



《操作系统实验》 实验报告

(实验四)

学 院 名 称 : 数据科学与计算机学院

专业 (班级) : 17 级计算机类教务 3 班

学 生 姓 名 : 姚森舰

学 号 : 17341189

时 间 : 2019 年 4 月 4 日

一. 实验题目

- (1) 设计一个汇编程序，实现时钟中断处理程序；
- (2) 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；
- (3) 编写键盘中断响应程序，完成相关的操作。

二. 实验目的

学习中断机制知识，掌握中断处理程序设计的要求。

三. 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。

四. 实验方案

实验环境：Windows10 +VMware

实验工具：NASM + Winhex + Tasm + Tcc + Dosbox + Notepad++

大概流程：用Notepad++编写C语言程序和汇编程序，用nasm把实验二中的用户程序和引导程序编译成二进制代码，在Dosbox下用TCC编译C代码，用TASM编译内核的部分汇编代码，在用Tlink将编译好的C和汇编代码link成.com文件。通过VMware创建带软盘的虚拟机，利用winhex将编译好的引导程序二进制代码写到虚拟机的软盘的第1个扇区，将文件数据写到第2个扇区，将内核代码放到3~12扇区，然后将用户程序放到第19~28扇区，然后在Vmware上运行测试虚拟机和程序。

大概思路：由引导程序引入到内核，内核里面最基础的输入输出、显示、读数据部分等等由汇编实现，再用 C 语言利用这些函数做一些复杂一些的工作，比如输出字符串等

等。再用 C 语言实现通过用户的输入，调用用户程序或者一些其他指令，实现类似命令行的功能。

中断部分，主要是将自己的中断的 offset 和 segment 作为中断入口地址，给到中断向量表中对应中断号乘 4 的 4 个字节处。另外应在调用用户程序之前改对应的中断，并且将原来的中断的入口地址保存到另外的地方，保护中断，以便在调用完用户程序后恢复原来的中断，使得系统正常运行。

关键代码及部分程序解释：

扇区安排：

- 1：引导程序
- 2：程序信息及批处理文件内容
- 3~18：内核，虽然内核还没有这么大
- 19~28：用户程序，每个用户程序占2个扇区

对于系统的部分功能已在实验三的报道中叙述了，这里不再赘述，这里主要写关于中断的部分。有关改中断和装填中断向量表的代码都在setint.asm文件中。

```
states db 0
char1 db '|'
char2 db '/'
char3 db '\'
delay0 equ 4
delay_count db delay0

Timer:
```

Timer标号处是在老师给的代码的基础上改的，引入变量states，有1,2,3三种取值，分别表示三个字符，在4次调用时钟后变成对应的下一个状态的符号并显示，实现风火轮的旋转。

```
push [es:35*4]
pop [es:27*4]
push [es:35*4+2]
pop [es:27*4+2]

push [es:36*4]
pop [es:28*4]
push [es:36*4+2]
pop [es:28*4+2]

mov word ptr [es:9*4], offset myint_09
mov ax,cs
mov word ptr [es:9*4+2],ax

mov word ptr [es:33*4], offset myint_33
mov ax,cs
mov word ptr [es:33*4+2],ax
```

在函数setMyInt中，将原有的9,33~36号中断的入口地址保存到保留的24~28号中断入口地址，在使用完后恢复(24~28都是保留中断，可自己修改)。然后就是将自己的中断的offset和segment填入对应中断的入口地址处。

```
xor ax,ax
mov es,ax
push [es:24*4]
pop [es:9*4]
push [es:24*4+2]
pop [es:9*4+2]

push [es:36*4]
pop [es:28*4]
push [es:36*4+2]
pop [es:28*4+2]

push [es:35*4]
pop [es:27*4]
push [es:35*4+2]
pop [es:27*4+2]

push [es:34*4]
pop [es:26*4]
push [es:34*4+2]
pop [es:26*4+2]
```

用户程序执行完后，同构resetInt()函数恢复原来的中断。

```

mov ah,13h
mov al,0
mov bl,0Eh
mov bh,0
mov dh,24
mov dl,50
mov cx,11
mov bp,offset ouch
int 10h

```

```

;清除该ouch位置的字符串
mov ah,6
mov al,0
mov ch,24
mov cl,50
mov dh,24
mov dl,60
mov bh,7
int 10H

```

这是myint_9的内容，修改9号中断，利用10h号中断，在myint_9中显示“ouch!”，并在适当延时后清除字符串。

```

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

