第6章 Windows编程

6.1 操作系统函数调用

6.2 控制台应用程序

6.3 图形窗口应用程序

第6章 Windows编程

- 熟悉汇编语言调用API函数的方法
- 掌握控制台输入输出函数
- 熟悉MASM的高级特性
- 理解Windows图形窗口程序的编写

6.1 操作系统函数调用

- 操作系统以其提供的系统函数（系统功能System function）支持程序员进行编程
- Windows的系统函数（功能）以动态连接库DLL（Dynamic-Link Library）形式提供，利用其应用程序接口API（Application Program Interface）调用DDL库中的函数
- API是一些类型、常量和函数的集合，提供了编程中使用库函数的途径
  - Win16：16位Windows的API
  - Win32：32位Windows的API

6.1.1 动态连接库

- 静态连接：连接程序从库文件中抽取需要的子程序插入到最终的可执行代码中
- 动态连接：程序运行时才将代码加载到主存
- 动态连接库：保存程序运行时需要重复使用的代码的文件
- 3个最重要的Windows动态连接库
  - KERNEL32.DLL：主要处理内存管理和进程调度
  - USER32.DLL：主要控制用户界面
  - GDI32.DLL：负责图形方面的操作
- 导入库（Import Library）：程序开发的连接阶段使用，与一个动态连接库DLL对应

6.1.2 MASM的过程声明和调用

- 过程声明伪指令PROTO：

  事先声明过程的结构

  （包括操作系统API函数、高级语言的函数）

  过程名 PROTO [调用距离] [语言类型]

  [,[参数]：类型]...

- 过程调用伪指令INVOKE

  实现过程调用

  INVOKE 过程名[,参数,...]

6.1.3 程序退出函数

- Win32程序员参考手册

```
VOID ExitProcess(
    UINT uExitCode      // exit code for all threads
   );
```

- 汇编语言声明

```
ExitProcess PROTO ,:DWORD
```

- 汇编语言调用

```
invoke ExitProcess,0
```

- 将函数调用定义成宏

```
exit  MACRO dwexitcode
      invoke ExitProcess,dwexitcode
      ENDM
```

- 宏调用

```
exit  0
```

6.2 控制台应用程序

- Windows应用程序开始运行
  - 创建控制台（Console）窗口
  - 或创建图形界面窗口
- 32位Windows控制台程序
  - 像增强版的MS-DOS程序
  - 使用标准控制台

    标准输入设备（键盘）

    标准输出设备（显示器）
  - 32位控制台程序运行在保护方式
  - 通过API使用Windows的动态链接库函数

6.2.1 控制台输出

- 编写控制台程序需要调用控制台函数，实现
  - 控制台（显示器）输出
  - 控制台（键盘）输入
- 几乎所有的控制台函数都要求将控制台句柄作为第一个参数传递给它们
- 本节介绍的控制台函数
  - 存在于KERNEL32.DLL动态库中
  - 程序开发需要使用KERNEL32.LIB导入库文件

1. 控制台句柄

- 句柄(Handle)是一个32位无符号整数
  - 用来唯一确定一个对象
  - 例如某个输入设备、输出设备或者一个图形
- 标准输入句柄

```
STD_INPUT_HANDLE   equ -10
```

- 标准输出句柄

```
STD_OUTPUT_HANDLE equ -11
```

- 标准错误句柄

```
STD_ERROR_HANDLE   equ -12
```

- GetStdHandle函数
  - 获取控制台输入或输出的句柄实例
  - 用于控制台程序的输入输出操作

2. 控制台输出函数

- 显示器输出API函数WriteConsole
  - 将一个字符串输出到屏幕上
  - 支持标准的ASCII控制字符，例如回车、换行等
- Win32 API中可以使用两种字符集
  - 8位ASCII字符集，函数名以字母A结尾
  - 16位Unicode字符集，函数名以字母W结尾
- WriteConsole参数
  - 第一个：控制台输出句柄实例
  - 第二个：指向字符串的指针、即缓冲区地址
  - 第三个：指明字符串长度，是一个32位整数
  - 第四个：指向一个整数变量，返回实际输出的字符数
  - 第五个：保留，设置为0

