实验四：具有中断处理的内核

学院名称：	数据科学与计算机学院
专业（班级）：	18计科教学3班
学生姓名:	夏一溥
学号:	18340178
时间:	2020 年 5 月 29 日
实验四：	具有中断处理的内核

实验内容

- [x] 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"|','/','\', (无敌风火轮)，适当控制显示速度。
- [x] 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。
- [x] 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!”。

实验过程

实验环境:

- 操作系统： Ubuntu 18.04.4 LTS
- 汇编编译器： NASM
- c语言编译器： GCC

设计思路:

Part 1： 实现时钟中断的响应处理程序

该中断响应处理程序要求在屏幕24行79列轮流显示'|','/','\',病控制显示速度。

于是基本思路是预先在汇编程序中申请一段地址依次储存要显示的字符:

```
mark db '\|/'
```

然后规定偏移量，偏移量每自增到mark的长度即3时置零:

```
offset dw 0
cmp byte[offset], 3  ;与mark 长度 3 比较，等于时置零
    jne end
    mov byte[offset], 0
end:
```

为了让风火轮转动起来并能控制其速度，我们不能让每次中断调用的时候都使偏移量自增，否则显示效果很不好，所以需要设置记数器，与stoneN.asm中延长显示时间类似，每调用一定次数的时钟中断才改变一次偏移量：

```
dec byte [count]          ;记数器
    jnz end
    ...
end:
    mov al,20h                  ; AL = EOI
    out 20h,al                  ; 发送EOI到主8529A
    out 0A0h,al                 ; 发送EOI到从8529A
    iret
```

此时控制风火轮转动速度可以通过此记数器的标准值实现，我实现的内核中暂时定位每3次时钟调用offset加1.

Part 2： 实现键盘中断响应处理程序

该响应处理程序要求的功能很简单，即显示OUCH！OUCH！在特定屏幕位置。

为了实现字符串的显示，我们可以调用系统中断int 10h:

```
mov ax, cs            ; 置其他段寄存器值与CS相同
    mov ds, ax            ; 数据段
    mov bp, msg     ; 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, len ; CX = 串长
    mov ax, 1300h         ; AH = 13h（功能号）、AL = 01h（光标不动）
    mov bx, 0007h         ; 页号为0(BH = 0) 黑底白字(BL = 07h)
    mov dh, 20            ; 行号=0
    mov dl, 40            ; 列号=0
    int 10h
```

并且为了方便用户观察到显示的字符，我们需要同Part 1 中一样增加中断响应时间：

```
loop:
    dec word[cnt]         ; 递减计数变量
    jnz loop               ; >0：跳转;
    mov word[cnt],num1
    dec word[dcnt]        ; 递减计数变量
    jnz loop
    mov word[cnt],num1
    mov word[dcnt],dnum1
```

之后再用空白符覆盖原本OUCH！OUCH！的位置并返回。

同时另外，特别说明的一点是在DOS上可运行的此程序，如果作为对应于int 09h的中断处理程序还需要有所改动，这一点会在下面详细说。

Part 3： 将中断响应程序写入内核

时钟中断

时钟中断号为08h，时钟中断由计算机按时钟频率自行调用。所以要实现风火轮的旋转，只需要能够让上述Part1实现的中断响应程序替代原有的09h时钟中断即可，这里采用重写中断向量表的方法实现：

```
%macro WriteIVT 2
    pusha
    mov ax,0
    mov es,ax
    mov ax,%1
    mov bx,4
    mul bx
    mov si,ax
    mov ax,%2
    mov [es:si],ax ; offset
    add si,2
    mov ax,cs
    mov [es:si],ax
    popa
%endmacro
```

由于要多次写/交换中断向量表，采用宏实现。

8086中一个中断向量占据4字节空间，所以将中断号*4得到其地址，再将我们的中断处理程序写入此地址即可。

时钟中断的写入我选择在监控程序中实现：

```
_start:
    WriteIVT 08h, Timer
    call dword OS
    jmp _start
```

