计算机技术丛书

Windows 程序设计

张之超　刘毅慧
孔凡敏　孙　路　编著

人民邮电出版社
PEOPLE'S POSTS & TELECOMMUNICATIONS PUBLISHING HOUSE

计 算 机 技 术 丛 书

Windows 程序设计

张之超　刘毅慧
孔凡敏　孙　路
编著

人 民 邮 电 出 版 社

内 容 提 要

本书是针对 Windows 程序设计的初学者和一般用户编写的，可以使他们在短时间内掌握如何用 C 语言在 Windows 环境下进行编程。本书共分 12 章，分别介绍了：Windows 程序设计基础，文本显示、控制窗口和文件，键盘与鼠标，内存管理，字符串、图标、光标、用户定义和位图资源，菜单和加速键资源，对话框，绘制图形，设备相关、设备无关的位图以及图元文件，字体与文本输出，打印输出和更高级的编程技术。

本书实例丰富、讲解深入浅出，对于没有任何程序设计经验的用户，可以通过本书学习并掌握 Windows 程序设计，从而达到应用的程度；对于具有一定 DOS 环境下程序设计经验的用户，可以通过本书迅速掌握 Windows 环境下的程序设计。

计算机技术丛书

Windows 程序设计

◆ 编　　著　张之超　刘毅慧　孔凡敏　孙　路

责任编辑　王亚明　段云洁

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号

北京鸿佳印刷厂印刷

新华书店总店北京发行所经销

◆ 开本：787 × 1092　1/16

印张：27

字数：672 千字　　1998 年 11 月第 1 版

印数：1 – 3 000 册　　1998 年 11 月北京第 1 次印刷

ISBN 7-115-07457-7/TP·885

定价：40.00 元

丛 书 前 言

世界上发达国家普遍重视发展以计算机和通信为核心的信息技术、信息产业和信息技术的应用，一些经济发达国家信息产业发展迅速。

当前，我国处于国民经济高速发展时期。与此相伴随，必将有信息技术、信息产业和信息技术应用的高速发展。各行各业将面临信息技术应用研究与发展的大课题以及信息化技术改造的大任务、大工程。

为了适应信息技术应用大众化的趋势，提高应用水平，我们组织编写、出版了这套“计算机技术丛书”。这套丛书以实用化、系列化、大众化为特点，介绍实用计算机技术。

这套丛书采取开放式选题框架，即选题面向我国不断发展着的计算机技术应用的实际需要和国际上的实用新技术，选题不断增添又保持前后有序。

这套丛书中有的著作还拟配合出版软件版本，用软盘形式向读者提供著作中介绍的软件，以使读者方便地使用软件。

我们希望广大读者为这套丛书的出版多提意见和建议。

前　言

Windows 作为新一代的操作系统，深为广大计算机用户所喜爱。在这一新的平台上，一大批十分优秀的应用软件逐渐被应用到人们的工作、学习和生活中，这不但使人们的工作、学习效率有了很大的提高，而且还大大地丰富了人们的生活。因此，开发 Windows 环境下的应用程序已经变得十分必要。

本书的目的就在于让 Windows 程序设计的初学者和一般计算机用户，在较短时间内，掌握 Windows 环境下用 C 语言进行程序设计的一般步骤、方法以及较为常用的技巧。

本书共分 12 章，分别介绍了：Windows 程序设计基础，文本显示、控制窗口和文件，键盘与鼠标，内存管理，字符串、图标、光标、用户定义和位图资源，菜单和加速键资源，对话框，绘制图形，设备相关、设备无关的位图以及图元文件，字体与文本输出，打印输出，更高级的编程技术。

在编写本书时，作者力求做到：

（1）深入浅出、循序渐进

使具有一定 DOS 环境下程序设计经验的读者，能更快更好地掌握 Windows 程序设计，而且达到一定的水平。

使没有任何程序设计经验的读者，学习并掌握 Windows 程序设计技巧，而且达到应用的程度。

（2）实例丰富，讲解细致

为了更好地帮助用户真正掌握 Windows 程序设计，作者在书中加入了较为丰富的实例，其中绝大部分都是作者实际工作的总结。

读者不仅可以使用实例中讲解的技巧，完成自己的程序设计，而且只要对这些实例略加修改，就可以应用到自己的工作中。

本书中的所有实例，均在 Borland C++3.1 环境中调试通过。读者可在该环境或更高的环境中使用。

对本书的不足和错误之处，欢迎广大读者批评指正。

目　录

第一章 Windows 程序设计基础

编写 Windows 程序和编写 MS-DOS 程序不尽相同，Windows 程序也很少直接调用 MS-DOS 的功能。如果 Windows 应用程序要获得键盘和鼠标输入，或在显示器和打印机上输出，或者管理磁盘文件，它都需依赖于 Windows 应用程序编程接口（API）。