〔例6-1〕控制台输出程序一1

```
                .686
                .model flat,stdcall
                option casemap:none
                includelib bin\kernel32.lib
ExitProcess     proto,:dword
GetStdHandle    proto,:dword
WriteConsoleA proto,:dword,:dword,:dword,:dword,:dword
WriteConsole    equ <WriteConsoleA>
STD_OUTPUT_HANDLE equ -11
                .data
msg             db 'Hello, Assembly!',13,10
count           equ  $-msg
outsize         dd 0
```

〔例6-1〕控制台输出程序一2

```
        .code
start:
        ;获得输出句柄
        invoke GetStdHandle,STD_OUTPUT_HANDLE
        ;显示信息
        invoke WriteConsole,eax,offset msg,count,offset outsize,0
        ;退出
        invoke ExitProcess,0
        end start
```

```
Hello, Assembly !
```

运行结果

6.2.2 控制台输入

- 键盘输入API函数ReadConsole
  - 将键盘输入的文本保存到一个缓冲区
  - 第一个：控制台输入句柄实例
  - 第二个：输入缓冲区指针
  - 第三个：要读取字符的最大数量
  - 第四个：实际输入字符数量的指针
  - 第五个：未使用，设置为0
- 调用ReadConsole函数时
  - 系统等待用户输入、并回车确认

（例如用户输入了3个字符，依次是123）

  - 第4个参数的变量保存用户输入字符个数再加2的结果

（例如本例是5）

（内容用十六进制数表达依次是31 32 33 0D 0A）

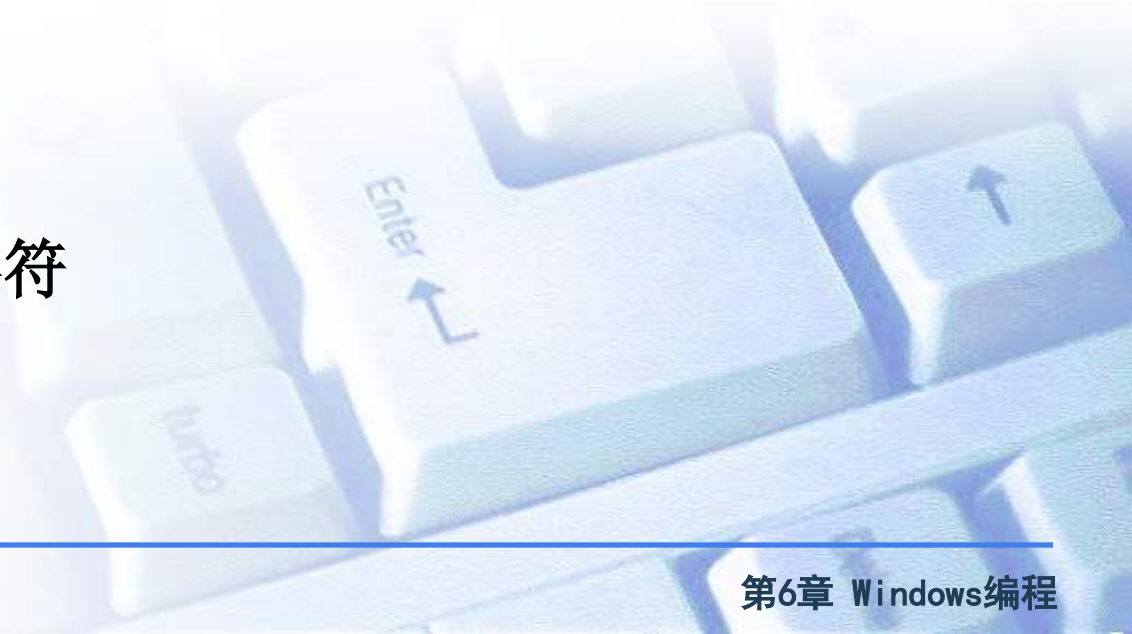
定义输入缓冲区
要多留两个字节

〔例6-2〕信息输入输出程序一1

```
                .686
                .model flat,stdcall
                option casemap:none
                includelib bin\kernel32.lib
ExitProcess     proto,:dword
exit            MACRO dwexitcode
                invoke ExitProcess,dwexitcode
                ENDM
GetStdHandle    proto,:dword
WriteConsoleA proto,:dword,:dword,:dword,:dword,:dword
WriteConsole    equ <WriteConsoleA>
```

