

C与汇编开发独立批处理的内核

C与汇编开发独立批处理的内核

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

实验原理

代码关键部分

标准库

stdio.asm

cstdio.c

内核

kfun.asm

ckfun.c

kernel.asm

统计字符串程序

cfun.c

count.asm

引导程序

实验过程

选择编译工具以及分析符号列表

写盘并且运行

运行虚拟机

m指令

l指令

运行统计字符串程序，输入5

c指令

运行字符反弹程序，输入1

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

C与汇编开发独立批处理的内核

实验目的

1. 加深理解操作系统内核概念
2. 了解操作系统开发方法
3. 掌握汇编语言与高级语言混合编程的方法

4. 掌握独立内核的设计与加载方法
5. 加强磁盘空间管理工作

实验要求

1. 知道独立内核设计的需求
2. 掌握一种x86汇编语言与一种C高级语言混合编程的规定和要求
3. 设计一个程序，以汇编程序为主入口模块，调用一个C语言编写的函数处理汇编模块定义的数据，然后再由汇编模块完成屏幕输出数据，将程序生成COM格式程序，在DOS或虚拟环境运行。
4. 汇编语言与高级语言混合编程的方法，重写和扩展实验二的的监控程序，从引导程序分离独立，生成一个COM格式程序的独立内核。
5. 再设计新的引导程序，实现独立内核的加载引导，确保内核功能不比实验二的监控程序弱，展示原有功能或加强功能可以工作。
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

实验内容

1. 寻找或认识一套匹配的汇编与c编译器组合。利用c编译器，将一个样板C程序进行编译，获得符号列表文档，分析全局变量、局部变量、变量初始化、函数调用、参数传递情况，确定一种匹配的汇编语言工具，在实验报告中描述这些工作。
2. 写一个汇编和c程序混合编程实例，展示你所用的这套组合环境的使用。汇编模块中定义一个字符串，调用C语言的函数，统计其中某个字符出现的次数（函数返回），汇编模块显示统计结果。执行程序可以在DOS中运行。
3. 重写实验二程序，实验二的的监控程序从引导程序分离独立，生成一个COM格式程序的独立内核，在1.44MB软盘映像中，保存到特定的几个扇区。利用汇编和c程序混合编程监控程序命令保留原有程序功能，如可以按操作选择，执行一个或几个用户程序、加载用户程序和返回监控程序；执行完一个用户程序后，可以执行下一个。
4. 利用汇编和c程序混合编程的优势，多用c语言扩展监控程序命令处理能力。
5. 重写引导程序，加载COM格式程序的独立内核。
6. 拓展自己的软件项目管理目录，管理实验项目相关文档








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

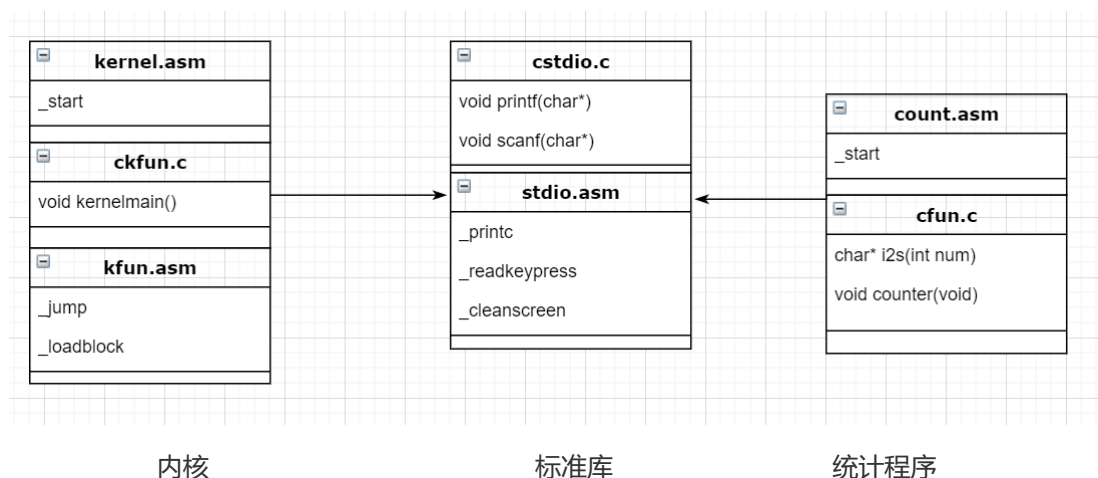
实验原理

即使是开发一个功能最简单的操作系统内核，也要大量的代码。单纯用汇编语言写是不现实的，为此我们引入了c语言，使用c语言的基本算术、逻辑运算以及控制来帮助开发。要让汇编语言和c语言能够联合编译，我们想要用各自的编译器（tcc和tasm）把它们转化成中间文件.obj，然后再用链接器