事实上，编写一个成功的 Windows 应用程序的关键就在于充分利用 Windows 现有的函数功能。在拥有 600 多个函数的 Windows 环境中编程，这看起来有点可怕。因此要想充分利用好系统内的这些资源，就需明白 Windows 环境是如何组织的。

1.1 Windows 环境概述

Windows 环境能够提供 3 种重要且最基本的服务。

(1)执行基本的输入输出功能，负责与键盘、鼠标、显示器、打印机、磁盘文件和串行通信设备等打交道。这其中的大多数功能虽然在 MS-DOS 中也同样支持，但 Windows 中的基本输入输出（I/O）函数远比 DOS 中的丰富。更重要的是，Windows 中的 I/O 函数都是与设备无关的。Windows 通过 API 函数的一个子集就能支持几百个 I/O 设备。因此，Windows 程序几乎全部依赖于现有的 API 函数完成输入输出。

(2)内存管理。Windows 允许程序动态申请和释放内存，Windows 内存管理 API 允许程序透明地存取扩充内存（Expanded Memory，即由 4.0 版的 LIM EMS 标准所支持的页方式内存）和延伸内存（Extended Memory，1MB 以上的可寻址空间，也译作扩展内存）。在基于 80386 和 80486 的主机上，Windows 还提供了对用户透明的虚拟内存（Virtual Memory，利用内存和磁盘之间进行交换的技术来实现的可共享内存）。

(3)支持多任务。Windows 支持多任务，即允许两个或多个程序共享 CPU、内存和 I/O 设备。Windows 允许多任务就意味着 Windows 的 I/O 和内存管理函数能够使多个程序协调地共享资源。

可以将 Windows 环境设想成一个能够支持 I/O、内存管理和多任务的管理者。事实上，Windows 的不同功能包括 I/O 管理程序、内存管理程序、任务管理程序、窗口管理程序等。虽然这是一些通俗的概念，但这些功能确实对应着 Windows 环境中的几个单独模块。

1.1.1 模块

在 Windows 中，模块是指任何能够装入内存的可执行代码和数据的集合。一个模块或是包含一个用户编制的应用程序，或是一个硬件设备驱动程序，或是一个动态连接库（DLL）函数，也可能是一个程序中所包含的能被另一个程序所存取的数据资源。Windows 应用程序，甚至于 Windows 环境本身，都是通过几个 Windows 模块来实现复杂操作的。

Windows 本身由几个相关的模块组成。Windows API 函数就是由 Windows 启动时装入内存的几个模块实现的。这些模块如表 1-1 所示。

表 1-1 几个模块的名称及其支持的函数

模 块	支持的函数
GDI.EXE	图形设备接口：图形图像输出、调色板管理
USER.EXE	窗口、象标、光标管理
KERNEL.EXE	内存管理、任务调度
KRNL286.EXE	内存管理、任务调度
KRNL386.EXE	内存管理、任务调度
COMM.DRV	串行通信设备驱动程序
KEYBOARD.DRV	键盘设备驱动程序
MOUSE.DRV	鼠标设备驱动程序
SOUND.DRV	声音设备驱动程序
SYSTEM.DRV	系统定时器、磁盘驱动器驱动程序

3 个主要模块是：

GDI.EXE；

USER.EXE；

KERNEL.EXE、RNL286.EXE 或 KRNL386.EXE（这三者分别对应于 Windows 的运行模式——实模式、标准模式和增强模式）。

在编写 Windows 应用程序时，不必考虑组成 Windows 环境的这些模块的名字。用户程序可以调用任何 Windows API 函数，而不必关心哪个模块包含了所需调用的函数。通常只需关心操作者明显涉及到的模块，即用户自己编写的应用程序或 DLL 模块。