如此，我们进入操作系统的时候已经完成了时钟中断响应程序的加载，从而实现风火轮的加载与旋转。

键盘中断

键盘中断的写入与时钟中断相似，不过相对复杂一些，其中有几点需要注意：

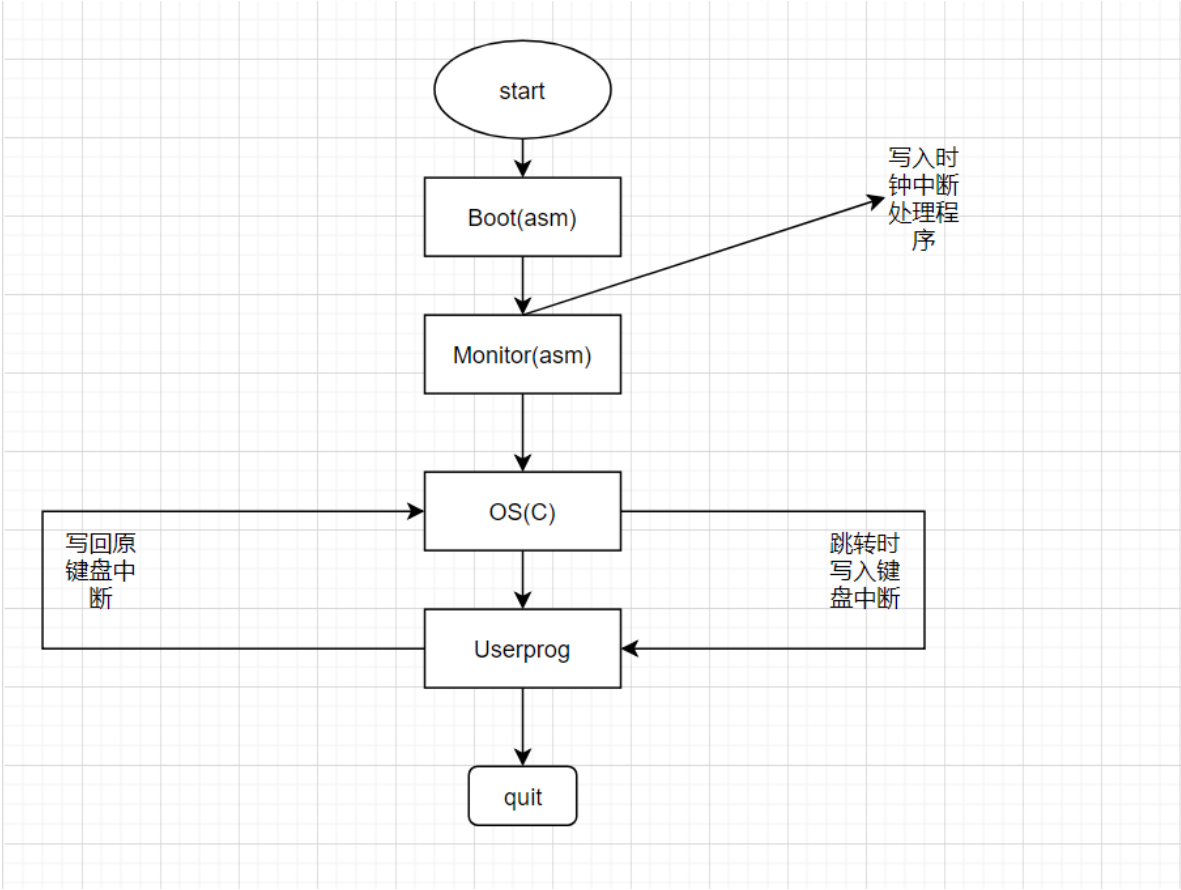
1. 键盘中断原处理程序并不为空，将处理程序写入后如果不调用原中断会导致键盘输入的阻塞，所以为了在用户程序中能正常使用键盘中断，在中断处理程序中需要手动调用原09h中断。

```
; 字符串显示
int 20h              ; 原来的BIOS int 09h
; 中断返回
```

