实验五：实现系统调用

19335174施天予

一、实验题目

实现系统调用

二、实验目的

1、学习掌握PC系统的软中断指令

2、掌握操作系统内核对用户提供服务的系统调用程序设计方法

3、掌握C语言的库设计方法

4、掌握用户程序请求系统服务的方法

三、实验要求

1、了解PC系统的软中断指令的原理

2、掌握x86汇编语言软中断的响应处理编程方法

3、扩展实验四的的内核程序，增加输入输出服务的系统调用。

4、 C语言的库设计，实现putch()、getch()、printf()等基本输入输出库过程。

5、编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

四、实验内容

1、修改实验4的内核代码，先编写save()和restart()两个汇编过程，分别用于中断处理的现场保护和现场恢复，内核定义一个保护现场的数据结构，以后，处理程序的开头都调用save()保存中断现场，处理完后都用restart()恢复中断现场。

2、内核增加int 20h、int 21h和int 22h软中断的处理程序，其中，int 20h用于用户程序结束是返回内核准备接受命令的状态；int 21h用于系统调用，并实现3-5个简单系统调用功能；int22h功能未定，先实现为屏幕某处显示INT22H。

3、保留无敌风火轮显示，取消触碰键盘显示OUCH!这样功能。

4、进行C语言的库设计，实现putch()、getch()、gets()、puts()、printf()、scanf()等基本输入输出库过程，汇编产生libs.obj。

5、利用自己设计的C库libs.obj，编写一个使用这些库函数的C语言用户程序，再编译，再与libs.obj一起链接，产生COM程序。增加内核命令执行这个程序。

```
int main(){
    char ch,str[80];
    int a;
    getch(&ch);
    gets(str);
    scnf("a=%d",&a);
    putch(ch);
    puts(str);
    printint("ch=%c, a=%d, str=%s", ch, a, str);
}
```

6、编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

五、实验方案

实验环境

1、实验运行环境：Windows10

2、虚拟机软件：VirtualBox和DOSBox

3、NASM编译器

4、TCC+TASM+TLINK混合编译

实验工具

1、编程语言：NASM，TASM，C

2、文本编辑器：Notepad++

3、软盘编辑软件：WinHex

实验思想

BIOS调用

1、其实它与内核子程序软中断调用方式原理是一样的。

2、每一种服务由一个子程序实现，指定一个中断号对应这个服务，入口地址放在中断向量表中，中断号固定并且公布给用户，用户编程时才可以中断调用，参数传递可以使用栈、内在单元或寄存器。

系统调用

1、因为操作系统要提供的服务更多，服务子程序数量太多，但中断向量有限，因此，实际做法是专门指定一个中断号对应服务处理程序总入口，然后再将服务程序所有服务用功能号区分，并作为一个参数从用户中传递过来，服务程序再进行分支，进入相应的功能实现子程序。

2、如果用户所用的开发语言是汇编语言，可以直接使用软中断调用如果使用高级语言，则要用库过程封装调用的参数传递和软中断等指令汇编代码。

3、我们的实验规定系统调用服务的中断号是21h。

模块结构

扇区号	扇区数（大小）	内容
1	1（512 B）	引导程序
2	1（512 B）	用户程序1
3	1（512 B）	用户程序2
4	1（512 B）	用户程序3
5	1（512 B）	用户程序4
6~7	2（893 B）	混合编译用户程序
8~13	6（2570 B）	操作系统内核

命令功能

命令	功能
help	显示支持的命令及其功能
cls	清屏
u1	运行用户程序1
u2	运行用户程序2
u3	运行用户程序3
u4	运行用户程序4
run	连续运行用户程序1-4
int22h	显示“INT22H”
test	运行混合编译用户程序（输入输出）
ls	显示文件列表
reboot	重启
quit	退出操作系统并关机

六、实验过程

1、save（）与restart（）

设计思路

save和restart过程的实质其实就是保护和恢复中断现场。在内核程序kc.c添加一个结构体保存我们想要保存的所有寄存器的值，每次保存现场就将内存中寄存器的值保存到结构变量中，恢复现场就将内存中结构变量的值写回寄存器中。ip寄存器，cs寄存器和程序状态字寄存器要利用栈里面的值，而不能直接将正在执行的程序的变量放在内存中。在程序调用中断的过程时，会将程序状态字，cs和ip自动压栈，所以需要做的时在栈中取出数据。在恢复现场的时候，同样要将这三个值重新压栈，这样才能利用iret指令返回。

数据结构设计

```
struct cpuRegisters {
    int ax;
    int bx;
    int cx;
    int dx;
    int di;
    int bp;
    int es;
    int ds;
    int si;
    int ss;
    int sp;
    int ip;
    int cs;
    int flags;
};
struct cpuRegisters current;
```

