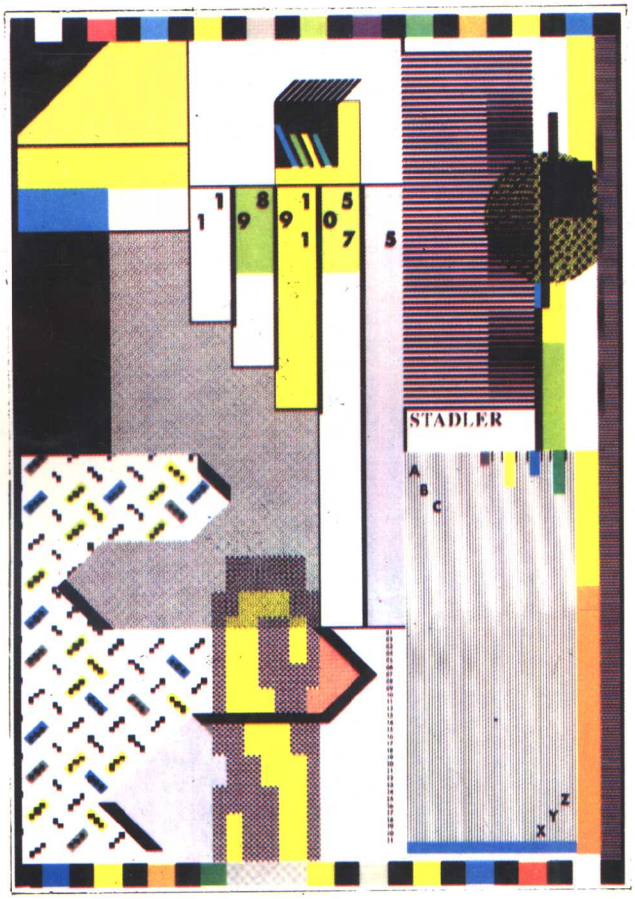
高　青　编著
秦作睿　陈篆生　审校



MS WINDOWS 3.X

软件开发实用教程

中国铁道出版社

MS WINDOWS 3. X

软件开发实用教程

高青　编著
秦作睿　陈箓生　审校

中　国　铁　道　出　版　社
1 9 9 5年·北京

（京）新登字 063 号

内 容 简 介

本书详细介绍了使用 C 语言编写 Windows 应用程序的方法。全书共分三部分：第一部分重点介绍如何编写 Windows 程序的 I/O 部分，如菜单、对话框、显示和打印输出。第二部分介绍映射、字体、绘图、鼠标、加速键、图标和图元文件等 Windows 编程技术。第三部分介绍内存、文件、打印设置、数据交换等有关技术。书中各章均配有示例程序（第十四章除外），且示例中有较详细的中文注解，方便读者理解掌握 Windows 的编程技术。

本书是从事 Windows 程序设计人员的必备参考书，也可做为大、中专计算机专业学习 Windows 编程的参考书。

MS WINDOWS 3. X 软件开发实用教程

高青 编著

秦作睿 陈箓生 审校

*

中国铁道出版社出版发行

（北京市东单三条 14 号）

责任编辑 刘 波 封面设计 陈东山

各地新华书店经售

中国铁道出版社印刷厂印

开本：787×1092 毫米 1/16 印张：14.75 字数：356 千

1995 年 4 月 第 1 版 第 1 次印刷

印数：1— 2000 册

ISBN7-113-01909-9/TP・200 定价：18.00 元

前　言

随着面向对象编程技术的广泛应用，目前市面上有很多书籍向读者介绍如何使用C++（如MSC++或Borland C++、Virsual BASIC或Virsual C等语言开发Windows应用程序。这些开发工具，对于有经验的Windows开发人员来说是非常有用的，因为程序员只需定义对象及它们的行为，具体的实现可由开发系统来完成。但由于Windows是一个以消息（事件）驱动的系统。因此在编程的技术上与传统的DOS或UNIX等操作系统有很大不同。根据作者本人的经验，如果读者不真正了解Windows程序的基本编程方法就直接使用那些高级的编程技术，不但不能提高编程效率，反而会事得其反。因为如果不了解Windows程序的运行（执行）原理，将无法解释在应用系统开发过程中出现的各种问题。本书编写的目的就是由浅入深地介绍各种Windows编程技术的实现原理和使用方法，在掌握了这些Windows编程的基本技术后，读者就可以使用高级的面向对象的编程技术来开发自己的Windows应用程序。

