School of Data and Computer Science, Sun Yat-sen University

《操作系统实验》

实验四：中断机制编程技术

17341190 叶盛源

2019 年 4 月 5 日

Contents

中山大學 SUN YAT-SEN UNIVERSITY

操作系统实验报告

1 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，实现时钟中断处理程序

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

2 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。

3 实验方案:

3.1 实验环境：

1) 实验运行环境：Windows10

2) 虚拟机软件：VMware Function和DOSBox

3) TCC+Tasm+TLink 混合编译链接

4) NASM 编译器

3.2 实验工具：

1) 汇编语言：NASM、TASM

2) 文本编辑器：VScode、notepad++

3) 软盘操作工具：WinHex

3.3 实验原理和思想：

1) 中断技术介绍：

中断(interrupt)是指对处理器正常处理过程的打断。中断与异常一样，都是在程序执行过程中的强制性转移，转移到相应的处理程序。中断包含包含两种，分别是软中断、硬中断和异常/程序中断：

i) 硬中断（外部中断）——由外部（主要是外设[即I/O设备]）的请求引起的中断，如时钟中断，I/O中断，硬件故障中断等

ii) 软中断（内部中断）——由指令的执行引起的中断，如中断指令（软中断int n、溢出中断into、中断返回iret、单步中断TF=1）

iii) 异常/程序中断（指令执行结果产生，如溢出、除0、非法指令、越界）

2) 修改中断的思想

i) 中断向量表：

X86计算机在启动的时候，会在内存的低位区（地址为0 1023[3FFH]，1KB）创建含256个中断向量的表IVT （每个向量[地址]占4个字节，格式为：16位段值:16位偏移值）如下图figure1。CPU根据指令的中断号找到中断向量表的位置，然后取出表中中断服务程序的入口CS:IP并跳转。在中断服务子程序结束后，我们需要使用指令iret来返回主程序。

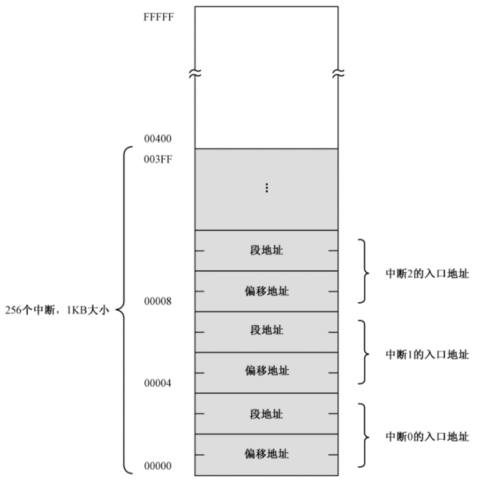

Figure 1: 中断向量表

ii) 中断开启和屏蔽：

当我们进入中断程序的时候，为了防止其他中断再进入，我们可以选择屏蔽中断。使用指令CLI就可以修改处理器状态字中的中断相关的对应位置，实现中断屏蔽。如果我们要开启中断的时候就可以使用STI指令。当我们编写的硬件中断程序结束的时候，我们需要给8258芯片发送EOI信号来将ISR对应位置清零来结束中断服务程序。

3) 键盘中断和时间中断

i) **键盘中断** 键盘中断是INT 09号中断是由于键盘按键输入触发的。当按键被按下时，kbc检测到并发出中断请求信号。这个中断包括通过60port读取操作扫描码，并转换成ASCII码，并将它放进键盘缓冲区。但要注意，如果修改了9号中断，就无法使用16号中断，因为16号中断是从键盘缓冲区读取数据，所以用户程序需要由原来的按键退出改成延时退出。

ii) **时钟中断** 时钟中断是20h号中断。每隔一段很短的时间就会触发一次，这样可以实现多进程的并发执行，利用这个特性我们可以实现出旋转的“风火轮”效果。

4) 模块结构

在上一次实验中，我们将我们的操作系统内核分为了以下4个模块，在本次实验中，我们只需要修改kernal模块的内容，增加一些中断服务子程序，并再monitor模块中将他们的CS:IP地址加入到中断向量表中即可：