保存寄存器内容的结构体我仿照老师的代码来写。在内存中我们可以将一个结构变量看成是一个整体，在内存中是一个连续的存储单元。所以在汇编中只要获取current的偏移量就可以访问。

save和restart代码

```
; save
public _save
_save proc
    push ds
    push cs
    pop  ds
    push si

    lea si,_current
    pop word ptr [si+16];保存si
    pop word ptr [si+14];保存ds

    lea si,return_place
    pop word ptr [si]   ;保存_save返回点

    lea si,_current
    pop word ptr [si+22];保存ip
    pop word ptr [si+24];保存cs
    pop word ptr [si+26];保存flags

    mov [si+18],ss
    mov [si+20],sp

    mov si,ds
    mov ss,si

    lea si,_current
    mov sp,si
    add sp,14

    push es
```

```
    push bp
    push di
    push dx
    push cx
    push bx
    push ax

    lea si,sp_place
    mov sp,[si]

    lea si,return_place
    mov ax,[si]
    jmp ax
_save endp

; restart
public _restart
_restart proc
    lea si,sp_place
    mov [si],sp
    lea sp,_current
    pop ax
    pop bx
    pop cx
    pop dx
    pop di
    pop bp
    pop es

    lea si,ds_place
    pop word ptr [si]
    lea si,si_place
    pop word ptr [si]

    lea si,bx_place
    mov [si],bx          ;保护bx，用它给ss赋值

    pop bx               ;恢复栈
    mov ss,bx
    mov bx,sp
    mov sp,[bx]

    add bx,2

    push word ptr [bx+4];flags,cs,ip压栈
    push word ptr [bx+2]
    push word ptr [bx]

    push ax
    push word ptr [si]  ;把bx压栈保存
    lea si,ds_place
    mov ax,[si]
    lea si,si_place
    mov bx,[si]
    mov ds,ax
    mov si,bx

    pop bx
```

```
    pop ax
    iret
_restart endp

datadef:
    return_place label byte
        dw 0
    si_place label byte
        dw 0
    sp_place label byte
        dw 0
    bx_place label byte
        dw 0
    ds_place label byte
        dw 0
```

save函数用来中断处理时保护现场。操作系统进入中断后，立即调用save把当前状态下cpu所有寄存器，标志寄存器。中断返回点都保存到内存的0-1023某处。并且save结束后，es、ds、cs、ss寄存器都是内核的，即进入了内核空间。使用示例：call __save

restart函数用于在中断响应结束前恢复现场，切换到用户空间，末尾要用iret返回。使用示例：jmp __restart

2、内核增加int 20h、int 21h和int 22h软中断的处理程序

设计思路

通过中断来实现系统调用，需要将系统调用的程序封装到内核的中断处理程序当中，这样程序只需要通过调用中断就可以执行我们定义好的中断处理程序，实现系统调用。

```
; 修改中断向量表
    xor ax, ax
    mov es, ax
    mov word ptr es:[20h], offset Timer
    mov word ptr es:[22h], cs

    mov word ptr es:[32*4], offset int20h
    mov word ptr es:[32*4+2], cs

    mov word ptr es:[33*4], offset int21h
    mov word ptr es:[33*4+2], cs

    mov word ptr es:[34*4], offset int22hprint
    mov word ptr es:[34*4+2], cs
```

INT 20H

从用户程序返回内核程序。在跳转到用户程序之前，已经把寄存器压栈保护，并且把内核的sp保存到sp_place处，调用int 20h返回时跳转到testret，让寄存器出栈，返回内核。

```
; int20h 用户程序返回
int20h:
    mov ax,cs
    mov ds,ax
    mov es,ax
    mov ss,ax
```

```
    lea si,sp_place
    mov sp,[si]
    jmp testret

testret:
    popf                            ;恢复寄存器
    pop si
    pop ds
    pop es
    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret
```

用户程序返回部分

```
Quit:
    mov ah,20H
    mov al,0CDH;int 20h机器码
    mov word[es:0],ax
    ret
```

INT 21H

int 21h我封装了之前ka.asm的5个库函数，将之前写的5个库函数改为系统调用的方式实现。0号功能输出一个字符，1号功能读入一个字符，2号功能清屏，3号功能关机，4号功能重启。

```
; int21h 完成系统内核的调用
int21h:
    cmp ah,0
    jz printChar
    cmp ah,1
    jnz next1
    jmp Readchar
next1:
    cmp ah,2
    jnz next2
    jmp clear
next2:
    cmp ah,3
    jnz next3
    jmp PowerOff
next3:
    cmp ah,4
    jnz quit
    jmp Reboot
quit:
    iret

printChar:
    push bp
    mov bp,sp
    mov al,[bp+10]
```

```
    mov bl,0
    mov ah,0eh
    int 10h
    mov sp,bp
    pop bp
    iret
Readchar:
    mov ah,0
    int 16h
    mov byte ptr [_x],al
    iret
clear:
    push ax
    push bx
    push cx
    push dx
    mov ax, 600h    ; AH = 6,  AL = 0
    mov bx, 700h    ; 黑底白字(BL = 7)
    mov cx, 0       ; 左上角: (0, 0)
    mov dx, 184fh   ; 右下角: (24, 79)
    int 10h         ; 显示中断

    mov ah,2
    mov bh,0
    mov dx,0
    int 10h

    pop dx
    pop cx
    pop bx
    pop ax
    iret
PowerOff:
    mov ax,5307H    ; 高级电源管理功能,设置电源状态
    mov bx,0001H    ; 设备ID1:所有设备
    mov cx,0003H    ; 状态3:表示关机
    int 15H
Reboot:
    mov al, 0FEh
    out 64h, al
    iret
```

