

实现系统调用

实现系统调用

个人信息

实验题目

实验目的

实验要求

实验内容

实验方案

所用工具

虚拟机配置

代码关键部分

内核

kfun.asm

ckfun.c

C语言库设计

libs.asm

clibs.c

测试程序1

entry.asm

test.c

测试程序2

实验过程

生成com文件

写盘并且运行

运行虚拟机

实验总结

参考资料

个人信息

- 院系：数据科学与计算机学院
- 年纪：2018
- 姓名：王天龙
- 学号：18340168

实验题目

实现系统调用

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

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()、scanf()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

实验内容

- 1. (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,在与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);
    scanf("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








实验方案

所用工具

实验平台是windows10系统，使用的软件有：虚拟机软件vmware workstation pro、汇编语言编译器tasm、c语言编译器tcc、链接器tlink、可视化编译十六进制文件内容工具winhex、代码编辑器visual studio 2019

虚拟机配置

虚拟机为1cpu，内存为4MB，使用一个1.44MB大小的软盘，选择从软盘启动

设备	摘要
 内存	4 MB
 处理器	1
 CD/DVD (IDE)	自动检测
 软盘	正在使用文件 D:\computer_op...
 网络适配器	NAT
 声卡	自动检测
 显示器	自动检测

代码关键部分

内核

- kernel.asm
 - _start 入口
- kfun.asm
 - _putc
 - _readkeypress
 - _cleanscreen
 - _loadblock
 - _jump
 - _LoadWhell
 - _LoadOuch
 - _save 用于中断处理的现场保护 (新增)
 - _restart 中断处理后恢复中断现场 (新增)
 - _LoadINT20H 加载20h号中断，返回操作系统 (新增)
 - _LoadINT21H 加载21h号中断，提供系统调用 (新增)
 - _LoadINT22H 加载22h号中断，显示INT22H (新增)
 - SYCALL_1H 1号系统调用，从键盘读一个字符 (新增)
 - SYCALL_2H 2号系统调用，显示一个字符 (新增)
 - SYSCALL_4CH 4ch号系统调用，返回操作系统 (新增)
- ckfun.c
 - struct cpuStat
 - void printf(char* s)
 - void scanf(char* s)
 - void kernelmain()

kfun.asm

_save的实现

这个函数用来中断处理时保护现场。操作系统响应21h号中断后，立即调用_save把当前状态下，cpu所有寄存器，标志寄存器，以及中断返回点都保存到内存的某处。并且，_save结束后，ss,cs,ds,es寄存器都是内核的了，即进入了内核空间。为了描述cpu状态，在ckfun.c里定义了一个c语言结构体，然后定义一个全局变量，用于保存cpu状态，该结构体如下

```
struct cpuStat
{
    int ax; /*偏移0*/
    int bx;
    int cx;
    int dx;
    int di;
    int bp;
    int es;
    /*最先保存的一批*/
    int ds;
    int si;
    int ss;
    int sp;
    int ip;
    int cs; /*cs, ip是中断处理结束后的返回点*/
    int flags; /*标志寄存器，偏移26*/
```

```
};
```

```
struct cpuStat CurrentState; /*定义一个全局变量，汇编部分可以直接访问*/
```

在汇编部分kfun.asm里，获取_CurrentState的偏移量就可以访问。

用call指令调用_save后，栈还在用户空间，从高地址到低地址为用户的标志寄存器，中断返回点的cs，中断返回的ip，以及_save返回的ip。

```
;代码部分
_save proc near
    ;刚开始时，还在用户空间
    push ds
    push cs
    pop ds;执行完后，ds=cs，属于内核空间，原来的ds在用户栈里
    push si

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

    lea si,ret_temp
    pop word ptr [si];保存_save的返回点到ret_temp中

    lea si,_CurrentState
    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;保存用户的栈以及栈顶指针
    mov [si+20],sp

    mov si,ds
    mov ss,si
    lea si,_CurrentState
    mov sp,si
    add sp,14
    ;此时，ss已经切换到内核里，sp指向_CurrentState里的ds

    push es;push具体操作为sp-2，把es的值移到ss:[sp]，即CurrentState.es中
    push bp
    push di
    push dx
    push cx
    push bx
    push ax

    lea si,kernelSp;kernelSp里保存有内核空间的栈顶指针
    mov sp,[si]
    ;至此ss,sp,ds,cs,ip都已经是内核的了

    lea si,ret_temp
    mov ax,[si]
    jmp ax;跳转到_save的返回点
_save endp
;数据段部分
_DATA segment word public 'DATA'
```

```

ret_temp label byte
    dw ?
kernel_sp label byte;在跳转到用户程序之前，保存内核栈顶指针sp
    dw ?
_DATA ends