〔例6-2〕信息输入输出程序—2

```
ReadConsoleA proto,:dword,:dword,:dword,:dword,:dword
ReadConsole  equ <ReadConsoleA>
STD_INPUT_HANDLE    equ -10
STD_OUTPUT_HANDLE   equ -11
                .data
msg1            db 'Please enter your name: ',0
msg2            db 'Welcome ',0
nbuf            db 80 dup(0)
msg3            db ' to Win32 Console!',0
                ;设置输入缓冲区最大255个字符
```



〔例6-2〕信息输入输出程序—3

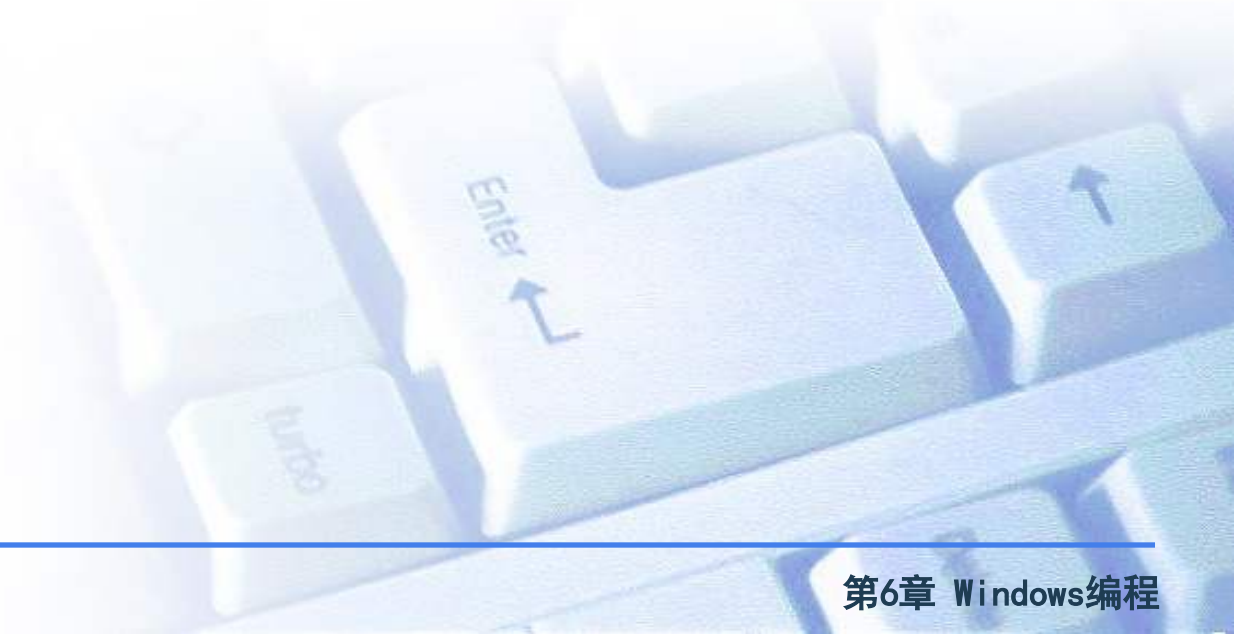
```
        .code
start:  mov eax,offset msg1     ;提示输入
        call dispmsg
        mov eax,offset nbuf     ;输入信息
        call readmsg
        mov eax,offset msg2
        call dispmsg
        mov eax,offset nbuf     ;显示输入信息
        call dispmsg
        mov eax,offset msg3
        call dispmsg
        exit 0
```

〔例6-2〕信息输入输出程序—4

```
            .data
_outsize    dd 0
_outhandle  dd 0
dispmsg     proc                    ;字符串显示子程序
            push eax                ;入口参数：EAX＝字符串地址
            push ebx
            push ecx
            push edx
            push eax                ;保存入口参数，即字符串地址
            invoke GetStdHandle,STD_OUTPUT_HANDLE
            mov [_outhandle],eax
            pop ebx                 ;EBX＝字符串地址
            xor ecx,ecx             ;计算字符串长度
```