```

向8259A发送EOI，表示中断结束。

```

myint_33:
    mov ax,1
    push ax
    call _loadp
    pop ax
    iret

```

在自己的33~36号中断中调用用户程序，下面是用户程序5，直接使用修改好的中断：

```

push ds
push es
push si
push bp

xor ax,ax                ; AX = 0    程序加载到0000: 100h才能正确执行
mov ax,cs                ; 开机这些寄存器都为0
mov es,ax                ; ES = 0
mov ds,ax                ; DS = CS
mov es,ax                ; ES = CS

int 33

int 34

int 35

int 36

pop bp
pop si
pop es
pop ds

```

```
setMyInt();

if(strlen(recv)==1){

for(i = 1; i < len0; i++){

resetInt();
```

setMyInt()函数即修改所要求的中断，然后运行用户程序，再调用resetInt()将中断还原。

```
mov ch,0
mov cl,byte ptr[bp+2+2+2+2+2+2+2] ; int\IP\cx\ax\dx\bp\bx\ex
mov bx,8800h ;8800
mov ax,400h
mul cx
add bx,ax
add cl,cl ; 1*2-1 2*2-1 3*2-1
dec cl ; 1 3 5

mov ah,2 ; 功能号
mov al,2 ; 扇区数
mov dl,0 ; 驱动器号 ; 软盘为0，硬盘和U盘为80H
mov dh,1 ; 磁头号 ; 起始编号为0
mov ch,0 ; 柱面号 ; 起始编号为0
int 13H ; 调用读磁盘BIOS的13h功能

call bx
```

因为磁盘布局发生了改变，加载用户程序处做了适当修改。每个用户程序占2个扇区，并且从第19扇区开始，从代码上看就是磁头号变为1，扇区数变为2，起始扇区cl分别为1,3,5,7,9, …， 同时每个用户程序加载到内存不同位置，所以最终图中bx的值，也就是用户程序加载到内存的位置，需要几次运算才能得到。

五. 实验过程与思想

1. 安装 VMware 并按要求创建一个无操作系统的裸机。



安装 Dosbox 并将 TCC, TASM, Tlink 等 exe 文件粘贴到对应文件

gc.bat	2014/10/8 13:18	Windows 批处理...	1 KB
gcc.exe	2013/10/6 1:17	应用程序	1,777 KB
ld.bat	2014/10/8 12:51	Windows 批处理...	1 KB
na.bat	2013/12/9 14:55	Windows 批处理...	1 KB
nasm.exe	2013/1/2 17:06	应用程序	726 KB
setting.bat	2014/10/8 12:31	Windows 批处理...	1 KB
startCmd.bat	2013/5/9 12:58	Windows 批处理...	1 KB
ta.bat	2013/5/18 15:04	Windows 批处理...	1 KB
TASM.EXE	1996/2/21 5:00	应用程序	133 KB
tc.bat	2013/12/10 10:51	Windows 批处理...	1 KB
TCC.EXE	2012/5/8 12:05	应用程序	177 KB
TLINK.EXE	2012/5/8 12:05	应用程序	22 KB

2. 编写汇编程序:

关键代码见实验方案, 具体代码见代码文件, 用户程序在实验三的基础上修改了 org, 使得用户程序被加载到内存的不同位置。

3. 编译:

将引导程序, 用户程序和文件信息编译成二进制:

```
C:\Users\user\AppData\Local\bin\NASM>nasm os4_1.asm -o 1
C:\Users\user\AppData\Local\bin\NASM>nasm os4_2.asm -o 2
C:\Users\user\AppData\Local\bin\NASM>nasm os4_3.asm -o 3
C:\Users\user\AppData\Local\bin\NASM>nasm os4_4.asm -o 4
C:\Users\user\AppData\Local\bin\NASM>nasm os4_5.asm -o 5
```

将二进制文件利用 Winhex 写到对应扇区:

000001E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00				
000001F0	00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA				U
00000200	31 2E 63 6F 6D 00 00 00 31 39 00 00 32 00 00 00	1.ccm	19	2	
00000210	32 2E 63 6F 6D 00 00 00 32 31 00 00 32 00 00 00	2.ccm	21	2	
00000220	33 2E 63 6F 6D 00 00 00 32 33 00 00 32 00 00 00	3.ccm	23	2	
00000230	34 2E 63 6F 6D 00 00 00 32 35 00 00 32 00 00 00	4.ccm	25	2	
00000240	72 20 31 20 32 20 34 20 31 00 00 00 00 00 00 00	r 1 2 4 1			
00000250	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00				
00000260	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00				

将内核的 C 代码和汇编代码在 Dosbox 下利用 tcc, tasm, tlink 编译链接:

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Unable to change to: os.