本书在章节内容的安排上考虑到一般程序员开发程序时对所需技术的要求，所以在第一部分介绍Windows程序的基本原理时，第一章重点介绍Windows程序的执行原理，使读者首先对Windows程序的控制转移有所了解，然后在第二章讨论菜单的使用，这样读完第二章读者就立刻可以为自己的应用程序建立基本的运行框架（因为大多数程序都是使用菜单来控制程序完成某一功能）。在第三章中，重点解决如何实现数据的输入（即对话框的使用），第四章介绍如何在程序的窗口中输出字符信息，而第五章介绍如何在打印机上输出信息。读者读完第一部分后，就基本上掌握了Windows输入、输出的基本编程技术，可以开始编写Windows的应用程序了。

第二部分介绍Windows的中级编程技术，包括：

第六章讨论键盘的使用；第七章介绍映射与字体；第八章讨论绘图的基本技术；第九章介绍鼠标的使用方法，第十章讨论加速键、图标资源和图元文件的使用方法。

读者在开发程序过程中可根据需要要阅读有关章节。

第三部分讨论了Windows编程中的一些高级技术。第十一章介绍在Windows中使用动态内存和文件的方法；第十二章介绍控制打印效果的方法；第十三、第十四章讨论Windows程序数据交换的两种常用方法。

虽然Windows3.1兼容Windows3.0但在某个技术的实现上Windows3.1则提供了一些更先进的手段。本书主要介绍使用Windows3.1提供的技术实现同一目的的方法。

在Windows3.1中系统还提供了对多媒体、连接与嵌入对象等多方面的支持，由于这些技术涉及到其它相关的技术，其内容相当广泛，因此在本书中没有讨论。作者准备在另一本书中专门论述这些技术。同时动态连接库属于编译系统讨论的内容，因此本书也没有讨论。

虽然各种开发软件都为开发人员提供一个调试工具，但对于相对较小的问题，程序员常习惯于在程序中增加一些调试用的显示语句来了解程序运行过程中某些变量的值。在调试Windows程序时，如要在程序中增加一些调试用的显示语句，不能象Dos环境那样只简单地加上

printf，必须使用第四章介绍的有关技术，同时还必须注意要显示的内容出现在窗口中的位置（如果显示的内容位置超出窗口范围，由于系统不会自动滚屏，因此将看不到输出结果）。这点请读者要特别注意。

本书各章均配有示例程序，全部包含在本书的软盘中，读者可以选择购买。读者可用下面的 DOS 命令将程序拷到硬盘上。

```
XCOPY A:  C:\  /S
```

或：

```
XCOPY B:C:\/S
```

上述命令在用户硬盘的根目录上建立一个 WINJC 子目录，在该子目录下本书的每章均有一个子目录，如：C1。

注意：XCOPY 是 DOS3.0 以上版本提供的外部命令。

读者可使用 Microsoft C++7.0 或 Borland C++3.1 编译、连接各章的示例程序。本书为 MSC、Borland C 分别提供了 make 文件。MSC 的 make 文件名以 m 开头，例如下面的命令为第一章的示例生成可执行程序。

```
nmake msan1
```

Borland C 的 make 文件以 b 开头，下例是使用 Borland C 为第一章示例生成可执行文件的方法。

```
make  -fbsam1
```