〔例6-2〕信息输入输出程序—5

```
dispm1:   mov al,[ebx+ecx]
          test al,al
          jz dispm2
          inc ecx
          jmp dispm1
dispm2:   invoke WriteConsole,[_outhandle],ebx,ecx,offset _outsize,0
          pop edx
          pop ecx
          pop ebx
          pop eax
          ret
dispmsg   endp
```



〔例6-2〕信息输入输出程序—6

```
            .data
_insize     dd 0
_inbuffer   db 255 dup(0)
readmsg     proc                ;字符串输入子程序
            push ebx            ;入口参数：EAX＝缓冲区地址
            push ecx
            push edx
            push eax            ;保护输入的缓冲区地址参数
            invoke GetStdHandle,STD_INPUT_HANDLE
            invoke ReadConsole,eax,offset _inbuffer,255,offset _insize, 0
            sub dword ptr [_insize],2
            xor ecx,ecx
            pop ebx             ;获得缓冲区地址
```

〔例6-2〕信息输入输出程序—7

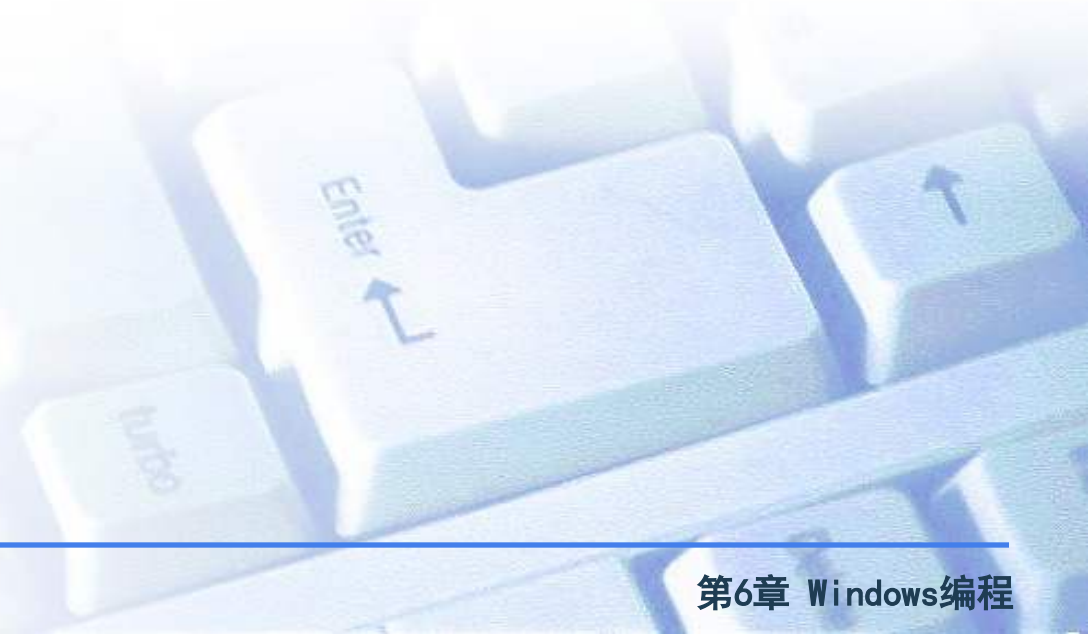
```
readm1:   mov al,_inbuffer[ecx]
          mov [ebx+ecx],al
          ;将输入的字符串复制到用户缓冲区
          inc ecx
          cmp ecx,[_insize]
          jb readm1
          mov byte ptr [ebx+ecx],0   ;最后填入结尾字符0
          mov eax,ecx
          pop edx
          pop ecx
          pop ebx
          ret
readmsg   endp
```

```
Please enter your name: Jerry
Welcome Jerry to Win32 Console!
```

运行结果

6.2.3 单字符输入

- 默认模式下
  - ReadConsole函数实现一行字符输入
  - 最后必须用回车键结束
- 设置为单字符输入模式后
  - ReadConsole函数实现一个字符输入
  - 自动结束调用
- 需要修改输入模式
  - 获取控制台模式函数GetConsoleMode
  - 设置控制台模式函数SetConsoleMode