i) **loader模块：** 内容类似实验二中的引导程序，用来引导内核加载进入内存并运行

ii) **monitor模块：** 类似实验二中的监控程序，但核心的代码改为使用C语言完成，这里只用来通过调用C语言实现的函数。这里还可以加载文件属性的表单和在启动系统前完成一些批处理指令。新增：重新填写中断向量表09、20h、33、34、35、36的中断服务程序入口地址

iii) **kernal模块：** 用C语言和汇编语言实现的函数，通过在C语言中调用汇编语言中写的I/O操作的基本库函数来实现内核主要功能。新增：自己编写时间中断20h，键盘中断09h和33、34、35、36号用户子程序

iv) **user模块：** 用户程序和一些批处理指令**list模块：** 用户程序的文件表单，用来实现简单的文件系统

3.4 实验过程：

1) 编写时间中断：

在syscall.asm文件中新增一个标号Timer作为这个子程序的入口。因为时间中断频率太快，我们需要使用延时的函数(每五次执行一次)来防止运行速度过快导致看不清效果。”风火轮”有旋转过程中有四种状态，我们设定一个计数器从1开始计数，它的值分别为1，2，3，4的时候进入四种状态分支并显示此时状态的字符。当状态大于4时就回到1，反复循环，就形成了一个时钟中断的烽火轮效果，代码和实验效果如下：

```
;执行五次中断才运行一次风火轮，防止运行速度过快
    dec byte ptr es:[ccount]
    jz  DUMP
    jmp end1
DUMP:
    mov byte ptr es:[ccount],ddelay
    inc byte ptr es:[state]
    cmp byte ptr es:[state],1
    jz  state1
    cmp byte ptr es:[state],2
    jz  state2
    cmp byte ptr es:[state],3
    jz  state3
    cmp byte ptr es:[state],4
    jz  state4
```

Figure 2: 延时函数和计数器

```
state1:
    mov bp,offset char1
    jmp show_state
state2:
    mov bp,offset char2
    jmp show_state
state3:
    mov bp,offset char3
    jmp show_state
state4:
    mov bp,offset char4
    mov byte ptr es:[state],0  ;当到达第四个状态之后将state归0
    jmp show_state
show_state:
    mov ah,13h
    mov al,0
    mov bl,0Fh
    mov bh,0
    mov dh,24
    mov dl,78
    mov cx,1
    int 10h
```

Figure 3: 风火轮四种状态

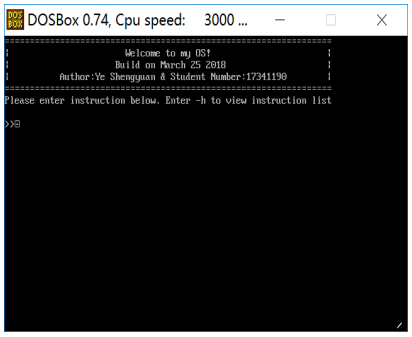

Figure 4: 风火轮效果1

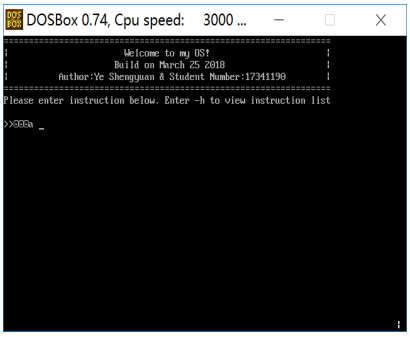

Figure 5: 风火轮效果2

2) 编写键盘中断

当我们敲击键盘的时候，就会触发键盘中断，我们可以直接用int10h的中断指令输出字符串OUCH!OUCH!但键盘中断的时间比较短，所以我们也需要一个延时函数。同时在字符显示结束后，我们还需要清空屏幕对应位置