Windows 具有动态装入模块的能力，且只在可执行程序需要存取这个模块的函数或数据时才会装入。Windows 提供了几个 API 函数（LoadLibrary、LoadModule 和 WinExec），以便让程序在执行时由人工装入一个模块。当另外一个程序涉及到一个模块内的函数时，Windows 也能隐式地装入这个模块。这种隐式动态装入模块的能力依赖于是否在不同模块内申明了引入函数（Imported Functions）和引出函数（Exported Functions）。

1.1.2 函数

大多数 Windows 模块包含一个或多个能被其他模块中的代码所调用的可执行代码，这些代码称为引出函数。由包含这个函数可执行代码的模块引出(Export)这个函数；而由调用这个函数的模块引入(Import)这个函数。引出函数能够被其他模块的多个程序所调用。

在 Windows 本身和 Windows 应用程序中大量使用了引出函数。整个 Windows API 就是由构成 Windows 环境的不同模块所引出的函数而组成的。Windows 要求应用程序定义那些 Windows 本身将调用的引出函数。特别地，Windows 调用应用程序中的引出函数用以将键盘和鼠标输入引导到一个应用程序或作为多任务运行的应用程序中。

1.1.3 任务和实例

Windows 支持协作性多任务。Windows 的任务管理程序维护一张任务表并且记录这些任务的执行顺序。Windows 将控制交给一个任务后，别的任务便不能运行，直到这个任务显式地将控制交还给 Windows 任务管理程序。Windows 的协作性多任务不同于 OS/2 中使用的抢先性多任务。在抢先性多任务中，操作系统以预定时间片为单位轮流执行每一个任务，而不管一个任务是否自动放弃控制权。

Windows 将所执行的应用程序作为不同的任务。即使运行一个应用程序的两个或多个实例，Windows 也把它们当作不同的任务。对于每个实例，Windows 将应用程序的缺省数据段分别装入内存。这是因为相应于每个实例，任务特定的数据（包括堆栈和局部数据）都存储在应用程序该实例的缺省数据段中。

尽管 Windows 为一个模块的每个实例都装入一个缺省数据段，但可执行代码部分却只装入一次。因此，Windows 应用程序中的可执行代码和数据有着天壤之别。当使用汇编语言编程时要特别注意这个问题。但如用高级语言，就不必考虑执行代码和数据的分离，它是由语言编译器来决定的。

Windows 的协作性多任务机制给应用程序设计增加了一些限制。每一个应用程序至少需要一个能被 Windows 任务管理程序所调用的窗口过程函数，还需要一个消息循环，以便将控制交还给任务管理程序。这些要求是很重要的，因为每一个 Windows 应用程序都遵循这一特定逻辑。

1.1.4 消息

在 Windows 环境中，消息是指从 Windows 调用的函数传递给应用程序窗口过程函数的固定不变的数据集。Microsoft Windows 软件开发包（SDK）将消息标识符加以符号化。WM_CREATE、WM_COMMAND、WM_PAINT 等符号化的名字表明了这些信息被传递给应用程序后，应用程序将执行什么样的相应动作。Windows SDK 文档详细解释了所有消息和它们的参数。

Windows 消息机制使人联想到事件驱动机制。不同的事件发送不同的消息。一些事件是很明显的。例如，当按了一个鼠标按钮或按了一下键盘，Windows 就会用一条或多条消息来精确地指出哪个鼠标或键盘事件发生过。于是，Windows 就会用像 WM_LBUTTONDOWN、WM_MOUSEMOVE、WM_KEYDOWN 等类的消息标识符去调用窗口过程函数。

另外，Windows 还用一些特定的消息使应用程序能处理特定事件。例如，Windows 用 WM_PAINT 消息通知应用程序在显示器上产生输出。一旦接收到这条消息，应用程序就会用 Windows API 函数在显示器上写一串字符或画一个图形。Windows 用其他别的消息来通知应用程序 Windows 环境发生了何种变化，并且在不同的应用程序之间顺利地传递通信信息。

Windows 传送一个消息给应用程序，既可以将消息放到应用程序消息队列中，也可以将消息直接发送到应用程序的窗口过程函数中。比如，通知应用程序有关键盘和鼠标的输入消息，常常将消息放到消息队列中。因此，要正确地处理消息，每个应用程序需有一个消息队列来检查顺序到达的消息。Windows API 提供了相应的函数集来完成这个工作，如 GetMessage、PeekMessage、WaitMessage。否则的话，Windows 会绕过应用程序程序消息队列，直接将窗口管理消息发送给应用程序。当一个消息传递（Post）到一个应用程序的消息队列后，应用程序不一定能立即处理这一消息，直到等到 Windows 任务管理程序将控制交给这个应用程序。而如果是直接发送（Send）给应用程序，应用程序就会立即处理。