调用int 21h

```
; 输出一个字符
public _printChar
_printChar proc
    mov ah,0
    int 21h
    ret
_printChar endp

; 读取一个字符
public _getChar
_getChar proc
    mov ah,1
    int 21h
```

```
    mov byte ptr [_x],al
    ret
_getChar endp

; 清屏
public _cls
_cls proc
    mov ah,2
    int 21h
    ret
_cls endp

; 关机
public _PowerOff
_PowerOff proc
    mov ah,3
    int 21h
    ret
_PowerOff endp

;重启
public _Reboot
_Reboot proc
    mov ah,4
    int 21h
    ret
_Reboot endp
```

INT 22H

int 22h需要实现在屏幕上输出"INT22H"，只要使用int 10h的13h功能即可输出字符串。

```
; int 22h中断处理程序：输出INT22H
int22hprint:
    push ds
    push es
    push ax
    push bx
    push cx
    push si

    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,6        ; 串长
    mov bp,offset int22hstr
    int 10h         ; 输出字符串

    pop si
    pop cx
    pop bx
    pop ax
    pop es
    pop ds
```

```
    iret

int22hstr db "INT22H"
```

调用int 22h

```
public _int22h
_int22h proc
    int 22h
    ret
_int22h endp
```

3、保留无敌风火轮显示，取消触碰键盘显示OUCH!这样功能

保留无敌风火轮

```
Timer:
    call _save              ; save保护中断现场
    mov ax,cs
    mov ds,ax
    lea bx,count            ; 判断count值是否为0，为0显示下一个"风火轮"
    mov al,[bx]
    dec al
    mov [bx],al
    cmp al,0
    jnz exit
    mov al,Delay
    mov [bx],al
    lea bx,wheel
    mov al,[bx]
    xor ah,ah
    inc ax
    cmp ax,4
    jnz next
    mov ax,1
next:
    mov [bx],al
    add bx,ax               ; [bx]是要显示的字符，加上偏移量获取
    mov ax,0b800h
    mov es,ax
    mov al,[bx]
    mov ah,0Fh              ; 黑底白字
    mov bx,((24*80+79)*2)   ; 24行79列
    mov es:[bx],ax
exit:
    mov al,20h              ; AL = EOI
    out 20h,al              ; 发送EOI到主8529A
    out 0A0h,al             ; 发送EOI到从8529A
    jmp _restart            ; restart恢复现场

    Delay equ 5
    count label byte
        db Delay
    wheel label byte
        db 1
        db '|'
```

```
        db '/'
        db '\'
```

我在时钟中断Timer中添加了save和restart，经检验“风火轮”依旧正常转动，说明之前设计的save和restart是有效的。

删去OUCH

删去上次实验OUCH部分的代码即可。

4、C语言的库的输入输出函数设计

libs.asm

putch输出一个字符。这里与之前内核的putChar实现方法相同。

```
public _putch
_putch proc
    push bp
    mov bp,sp
    mov al,[bp+4];char\ip\bp
    mov bl,0
    mov ah,0eh
    int 10h
    mov sp,bp
    pop bp
    ret
_putch endp
```

readch读取一个字符。注意readch不是getch，readch只是单纯读入，getch还需将读取的输出。

```
public _readch
_readch proc
    mov ah,0
    int 16h
    mov byte ptr [_x],al
    ret
_readch endp
```

clibs.c

首先声明两个全局变量，x是读取的单个字符，snum是保存数字对应的字符串，snum的用处在之后的printf和scanf会再提到。

```
char x;
char snum[6];
```

void puts(char *s)

利用循环调用putch即可实现。

```
void puts(char *s) {
    int i;
    for (i = 0; s[i] != 0; i++)
        putch(s[i]);
}
```

void getch(char *c)

输入一个字符，同时要将该字符显示在屏幕上。通过调用readch实现，也设计了回退的功能，最后需要按回车键确认。

```
void getch(char *c) {
    char t = 0;
    x = 0;
    while (1) {
        t = x;
        readch();
        if(x == 8) {
            putCh(x);
            putCh(32);
            putch(x);
            continue;
        }
        if (x == 13) {/*回车*/
            x = t;
            putch(10);/*换行*/
            putch(13);
            break;
        }
        putch(x);
    }
    *c = x;
}
```

void gets(char* s)

