实验四：中断处理与异步事件响应

1 实验题目

学习中断处理，用系统时钟中断实现“-\|/”循环输出的无敌风火轮、用键盘响应中断实现在用户程序执行期间按键盘会在屏幕上提示信息。把它们加进内核。

具体来说，共有两项：

1. 设计一个汇编程序，利用系统时钟中断，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，以方便观察效果
2. 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，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!”

2 实验目的

1. 理解 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。

3 实验要求

1. 掌握 x86 汇编语言对时钟中断的响应处理编程方法；
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，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。

4 实验方案

工具与环境：Windows（x86_64），gcc，NASM，ld 2.25.1，bochs 虚拟机（1MB 内存、1.44MB 软驱）

中断的相关原理技术在《x86汇编语言：从实模式到保护模式》第九章[1]有详述。本实验中，需要重新编写系统时钟中断（`int 08h`）、键盘响应中断（`int 09h`）的处理程序，并修改中断向量使其指向这两个程序。

（注：时钟中断有系统时钟中断（`int 08h`，约每 1/18 秒发生一次）、实时时钟中断（`int 70h`，每秒发生一次），本次实验使用前者。）

重难点是中断程序的编写规范。

实验过程部分将以流畅的方式叙述实验细节，更多试错过程写在实验总结部分。

5 实验过程

5.1 中断原理学习

详细原理需阅读《x86汇编语言：从实模式到保护模式》第九章[1]，此处仅简述。

中断，就是由于种种原因（硬件信号、运算错误、显式调用处理程序）使得 CPU 执行完当前指令后，响应这个事件，去执行相应的中断处理程序。

例如，计算机硬件每隔一定时间会发出一个时钟中断信号，这是外部硬件中断；如果 CPU 发生了除 0 等错误，会发出信号，这是内部中断；还可以显式调用中断处理程序，如之前的实验调用了 BIOS 中断服务，这是软中断。

实模式下可使用最多 256 个中断（编号 0 至 255），每个中断的处理程序的偏移地址、段地址（共 4 字节）依次从内存 0 开始存放，共需 1 KB。这段内存称为“中断向量表”，如图：

```
<bochs:52> xp /96bx 0x0
[bochs]:
0x0000000000000000 <bogus+       0>:    0x53    0xff    0x00    0xf0    0x53    0xff    0x00    0xf0
0x0000000000000008 <bogus+       8>:    0x53    0xff    0x00    0xf0    0x53    0xff    0x00    0xf0
0x0000000000000010 <bogus+      16>:    0x53    0xff    0x00    0xf0    0x53    0xff    0x00    0xf0
0x0000000000000018 <bogus+      24>:    0x53    0xff    0x00    0xf0    0x53    0xff    0x00    0xf0
0x0000000000000020 <bogus+      32>:    0xa5    0xfe    0x00    0xf0    0x87    0xe9    0x00    0xf0
0x0000000000000028 <bogus+      40>:    0xe6    0xe9    0x00    0xf0    0xe6    0xe9    0x00    0xf0
0x0000000000000030 <bogus+      48>:    0xe6    0xe9    0x00    0xf0    0xe6    0xe9    0x00    0xf0
0x0000000000000038 <bogus+      56>:    0x57    0xef    0x00    0xf0    0xe6    0xe9    0x00    0xf0
0x0000000000000040 <bogus+      64>:    0x52    0x01    0x00    0xc0    0x4d    0xf8    0x00    0xf0
0x0000000000000048 <bogus+      72>:    0x41    0xf8    0x00    0xf0    0xfe    0xe3    0x00    0xf0
0x0000000000000050 <bogus+      80>:    0x39    0xe7    0x00    0xf0    0x59    0xf8    0x00    0xf0
0x0000000000000058 <bogus+      88>:    0x2e    0xe8    0x00    0xf0    0xd2    0xef    0x00    0xf0
```

图 1: 初始中断向量表

当中断发生时（信号到达、调用 `int` 语句），依次压栈 `flags,cs,ip`，然后根据中断号×4 得到中断向量（即其在表中的位置），再到中断向量表中获得相应的处理程序地址，跳转执行。中断处理程序的最后一句指令为 `iret`，功能是依次退栈 `ip,cs,flags`，这样就返回原处继续执行原来的代码了。

初始时大多数中断所指向的处理程序是没有意义的，例如前 8 个指向的 `0xf000:0xff53` 实际上只是一条 `iret` 语句。也有些是 BIOS 写好的，例如 `int 16h`。

本实验需要编程处理的是两个外部硬件中断（系统时钟中断、键盘响应中断），它们属于可屏蔽中断，信号会先到达 8259 中断控制芯片，再到达 CPU。因此 CPU 响应它们需要满足两个条件：8259 没有拦着它们