由于时间仓促，作者水平有限，书中难免有缺点和错误，欢迎广大读者给予批评和指正。

本书的出版得到了北方交通大学秦作睿教授、陈箓生教授的大力支持，在此表示衷心的感谢！

作　者

1994 年 7 月于深圳

目　录

第一部分　Windows 编程入门

第一章　Windows 的程序结构分析、运行原理和程序基本框架

1.1　Windows 简介

Microsoft 公司自 1990 年 5 月推出 MS Windows3.0 后，又于 1992 年 4 月推出了功能更强的 MS　Windows3.1，从此 MS　Windows 以其生动友好的用户图形界面、方便的操作方式和可以在不同的硬件环境下运行等特点，成为当今 PC 机操作系统的主流。

概括起来 MSW indows3.x 有以下主要特点：

1. MSWindows 一改 MSDOS 字符用户界面呆板的面孔，采用了图形用户界面，因此用户面对的不再是几个毫无生气的不容易理解其含义的字符，而是丰富多采的一看就知其含义的代

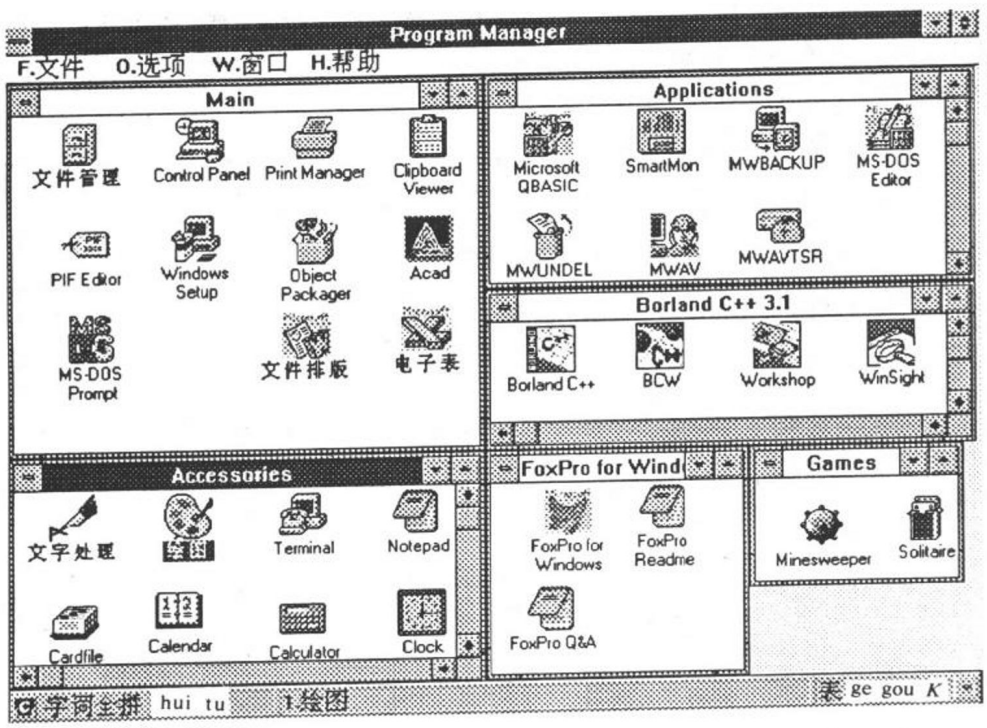


图　1.1

表该程序的图标。例如一个文字处理程序可用一支钢笔形象地表示(见图 1.1)。这样用户通过代表某个程序的图标了解该程序的功能后,就可以根据自己的需要,选择适用的程序完成所要进行的工作,而不需再记忆大量的命令。

2. MS Windows 采用一致的菜单结构,使用户学会一个 Windows 程序的操作后,就可以很快学会其它 Windows 程序的使用。

3. 与设备无关的图形界面。由于 PC 机的设计是基于开放式的系统结构,所以它鼓励第三方硬件商为 PC 机开发外部设备,因此目前产生了大量的性能优良、控制方式各异的外设(如显示卡、打印机、绘图仪、数字化仪等)。但这也带来了一个问题,如果想开发一个在各种外设上都能运行的软件,该软件必须为这些不同的外设提供一个驱动程序。而在 Windows 系统下,由于外设是由 Windows 管理控制的,应用软件只需与 Windows 打交道,因此,从理论上讲,该软件可以在所有 Windows 支持的外设上运行,这就为软件移植创造了很好的条件。

