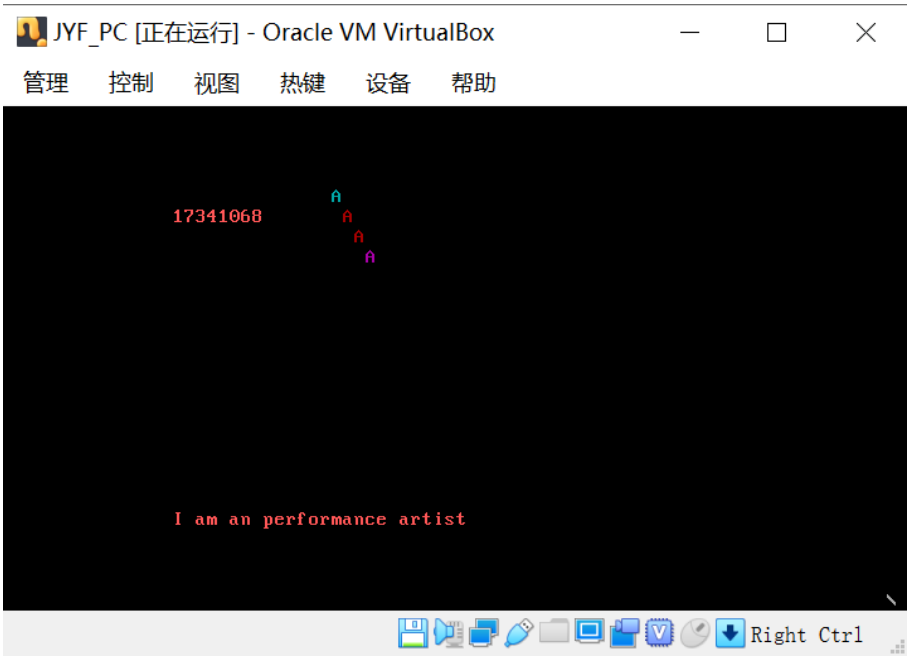
与实验 3 一样，可以输入 run #或者 table 来实现不同的功能，在主菜单界面，由于没有设置自定义的键盘中断，可以输入命令，与实验 3 的效果一致。

运行程序后，自定义的键盘中断将在用户程序运行期间被设置，在退出之后被重置为系统自带的 int9，让键盘可以正常输入。在运行期间，右下角风火轮一直在转（太小了，不是很明显），而键盘有输入后会显示不同的信息在下方，以 run 1 为例，实验截图如下：


JYF_PC [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Welcome to my OS...
---------- Introduction ----------
You can use the following commands:
1.Enter 'run #' to run different programs
2.Enter 'table' to get information about the programs
3.Run multiple programs at once By press 'RUN #numbers'
for example,enter 'run 213' to run program 2,1,3 in order
>>run 1
Right Ctrl


JYF_PC [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
17341068_
A
A
A
A
OUCH!OUCH!
Right Ctrl


JYF_PC [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
17341068_
A
A
A
A
My name is Van
Right Ctrl



用户程序结束后，跳出提示信息，任意键返回主菜单，恢复键盘输入的功能。

程序 2、3、4 与实验 2 的用户程序功能一致，稍加修改即可编译复制到软盘中。具体效果可使用虚拟机加载附件中的 myos.img 文件（vbox 下测试通过），输入 run 1234 即可看到相应的效果。

【实验总结】

有了前几次实验的踩坑经历，加上这次实验要求实现的功能比较单一容易，这次实验相比以前显得容易了不少。

实现无敌风火轮的时钟中断程序，本质上就是在屏幕固定位置轮流显示一些字符，这个在前几次实验中已经实现过很多类似的功能了，只需要记得保护寄存器，在最后加上一些中断程序的特有的操作就可以了。键盘中断的实现遇到了一些问题，主要的坑在于最后需要一条 in al,60h 指令读端口。不过由于之前的用户程序都是在用 nams 编译，而这次这些中断程序写在内核中，是 16 位的 86 汇编，语法上有一些不一样。中断调用用户程序的实现也比较简单，直接在中断程序里调用相关的函数即可。

在一开始本想着直接在内核菜单程序里，轮流调用四个中断加载用户程序就好了，但是考虑到之前花了不少时间做命令行的功能，感觉不用的话挺可惜的…遂打算让自定义键盘中断只在程序运行期间实现，回到主菜单后恢复系统自带的

中断，恢复键盘读入的功能。这里花了一些时间，主要是没有什么想法，一开始尝试在寄存器中保存中断程序的入口地址和代码段数据，但是发现不能实现，可能是因为常用寄存器太容易被使用、破坏，要想实现寄存器的保护也比较麻烦，于是在一篇博客里（博客地址见本篇报告‘四、程序设计’部分）找到了另一种办法，就是把相关数据保存在内存的固定地址，需要恢复的时候，从内存读取数据，保存到 int9 的中断向量中即可。

最后实现的功能相当于在实验 3 的基础上，添加了一个时钟中断（无敌风火轮）、只在用户程序中实现自定义的键盘中断以及使用中断的方式进行程序调用。

本次实验的感受就是，寄存器的保护很重要，由于一开始发现不少问题是寄存器预期值变化导致的，后来就干脆在所有过程前后，都将那几个常用的寄存器(es,ds,bp,dx 等等)push pop 一次，避免了这一类错误。另外还需要注意，在调用用户程序的中断程序中，需要 sti 指令开中断，让自定义的 int9 和时钟中断可以打断当前的程序。