的字符串，用空白填充即可，最后发送EOI信号给20h端口，再用iret返回，就结束中断程序了，如下图：

```
mov ah,13h                          ; 功能号
mov al,0                            ; 光标返回起始位置
mov bl,0Fh                          ; 0000：黑底、1111：亮白字
mov bh,0                            ;
mov dh,15                           ; 行
mov dl,45                           ; 列
mov cx,11                           ; 串长
mov bp,offset me
int 10h

;延时   dos int 15h功能调用   cx 和 dx的值大小影响延时的时间
push dx
push cx
push ax
mov ah,86h
mov cx,02h
mov dx,9999h
int 15h
pop ax
pop cx
pop dx
```

Figure 6: 输出字符串并延时

```
;清除该ouch位置的字符串
mov ah,6
mov al,0
mov ch,15
mov cl,45
mov dh,15
mov dl,55
mov bh,7
int 10H

in al,60h
mov al,20h                          ; AL = EOI
out 20h,al                          ; 发送EOI到主8529A
out 0A0h,al                         ; 发送EOI到从8529A
```

Figure 7: 清除字符发送EOI

但因为int09号中断的更改会影响我们的输入功能，所以在用户子程序中我们需要采用延时退出的方式。而且为了保证在结束子程序后，主程序还能正常输入字符，我们在调用runProm函数开头要对原来的键盘09号中断做一个保存，在结束的地方再将它恢复即可，如图：

```
;保护键盘中断
xor ax, ax
mov es, ax
mov bp,offset int_09_saved
mov word ptr ax,es:[36]
mov word ptr [bp],ax
mov word ptr ax,es:[38]
mov word ptr [bp+2],ax
;修改键盘中断
mov word ptr es:[9*4], offset int_09
mov word ptr es:[9*4+2], cs
```

Figure 8: 保护键盘中断

```
;恢复键盘中断
xor ax, ax
mov es, ax
mov bp,offset int_09_saved
mov word ptr ax,[bp]
mov word ptr es:[36],ax
mov word ptr ax,[bp+2]
mov word ptr es:[38],ax

pop  bp
pop  es
pop  ds
pop  ax
ret  ;中断用iret
```

Figure 9: 恢复键盘中断

四个用户程序运行效果如下：

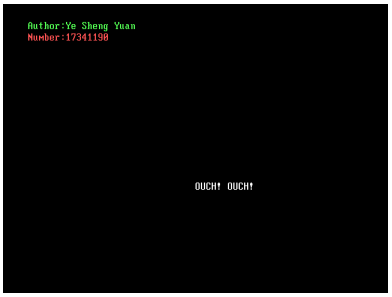

Figure 10: 用户程序1

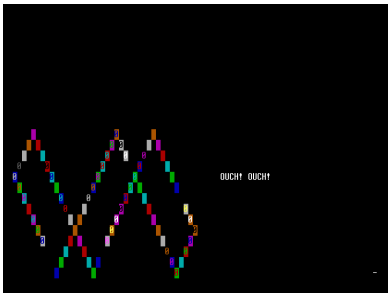

Figure 11: 用户程序2

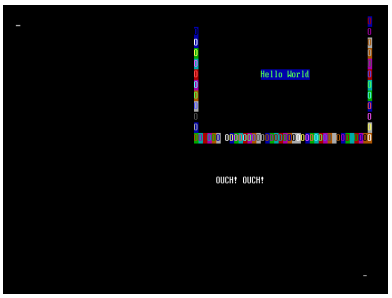

Figure 12: 用户程序3

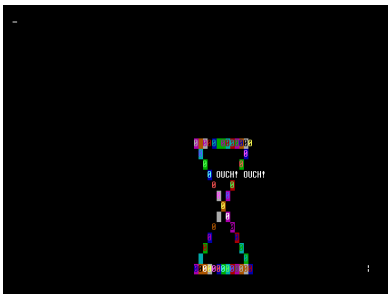

Figure 13: 用户程序4

3) 编写四个用户自定义中断

我们需要将int33、int 34、int 35、int 36四个中断修改为我们的四个用户子程序，但因为在上一个实验中实现的文件系统中要求我们的用户子程序只有要用时候才通过传递参数扇区号来调进内存。

```
int_33:
    mov ax,8
    push ax
    call _RunProm
    pop ax
    iret
int_34:
    mov ax,9
    push ax
    call _RunProm
    pop ax
    iret
```