输入一个字符串，通过循环调用readch实现，也有回退功能，最后需要按回车键确认。

```
void gets(char* s) {
    int i = 0;
    readch();
    while(x != 13) {
        if(x == 8) {
            putCh(x);
            putCh(32);
            i--;
            putch(x);
            readch();
            continue;
        }
        putch(x);
        s[i] = x;
        i++;
        readch();
    }
    putch(10);
```

```
    putch(13);
    s[i] = '\0';
}
```

scanf()和printf()和之前的不同，这里的功能更丰富，实现了格式化输入和输出，类似C中的stdio库。要实现这两个函数，首先要写两个将字符串转为数字和将数字转为字符串的函数。

char *itoa(int n)

将数字转为字符串

```
char *itoa(int n) {
    int i = 0, t = 10, s = 0;
    if (n == 0) {
        snum[0] = '0';
        snum[1] = 0;
        return snum;
    }
    while (n != 0) {
        snum[i] = n % t + '0';
        n /= 10;
        i++;
        s++;
    }
    for (i = 0; i < s / 2; i++) {
        char temp = snum[i];
        snum[i] = snum[s - 1 - i];
        snum[s - 1 - i] = temp;
    }
    snum[s] = 0;
    return snum;
}
```

int atoi(void)

将字符串转为数字

```
int atoi(void) {
    int i, s = 0;
    for (i = 0; snum[i] != 0; i++) {
        s *= 10;
        s += (snum[i] - '0');
    }
    return s;
}
```

void printf(char* str, ...)
printf格式化输出,类似于stdio库中的printf。我在网上搜集了大量关于变长参数的资料，对于变长参数，调用时仍然是从右到左压栈。对于参数的个数，可以通过判断'%'的个数实现。其它参数可通过str的地址加上偏移量的形式访问，例如&str+1是str后面一个参数的地址，*(&str+1)即可访问该参数。

```
void printf(char *str, ...) {
    int i;
    int offset = 1;
    for (i = 0; str[i] != 0; i++) {
        if (str[i] != '%')
```

```
            putch(str[i]);
        else {
            i++;
            switch (str[i]) {
                case 'd':
                    puts(itoa((int)*(&str + offset)));
                    break;
                case 'c':
                    putch((char)*(&str + offset));
                    break;
                case 's':
                    puts((char *)*(&str + offset));
                    break;
                default:
                    continue;
            }
            offset += 1;
        }
    }
}
```

void scanf(char* str, ...)

scanf格式化输入,类似于stdio库中的scanf。具体实现方法与printf差不多。

```
void scanf(char *str, ...) {
    int i;
    int offset = 1;
    for (i = 0; str[i] != 0; i++) {
        if (str[i] == '%') {
            i++;
            switch (str[i]) {
                case 'd':
                    gets(snum);
                    *((int *)(*(&str + offset))) = atoi();
                    break;
                case 'c':
                    getch((char *)*(&str + offset));
                    break;
                case 's':
                    gets((char *)*(&str + offset));
                    break;
                default:
                    continue;
            }
            offset += 1;
        }
    }
}
```

5、混合编译链接生成COM格式用户程序

enter.asm

混合编译生成的COM程序都要org 100h。下面的代码是test.com用户程序的入口，最后通过int 20h返回。

```
org 100h
start:
    mov ax,cs
    mov ds,ax
    mov es,ax
    call near ptr _main

    ret
    include libs.asm
```