```

系统调用_save函数后，就切换到内核空间里了，可以执行其它操作

_restart的实现

这个函数用于在中断响应结束前，恢复现场，切换到用户空间了。注意，要用jmp指令跳转到_restart执行，而不是用call，防止栈里有一个多余的返回点，具体代码如下

```

_restart proc near
    lea si, kernel_sp
    mov [si], sp;把内核空间的栈顶保护起来
    lea sp, _CurrentState
    pop ax;用pop操作恢复寄存器
    pop bx
    pop cx
    pop dx
    pop di
    pop bp
    pop es;此时sp指向ds

    lea si, ds_temp;暂存ds和si，ds和si最后才恢复
    pop word ptr [si]
    lea si, si_temp
    pop word ptr [si]

    lea si, bx_temp
    mov [si], bx;保护bx，用它给ss赋值
    ;恢复栈
    pop bx
    mov ss, bx
    mov bx, sp
    mov sp, [bx];ds:[bx]就是用户空间的sp

    add bx, 2;bx指向返回点的ip
    ;flags, cs, ip按顺序压栈
    push word ptr [bx+4]
    push word ptr [bx+2]
    push word ptr [bx]

    ;恢复ds, si
    push ax
    push word ptr [si];把bx压栈保存
    lea si, ds_temp
    mov ax, [si]
    lea si, si_temp
    mov bx, [si]
    mov ds, ax
    mov si, bx

    pop bx
    pop ax

```

`iret;iret`依次把ip,cs,flags弹出

```
_restart endp
;数据段部分
_DATA segment word public 'DATA'
ds_temp label byte
    dw ?
si_temp label byte
    dw ?
bx_temp label byte
    dw ?
kernel_sp label byte;在跳转到用户程序之前，保存内核栈顶指针sp
    dw ?
_DATA ends
```

_LoadINT20H的实现

这个函数用于装载20h号中断，功能是用户程序返回操作系统。需要和操作系统跳转到用户程序的_jump函数配合。在跳转到用户程序之前，_jump里就已经把寄存器压栈保护，并且把sp保存到kernel_sp处，并且_jump里有一个地址标号retp，然后跳转用户程序。系统响应20h号中断时，只要把cs的值赋给ss，并且把kernel_sp的值给sp就恢复内核栈，然后跳转到retp即可。具体代码如下

_jump代码

```
public _jump
_jump proc near;两个参数,要跳到的cs:ip
    push bp
    mov bp,sp
    ;寄存器压栈保护
    push ax
    push bx
    push cx
    push dx
    push di
    push es
    push ds
    push si
    pushf

    mov bx,[bp+4];获取参数cs
    mov ax,[bp+6];获取参数ip

    ;把PSP写到cs:0处
    mov es,bx
    lea si,PSPBegin
    mov di,0
    ;movsb 把ds:si->es:di
    lea cx,PSPEnd
    sub cx,si;循环次数
    rep movsb;循环写

    lea si,kernel_sp
    mov [si],sp

    ;把这里的sp放在kernel_sp里
    ;程序前缀用系统调用返回到内核，然后跳转一下就到了
    ;用户栈只需提前压一个0x0000进去即可
```

```

;切换到用户栈
mov ss,bx
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
retp:
;恢复寄存器
popf
pop si
pop ds
pop es
pop di
pop dx
pop cx
pop bx
pop ax
pop bp
ret

;程序前缀PSP的内容
PSPBegin:
    int 20h
PSPEnd:nop

_jump endp

```

_LoadINT20H的代码

```

public _LoadINT20H
_LoadINT20H proc near
    push ax
    push es
    CLI
    ;装填中断向量
    xor ax,ax
    mov es,ax
    lea ax,INT20H

    mov [es:80h],ax
    mov ax,cs
    mov [es:82h],ax
    mov es,ax
    STI

    pop es
    pop ax
    ret

INT20H:
    mov ax,cs
    mov ds,ax

```

```

;恢复内核栈
mov ss,ax
lea si,kernel$sp
mov sp,[si]
jmp retp;注意最后要跳转到_jump的retp,而不是用iret返回
_LoadINT20H endp