〔例6-3〕单字符输入程序－1

```
                ...
GetConsoleMode proto,:dword,:dword
SetConsoleMode proto,:dword,:dword
                .data
msg             db 'Press any key to end'
count           equ $-msg
outsize         dd 0
inhandle        dd 0
savemode        dd 0                    ;保存控制台模式
insize          dd 0
inbuffer        db 255 dup(0)
                ;设置输入缓冲区最大255个字符
```

〔例6-3〕单字符输入程序一2

```
        .code
start:
        invoke GetStdHandle,STD_OUTPUT_HANDLE
        ;提示按任意键
        invoke WriteConsole,eax,offset msg,count,offset outsize,0
        invoke GetStdHandle,STD_INPUT_HANDLE
        mov [inhandle],eax
        invoke GetConsoleMode,[inhandle],offset savemode
        ;获得控制台模式
        invoke SetConsoleMode,[inhandle],0
        ;设置为单字符输入模式
```

〔例6-3〕单字符输入程序一3

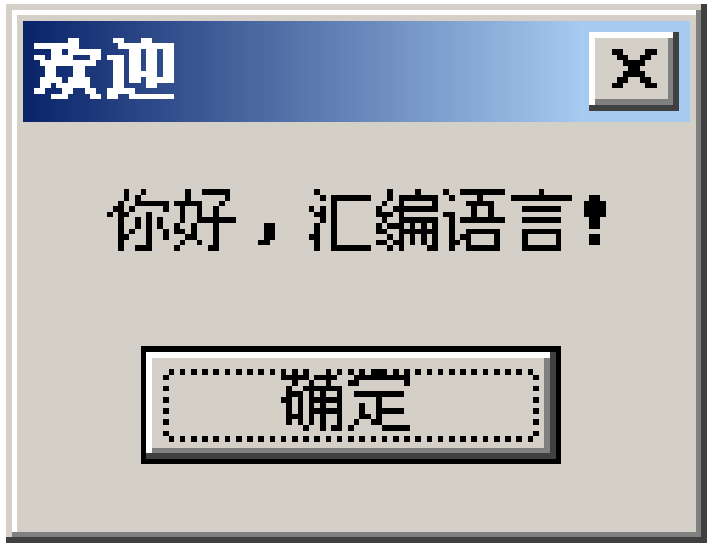
```
invoke ReadConsole,[inhandle],offset inbuffer,1,offset insize,0
;输入字符
invoke SetConsoleMode,[inhandle],[savemode]
;恢复原控制台模式
invoke ExitProcess,0                    ;退出
```

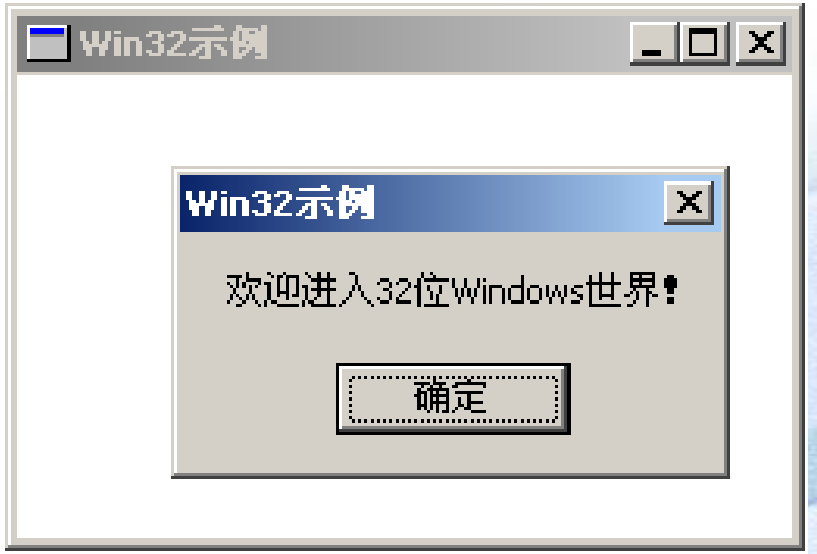
```
Press any key to end
```

运行结果

6.3 图形窗口应用程序

- Windows图形界面以窗口、对话框、菜单、按钮等实现用户交互
- 用汇编语言编写图形窗口应用程序就是调用这些API函数





6.3.1 消息窗口

- 消息窗口是常见的Windows图形窗口显示形式
  - 使用MessageBox函数
  - Win32程序员参考手册的定义

```
int MessageBox(
    HWND hWnd,
    LPCTSTR lpText,
    LPCTSTR lpCaption,
    UINT uType
    );
```