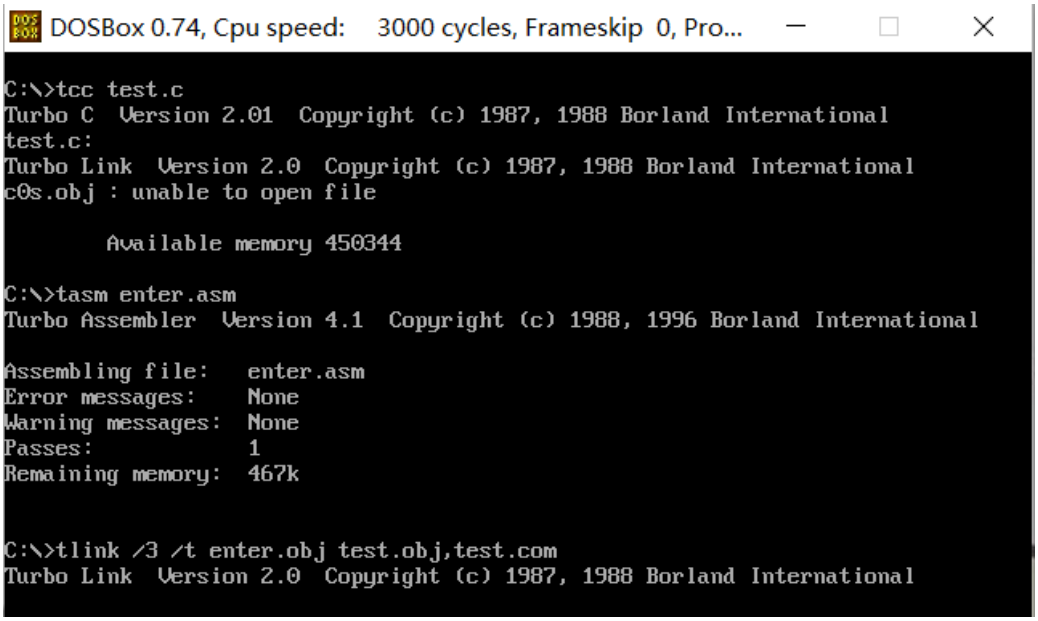
test.c

这是老师给的main函数，我在最后加了一个readch，用于按下任意键返回，否则会直接返回内核看不到结果。

```
#include"clibs.c"

int main() {
    char ch, str[80];
    int a;
    printf("ch=");
    getch(&ch);
    printf("str=");
    gets(str);
    printf("a=");
    scanf("%d", &a);
    putch(ch);
    printf("\r\n");
    puts(str);
    printf("\r\n");
    printf("ch=%c,a=%d,str=%s\r\n", ch, a, str);
    readch();/*用readch()按下任意键返回，否则会立即返回内核看不到输出结果*/
}
```

混合编译

利用TCC+TASM+TLINK组合



混合编译用户程序的调用

在完成混合编译用户程序的生成后，就该思考如何调用test.com了。不同于之前的汇编用户程序，在loadUser函数中只需加载到内存位置，确定好扇区等参数后使用call就行，混合编译生成的用户程序test.com如果直接使用call来调用会出现乱码，字符显示完全不正确。所以我写了两个函数loadTest和jump。

loadTest用于将混合编译生成的用户程序加载到指定内存位置。

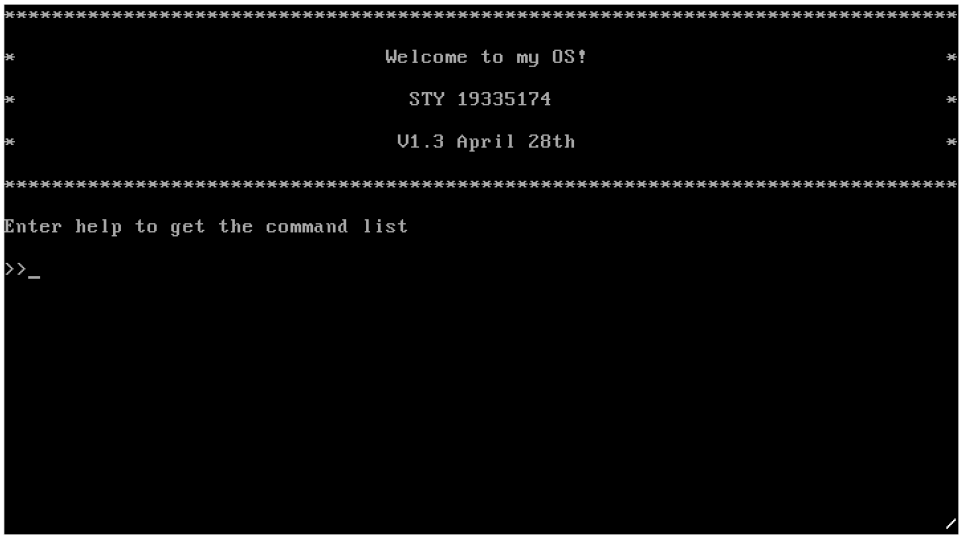
```
public _loadTest
_loadTest proc
    push ax
    push ds
    push es
    push bp

    mov bp,sp
    mov ax,2000h              ;段地址 ; 存放数据的内存基地址
    mov es,ax                 ;设置段地址（不能直接mov es,段地址）
    mov bx,100h               ;偏移地址; 存放数据的内存偏移地址
    mov ah,2                  ;功能号2
    mov al,2                  ;扇区数2
    mov dl,0                  ;驱动器号 ; 软盘为0，硬盘和U盘为80H
    mov dh,0                  ;磁头号 ; 起始编号为0
    mov ch,0                  ;柱面号 ; 起始编号为0
    mov cl,[bp+10]            ;起始扇区号 ; 起始编号为1
    int 13H ;                 调用读磁盘BIOS的13h功能
    ;用户程序test.com已加载到指定内存区域

    pop bp
    pop es
    pop ds
    pop ax
    ret
_loadTest endp
```

jump用于跳转到该内存位置，调用混合编译生成的用户程序。注意要初始化用户程序ss=cs，sp=0000h，并且把这里的ss和sp压到用户程序的栈里，然后再压cs和ip，最后用int 20h即testret将寄存器出栈，返回内核。