```

_LoadINT22H实现

装载22h号中断,这个中断功能简单,知识在屏幕某处显示字符串INT22H,代码如下

```

public _LoadINT22H
_LoadINT22H proc near
    push ax
    push es
    CLI
    ;装填中断向量
    xor ax,ax
    mov es,ax
    lea ax,INT22H
    mov [es:88h],ax
    mov ax,cs
    mov [es:8ah],ax
    mov es,ax
    STI

    pop es
    pop ax

    ret
INT22H:
    push ds
    push es
    push ax
    push bx
    push cx
    push si

    mov ax,cs
    mov ds,ax

    mov ax,0b800h;显存地址
    mov es,ax

    lea si,int22hstrlen;获取字符串长度
    mov cx,[si]

    lea si,int22hstr;获取字符串地址

    mov bx,3800;在屏幕倒数第二行打印INT20H
    mov ah,07h

again3:;打印字符
    mov al,[si]
    mov es:[bx],ax
    inc si
    inc bx

```



```

    inc bx
    loop again3;循环打印
    pop si
    pop cx
    pop bx
    pop ax
    pop es
    pop ds

    iret

_LoadINT22H endp
;数据段部分
_DATA segment word public 'DATA'
int22hstr label byte;字符串
    db "INT22H"
int22hstrlen label byte;字符串长度
    db $-int22hstr
_DATA ends

```

_LoadINT21H的实现

装载21h号中断，该中断用于提供系统调用，系统调用号参考了DOS的，这里实现了3个系统调用，分别是1h：从键盘输入一个字符，放在al中；2h：显示一个字符，dl为要显示的字符；4ch：返回操作系统。处理21h号中断时，系统首先要调用_save保护现场，然后根据ah的值选择调用相应的系统调用，最后用_restart返回到用户程序。具体代码如下

_LoadINT21Hd代码

```

_LoadINT21H proc near
    push ax
    push es
    CLI
    ;装填中断向量
    xor ax,ax
    mov es,ax
    lea ax,INT21H
    mov [es:84h],ax
    mov ax,cs
    mov [es:86h],ax
    mov es,ax
    STI

    pop es
    pop ax

    ret

INT21H:
    call _save;保护现场

    lea si,_CurrentState
    mov ax,[si];获取ax

    cmp ah,1h;选择系统调用
    jz SYCALL_1H

```

```

cmp ah,2h
jz SYCALL_2H

cmp ah,4ch;返回操作系统
jz SYCALL_4CH

```

SYCALLEND:

```

jmp _restart;恢复现场，并且返回用户程序，注意是jmp而不是call,call会把ip压栈
_LoadINT21h endp

```

SYCALL_1H系统调用

```

SYCALL_1H proc near;从键盘输入一个字符，al为输入的字符
mov ax,cs
mov ds,ax
mov ah,0
int 16h;调用BIOS调用，读一个字符
xor ah,ah
lea si,_CurrentState
mov [si],ax;修改ax
jmp SYCALLEND
SYCALL_1H endp

```

SYCALL_2H系统调用

```

SYCALL_2H proc near;显示一个字符，DL为要显示的字符
mov ax,cs
mov es,ax

mov bh,0
mov ah,3
int 10h;获取当前光标的位置
lea bp,_CurrentState
add bp,6;es:[bp]为DL的值，即要显示的字符
mov ax,1301h
mov bx,0007h
mov cx,1
int 10h;显示字符
jmp SYCALLEND
SYCALL_2H endp

```

SYCALL_4CH系统调用

```

SYCALL_4CH proc near;和int 20h的功能以及实现一致，返回操作系统
mov ax,cs
mov ss,ax
lea si,kernel$sp
mov sp,[si]
jmp ret
SYCALL_4CH endp

```

ckfun.c

这个文件包含了内核的主要代码，这次实验没有太大的修改。只是装载了20H，21H，22H三个中断，并且定义了保存cpu寄存器的结构体，声明了全局变量。同时，消除碰键盘显示"OUCH!OUCH!"的功能保留了下来，修改如下

```
extern void LoadINT20H();
extern void LoadINT21H();
extern void LoadINT22H();
...

struct cpuStat
{
    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 cpuStat CurrentState;
...
void kernelmain()
{
    ...
    LoadINT20H();
    LoadINT21H();
    LoadINT22H();
    ...
}
```

C语言库设计

C语言库包含了2个文件，分别为libs.asm和clibs.c，实现了putch(), getch(), puts(), gets(), scanf(), printf()函数。注意，scanf()和printf()和内核里的scanf()和printf()不一样，这里的功能更加强大，实现了格式化输入和输出。文件结构如下

- libs.asm
 - _putch
 - _readkeypress
- clibs.c
 - char sum[10];
 - char* i2s(int n)
 - int s2i(void)
 - void puts(char*s)
 - void getch(char*c)
 - void gets(char*s)
 - void printf(char*str,...)

