实验五报告

by 12348006 蔡燕芝 12级计科1班

# 一．系统调用表设计如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 功能号 | 入口参数 | 出口参数 | 功能描述 |  |
| 0 | Ah = 0 | 无 | 在屏幕中心显示 “OUCH” |  |
| 1 | Ah=1;  dx=串首偏移；  es=串段址； | 无 | 将es:dx位置的一个字符串中的小字母变为大字 |  |
| 2 | Ah=2;  dx=串首偏移；  es=串段址； | 无 | 将es:dx位置的一个字符串中的大字母变为小字 |  |
| 3 | Ah=3;  dx=串首偏移；  es=串段址； | Ax=数值 | 将es:dx位置的一个数字字符串转变对应的数值 |  |
| 4 | Ah=4;  Bx=数值  dx=串首偏移；  es=串段址； | 无 | 将bx的数值转变对应的es:dx位置的一个数字字符串 |  |
| 5 | Ah=5;  Cx=ch:行号cl:列号  dx=串首偏移； | 无 | 将es:dx位置的一个字符串显示在屏幕指定位置(ch:行号cl:列号) |  |
| 6 | Ah = 6  dx=串首偏移；  es=串段址； | Ax = 字符串长度 | 读入字符串，放在es：dx位置 |  |
| 7 | AH = 7 | Ax = 字符 | 读入一个字符放在ax |  |
| 8 | AH= 8 |  | 在屏幕的最下方中间  显示当前时间 |  |
| 9 | Ah = 9  dx=串首偏移；  es=串段址； | 无 | 将es:dx位置的字符串显示在屏幕 |  |

# 二.上述系统调用的实现：

上述系统调用的中断服务例程安装在33号中断上面。这些功能在kernal中调用正常。

kliba中安装int33

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; void setInt33()

; ; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; 33号中断的安装，实现自己的系统能调用

public setint33

setint33 proc

push ax

push es

mov ax, cs

push ax

mov ax, 33

push ax

mov ax, offset int33

push ax

call \_setNewInt

pop ax

pop ax

pop ax

pop es

pop ax

ret

int33 ;33号中断

int33Begin:

push bp

push bx

push cx

push dx

push es

push ds

cmp ah, 0 ;根据功能号实现对应的功能

je int33\_00

cmp ah, 1

je int33\_01

cmp ah, 2

je int33\_02

cmp ah, 3

je int33\_03

cmp ah, 4

je int33\_04

cmp ah, 5

je int33\_05

cmp ah, 6

je int33\_06

cmp ah, 7

je int33\_07

cmp ah, 8

je int33\_08

cmp ah,9

je int33\_08

int33\_00: ;0号功能：在屏幕中间显示"OUCH"

call int33\_00\_1

jmp int33end

int33\_01: ;1号功能: 将es:dx位置的一个字符串中的小字母变为大字

call int33\_01\_1

jmp int33end

….其他的省略功能号详见kliba.asm

…

具体的功能号实现如下：

int33\_00\_1:

mov ax, 0b800h

mov es,ax

mov di, (80\*13+38)\*2

mov ah, 0Fh

mov al, 'O'

mov word ptr es:[di], ax

mov al, 'U'

mov word ptr es:[di+2], ax

mov al, 'C'

mov word ptr es:[di+4], ax

mov al, 'H'

mov word ptr es:[di+6], ax

ret

int33\_01\_1:

mov ax, es

mov ds, ax

push dx

call \_upper

pop ax

ret

myInt33.c中为int33功能提供外部函数：

/\*程序源代码（myInt33.c）\*/

/\*time: 2014年4月19日14:49:19

/\*Auth: 蔡燕芝

/\*Describe: 实现33号中断的 1 2 3 4 功能号的子程序\*/

/\*将一个字符串中的小字母变为大写字母\*/

void myprintf(char \*Message);

void upper(char \*Message){

int i=0;

while(Message[i]) {

if (Message[i]>='a'&&Message[i]<='z')

Message[i]=Message[i]+'A'-'a';

i++;

}

}

/\*将一个字符串中的大字母变为小写字母\*/

void lowwer(char \*Message)

{

int i=0;

while(Message[i]) {

if (Message[i]>='A'&&Message[i]<='Z')

Message[i]=Message[i]-'A'+'a';

i++;

}

}

/\*数值num变成对应的数字字符串存储在Message中\*/

void int2Chars(char\* Message, int num)

{

int index = 0;

int i = 0;

int result[15];

while(num)

{

result[index++] = num%10;

num /= 10;

}

for(i = 0; i < index ; i++)

Message[i] = result[index - 1 - i] + '0';

Message[index] = '\0';

}

/\*将一个数字字符串转变对应的数值\*/

int chars2Int(char \*Message)

{

int i=0;

int result = 0;

while(Message[i]) {

if(i!=0)

result \*= 10;

result += (Message[i] - '0');

i++;

}

return result;