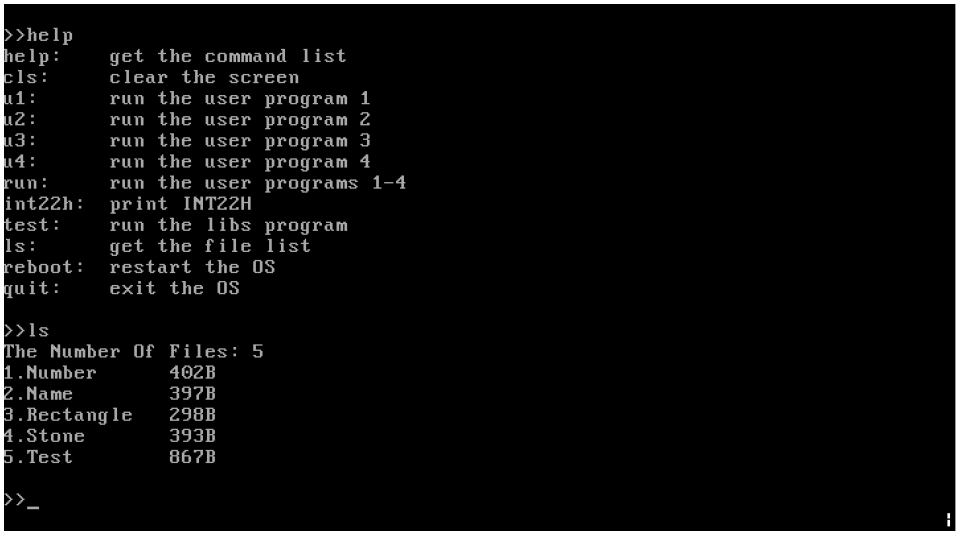
```
public _jump
_jump proc
    push bp
    mov bp,sp

    push ax
    push bx
    push cx
    push dx
    push di
    push es
    push ds
    push si
    pushf

    mov bx,2000h                ;段地址
    mov ax,100h                 ;偏移地址

    mov es,bx
    lea si,testbegin
    mov di,0
    lea cx,testend
    sub cx,si                   ;循环次数
    rep movsb                   ;循环写

    lea si,sp_place
    mov [si],sp

    mov ss,bx                   ;切换到用户栈
    mov sp,0

    xor cx,cx
    push cx                     ;压0x0000
    push bx                     ;要跳转的cs
    push ax                     ;要跳转的ip
    retf                        ;跳转

testret:
    popf                        ;恢复寄存器
    pop si
    pop ds
    pop es
    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret

testbegin:
    int 20h                     ;用int 20h返回
testend:nop

_jump endp
```