1.1.5 窗口

在 Windows 环境中，窗口有两个逻辑组成部分，一是描述窗口特性的数据结构，另一个是处理消息的窗口过程函数。数据结构由 Windows 窗口管理程序创建和维护。窗口过程函数是一个引出函数，直接被 Windows 调用。

直观看来，窗口是一个视觉上的窗口，它能进行图形处理。而将窗口看作一个消息处理单元则更为确切。一个应用程序可以创建一个窗口，它能处理各种消息，而不在屏幕上显示出来。例如，与其他应用程序通过 DDE 协议交换消息的应用程序可以创建多个不可见窗口，以此封装 DDE 消息处理。

Windows 编程的好处也在于它能创建多个窗口，每一个都可具有不同的功能。Windows 窗口管理程序通过维护一张所有应用程序创建的窗口表来实现这一功能。窗口管理程序将任务和它们创建的窗口联系起来。任务结束后，如果它不将自己创建的窗口撤消，则由窗口管理程序将它们撤消掉。

一个任务的多个窗口按拥有者和被拥有者之间的继承关系组织起来，也就是说，每个窗口都能被其他窗口所拥有，而每个窗口也可以拥有一个或多个窗口。拥有者和被拥有者的关系影响着一个窗口的生命周期以及显示出来的窗口风格，窗口特性如表 1-2 所示。

表 1-2　　窗口特性

显示风格	是否由拥有者确定	生命周期	显示关系
WS_OVERLAPPED	否	任务周期	覆盖其他窗口。通常由标题和边框组成。由 CreateWindow 确定位置和大小
WS_OVERLAPPED	是	拥有者	覆盖拥有者。通常由标题和边框组成。若拥有者不可见，它也不可见
WS_POPUP	否	任务周期	覆盖其他窗口
WS_POPUP	是	拥有者	覆盖拥有者。若拥有者不可见，它也不可见
WS_CHILD	是	拥有者（双亲）	在双亲的范围内，尺寸大于它，则被剪裁掉

在 Windows 环境中，窗口之间的关系被跟踪记录下来，这种关系对应用程序是透明的。窗口管理程序维护一张数据结构表，每一个表项描述一个窗口。从应用程序的观点看，每个窗口由一个窗口管理程序所确定的数值唯一地标识。这个数值就是窗口句柄。

1.1.6 句柄

在 Windows 环境中存在许多对象，窗口仅仅是由句柄所标识的对象中的一类。其他对象也都需句柄来标识，它们是：

模块（Module）；
任务（Task）；
实例（Instance）；
文件（File）；
内存块（Block Of Memory）；
菜单（Menu）；
控制（Control）；
字体（Font）；
资源（Resource），如：图标（Icon）、光标（Cursor）、字符串（String）等；
GDI 对象，如：位图（Bitmap）、刷子（Brush）、元文件（Metafile）、调色板（Palette）、画笔（Pen）、区域（Region）以及设备描述（Device Context）。

Windows 程序并不用物理地址来标识一个内存块、文件、任务或动态装入模块。相反地，Windows API 给这些对象分配确定的句柄，并将句柄返回给应用程序。

一个 Windows 应用程序可用不同的方法获取一个特定对象的句柄。许多 API 函数，包括 CreateWindow、GlobalAlloc 和 OpenFile 都返回一个句柄。另外，Windows 也能通过应用程序的引出函数将一个句柄作为参数传送给应用程序。一旦获得了一个确定对象的句柄，便可在 Windows 环境的任何地方对这个句柄进行操作。这种大量使用句柄的方式影响着每一个 Windows 程序的设计。

句柄值只有唯一地确定了一个对象时才有意义。实际上，句柄值对应着对象表中的某一项，而只有 Windows 本身才能直接存取这个表。应用程序只能通过 Windows API 函数处理不同的句柄。例如，应用程序可为自己申请一块内存，通过调用 API 函数——GlobalAlloc，来返回一个指向内存块的句柄。

```
hMem    =    GlobalAlloc(…);
```