（如果一个中断被送去 CPU 处理了，但还没有收到 CPU 的结束信号，那么芯片就会拦下其他中断）、CPU 的 IF 标志位为 1（表示允许响应非屏蔽中断）。但如果显式调用处理程序，则不需要这些条件。

对于程序员来说，主要是做两件事：1、编写中断处理程序；2、修改中断向量。

5.2 无敌风火轮

将其编写为引导程序格式，以方便测试。

首先是编写中断处理程序。建立一个字符串 wheel='-\|/'，和一个下标 wheelcnt，表示在 24 行 79 列输出 wheel[wheelcnt]。每 100 次触发系统时钟中断，wheelcnt 向前循环推进一位。（出于 bochs 的原因，系统时钟中断并没有“每 1/18 秒发生一次”，因此采用了效果较好的 100 次延迟）

```
HotWheel:
        inc word [delaycnt]
        cmp word [delaycnt], 100
        jnz End
        mov word [delaycnt], 0                  ; reset delaycnt
        inc byte [wheelcnt]
        and byte [wheelcnt], 3                  ; wheelcnt = wheelcnt mod 4
        movzx bx, byte [wheelcnt]               ; cl = wheel[wheelcnt]
        mov byte cl, [ds:wheel+bx]
        mov ax, 0xb800                          ; print wheel
        mov es, ax
        mov byte [es:(23*80+79)*2], cl
End:
        mov al, 0x20                            ; tell 8259 that we finished
        out 0x20, al
        iret                                    ; interrupt return

Data:
        wheel db '-\|/'
        wheelcnt db 0
        delaycnt dw 0
```

需要注意的是，处理结束后，要向 8259 芯片发送 EOI 信号（0x20），告诉它这个中断处理完了，可以放行其他中断了。实际上 8259 分为主芯片和从芯片，而系统时钟中断信号来自主芯片，因此只需向主芯片（端口号 0x20）发送 EOI 即可。

接下来编写主程序，主程序的主要任务就是修改中断向量，使 int 08h 指向 HotWheel：

```
resetInt08h:                                    ; reset int 08h
        mov ax, 0
        mov es, ax
        cli
        mov word [es:0x20], HotWheel
        mov word [es:0x22], cs
        sti
        jmp $
```

注意一个细节，正在修改中断向量时，不应响应中断，否则程序将会去到不可控的地方。因此，修改中断向量前用 cli 指令置 IF 为0，修改后用 sti 指令置 IF 为 1。

不过作为引导程序，cli 指令是不必要的，因为初始时 IF 就为 0。但 sti 是必要的，否则后面无法响应中断。

完整代码见 src\hotwheel\hotwheel.asm。运行虚拟机，效果如下：（助教可运行 hotwheel.bxrc 查看动态效果）

可以看到右下角的无敌风火轮转起来了，该步骤成功。

5.3 内核修改

现需要把风火轮加入内核，以及实现在用户程序执行期间响应键盘中断。然后再完善一下之前实验未完善的地方。

5.3.1 键盘中断响应

用户程序执行期间，若触碰键盘，则在屏幕中间（10 行 35 列）显示字符串“OUCH!OUCH!”。

中断处理程序伪代码如下：

```
int09h_seg_addr equ 0xf000
int09h_offset_addr equ 0xe987

Kbhit_OUCH:
        push ds, es, ax, bx, edi                        ; protection
        pushf                                           ; to clear buffle
        call int09h_seg_addr:int09h_offset_addr
        mov ds, cs                                      ; reset ds=cs, es=0xb800
        mov es, 0xb800
        mov bx, 10
        For1:
                ; here loop bx from 10 to 1 to print ouch[bx-1] on (10,34+bx)
        call delay                                      ; delay about 0.5s
        mov bx, 10
        For2:
                ; here loop bx from 10 to 1 to print ' ' on (10,34+bx)
        mov al, 0x20                                    ; end
        out 0x20, al
        pop edi, bx, ax, es, ds
        iret

section .data
        ouch db 'ouch!ouch!'
```

要点如下：

1、现在不是引导程序了，而是由用户程序突然进入此过程，因此必须保护现场。风火轮同理。

2、原本的 int 09h 带有类似于清空缓冲区的功能，此功能为必需，否则将会无限触发 int 09h 或导致以后的 int 16h 读入出错。因此在执行自己的程序前，应调用原本的 int 09h，地址可在初始中断向量表查到。

5.3.2 修改中断向量

内核一开始便修改 int 08h，并打开 IF 标志：

```
_start:
        mov ax, 0
        mov gs, ax
        mov word [gs:0x20], HotWheel            ; reset int 08h
        mov word [gs:0x22], cs
        sti
        push 0
        call _ker
        ret
```