4. Windows 提供了大量的 API 实用函数用于创建窗口、对话框、列表框等对象,这样程序员只需把精力集中在程序的内部逻辑的组织上,而人机界面的设计可以直接调用 Windows 提供的 API 函数来完成,使软件的开发过程缩短很多,这使程序员们受益非浅。

5. 对于程序员来说,由于 Windows 突破 DOS 640KB 内存的限制,同时采用了虚拟内存管理技术,使程序员不必再考虑内存对应用程序的限制。

1.2 Windows 程序结构分析与程序运行原理

熟悉 Windows 的读者都知道,每一个 Windows 程序一般都在屏幕上建立一个属于自己的矩形窗口,窗口由标题条、菜单条、滚动条、边框等特征组成。

在讲解 Windows 程序运行原理之前,首先让我们来看一段最简单的 Windows 程序,它只完成在屏幕上显示"你好"这个工作。程序见清单 1.1。程序的输出结果见图 1.2。

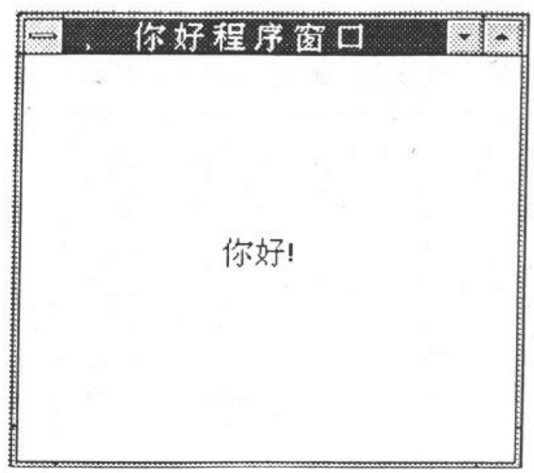


图 1.2

清单 1.1

```
//程序名是 SAMPLE1.C
#include<windows.h>
long FAR  PASCAL _export WndProc(HWND,UINT,UINT,LONG);
int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,
        LPSTR lpszCmdParam,int  nCmdShow)
```

```
{
  HWND      hWnd;
  MSG       msg;
  WNDCLASS  wndclass;
  if(! hPrevInstance){
    wndclass.style         =CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc   =WndProc;
    wndclass.cbClsExtra    =0;
    wndclass.cbWndExtra    =0;
    wndclass.hInstance     =hInstance;
    wndclass.hIcon         =LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor       =LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground=GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName=NULL;
    wndclass.lpszClassName="Howareyou";
  RegisterClass(&wndclass);
}
hWnd=CreateWindow("Howareyou",
                  "你好程序窗口",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
if(! hWnd)return;
ShowWindow(hWnd,nCmdShow);
UpdateWindow(hWnd);
while(GetMessage(&msg,NULL,0,0)){
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    }
  return  msg.wParam;
}
long FAR  PASCAL _export WndProc(HWND hWnd,UINT message,UINT wParam,
                        LONG lParam)
```

```
{
  HDC      hdc;
  PAINT  STRUCT  ps;
  switch(message)
    {
    caseWM_PAINT:
      hdc=BeginPaint(hWnd,&ps);
      TextOut(hdc,100,100,"你好!",5);
      EndPaint(hWnd,&ps);
      return  0;
    caseWM_DESTROY:
      PostQuitMessage(0);
      return  0;
    }
  return  DefWindowProc(hWnd,message,wParam,lParam);
  }
```

从上述程序清单中可以看到，每一个 Windows 程序都有一个象 C 语言 main 一样的主函数，并被命名为 WinMain。Windows 程序从 WinMain 开始执行。WinMain 从系统中接收四个参数，其中第一个参数 hInstance 是本实例的句柄(句柄的概念在本章 1.3.1 节介绍)；第二个参数 hPrevInstance 是本程序前一个实例的句柄。由于 Windows 允许同一程序的多个实例同时运行，所以如果运行本实例之前已有同一程序的另一个实例在运行，则该参数返回一个非 0 的前一个实例句柄，否则返回 0；第三个参数 lpszCmdParam 是命令行参数指针，相当于 C 语言 main 主函数中的 argv；第四个参数 nCmdShow 是窗口的显示方式，它的含义将在下面的章节介绍。