- o void scanf(char*str,...)

libs.asm

这里实现了两个最基本的函数，_readkeypress和_putch。_readkeypress从键盘读一个按键，不显示；_putch显示一个字符。这两个函数都是使用系统提供的系统调用实现

_putch

```
_putch proc near;只有一个参数，即要显示的字符
    push bp
    mov bp,sp
    push ax
    push dx
    mov dl,[bp+4];把字符送到dl
    mov ah,2
    int 21h;使用2号系统调用
    pop dx
    pop ax
    pop bp
    ret
_putch endp
```

_readkeypress

```
_readkeypress proc near
    mov ah,1
    int 21h;s使用1号系统调用
    cbw;按al最高位扩展
    ret
_readkeypress endp
```

通过循环调用上述函数，就可以实现输入字符串和显示字符串了。

clibs.c

char* i2s(int n)实现

这个函数，主要用于把数值转成字符串，用于显示数字

```
/*全局变量，用于保存数字对应的字符串*/
char snum[6];/*0xffff=65535，最多5位*/

char *i2s(int n)
{
    int i = 0, t = 10, s = 0;
    if (n == 0)
    {
        snum[0] = '0';
        snum[1] = 0;
        return snum;
    }
    while (n != 0)/*把数字从最低位开始，依次转成字符*/
    {
        snum[i] = n % t + '0';
        n /= 10;
        i++;
    }
}
```

```

        s++;
    }
    for (i = 0; i < s / 2; i++)/*字符串反转*/
    {
        char temp = snum[i];
        snum[i] = snum[s - 1 - i];
        snum[s - 1 - i] = temp;
    }
    snum[s] = 0;
    return snum;
}

```

int s2i(void)

这个函数用于把snum字符串转换成对应的数字，代码如下

```

char snum[10];
int s2i(void)
{
    int i, s = 0;
    for (i = 0; snum[i] != 0; i++)
    {
        s *= 10;
        s += (snum[i] - '0');
    }
    return s;
}

```

void puts(char*s)

这个函数用于把字符串s显示出来，通过循环调用putch实现，代码如下

```

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

```

void getch(char*c)

这个函数用于读取一个字符。通过调用readkeypress实现。注意，输入字符后，需要按回车键确认。

```

void getch(char *c)
{
    char t = 0;
    *c = 0;
    while (1)/*不停的读按键，只到有回车键确认，才把前一个按键返回*/
    {
        t = *c;
        *c = readkeypress();/*读按键*/
        if (*c == 0x0d)/*判断是否是回车*/
        {
            *c = t;
            putch(0x0a);/*换行*/
        }
    }
}

```

```

        putchar(0x0d);
        break;
    }
    putchar(*c); /*把按下的按键显示出来*/
}
return;
}

```

void gets(char*s)

这个函数用于输入一个字符串，通过循环调用readkeypress实现，知道有回车按下结束。支持用Backspace删除已经输入的字符。代码如下

```

void gets(char *s)
{
    int i = 0;
    char c;
    while (1)
    {
        c = readkeypress();
        if (c == 0x0d) /*回车*/
        {
            s[i] = 0;
            putchar(0x0a); /*换行*/
            putchar(0x0d);
            break;
        }
        else if (c == 0x08) /*按下Backspace*/
        {
            if (i == 0)
            {
                continue;
            }
            i--;
            putchar(0x08); /*光标向前移动一格*/
            putchar(' '); /*在前一个字符的位置显示空格，看起来就像是被删除了*/
            putchar(0x08);
            continue;
        }
        s[i] = c;
        i++;
        putchar(c);
    }
    return;
}

```

void printf(char*str,...)