2. 为了让该中断只在用户程序运行时写入，我选择在加载用户程序时加载，返回内核前一步还原原中断。

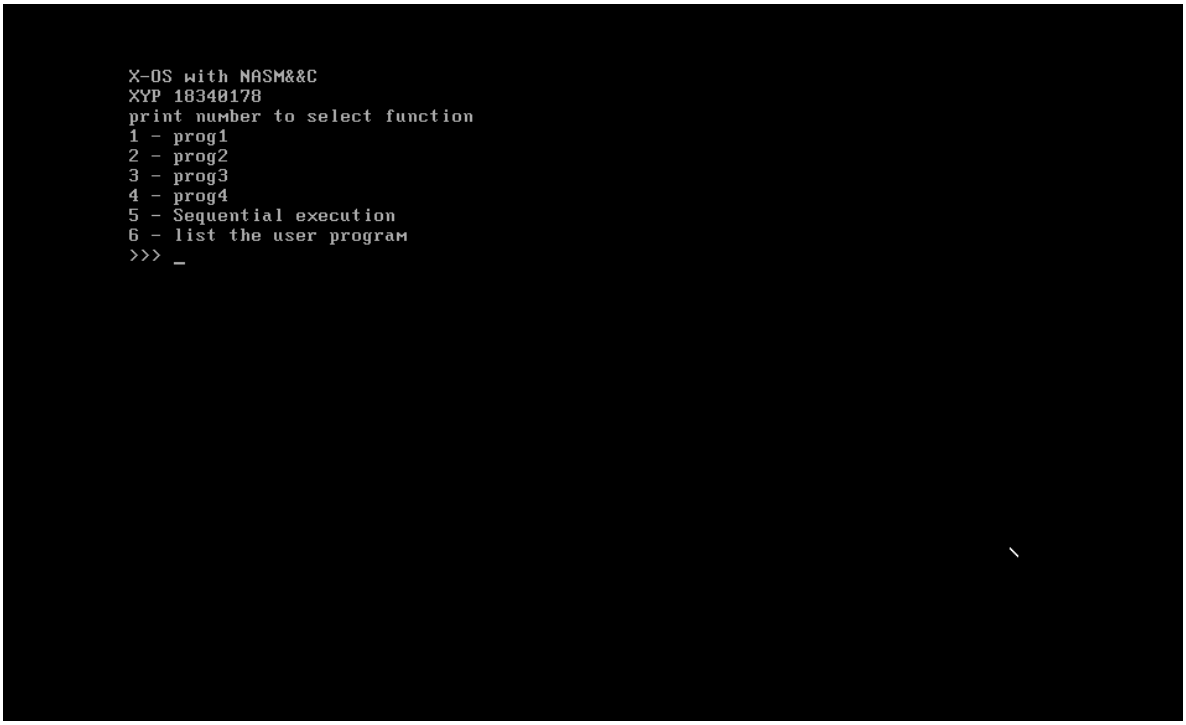
```
 SwitIVT 20h, 09h
    WriteIVT 09h, keyboard
    jmp [bp+4]
    ip_new:
    SwitIVT 09h, 20h
    popa
    retf
```