}

# 三．设计C程序库，实现系统调用的封装，并开发一个测试系统调用所有功能的C程序。

**我将int 33的0-9号功能封装成在userlib.asm 中，以供其他程序调用**

**userlib.asm 如下：**

**; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**; userlib.asm**

**; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**;\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**; \*SCOPY@ \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**; 实参为局部字符串带初始化异常问题的补钉程序**

**public SCOPY@**

**SCOPY@ proc**

**arg\_0 = dword ptr 6**

**arg\_4 = dword ptr 0ah**

**push bp**

**mov bp,sp**

**push si**

**push di**

**push ds**

**lds si,[bp+arg\_0]**

**les di,[bp+arg\_4]**

**cld**

**shr cx,1**

**rep movsw**

**adc cx,cx**

**rep movsb**

**pop ds**

**pop di**

**pop si**

**pop bp**

**retf 8**

**SCOPY@ endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* void \_myShowOuch() \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myShowOuch**

**\_myShowOuch proc**

**push ax**

**mov ah, 0**

**int 33**

**pop ax**

**ret**

**\_myShowOuch endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* void \_myUpper(char \*pChars) \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myUpper**

**\_myUpper proc**

**push bp**

**mov bp,sp**

**push dx**

**push ax**

**mov ax, ds**

**mov es, ax**

**mov dx, [bp+4]**

**mov ah, 1**

**int 33**

**pop ax**

**pop dx**

**pop bp**

**ret**

**\_myUpper endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* void \_myLowwer(char \*pChars) \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myLowwer**

**\_myLowwer proc**

**push bp**

**mov bp,sp**

**push ax**

**push dx**

**mov ax, ds**

**mov es, ax**

**mov ah, 2**

**mov dx, [bp+4]**

**int 33**

**pop dx**

**pop ax**

**pop bp**

**ret**

**\_myLowwer endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* int \_myChars2Int(char \*pChars) \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myChars2Int**

**\_myChars2Int proc**

**push bp**

**mov bp,sp**

**push dx**

**mov ax, ds**

**mov es, ax**

**mov ah, 3**

**mov dx, [bp+4]**

**int 33**

**pop dx**

**pop bp**

**ret**

**\_myChars2Int endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* void \_myInt2Chars(int num, char \*pChars) \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myInt2Chars**

**\_myInt2Chars proc**

**push bp**

**mov bp,sp**

**push dx**

**push bx**

**push ax**

**mov ax, ds**

**mov es, ax**

**mov ah, 4**

**mov bx, [bp+4]**

**mov dx, [bp+6]**

**int 33**

**pop ax**

**pop bx**

**pop dx**

**pop bp**

**ret**

**\_myInt2Chars endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* void myPrintCharsInPosition(int row,int col, char \*pChars) \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myPrintCharsInPosition**

**\_myPrintCharsInPosition proc**

**push bp**

**mov bp,sp**

**push ax**

**push dx**

**push cx**

**mov ax, ds**

**mov es, ax**

**mov ah, 5**

**mov ch, [bp+4]**

**mov cl, [bp+6]**

**mov dx, [bp+8]**

**int 33**

**pop cx**

**pop dx**

**pop ax**

**pop bp**

**ret**

**\_myPrintCharsInPosition endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* char myGetChar() \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myGetChar**

**\_myGetChar proc**

**push bp**

**mov ah, 7**

**int 33**

**pop bp**

**ret**

**\_myGetChar endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* int myGetChars(char \*pChars) \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myGetChars**

**\_myGetChars proc**

**push bp**

**mov bp,sp**

**push dx**

**mov ax, ds**

**mov es, ax**

**mov dx, [bp+4]**

**mov ah, 6**

**int 33**

**pop dx**

**pop bp**

**ret**

**\_myGetChars endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* void myShowTime() \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myShowTime**