而键盘响应中断不同，只在用户程序执行期间输出 ouch，因此在调用用户程序前修改 int 09h，用户程序结束后恢复原样：

```
_load_client:
        ...
        int 13h                                 ; load client to memory
        cli                                     ; reset int 09h
        mov ax, 0
        mov gs, ax
        mov word [gs:0x24], Kbhit_OUCH
        mov word [gs:0x26], cs
        sti
        ...
        jmp client_seg_addr: 0x100
end_client:
        cli                                     ; recover int 09h
        mov ax, 0
        mov gs, ax
        mov word [gs:0x24], int09h_offset_addr
        mov word [gs:0x26], int09h_seg_addr
        sti
        ...
```

5.3.3 其他更新

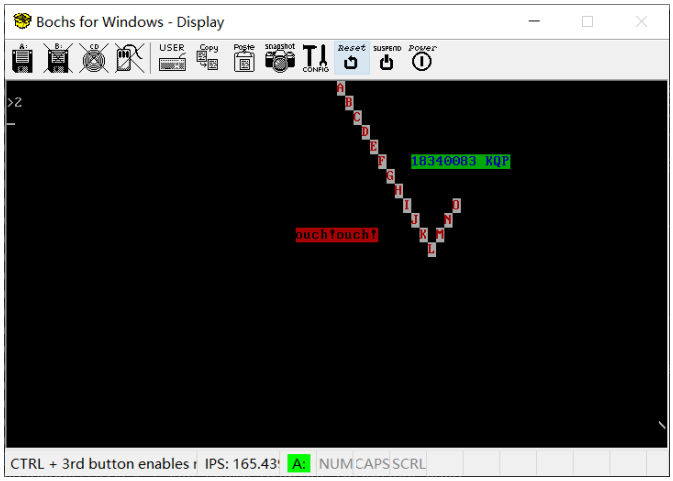
规范用户程序的进入与退出 将原本的“内核 call 到用户程序、用户程序 retf 返回”改成较为正式的 DOS 方式，跳转前在用户栈中依次压入 cs,end_client,0，在用户程序地址开头（空出来的 100h）写入 retf 语句（0xcb），然后 jmp client_seg_addr:100h 过去，用户程序的 retf 改为 ret。这样，用户程序结束后 ret 到其开头，再执行 retf 返回操作系统。

命令输入支持退格 突然发现退格是有 ascii 码的，于是就可以实现退格了。整个输入缓冲区视为一个栈，退格即为退栈与输出“退格、空格、退格”。

代码结构调整 把 C 中的读入字符串放入 myprintf.c 中，并更名为 myOS_IO.c。

5.3.4 测试效果

4 个用户程序依然为之前的射字母。测试效果如下：（助教可运行 myOS.bxrc 查看动态效果）



可以看到，输入命令正常执行射字母程序，在此期间按了键盘，显示了“OUCH!OUCH!”，并且右下角的风火轮始终在转，该步骤成功。

6 实验总结

这次实验相对就简单了，参考书写得非常详细，读完之后基本上明白所有步骤了。并且参考书上还有很多编程好习惯细节，例如修改中断向量前要 `cli` 关闭中断。

6.1 试错

1. 一开始写好风火轮，却是转不起来的。调试发现如果手动调用 `int 08h` 却是可以转的，那么真相只有一个，系统默认 `IF` 是关的，需要自己 `sti` 打开。

2. 做好风火轮之后，想要把它做成一个 COM 格式用户程序，功能为“开启风火轮”，即重置 `int 08h` 然后离开。但是发现这样之后进入用户程序就会发生错误，风火轮也不转了。后来醒悟过来，用户程序都把风火轮程序都覆盖了，我还搁这 `int 08h` 呢。

3. 把风火轮写进内核之后，会有弹射字母位置错乱的问题。调试发现，是没有保护现场。

4. 把 ouch 写进内核之后，会有结束后无法继续读入等问题。调试发现，如果一开始就使用我的 `int 09h`，那么输入命令阶段只能输入一个字符，然后 `int 16h` 死循环。那么真相只有一个，键盘与内存的输入缓冲之间有一些正常的交流操作我没有做。调用原本的 `int 09h`，问题解决。

6.2 未解决问题

射字母程序一开始就输出 ouch，据同学称是因为键盘的按下、弹起均会触发中断。但我觉得这个解释不完全，因为调试得到，只有长时间的程序（如射字母）才会这样，短的程序（如简单的 a+b）是不会触发的，触发基本都在 `delay` 的时候，这与时间有关。

目前比较好的解释是，外设太慢了，键盘最后一下弹起被送到 CPU 的时候，正值用户程序 `delay`。该说法有待验证，bochs 的时间本身比较玄学。

参考文献

[1] 李忠等,x86汇编语言：从实模式到保护模式,电子工业出版社,第九章 中断和动态时钟显示

[2] 闫俊等,键盘中断INT09H与INT16H的分析和应用,http://www.doc88.com/p-2798102426737.html