（tlink）把它们链接生成可执行程序.com文件。程序中汇编语言部分作为程序的主入口，然后跳转到c语言部分，c语言完成内核的大部分工作，汇编语言只用作BIOS调用和跳转。这样，内核功能变的更加强大，当时占用盘块也增加，不能简单地把它放在引导扇区里面。要把内核和引导程序分开，用引导程序把内核加载进内存。

代码关键部分

由于使用了联合编译，所以代码不需要都装在同一个文件里，可以充分发挥多文件的优势，增加代码的重用性。这里主要介绍内核代码和统计字符串中某一个字符出现的次数的代码以及引导程序。在屏幕四个角落反弹碰撞的代码和之前的实验基本相同。组织结构如下



标准库

这部分的代码实现了在屏幕打印字符串的printf函数，从键盘读字符串或字符的scanf函数，把屏幕清空的cleanscreen函数

stdio.asm

这里实现了在屏幕打印一个字符的_princ函数，从键盘读一个字符的_readkeypress函数，以及清空屏幕的_cleanscreen函数，cstdio.c里的printf和scanf可以调用这些函数打印和输入字符串。

首先是能够联合编译的特定格式

```
.8086;伪指令
public _princ;让其他文件可以调用这个函数
public _readkeypress
public _cleanscreen

_TEXT segment byte public 'CODE';代码段
DGROUP group _TEXT,_DATA,_BSS
assume cs:_TEXT,ds:DGROUP,ss:DGROUP

_princ proc near
;函数实现
_princ endp
_TEXT ends;代码段结束

_DATA segment word public 'DATA';数据段
```

```
_DATA ends;数据段结束
```

```
_BSS segment word public 'BSS';存放程序中未初始化的全局变量  
_BSS ends  
end
```

如果格式不对就会产生编译报错或者其他程序无法调用的错误。还有要注意的是，这是tasm的格式，对于nasm不一定适用。

_putc函数实现

```
_putc proc near  
;一个参数c为要显示的字符的偏移量,用int 10h把字符c输出,21h需要dos支持  
    push bp;把bp寄存器压栈保护  
    mov bp,sp;把栈顶指针的值赋给bp,参数的传递是用压栈进行,此时,[bp+4]就指向传进来的参数了  
    ;把函数要用到的寄存器压栈保护  
    push ax  
    push cx  
    push bx  
  
    mov bh,0;页码  
    mov ah,3;使用3号功能  
    int 10h;获取当前光标位置  
  
    mov ax,1301h;使用13h号功能,写模式为1  
    mov bx,0007h;页码为0,b1为颜色  
    add bp,4;要显示的字符串偏移量,bp=bp+4就指向要显示的字符串的偏移量  
    mov cx,1;因为只打印一个字符,所以长度为1  
    int 10h;打印字符  
  
    ;把压栈的寄存器恢复  
    pop bx  
    pop cx  
    pop ax  
    pop bp  
    ret;返回  
_putc endp
```

使用_putc时,把要显示的字符的地址传进去,然后就会在光标的当前位置显示出来,光标移动到后一位。c语言中要调用这个函数时,函数名为void putc(char*),没有下划线。在这里要注意的是不能用21h号中断,因为21h号中断需要Dos操作系统的支持,在虚拟机上无法使用。

_readkeypress函数实现

```
_readkeypress proc near;有一个返回值,返回从键盘读一个按键ascii码  
    mov ah,0  
    int 16h;检测按下的按键,ascii码在al寄存器  
    xor ah,ah  
    ret;返回  
_readkeypress endp
```

注意,在c语言中,函数名为char readkeypress(void),有一个返回值。函数的返回值是在ax寄存器,不像参数那样在栈里面。

_cleanscreen函数实现

```

_cleanSCREEN proc near
    ;把用到的寄存器压栈保护
    push bx
    push cx
    push dx
    push ax

    mov ah,6;功能号
    mov al,0;
    mov ch,0;左上角(ch,cl)
    mov cl,0
    mov dh,24;右下角(dh,d1)
    mov dl,79
    mov bh,7
    int 10h;把左上角和右下角围起来的长方形清空

    mov bh,0;页码
    mov dx,0
    mov ah,02h
    int 10h;把光标设置到(0,0)处
    ;恢复寄存器
    pop ax
    pop dx
    pop cx
    pop bx
    ret
_cleanSCREEN endp