下面简单介绍 sample1 程序中各主要部分的功能。

程序的第一部分如下：

```
if(! hPrevInstance){
    wndclass.style       =CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc   =WndProc;
    wndclass.cbClsExtra    =0;
    wndclass.cbWndExtra    =0;
    wndclass.hInstance     =hInstance;
    wndclass.hIcon         =LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor       =LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground=GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName  =NULL;
    wndclass.lpszClassName=szAppName;
    RegisterClass(&wndclass);
}
```

if 语句判断是否有本程序的另一个实例在运行，如果没有，就执行下面的语句向 Windows

注册本程序所建立该类窗口具有的共同特征，如窗口内光标的形状、背景颜色等。

程序的第二部分如下：

```
hWnd=CreateWindow(szAppName,
                  "你好程序窗口",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  CW_USEDEFAULT,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);
if(! hWnd)return;
ShowWindow(hWnd,nCmdShow);
UpdateWindow(hWnd);
```

由于几乎每个 Windows 程序都有一个主窗口，因此这部分程序首先调用 CreateWindow 函数创建程序的主窗口，然后判断窗口创建是否成功，若成功，就调用 ShowWindows 函数显示该窗口，UpdateWindow 函数的作用将在以后的章节中介绍。

第三部分程序如下：

```
while(GetMessage(&msg,NULL,0,0)){
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return   msg.wParam;
```

这一部分是个 while 循环语句，我们叫做"消息循环"。GetMessage 函数从用户消息队列中取出一条消息后，先调用 TranslateMessage 函数对消息中有关虚拟键盘的消息进行翻译(虚拟键盘的概念将在后面的章节介绍)，然后再调用 DispatchMessage 函数，请求 Windows 把该消息传给本窗口的窗口处理函数。看到这里读者会问，往下程序将如何运行？该程序只有 WndProc 一个函数，而在 WinMain 函数中并没有调用 WndProc 函数，那么 WinMain 函数怎样才能调用 Wndproc 窗口处理函数呢？这里也许是让初次接触 Windows 程序的程序员们最迷惑不解的地方。

在 DOS 环境下编过程序的读者都知道，当程序开始执行时，它就从程序的第一条可执行语句开始获得了对 CPU 等计算机资源的控制权。以后只有当发生用户强行中断程序的运行、硬件中断或程序运行结束等情况，操作系统才重新获得计算机资源的控制权。对于 UNIX 这类以时间片为单位的抢占式多任务系统，每个进程一次只运行一个时间片，当一个进程运行了一个时间片后，系统硬件产生中断使操作系统获得系统资源的控制权，此时操作系统根据各进程的优先级和资源使用情况确定下一个将要运行的进程。但从程序员的角度看，由于每个时间片的时间很短，且进程的挂起和激活是由操作系统自动完成的，所以对于分时多任务系统的每个

程序来说就好象全部资源由自己独占一样，因此在程序设计中除不允许应用程序直接访问硬件设备外，其它方面与单任务系统中的程序无多大区别。而 MS Windows3.x 是一个基于单任务 DOS 环境下的多任务系统，因此它不能采用类似 UNIX 那样的以时钟中断为基础、以时间片为抢占单位的用抢占方式实现的多任务系统。

Windows 采用了一种叫做非抢占式的多任务系统，简单地讲就是一个进程运行一段时间后，自己将控制权主动地交给另一个进程实现的多任务。所以这种多任务系统要求 Windows 与应用程序一起对所有计算机资源进行管理，以保证所有 Windows 下的应用程序都能很好地运行。