C:\ASM\OS>tcc -mt -c -ok.obj kernel.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:

    Available memory 438500

C:\ASM\OS>tcc -mt -c -ok.obj kernel.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:

    Available memory 438500

C:\ASM\OS>tasm hunbian.asm h.obj
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:  hunbian.asm to h.obj
Error messages:  None
Warning messages: None
Passes:         1
Remaining memory: 466k

C:\ASM\OS>
```

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:

    Available memory 438500

C:\ASM\OS>tcc -mt -c -ok.obj kernel.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:






    Available memory 438500

C:\ASM\OS>tasm hunbian.asm h.obj
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file: hunbian.asm to h.obj
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 466k

C:\ASM\OS>tlink /3 /t h.obj k.obj,test.com,,
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
C:\ASM\OS>

```

-  H.MAP
-  H.OBJ
-  hunbian.asm
-  K.OBJ
-  kernel.c

将生成的.com 文件，即内核，写到 3~18 扇区，实际上不需要这么多，但是为了以防万一就给了这么多个扇区放内核：

000003E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000003F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000400	26 C7 06 20 00 22 01 8C	C8 26 A3 22 00 8C C8 8E	&Ç " GE&f" GE
00000410	D8 8E C0 8E D0 BC 00 01	E8 22 08 EB FE 00 7C 2F	øŽÀŽĐ% è" ëþ
00000420	5C 04 50 53 51 52 55 06	1E 8C C8 8E D8 8E C0 2E	\ PSQRU GEŽøŽÀ
00000430	FE 0E 21 01 75 4F 2E C6	06 21 01 04 2E FE 06 1D	p ! uO.Æ ! .þ
00000440	01 2E 80 3E 1D 01 01 74	10 2E 80 3E 1D 01 02 74	.e> t .e>
00000450	0E 2E 80 3E 1D 01 03 74	0C BD 1E 01 EB 13 90 BD	.e> t % ë :
00000F50	20 20 20 20 20 20 20 20	20 20 20 20 20 20 20 20	
00000F60	20 20 20 20 7C 0A 0D 00	20 2D 2D 2D 2D 2D 2D 2D	-----
00000F70	2D 2D 2D 2D 2D 2D 2D 2D	2D 2D 2D 2D 2D 2D 2D 2D	-----
00000F80	2D 2D 2D 2D 2D 2D 2D 2D	2D 2D 2D 2D 2D 2D 2D 2D	-----
00000F90	2D 2D 2D 2D 2D 2D 2D 2D	2D 2D 2D 2D 2D 2D 2D 2D	-----
00000FA0	2D 2D 2D 2D 2D 2D 2D 2D	2D 2D 2D 2D 2D 2D 2D 2D	-----
00000FB0	2D 2D 0A 0D 0A 0D 00 00	00 00 00 00 00 00 00 00	--
00000FC0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000FD0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

运行测试:

有关系统指令部分在实验三中已详细描述，这里只叙述实验四的内容：

```
-----
:                               Welcome to YaoOS                               :
-----

Command help:
-l: to get some help imformation.
-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help imformation.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.
user $:
```

可见右下角的风火轮在旋转。



运行用户程序时，风火轮也在旋转。



执行用户程序时，按键盘时跳出“ouch! ouch!”



执行用户程序时，按键盘时跳出“ouch! ouch!”

```
Command help:
-l: to get some help imformation.
-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help imformation.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.
user $: r5 _
```

调用用户程序 5，通过在用户程序 5 中调用 33~36 号中断调用用户程序 1~4，执行用户程序时，按键盘时跳出“ouch! ouch!”，通过运行测试可见，以下是用户程序 5 的内容，其实就只需要写 int 33~36 即可。

```
org 09c00h
```

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

```
xor ax,ax
mov ax,cs
mov es,ax
mov ds,ax
mov es,ax
```

```
int 33
```

```
int 34
```

```
int 35
```

```
int 36
```

```
pop bp
pop si
pop es
pop ds
pop dx
pop cx
pop bx
pop ax
ret
```

六. 实验心得和体会

这个实验相对来说简单一些，但也是有很多坑。由于之前计组课上用 masm 写过类似修改中断的程序，所以对整个流程也有大概的印象，通过实验，我对这个过程也更加的熟悉了。