实现了类似<stdio.h>的printf格式化输出。对于变长参数，调用时仍然是从右到左压栈。为了访问除str以外的参数，可以通过参数str的地址加上偏移来实现，获取str的地址可以通过&实现，例如&str+1，是str后面第一个参数的地址，*(&str+1)即可访问该参数。而要确定有多少个参数，可以通过判断str里字符'%'的个数实现。具体代码如下

```

void printf(char *str, ...)
{
    int i;

```

```

int parOffset = 1; /*偏移量*/
for (i = 0; str[i] != 0; i++)
{
    if (str[i] != '%') /*如果字符不是'%', 就把它显示出来*/
    {
        putchar(str[i]);
    }
    else /*如果是'%', 则判断其后跟的是什么*/
    {
        i++;
        /* *(&str+parOffset)就可以访问到对应的参数, 同时要加上类型转换*/
        switch (str[i])
        {
            case 'd': /*显示整数*/
                puts(i2s((int)*(&str + parOffset)));
                break;
            case 'c': /*显示字符*/
                putchar((char)*(&str + parOffset));
                break;
            case 's': /*显示字符串*/
                puts((char *)(&str + parOffset));
                break;
            default:
                continue;
        }
        parOffset += 1; /*注意偏移量加的是1*/
    }
}
}

```

void scanf(char*,...)

实现了类似<stdio.h>的scanf格式化输入。对于变长参数的处理, 类似printf(), 这里不再赘述, 具体代码如下

```

void scanf(char *str, ...)
{
    int parOffset = 1;
    int i;
    for (i = 0; str[i] != 0; i++)
    {
        if (str[i] == '%')
        {
            i++;
            switch (str[i])
            {
                case 'd': /*输入整数*/
                    gets(snum);
                    *(((int *)(&str + parOffset))) = s2i();
                    break;
                case 'c': /*输入字符*/
                    getch((char *)(&str + parOffset));
                    break;
                case 's': /*输入字符串*/
                    gets((char *)(&str + parOffset));
                    break;
                default:

```

```

        continue;
    }
    parOffset += 1;
}
}
}

```

测试程序1

为了测试C语言库是否能正常运行，设计了以下测试程序

entry.asm

程序入口

```

_start:
    mov ax,cs
    mov ds,ax
    mov es,ax
    call near ptr _main

    ret;返回操作系统，提供多种返回方式
    ;或者直接用中断返回
    ;int 20h
    ;或者用系统调用返回
    ;mov ah,4ch
    ;int 21h

```

test.c

主体函数

```

extern void printf(char *s, ...);
extern void scanf(char *s, ...);
extern void puts(char *s);
extern void putchar(char c);
extern void getch(char *c);
extern void gets(char *str);

int main()
{
    char ch, str[80];
    int a;
    printf("ch=");
    getch(&ch);/*输入字符*/
    printf("str=");
    gets(str);/*输入字符串*/
    printf("a=");
    scanf("%d", &a);/*输入数字*/
    putchar(ch);
    printf("\r\n");
    puts(str);
    printf("\r\n");
    printf("ch=%c,a=%d,str=%s\r\n", ch, a, str);
}

```

测试程序2

这个程序用于测试22h号中断，代码如下

test1.asm

```
.8086
_TEXT segment byte public 'CODE'
DGROUP group _TEXT,_DATA,_BSS
assume cs:_TEXT,ds:DGROUP,ss:DGROUP

org 100h

_start:

    int 22h

    ret
_TEXT ends

_DATA segment word public 'DATA'
_DATA ends

_BSS segment word public 'BSS'
_BSS ends

end _start
```

实验过程

生成com文件

在DoxBox里使用命令

```
tasm kernel.asm
tasm kfun.asm
tcc -c ckfun.c
tlink /3 /t kernel.obj kfun.obj ckfun.obj, kernel.com
```

即可生成内核程序的com执行体。用户程序也类似。

写盘并且运行

写盘式，引导程序在首扇区，监控程序在2、3、4、5扇区，用户程序在6-14号扇区，可以用指令查看用户程序所在的扇区

运行虚拟机

风火轮显示效果

```
m for message, l to list, c to clean screen, 1-5 to select program: _
```

```
m for message, l to list, c to clean screen, 1-5 to select program: _
```

按下键盘

```
m for message, l to list, c to clean screen, 1-5 to select program: aaaaaaaaaa
aa
OUCH! OUCH!
```

```
m for message, l to list, c to clean screen, 1-5 to select program: 1
A
A
OUCH! OUCH!
```

运行测试程序1，输入6

实验总结

通过这次实验，对系统调用的含义有了更深入的理解，同时，了解了c语言的变长参数的处理过程。这次实验困难的地方在于相应系统调用前的保护现场和调用完成后的恢复现场，这两个过程需要在用户空间和内核空间进行巧妙的切换。而在处理变长参数是，关键要用取地址符&来获取参数的地址，实现访问其它参数。

参考资料

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