```

调用_cleanSCREEN函数可以把屏幕清空，并且把光标设置到第一行第一列

cstdio.c

这个文件里的函数利用stdio.asm里面的函数实现更加高级的功能。

void printf(char* s)函数的实现

```

extern void printc(char c);/*声明这个函数来自外部文件*/

void printf(char *s)/*参数为要打印的字符串的首地址*/
{
    int i;
    for (i = 0; s[i] != 0; i++)/*直到读到0结束*/
    {
        printc(s[i]);
    }
}

```

这里要注意c语言调用汇编里面的函数的时候没有下划线。还有就是用tcc编译的时候，写注释好像不能用//要用/**/, 否则会报错

char* scanf(void)函数的实现

```

extern char readkeypress(void);
void scanf(char *s)/*传进一个地址，把字符串写在那里*/
{
    int i = 0;
    char c;

```

```

while (1)
{
    c = readkeypress();
    if (c == 0x0d)/*读到回车就停止*/
    {
        s[i] = 0;/*字符最后面置为0*/
        /*换行*/
        printc(0x0a);
        printc(0x0d);
        break;
    }
    s[i] = c;
    i++;
    printc(c);
}
return;
}

```

内核

kfun.asm

这个文件实现了跳转函数_jump和加载盘块的函数_loadblock

_jump函数的实现

```

_jump proc near;两个参数,要跳到的cs:ip
    push bp
    mov bp,sp

    mov ax,[bp+4];cs
    push ax
    mov ax,[bp+6];ip
    push ax
    retf;用retf跳转到目标地址
    ret

_jump endp

```

_jump函数有两个参数，一个是目标地址的段，一个是目标地址的偏移地址。在tasm中，call、jmp等无法实现段跳转，所以利用retf来跳转。retf指令执行时，先从栈中弹出ip的值，然后弹出cs的值，这样就实现了段间转移。在c语言中，函数名为void jump(int cs,int ip)，注意到cs在前面，这是因为c语言调用函数时，后面的参数先进栈。

_loadblock函数的实现

```

_loadblock proc near;有三个参数,es:bx,一个要加载的块号从1开始编号loadblock(es,bx,id)
    push bp
    mov bp,sp
    ;保护寄存器
    push ax
    push bx
    push cx
    push dx
    push es
    mov ax,[bp+4];加载地址的段地址
    mov es,ax

```

```

mov bx,[bp+6];加载地址的偏移量
mov ah,2
mov al,1
mov ch,0
mov cl,[bp+8];要加载的盘块
mov dl,0
mov dh,0
int 13h
;恢复寄存器
pop es
pop dx
pop cx
pop bx
pop ax
pop bp
ret
_loadblock endp

```

在c语言中，函数名为void loadblock(int es,int bx,id)，会把块号为id的盘块加载到es:bx处

ckfun.c

这里的kernelmain函数通过调用kfun.asm里的函数，实现了内核的大部分功能。使用了c语言的控制语句、算术逻辑运算。

```

extern int loadblock(int es, int bx, char id);
extern int jump(int cs, int ip);
extern void cleanscreen();
extern void scanf(char *s);
extern void printf(char *s);
void kernelmain()
{
    char c;
    while (1)
    {
        /*打印一个字符串*/
        printf("m for message, l to list, c to clean screen, 1-5 to select
program: ");
        scanf(&c);/*输入一个字符*/
        switch (c)
        {
            case 'm':
                printf("Designed by 18340168 wangtianlong\n\r");
                break;
            case 'l':/*打印用户程序以及所在块号*/
                printf("1.Up_Lt, block: 4\n\r");
                printf("2.Up_Rt, block: 5\n\r");
                printf("3.Dn_Lt, block: 6\n\r");
                printf("4.Dn_Rt, block: 7\n\r");
                printf("5.Count, block: 8\n\r");
                break;
            case 'c':
                cleanscreen();/*清除屏幕*/
                break;
            case '1':
                loadblock(0x900, 0x100, 4);/*加载盘块*/
                jump(0x900, 0x100);/*跳转到用户程序执行*/