七、实验结果

操作系统界面



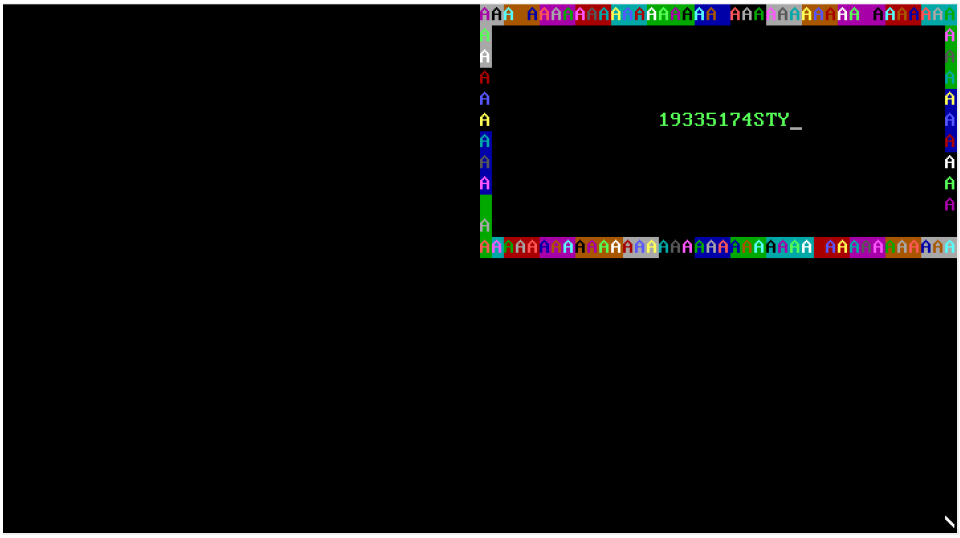
帮助菜单和文件列表



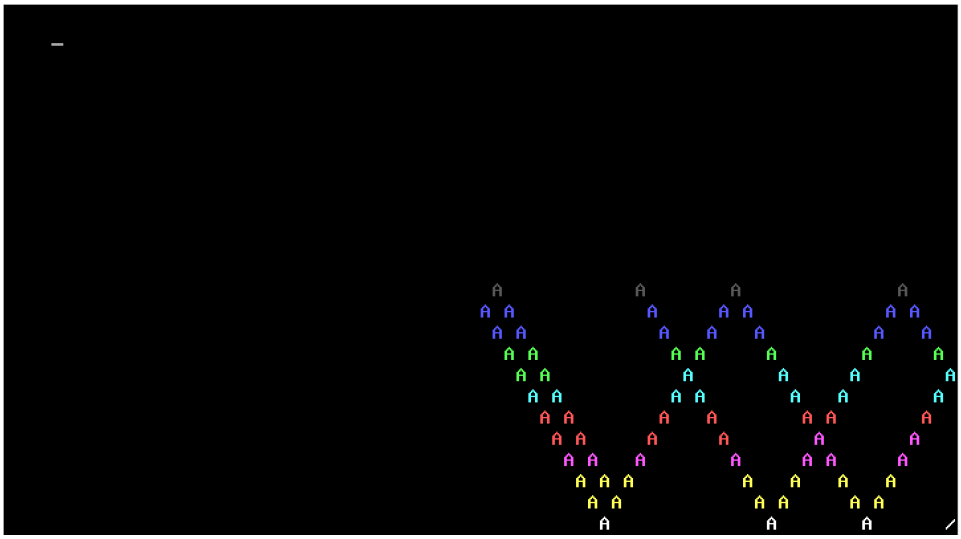
用户程序1

用户程序2

用户程序3

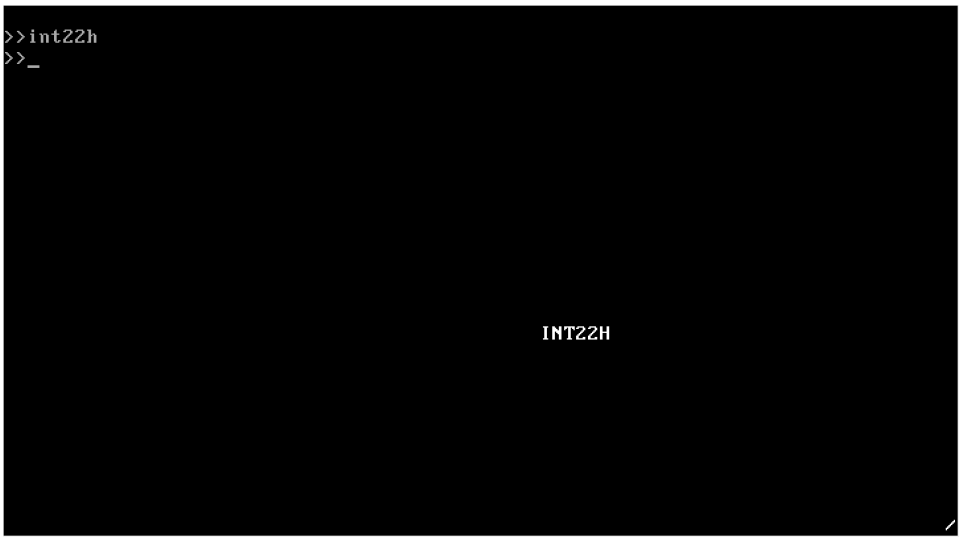


用户程序4

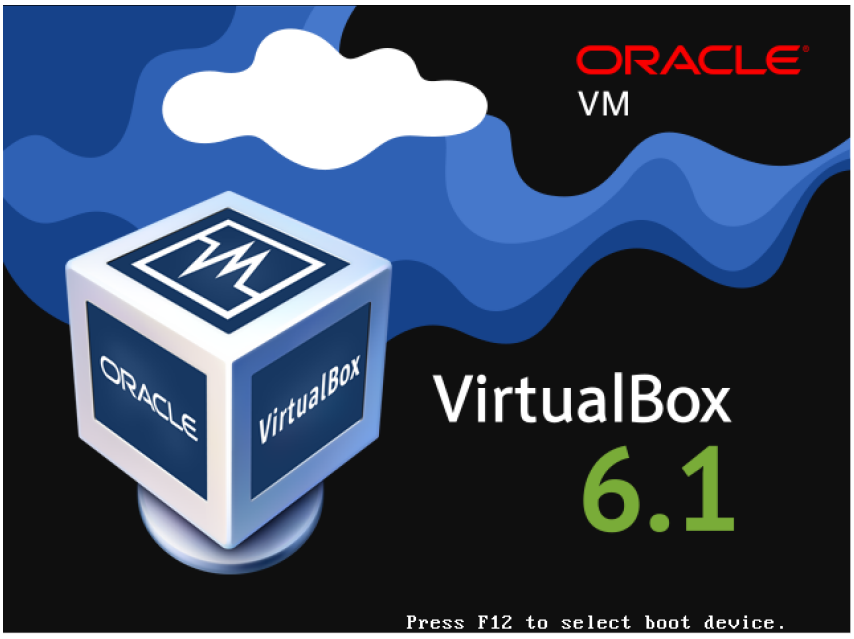


用户程序5（libs program）

INT 22H

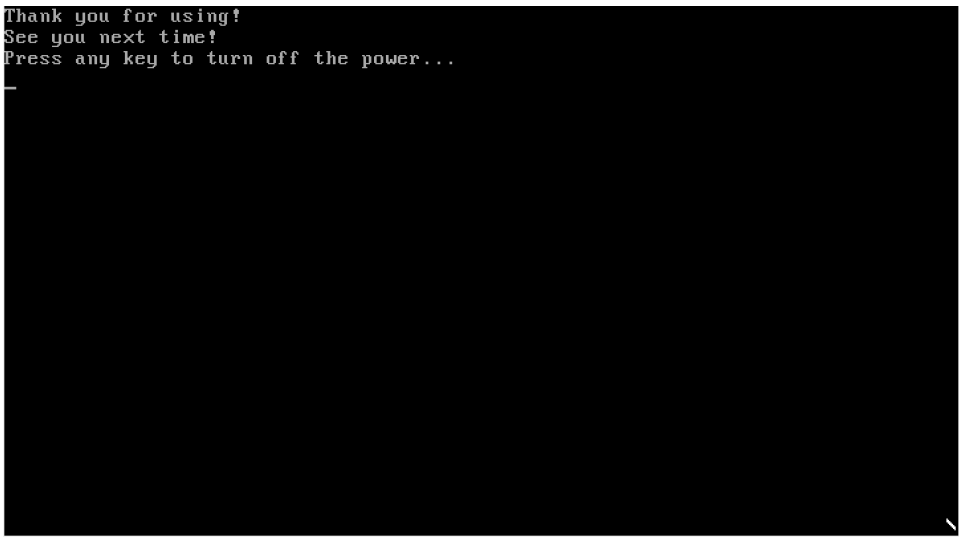


重新启动（int 21h的4号功能）



关闭电源（int 21h的3号功能）

按下任意按键后关闭电源!



八、实验总结

这次实验比较困难，但也让我学到了很多东西。我对中断向量表的修改和中断处理程序的运用更加熟练了，也学会了系统调用的实现，以及中断现场的保护和恢复。但是我对栈的理解还是不够深入，在实验中稍有不慎就会出现奇奇怪怪的错误，导致我花费了大量时间。

在保护现场和恢复现场的过程中，需要注意的是ip寄存器，cs寄存器和程序状态字寄存器要用栈里面的值，而不能直接将正在执行的程序的变量放在内存中。因为我们要保存的不是save和restart过程中的程序执行的ip，cs和程序状态字，而是调用中断的程序的ip,cs和程序状态字。在程序调用中断的过程时，会将它们自动压栈，需要时在栈中取出数据。在恢复现场的时候，同样要将这三个值重新压栈，再用iret返回。

int 20h我一开始以为把ret写在中断处理程序即可，但是出现了莫名其妙的错误。后来我尝试用push和pop实现对寄存器的保护，调整偏移量返回，但是我发现我的程序直接跳回了内核的起点。虽然这在执行单个用户程序时不会出现什么大的问题，但我在执行我的run命令（连续执行4个用户程序）时却会出现只执行一个用户程序死循环的情况。好在后来改进了我的save和restart，并且正确保存了内核的sp，终于正确完成了这个功能。int 21h这次我有点偷懒，没有实现新的功能，只是把原来实现的输出一个字符，输入一个字符，清屏，关机，重启这5个功能封装，以系统调用的形式实现了，这样也让我的代码更加规整。int 22h的实现比较简单，只要使用int 10h的13h功能就能输出字符串。