  - hWnd：父窗口的句柄
  - lpText：要显示字符串的地址指针，字符串的首地址
  - lpCaption：消息窗标题的地址指针
  - uType：指明该消息窗的类型

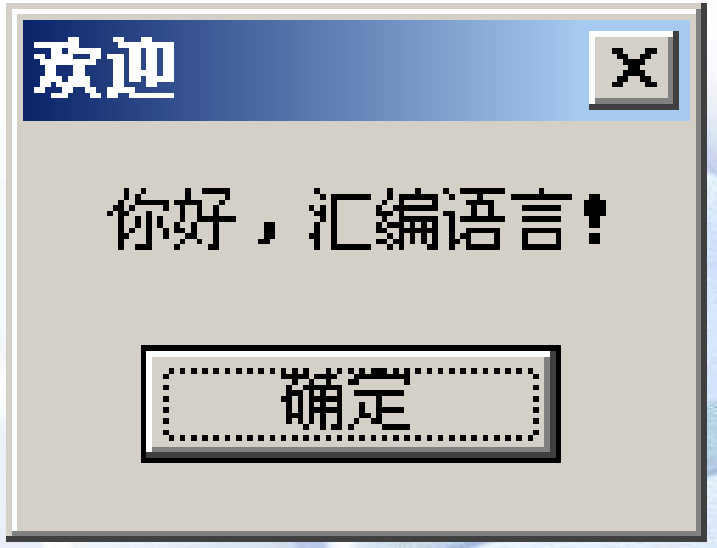
〔例6-4〕消息窗口程序一1

```
                .686
                .model flat,stdcall
                option casemap:none
                includelib bin\kernel32.lib
                includelib bin\user32.lib
ExitProcess     proto,:dword
MessageBoxA     PROTO,:dword,:dword,:dword,:dword
MessageBox      equ <MessageBoxA>
NULL            equ 0
MB_OK           equ 0
                .data
szCaption       db  '欢迎',0
outbuffer       db  '你好，汇编语言！',0
```

〔例6-4〕消息窗口程序一2

```
        .code
start:
        invoke MessageBox,NULL,offset outbuffer,offset szCaption,MB_OK
        invoke ExitProcess,NULL
        end start
```

连接时应该使用参数
/subsystem:windows



6.3.2 结构变量

- 结构类型的说明

  结构名 STRUCT

  ……　　;数据定义语句

  结构名 ENDS

- 结构变量的定义

  变量名　结构名　〈字段初值表〉

- 结构变量及其字段的引用
  - 引用结构变量，只要直接书写结构变量名
  - 引用其中某个字段，采用圆点“.”操作符

  结构变量名.结构字段名

〔例6-5〕系统时钟显示程序—1

```
                    ;系统时间的结构类型说明
SYSTEMTIME      struct
   wYear        dw 0          ;年（4位数）
   wMonth       dw 0          ;月（1～12）
   wDayOfWeek   dw 0          ;星期（0～6，0＝星期日，……）
   wDay         dw 0          ;日（1～31）
   wHour        dw 0          ;时（0～23）
   wMinute      dw 0          ;分（0～59）
   wSecond      dw 0          ;秒（0～59）
   wMillisconds dw 0          ;毫秒（0～999）
SYSTEMTIME      ends
                    ;函数声明，参数是指向结构变量的指针
GetLocalTime    proto,:dword
```

〔例6-5〕系统时钟显示程序—2

```
                .data
mytime          SYSTEMTIME <>      ;系统时间的结构变量定义
timestring      db '--:--:--',0
timecaption     db '当前时间',0
                .code
start:          invoke GetLocalTime,offset mytime
                ;获得当前时间
                mov ebx,offset timestring
                ;EBX指向“时”的保存位置
                writedec [mytime.wHour]   ;转换为ASCII字符
                …
                invoke MessageBox,0,offset timestring,offset timecaption,1
```

writedec