Figure 14: 自定义中断33 34

```
int_35:
    mov ax,10
    push ax
    call _RunProm
    pop ax
    iret
int_36:
    mov ax,11
    push ax
    call _RunProm
    pop ax
    iret
```

Figure 15: 自定义中断35 36

因此我们不能用时钟中断键盘中断的方法来实现。我们通过传递扇区号的参数，再调用runProm函数，来调用我们自己编写的四个用户程序，如上图figure 14和15所示。

4) 新增用户程序：

我们要编写一个用户程序使用中断来调用我们的用户子程序。但要注意的是，因为之前我们都把所有的用户子程序加载到内存中的同一个位置运行，如果这个程序是作为用户程序调用，他使用中断调用其他用户程序就有覆盖它自身的风险，导致程序出错，因此我们需要给它单独写一个函数存入内存中一个特定的区域，我选择了0DC00H的内存空间，如下图：

```
org 0DC00H
Program:
    int 33
    ; int 34
    ; int 35
    ; int 36

Quit:
    ret

times 512 - ($ - $$) db 0 ;将前510字节不是0就填0
```

Figure 16: 新的用户程序

我们在内核中也要新增调用这个用户程序的指令"-p6",这样我们可以用指令来调用这个程序，这个程序在使用中断33 34 35 36来分别启动我们之前写的4个用户子程序。

5) 修改中断向量表

所有的中断子程序都已经写完了，现在们需要修改中断向量表来让我们可以通过中断进入我们的子程序中，我在monitor.asm文件中在调用cmain函数前使用语句修改了向量表。我们通过中断向量号计算出内存的位置，再依次插入IP和CS的值完成修改，如下图：

```
mov word ptr es:[20h],offset Timer
mov word ptr es:[22h],cs

mov word ptr es:[33*4], offset int_33
mov word ptr es:[33*4+2], cs

mov word ptr es:[34*4], offset int_34
mov word ptr es:[34*4+2], cs

mov word ptr es:[35*4], offset int_35
mov word ptr es:[35*4+2], cs

mov word ptr es:[36*4], offset int_36
mov word ptr es:[36*4+2], cs
```

Figure 17: 修改中断向量表

6) 用软盘启动裸机:

将内核用tcc+tasm+tlink编译链接成.com文件，再把5个用户程序、list、batch和loader用nasm编译形成bin文件，加入到1.44MB软盘的合适扇区中并运行程序，虚拟机运行如下图：

Figure 18: 开机界面

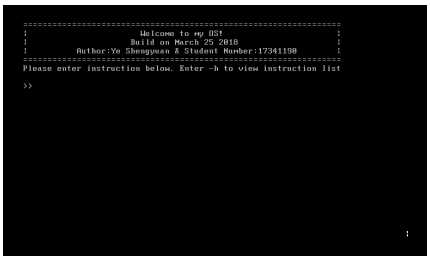

Figure 19: 操作系统界面

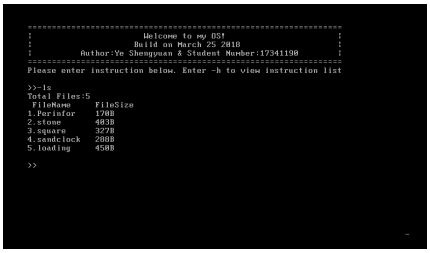

Figure 20: 文件列表界面

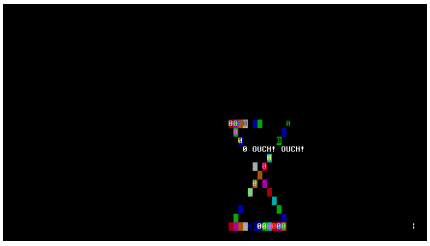

Figure 21: 用中断开启的用户程序

4 实验心得和总结：

本次实验也是基于上一次实验的，操作系统内核那些重要部分都是依靠实验三中编写的程序。但难度在于，我们以前是依靠输入指令或者是批处理来调用我们的用户程序，但现在是我们需要将他们编写成中断服务子程序,并修改中断向量表，这样我们就能通过int指令来调用我们的用户程序了。