```

```

        case '2':
            loadblock(0x900, 0x100, 5);
            jump(0x900, 0x100);
        case '3':
            loadblock(0x900, 0x100, 6);
            jump(0x900, 0x100);
        case '4':
            loadblock(0x900, 0x100, 7);
            jump(0x900, 0x100);
        case '5':
            loadblock(0x900, 0x100, 8);
            jump(0x900, 0x100);
        default:
            printf("error\r\n");/*非法输入*/
            break;
    }
}
}

```

kernel.asm

_start函数实现

_start是内核的入口，功能就是调用kernelmain函数

```

_start:
    mov ax,cs
    mov es,ax
    mov ds,ax
    mov ss,ax
    mov sp,100h
    call near ptr _kernelmain;跳到kernelmain函数
    jmp _start

```

统计字符串程序

cfun.c

这里实现了char* i2s(int n)函数和void counter(void)函数，前者把数字转化成字符串，后者就是统计字符串的程序

char* i2s(int n)函数的实现

```

char snum[10];/*限制字符串大小为10*/
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)
    {

```



```

    snum[i] = n % t + '0';
    n /= 10; /*n每次除于10，就可以获取每一位上的数字*/
    i++;
    s++;
}
/*通过上面获取的字符串是倒序的，要转化过来*/
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;
}

```

void kernelmain(voic)函数的实现

```

extern void scanf(char *s);
extern void printf(char* s);
void counter()
{
    char input[20];
    int i;
    char c;
    int num = 0;
    printf("Input a string: ");
    /*输入字符串*/
    scanf(input);
    /*输入一个字符，然后统计这个字符在字符串中出现的次数*/
    printf("Input a character: ");
    scanf(&c);
    for (i = 0; input[i] != 0; i++)
    {
        if (input[i] == c)
        {
            num += 1;
        }
    }
    printf("The number is: ");
    printf(i2s(num)); /*打印字符出现的次数*/
    printf("\n\r");
    return;
}

```

count.asm

这里实现了_start函数，_start的作用是跳转到counter函数，并且当counter函数执行完时，跳转回到监控程序

```

_start:
    mov ax,cs
    mov es,ax
    mov ds,ax
    mov ss,ax
    mov sp,100h
    call near ptr _counter;汇编中调用c语言中的函数时，函数名前要加下划线
    mov ax,800h
    push ax
    mov ax,100h
    push ax
    retf;返回监控程序

```

引导程序

引导程序功能很简单，就是把监控程序加载到内存，并且跳转到那里执行，用nasm编译

```

mov ax,cs
mov ss,ax
mov ds,ax
mov es,ax

mov ah,2
mov al,2
mov ch,0
mov cl,2;监控程序占了两个盘块
mov dh,0
mov dl,0
mov bx,8100h;加载到8100h处
int 13h

jmp 0x800:0x100
times 510-($-$$) db 0
db 0x55,0xaa

```

实验过程

选择编译工具以及分析符号列表

这次实验我选择的是老师提供的tcc、tasm和tlink，在DosBox里面使用。有一个样板c文件test.c，用tcc把它编译成汇编语言，然后分析。具体如下

test.c文件

```

extern int sum(int a, int b);

int global1;
int global2 = 5;

int main()
{
    int a, b = 1, c;
    a = 3;
    c = sum(a, b);
    return c;
}

```

利用tcc把test.c编译成汇编代码，生成test.asm文件

```
tcc -S test.c
```

打开test.asm文件，分析如下，具体见注释

```

    ifndef ??version
?debug macro
    endm
    endif
    ?debug S "test.c"
_TEXT segment byte public 'CODE';代码段
DGROUP group _DATA,_BSS
    assume cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT ends
_DATA segment word public 'DATA';数据段
d@ label byte
d@w label word
_DATA ends
_BSS segment word public 'BSS';BSS段
b@ label byte
b@w label word
    ?debug C E9E2B1AF5006746573742E63
_BSS ends
_DATA segment word public 'DATA';数据段
_global2 label word;全局变量global2定义在这里，并且初始化为5
    dw 5
_DATA ends
_TEXT segment byte public 'CODE';代码段，main函数
; ?debug L 6
_main proc near;注意main函数的名字加了一个下划线
    push bp
    mov bp,sp
    sub sp,2;sp减二，留出了未初始化的c的空间，[bp-2]就指向了c
    push si
    push di
; ?debug L 8
    mov di,1;局部变量b
; ?debug L 9
    mov si,3;局部变量a
; ?debug L 10
    push di;a和b是sum函数的参数，并且a在前，b在后，注意到入栈顺序是b先a后
    push si