示意图：

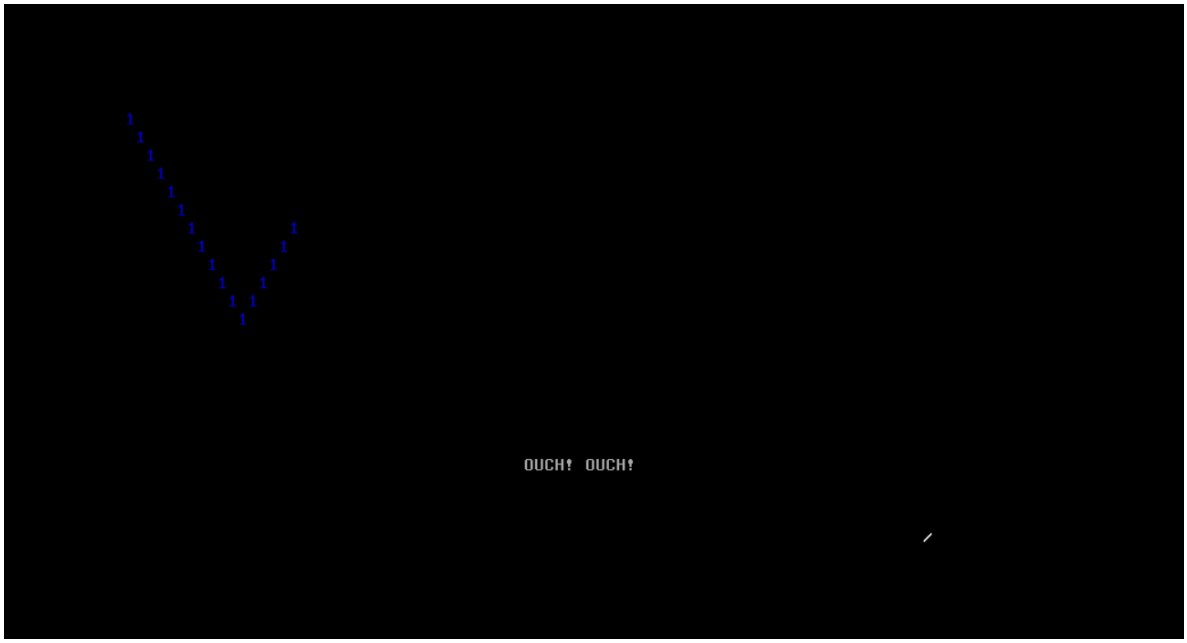


实验结果：

进入监控程序：



用户程序并有按键：



具体完成结果附有视频 ex.mp4

纠错过程：

1. binary output format does not support external references

在对.o文件写入.bin文件时有如此报错，发现是编译格式的问题，指定格式为 -elf32，并预先生成空的二进制文件.bin再写入。

2. 在键盘中断写入后用户程序失去对键盘输入的反应

即将键盘中断原响应程序覆盖后，失去了对键盘输入的处理与输出，所以需要在自己实现的中断响应处理程序附件原中断处理程序。其解决办法在设计思路-->Part 2 中已详细说明。

3. 在实现2的修改后，依然会有在用户程序运行途中卡住的情况

发现是在中断调用的过程中改变了ds，es的值，发现是我最开始只进行了pusha与popa的处理，但实际上pusha/popa只保护了ax、cx、dx、bx、sp、bp、si、di ,而不能保护ds，es的值，需要手动入栈保护用户程序的寄存器的值。

实验总结

本次实验建立在上一次的实验基础之上困难少了很多。值得注意的是BIOS的原中断可以通过自己实现中断响应处理程序来完成自己需要的效果。而基于时间中断处理程序的重写，我想大概可以实现在多个程序段间快速切换（需要保存当前各寄存器状态），从而实现程序运行的并行。而同时也有疑问，就是中断处理程序写在中断向量表中，他是惟一的入口。比如时钟中断，于是会引出一个问题，就是当你有多个程序需要配和时钟中断的调用，会有入口的冲突，这种冲突如何解决我还没有找到合适的解决方法，唯一能想到的只是在一个处理程序中实现所有功能（可这显然只是治标不治本）。

Reference

1. BIOS 中断向量表 https://blog.csdn.net/jackailson/article/details/82469746
2. 中断处理程序 https://blog.csdn.net/qq_42146775/article/details/86704153
3. 《03实验课.ppt》