《操作系统实验》

实验报告

（实验五）

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　19　日

一. 实验题目

系统调用功能实现可以用汇编与c语言混合编程

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

在内核增加一个过程作为系统调用的总入口，用以获取参数和分析功能号，再根据功能号产生分枝结构，根据系统调用号决定选择对应的分支完成相应的服务。通常每个分枝实现一种系统调用功能，简单的功能可以用汇编实现，也可以用c程序实现。

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。

不过，这次稍微改进了一下方法，采用DOS批处理的形式自动化编译过程，详细见后。

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。

由于此次实验将多个中断功能写到同一个中断号中，所以还需要 ah 寄存器作为功能

号，辅助判断具体的中断。

关键代码即及部分程序解释：

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之前的报告中叙述了，这里不再赘述，这里主要写关于中断，功能号以及新增功能部分的部分。有关改中断和装填中断向量表的代码都在setint.asm文件中。

```
 ------------------------------------------------------------------------
|                             Welcome to YaoOS                           |
 ------------------------------------------------------------------------

Command help:
-l: to get some help information.
-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 information.
-s: shut down my OS.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch commmand in the disk.
user $:
```

新增指令“s ”，即shutdown，关机指令，实现见下：

```
else if(strcmp(recv,shutdown) == 1){
    shutDown();
    }
```

C中新增指令“s”

```
public _shutDown
_shutDown proc

    call _setMyInt
    mov ah,8
    int 21h
    call _resetInt
    ret
_shutDown endp
```

```
fun8:
    mov         ax, 2001h
    mov         dx, 1004h
    out         dx,ax
    iret
```

关机操作，向 1004H 端口，写入 2001H，实现关机，右图则是在int 21h里面的8号功能，shutDown()函数中先装填中断，再调用int 21h中的8号功能，实现关机，然后恢复中断，但是没有测试执行关机的中断后是否还会返回，恢复中断不一定有意义。另外重启功能的实现也类似，只不过是向 64H 端口，写入 0FEH，因为方法都是类似的，所以只实现了关机功能用于展示。

```
myint_21:

    STI
    cmp ah,0
    jz helpJumpFun0

    cmp ah,1
    jz helpJumpFun1

    cmp ah,2
    jz helpJumpFun2

    cmp ah,3
    jz helpJumpFun3

    cmp ah,4
    jz helpJumpFun4

    cmp ah,5
    jz helpJumpFun5

    cmp ah,6
    jz helpJumpFun6

    cmp ah,7
    jz helpJumpFun7

    cmp ah,8
    jz helpJumpFun8

helpJumpFun0:
    jmp fun0

helpJumpFun1:
    jmp fun1

helpJumpFun2:
    jmp fun2
```

21号中断内部，先判断ah内容，然后跳转到对应功能处执行，由于代码长度较长，通过jz无法直接跳转，故进入helpJumpFun作为跳板辅助跳转。

```
fun5:
    call _cls
    iret

fun6:
    call _printChar
    iret

fun7:
    call _getChar
    iret

fun8:           ;shutdown
    mov        ax, 2001h
    mov        dx, 1004h
    out        dx,ax
    iret
```

Int 21h中的部分功能，0~3为用户程序5中测试用的中断，仅显示一些字符，功能号为4为实验4中的“ouch!”，也就是9号中断的改写，代码稍长，就没有截图了。

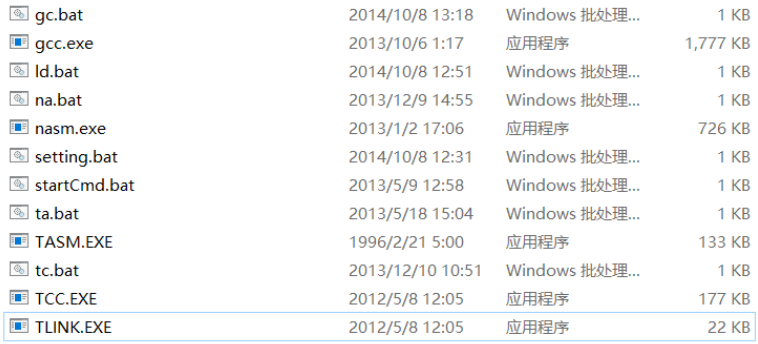
```
    mov ah,0        ;test0
    int 21h

    mov ah,1        ;test1
    int 21h

    mov ah,5        ;test5 clear screen
    int 21h

    mov ah,2        ;test2
    int 21h

    mov ah,3        ;test3
    int 21h
```

用户程序5，调用int21h，测试不同的功能号功能，至于输入输出的中断，没有专门写程序测试。

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



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。

3. 编译：

利用 DOS 批处理自动化编译，其实没有实现完全自动化，但是自动化了混编 c，nasm 编译和将二进制文件粘贴到扇区，这些操作已经能大大提高效率了：

打开 DOSBox，输入写好的批处理文件名，即可完成内核部分的混编，错误信息在对应的 txt 文件中可见：

```
cd os
del *.obj
del *.map
del *.com

tcc -mt -c -ok.obj kernel.c>tcc.txt
tasm os.asm os.obj>tasm.txt
tlink /3 /t os.obj k.obj,os.com,,>tlink.txt
```