```

```

    call    near ptr _sum;调用函数
    pop cx;把参数弹出来
    pop cx
    mov word ptr [bp-2],ax;这里可以看到ax是sum函数的返回值
;   ?debug  L 11
    mov ax,word ptr [bp-2];main函数通过ax寄存器返回c
    jmp short @1;返回return
@1:
;   ?debug  L 12
    pop di;把用过的栈恢复
    pop si
    mov sp,bp
    pop bp
    ret
_main    endp
_TEXT    ends
_BSS     segment word public 'BSS'
_global1 label word;为初始化的全局变量global1定义在这里
        db 2 dup (?)
_BSS     ends
        ?debug  C E9
_DATA    segment word public 'DATA'
s@ label byte
_DATA    ends
_TEXT    segment byte public 'CODE'
        extrn  _sum:near;说明sum函数的实现在其他文件
_TEXT    ends
        public _main;外部可以调用的函数或值
        public _global2
        public _global1
        end

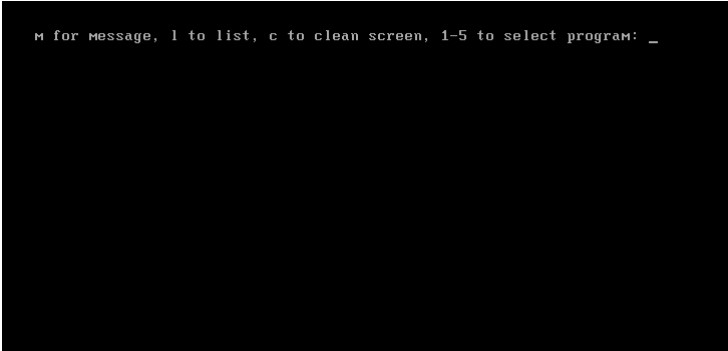
```

有几个地方要特别注意，首先是c语言的函数名在汇编里多了个下划线，全局变量是在_BSS段或者_DATA段，其中初始化的在_DATA段，未初始化的在_BSS段，而局部变量是在栈和寄存器里。调用函数时，通过栈传递参数，其中，后面的参数先进栈，前面的参数后进栈，函数返回时，返回值在ax寄存器里。

写盘并且运行

写盘式，引导程序在首扇区，监控程序在2、3扇区，用户程序在4-8号扇区，可以用I指令查看。

运行虚拟机



```

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

```

m指令

功能是打印个人信息

```
m for message, l to list, c to clean screen, 1-5 to select program: M
Designed by 18340168 wangtianlong
m for message, l to list, c to clean screen, 1-5 to select program: S
error
m for message, l to list, c to clean screen, 1-5 to select program: _
```

指令

功能是列出用户程序编号以及所在盘块

```
M for message, l to list, c to clean screen, 1-5 to select program: M
Designed by 18340168 wangtianlong
M for message, l to list, c to clean screen, 1-5 to select program: S
error
M for message, l to list, c to clean screen, 1-5 to select program: l
1.Up_Lt, block: 4
2.Up_Rt, block: 5
3.Dn_Lt, block: 6
4.Dn_Rt, block: 7
5.Count, block: 8
M for message, l to list, c to clean screen, 1-5 to select program: _
```

运行统计字符串程序，输入5

```
S count, back: 0
M for message, l to list, c to clean screen, 1-5 to select program: 5
Input a string: akdfjkoisdka
Input a character: a
The number is: 2
M for message, l to list, c to clean screen, 1-5 to select program: S_
```

c指令

功能是清除屏幕

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

运行字符反弹程序，输入1

[illegible]

实验总结

通过这次试验，学习到了如何联合编译c语言和汇编，学到了一些关于编译原理的知识。从代码到可执行文件主要有2个过程

1. 编译器把代码编译成中间代码.obj
2. 链接器把.obj链接起来生成可执行文件

在使用tcc和tasm过程中，遇到了大小小很多bug，这里记录一下，避免以后再犯

1. c语言的注释要用/**/
2. jmp不能实现段间跳转，要用retf
3. mov bx,offset string不能正确获取string的偏移量，要用lea bx,string
4. 汇编调用c语言的函数或变量时，名字前要加下划线
5. 汇编要注意格式
6. 虚拟机不能使用int 21h中断，因为21h号中断需要Dos系统支持

参考资料

<https://blog.csdn.net/BXD1314/article/details/38433837>

https://www.viseator.com/2018/07/02/80x86_asm_c/

<https://blog.csdn.net/songjinshi/article/details/6554187>