下面就是碰到的一些问题的解决方案和感想：

1. tasm 中使用数据时常常要加 word/byte ptr。最开始把改 9 号中断的代码放到引导扇区，风火轮可以正常运行，但后来移到内核之后就出现了问题，进入系统就无法进行输入了，结果卡了很久，在所有通过寻址拿数据的地方加上了 word/byte ptr 之后程序就能运行了。因为引导程序使用 nasm 而内核部分用的是 tasm，思考后觉得这应该是 nasm 和 tasm 语法不同的地方之一，不写 ptr 不会报错，但数据不一定对，可能就会

导致程序异常。

2. 风火轮显示一次就换行。以为是代码错了，最后发现如果利用 10h 中断在 79 列显示风火轮的话，显示完后光标会后移，就换行了，然后会一行一行的上移，改成在 78 列显示或者通过把字符搬到显存对应位置显示即可。

3. 数据段的问题。遇到过风火轮在用户程序中不能转起来，原因是 ds 没有改成当前的数据段，使得定义的字符数据不能用，只需要在合适位置加上这三条指令即可：

```
mov ax, cs;      mov ds, ax;      mov es, ax
```

意思是将数据段改为当前 cs 处的数据段，明白了这个，就有种豁然开朗的感觉，之前遇到的一些问题也可以由这个来解释。

4. 扇区重新分配。因为之前的用户程序定义的字符串有点多，又加上一些语句之后，二进制代码就超过了 512B，就需要 2 个扇区，同时又要增加新的用户程序，就越过了 18 扇区，需要改磁头，不太方便，所以扇区需要重新安排。查阅资料后，将用户程序都搬到了 19 扇区以后，下面这张图就很直观明了地解释了柱面，磁头，扇区的关系。

0柱面, 0磁头, 1扇区	0
0柱面, 0磁头, 2扇区	1
.....	
0柱面, 0磁头, 18扇区	17
0柱面, 1磁头, 1扇区	18
.....	
0柱面, 1磁头, 18扇区	35
1柱面, 0磁头, 1扇区	36
.....	
1柱面, 0磁头, 18扇区	53
1柱面, 1磁头, 1扇区	54
.....	
1柱面, 1磁头, 18扇区	71
2柱面, 0磁头, 1扇区	72

5. 按键盘显示“ouch!”时，需要延迟一下，但不知为什么此时 int 15h 延时中断不可用了，在其他的用户程序中可用，也许是和更改了 int 9 有关，最终自己写了一个双重循环来延时。

6. 之前所有的用户程序都加载到了内存同一个地方，这就造成了问题，用户程序 5 中要利用中断来调用前 4 个程序，如果都加载到了同一个地方，那么调用的第一个程序运行时就会把用户程序 5 给覆盖掉，最终得不到想要的结果。所以我就修改了用户程序和 loadp() 函数，也就是加载用户程序的函数，将程序加载到不同位置。而在这其中又遇到了问题，就是 mul 指令的使用：

```
mov ch,0
mov cl,byte ptr[bp+2+2+2+2+2+2+2] ;
mov bx,8800h ;8800
mov ax,400h
mul cx
add bx,ax
add cl,cl ; 1*2-1 2*2-1 3*2-1
dec cl ; 1 3 5
```

cl 里的值就是第几个用户程序，第一个用户程序加载到 8c00h，每隔 400h 加载一个用户程序，把 400h 放到 ax，用 $cx \times ax + 8800h$ 来算用户程序加载的位置。最开始用的是 $cl \times ax$ ，结果总是有问题，无法显示定义的字符串，但其他没问题，也就是说 ds 不对，而 ds 和 cs 是同样的值，也就是说跳错了位置，翻书看到，如果用 `mul cl`，实际上算的是 $al \times cl$ ，而不是自己认为的 $ax \times cl$ ，ah 中的高位就根本没用到，改成 `mul cx`，才是 $ax \times cx$ ，才解决了问题。

总的来说，比起前面的实验，这个实验要稍微容易一些，每次踩坑都会学到一些新东西，慢慢的，前面感觉模糊的东西也清楚了起来，比如 ds 的作用等等。另外，感谢老师和助教延期了实验 DDL，让我们不用天天熬夜。

七. 参考资料

- [1] [int 13h 参数详解](#)
- [2] [8086 DS 和 ES 寄存器](#)
- [3] [扇区，柱面，磁头相关](#)
- [4] 汇编语言(第 3 版) 王爽著