效果如图：

```
C:\ASM\OS>tcc -mt -c -ok.obj kernel.c>tcc.txt

C:\ASM\OS>tasm os.asm os.obj>tasm.txt

C:\ASM\OS>tlink /3 /t os.obj k.obj,os.com,,>tlink.txt

C:\ASM\OS>CD..

C:\ASM>DOSBOX.BAT
C:\ASM>cd os

C:\ASM\OS>del *.obj

C:\ASM\OS>del *.map

C:\ASM\OS>del *.com

C:\ASM\OS>tcc -mt -c -ok.obj kernel.c>tcc.txt

C:\ASM\OS>tasm os.asm os.obj>tasm.txt

C:\ASM\OS>tlink /3 /t os.obj k.obj,os.com,,>tlink.txt

C:\ASM\OS>_
```

然后将生成的 os.ocm 文件复制到 nasm 所在文件夹中，在执行下面的批处理脚本，实现引导程序，用户程序的编译和将代码包含到 flp 文件中：

```
@echo off
title test
color 0
pushd %~dp0

set NASM="C:\Users\user\AppData\Local\bin\NASM\nasm.exe"

%NASM% 1.asm||goto error
%NASM% 2.asm||goto error
%NASM% 3.asm||goto error
%NASM% 4.asm||goto error
%NASM% 5.asm||goto error
%NASM% data.asm||goto error
%NASM% BootLoader.asm -o PC_5.flp||goto error

echo success
pause
exit

:error
REM del 1 2 3 4
echo error!
pause
exit
```

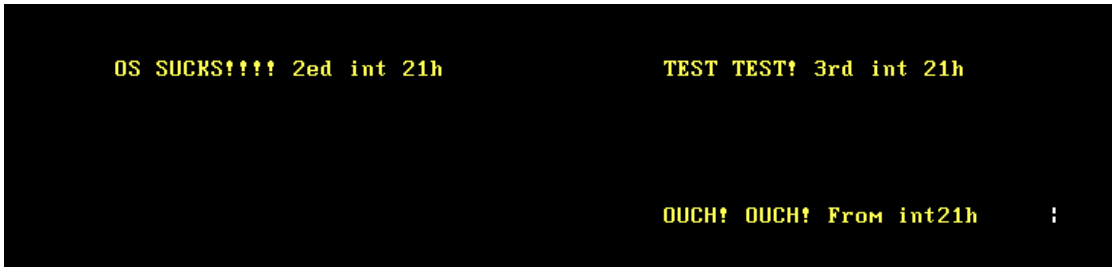
要注意的是，对扇区的安排实际是通过引导程序中的代码决定的：

```
times 510 - ($ - $$) db 0
db 0x55
db 0xaa
incbin 'data'
incbin 'OS.COM'
times 2400h - ($ - $$) db 0
incbin '1'
times 2800h - ($ - $$) db 0
incbin '2'
times 2c00h - ($ - $$) db 0
incbin '3'
times 3000h - ($ - $$) db 0
incbin '4'
times 3400h - ($ - $$) db 0
incbin '5'
times 3800h - ($ - $$) db 0
```

运行测试:

中断测试:

用户程序 5 测试时，先调用了前两个中断显示一些信息后，又调用了清屏中断功能，再测试后面的中断，同时还测试了键盘中断。



六. 实验心得和体会

这个实验相对来说简单一些，过程也比较顺利，所以花了一些时间学了一点 DOS 批处理的知识，如前面描述的，实现了编译的半自动化，大大提高了效率。

下面就是碰到的一些问题的解决方案和感想:

1. 一开始在测试中断功能号时，发现键盘中断显示“ouch!”有点奇怪，似乎有点延迟，后来发现我是在自己的 int 9 中调用了 int 21h 的 4 号功能，也就是中断嵌套了，需要用 STI 指令允许中断里面去执行其他中断才行。

```
myint_21:

    STI
    cmp ah,0
    jz helpJumpFun0
```

2. 如下图:

```
public _printf
_printf proc
        mov ax,0500h
        push ax
        int 21h        ；产生中断，
_printf end
```

老师的例子中有 push ax 的操作，这样做的话似乎更保险一些，稍微麻烦一点的是实现中断时，要从栈里面取出 ah，再做比较。如果不 push，理想情况下是可以直接用 ah 的，所以我就直接用了 mov，也可正常运行。

总的来说，比起前面的实验，这个实验要容易很多，不过学会了一点批处理的知识，并且实现了编译半自动化，也算是很大的收获了。

七. 参考资料

[1] 汇编强制关机和重启