为了实现多任务，Windows 将计算机可处理的事件细分成很多的消息，所以 Windows 要求每个 Windows 应用程序在处理完某个消息(事件)后，就主动将计算机资源的控制权交回给 Windows，Windows 再根据当前各应用程序对系统资源请求情况，将系统资源连同请求处理的消息(事件)，一同交给这个应用程序。因此可以认为 Windows 的一个消息就相当于抢占多任务环境下的一个时间片。

在具体实现时，Windows 为每个应用程序建立一个“消息队列”，同时系统自己保留一个“系统消息队列”。为了处理窗口中产生的各种消息，Windows 程序总是将对消息进行处理的这部分程序放到一个叫做“窗口处理函数”的函数中。当程序开始运行时，它首先创建主窗口，然后用一个循环等待接收和处理由本窗口产生或由其它程序发送到本窗口的消息。当有消息产生时(如按下键盘)，系统就将这一消息收集到系统消息队列，同时也将这一消息放入用户的消息队列，一旦应用程序从 GetMessage 函数获得控制权，它就从自己的消息队列中取出消息，然后使用 DispatchMessage 函数要求 Windows 调用本窗口的窗口处理函数(如清单 1.1 中的 WndProc)，窗口处理函数在处理完这一事件后，如没有消息需要处理，就将 CPU 等计算机资源的控制权交回给 Windows。

下面我们再通过程序中对一个菜单选择的处理过程说明程序与 Windows 之间 CPU 控制权的传递过程。程序见清单 1.2。该清单只列出了对菜单项进行响应的部分。

清单 1.2

```
long FAR PASCAL _export WndProc(HWND hWnd,UINT message,UINT wParam,
                        LONG   lParam)
{
  switch(message){
    case  WM_COMMAND:
        switch(wParam){
            case  IDM_ADDMEN:
                addmanproc;//调用增加人员处理函数
                break;
            case  IDM_DELEMAN:
                delemanproc;//调用删除人员处理函数
                break;
        }
```

```
            return 0;
        case  WM_DESTROY:
            PostQuitMessage(0);
            return  0;
        }
        return DefWindowProc(hWnd,message,wParam,lParam);
    }
}
```

在 Windows 下菜单是由 Windows 维护的一个系统资源，因此它不象在 DOS 条件下用一读语句等待用户选择。在 Windows 条件下采用的方式是：当用户在菜单条上选择一菜单项后，系统将这一事件变成消息放入“用户消息对列”，经过一系列的处理，Windows 将这一选择变成 WM_COMMAND 消息，并将所选菜单的标号(如 IDM_ADDMAN)放入 wParam 参数中。当应用程序从“消息循环”中的 GetMessage 函数获得控制后，它就从“用户消息队列”中取出这一消息，然后由 DispatchMessage 函数通知 Windows 调用窗口函数(如本例中的 WndProc)对消息进行处理，窗口函数首先通过 switch 语句判断 message 参数中的消息是否为 WM_COMMAND，如果是就再判断 wParam 变量的值是否为所选菜单的标号(如 IDM_ADDMAN)，如果是就调用 addmanproc 函数进行处理。当处理完后，该处理函数用 return 语句返回到“消息循环”，这时用户又可以选择下一个菜单。如果用户没有选择下个要处理的菜单，用户的消息队列中就可能无消息供 GetMessage 函数取出，在等待下一个要被处理的消息时，它将 CPU 的控制权交回 Windows，这时 Windows 允许其它程序获得控制权。Windows 就是采用这种方式实现了多任务。