本次实验要实现三个问题，第一个是修改时钟中断。因为有给时间中断的模版，所以我直接使用了模版代码进行修改，但一直都无法显示字符。这个卡了很久，花了很多时间后才发现，因为没有使用push pop保护寄存器，导致段寄存器的值出现了问题，这个问题真的很难发现，而且也无法debug，所以后面编写的时候我都很细心的做了push和pop的保护工作，而且每次写完一个程序就立刻测试，防止bug的累积。同时时钟中断因为太快，所以要设置延时函数，也是尝试了多次才选择了5次显示一次这个作为计数器的初值。一开始做这个风火轮觉得只是完成任务，后面竟然发现它可以拿来判断操作系统是否死机了，如果停转可能就需要重启了，反而对我有所帮助。

第二个要做的是修改键盘中断，做之前我以为这个和时间中断一样，直接把中断向量表改一改就能用了，后来才发现没这么简单，因为9号中断改了，这样我的用户程序也必须改成延时退出，同时我为了在操作系统中还能输入指令，必须在用户程序结束前将键盘中断再改回来。思路到是比较清晰，但需要花时间去实现，中间也遇到了一些小bug，比如调试的时候发现一进入用户程序风火轮就不转了，后面也发现是程序里有一些死循环的问题。

第三个要做的是修改原来的33 34 35 36号中断，一开始想直接把用户程序填进来，但发现没必要，可以在中断服务程序中再调用用户程序的函数就能实现，大大提高了代码的复用性。在写新的用户程序的时候，本来以为直接int连着调用4次就完事，结果只调用一个就死循环了，一直出不去，而且因为这些程序都是实验三中用过的，找不出毛病，当时十分的急躁。后面静下来好好想想，

才发现新写的程序其实也是用户程序，如果把他们都加载到同一片内存很容易就相互覆盖然后出现问题，一开始还想在调用完一个用户程序后再把新的用户程序加载回去，发现太麻烦，不如直接单独给新的用户程序一个自己的内存空间，就省事很多了。

除了完成这三个工作，我还遇到很多问题，比如说因为内核的变大，我需要更多的软盘空间来填装我的内核，所以我的用户程序就被迫往后移动位置，首先在WinHex中移动操作就已经十分的头疼，后面当我把扇区移到18后面的时候就一直无法加载，后来在操作系统群里问了同学才明白，竟然是扇区号到了18之后就要归1，然后磁头号加一，这也是以前没想到的，一直都是直接照老师的代码使用也没有深究，可见学习还是需要刨根问底，打好基础。同时因为内核增加，我想将内核存在另外一个内存地址，但一改就有问题，打开程序就是一片黑，加什么调试测试语句都没有效果，推测是跳转错误了。后来研究了半天才明白，以前内核保存在0A100H的地方，但我改了之后后面三位不是100了，可在com文件中，有一句org 0100h语句，同学告诉我说是要对齐才能成功跳转。

虽说这次实现最终还是完成了老师安排的任务，也顺利的实现了键盘中断和时钟中断，但还有很多的问题需要改善，代码并不是很精简美观，还有很多地方可以提高。在几周的汇编练习中，因为没有下载汇编语言的调试工具bochs，全程都是人脑调试，靠自己的观察和思考去调试，所以虽然遇到了很多困难，花了很多时间，但还是收获满满，汇编编程水平也大幅度提高了，希望可以再接再厉，期待下一个实验!

5 参考文献:

1) nasm手册

2) https://blog.csdn.net/longintchar/article/details/79511747

3) https://blog.csdn.net/cielozhang/article/details/6171783/

4) https://blog.csdn.net/lulipeng_cpp/article/details/8161982

5) https://www.cnblogs.com/alwaysking/p/7789282.html

6) https://blog.csdn.net/cielozhang/article/details/6171783

7) https://blog.csdn.net/lulipeng_cpp/article/details/8161982 https://stackoverflow.com/questions/12882342/override-default-int-9h