**\_myShowTime proc**

**push bp**

**mov bp,sp**

**push ax**

**mov ah, 8**

**int 33**

**pop ax**

**pop bp**

**ret**

**\_myShowTime endp**

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;\* void myprintf() \***

**;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**;**

**public \_myprintf**

**\_myprintf proc**

**push bp**

**mov bp,sp**

**push ax**

**push dx**

**mov ax, ds**

**mov es, ax**

**mov ah, 9**

**mov dx, [bp+4]**

**int 33**

**pop dx**

**pop ax**

**pop bp**

**ret**

**\_myprintf endp**

**而测试int33所提供的服务，我用c实现，在user1.c中定义了以下的外部函数调用userlib.asm的封装，效果在第四部分中展示**

**extern void myShowOuch();**

**extern void myUpper(char \*pChars);**

**extern void myLowwer(char \*pChars);**

**extern int myChars2Int(char \*pChars);**

**extern void myInt2Chars(int num, char \*pChars);**

**extern void myPrintCharsInPosition(int row,int col, char \*pChars);**

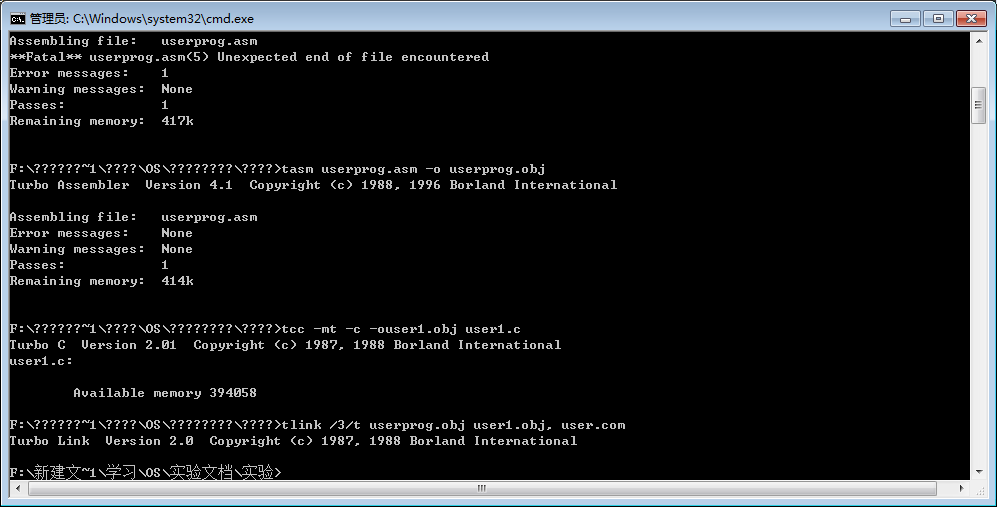
**extern char myGetChar();**

**extern int myGetChars(char \*pChars);**

**extern void myShowTime();**

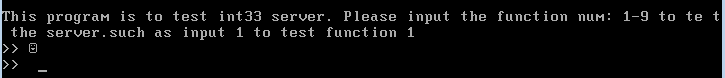
**extern void myprintf(char \*pChars);**

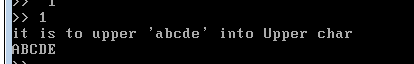
**生成测试文件user.com的过程：**

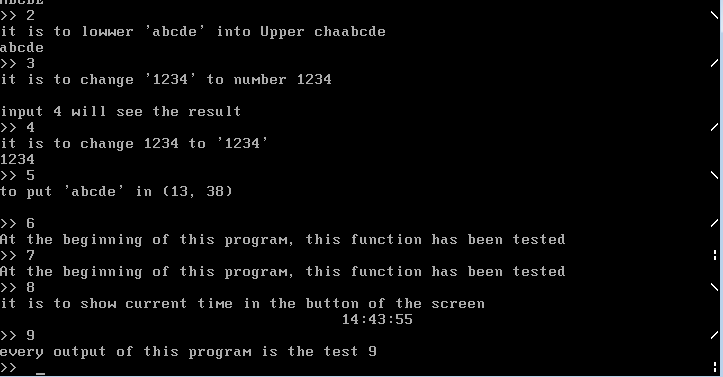


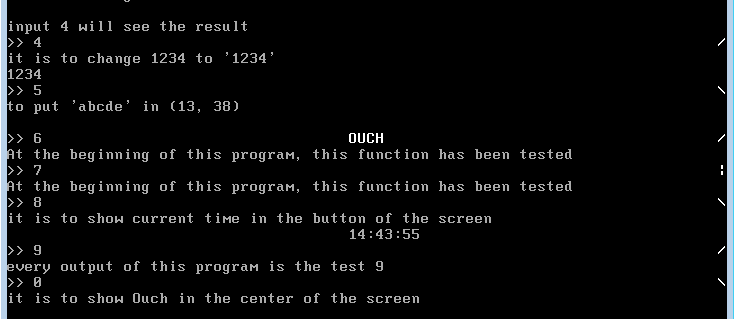
# 四．效果

**进入内核后，跳入run程序的分支，键入run a来查看效果，输入详看效果图**









# 五．体会

**实现系统调用的内容很容易，只需安装好中断就好。但是将系统调用封装成c库，我一开始总以为是要弄成一个obj文件弄成lib文件以后直接在c中include头文件然后使用，折腾了特别久特别久，用了tlib生成了lib文件，但是用tcc编译c文件后，放到虚拟机中总是进不到程序里面去，最后弄得特别烦就放弃了这种方法。后来直接像写内核那样子，去用asm写了一个userlib文件然后直接和c文件混合汇编，结果问题也解决了。在实验过程中，出现过一次调用九号功能时字符串连续打印了多次，令人匪夷所思，后来仔细分析了代码后发现在在系统调用后忘记添加了ret这个语句，虽然至今不知道为什么不返回主程序会导致字符串多次打印，但是经过这次纠错，深刻明白了汇编语言的难点，就是有些错误不像理解c的问题那么简单。说到底，还是要多多思考，多多犯错，培养一下汇编编写的感觉。**