应用程序与 Windows 之间控制权转移的示意图见图 1.3。

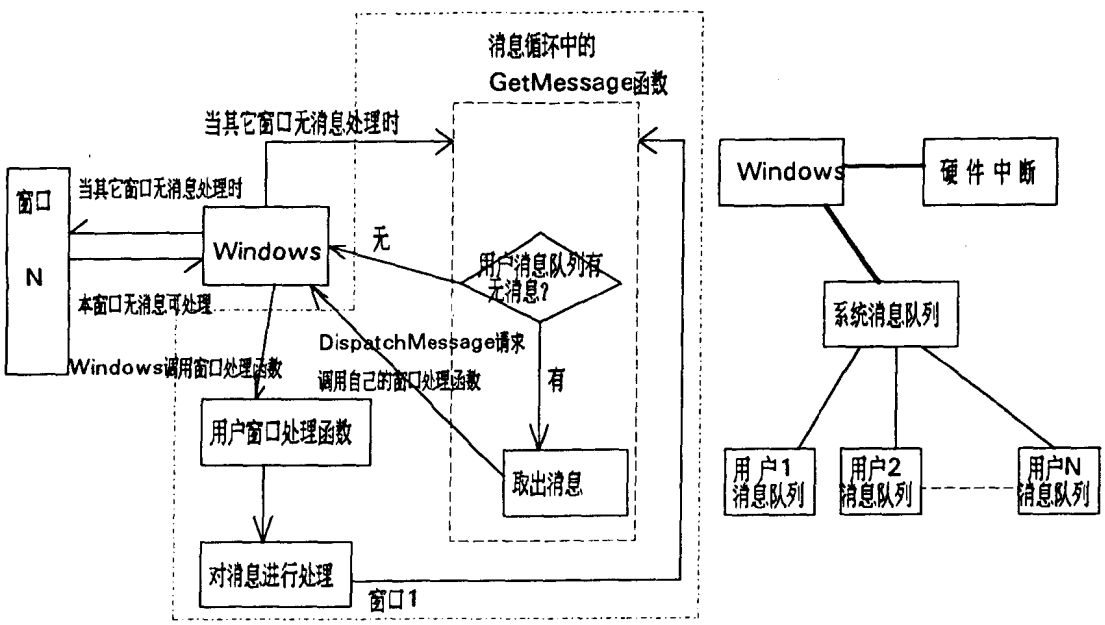


图 1.3

从图 1.3 可以看出一般情况下“消息循环”中的 GetMessage 函数是应用程序与 Windows 之间控制权的交换点。在以后的章节还会讨论另外几个可与 Windows 交接控制权的函数。

1.3 Windows 程序编写的基本框架

虽然用户编写的应用程序功能各异，但从程序设计的角度看，其程序结构都是由固定的几个部分组成(详见本章 1.3.2 节)。因此读者可用清单 1.1 为模板开发自己的应用程序。本书的所有示例均以清单 1.1 为模板经修改而成。在介绍 Windows 程序编写的基本框架之前，先介绍几个 Windows 编程中常用的技术术语。

1.3.1 有关编写 Windows 程序中的几个常见术语

1. PASCAL 调用规则

C 语言函数的调用规则是：参数从右至左压入栈，即参数表中的最后一个参数首先入栈，函数返回后，由调用函数进行栈指针的调整；PASCAL 语言的调用规则是：参数从左至右入栈，被调函数返回在返回之前，自行调整栈指针。Windows 与应用程序之间采用 PASCAL 调用规则，因此 WinMain 函数和所有窗口处理函数、对话框函数及须由 Windows 调用的函数均要使用 PASCAL 调用规则。但用户自己编写的并由自己直接调用的函数可象标准 C 语言库函数那样使用 C 语言调用规则。我们通过在函数名前加上 PASCAL 标识符指出该函数采用 PASCAL 调用规则。

2. windows.h

在每个 Windows 程序的顶部必须有条预处理命令：

#include <windows.h>

windows.h 是一个很大的头文件，在这个头文件中定义了大量的 Windows 程序中使用的数据结构和常量，如：窗口关闭消息 WM_QUIT 被定义成

#define WM_QUIT 0x0012

windows 中大量使用的消息数据结构 MSG 定义如下：

```
typedef struct tag MSG{
        HWND    hWnd;       //获得消息的窗口句柄
        WORD    message;    //消息序号
        WORD    wParam;     //一个字的消息附加信息
        LONG    lParam;     //双字的消息附加信息
        DWORD   time;       //消息被传送的时间
        POINT   pt;         //消息被发送时的光标位置
}MSG;
```

因此，当在程序中加上 windows.h 头文件后，就可以使用所有 Windows 定义的数据结构和标识符。