混合编译的C用户程序是我这次实验遇到的最大问题。首先要实现C语言stdio库中的scanf和printf就让我头痛欲裂，不知道%d和%s这样的功能究竟是怎样实现的。好在后来我在网上查询了大量资料，发现了c语言的变长参数的相关介绍，关键是要用取地址符&来获取参数的地址，访问其他参数。而且我开始在实现gets的时候忽略了“回退”这个按键的设计，导致我输入的时候按下“回退”只是光标往左移了一格，要再重新输入才会显示我删去字符的改变，好在后来我改进了代码成功完成了。

虽然我编写好的混合编译用户程序能在DOSBox中正确运行，但当我在虚拟机中时却出现了各种千奇百怪的错误。因为偏移量不正确，入栈出栈顺序不正确，链接时重定向位置和用户程序加载到内存位置不匹配等等问题，我的新用户程序遇到了无法正确跳入、乱码、返回时底层BIOS函数被改变等诸如此类的错误。这是我在本次实验中遇到的最大问题，也让我心力交瘁了很久。后来我发现混合编译生成的用户程序与纯汇编用户程序不同，不能直接用call，用call就会乱码，而是要用jmp跳转到加载好的内存指定位置，且要小心对栈的操作和寄存器的保护。而在混合编译生成的用户程序返回内核时，也不能直接用ret，因为栈已经被改变，我开始用ret出现了重复3次循环这个用户程序然后死机的离谱错误。后来我想起来int 20h有对寄存器的保护并且返回内核，所以我最后用int 20h就能成功返回内核了。

总而言之，这次实验花费了我很多时间，但也让我受益匪浅。对于中断处理程序的运用我更加熟练了，也学会了系统调用的使用，和混合用户程序的编写与加载。后面的实验马上就是多进程了，希望我也能迎难而上，再接再厉，勇往直前！

九、参考文献

《汇编语言 （第3版）》

《X86汇编语言：从实模式到保护模式》

https://zhuanlan.zhihu.com/p/45153186

https://wenku.baidu.com/view/9ac0446c48d7c1c708a145aa.html

https://blog.csdn.net/qq_35212671/article/details/52761585