这个句柄确定了一个内存块，但这时应用程序并不能直接存取这块内存。应用程序还需调用 API 函数 GlobalLock，并返回内存块的物理地址：

```
lpMem    =    GlobalLock(hMem);
```

一种通用的句柄类型是 HANDLE，在 Windows 3.1 以前的版本中，它可以标识所有种类的句柄。在 Windows3.1 以后的版本又派生出一些新的句柄数据类型，每种类型的句柄用于标识一种类型的对象：

- HANDLE　　通用句柄类型
- HWND　　标识一个窗口对象
- HDC　　标识一个设备对象
- HMENU　　标识一个菜单对象
- HICON　　标识一个图标对象
- HCURSOR　　标识一个光标对象
- HBRUSH　　标识一个刷子对象
- HPEN　　标识一个画笔对象
- HFONT　　标识一个字体对象
- HINSTANCE　　标识一个应用程序模块的一个实例
- HLOCAL　　标识一个局部内存对象
- HGLOBAL　　标识一个全局内存对象

1.1.7 内存管理

Windows 内存管理程序控制着系统的所有可用内存。它动态分配和释放模块中的代码段和数据段所用的内存。内存管理程序还支持一些 API 函数，以供 Windows 程序为自己分配内存块。

从 Windows 程序的角度看，内存管理程序将可用内存分为两种堆（Heap）：全局堆和局部堆。例如，如果一个应用程序有两个可执行代码段和一个缺省数据段，Windows 将 3 个段装入到全局堆中的 3 个不同的内存块中。类似地，每一次调用 GlobalAlloc 函数都会分配一个返回远指针的内存块。

局部堆是局限于一个缺省数据段中的内存，这些内存在模块装入内存后的全局堆中分配。程序可通过调用 LocalAlloc 函数在局部堆中分配内存。这些内存块由近指针寻址。由于每个模块的局部堆使得该模块的堆栈和静态数据变量都共享同一个内存段，因此，局部堆的大小

不得超过物理段的大小（64kB）。

局部堆和全局堆最大的区别在于容量的大小。如果在应用程序的一个实例中分配较小数据量，则最好使用局部堆。而如果申请大块内存或在两个以上模块中共享数据，则需使用全局堆。

1.2 Windows 程序设计

Windows 程序的设计对于初学者来说的确是件头疼的事，但是 Windows 程序的设计并不是想象的那样困难。对于大多数程序员来说，学习 Windows 程序设计的关键是打破传统的程序设计方法，并全心地接受 Windows 程序设计思想。下面介绍了用 Borland C++进行 Windows 应用程序编程的基本思想和基本方法。

1.2.1 事件驱动程序设计

DOS 程序主要使用顺序的、过程驱动的程序设计方法。顺序的、过程驱动的程序有一个明显的开始、明显的过程及一个明显的结束，因此程序能直接控制程序事件或过程的顺序。

假如用户要编写一个程序来计算一个学期中进行了 3 次测验后一个班的平均成绩，采用顺序的、过程驱动的程序设计方法的逻辑流程图如图 1-1 所示。

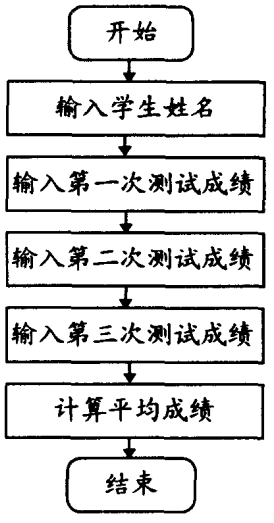

图 1-1 顺序的、过程驱动程序设计的方法示例

从图 1-1 可以看出，这个步骤遵循事件的顺序，用户必须按照事件的顺序来完成各个步骤，它不可能允许在计算平均分后再去修改测试成绩。虽然在顺序的、过程驱动的程序中也有许多异常处理的方法，但这样的异常处理也仍然是顺序的、过程驱动的结构。

Windows 程序设计方法与 DOS 程序设计方法的不同在于 Windows 程序是事件驱动的。事件驱动的程序不由事件的顺序来控制，而是由事件的发生来控制。事件驱动程序的设计是以一种非顺序的方式处理事件，从而回避了顺序的、过程驱动的方法。

事件驱动程序的设计是围绕着消息的产生与处理而展开的，每一条消息都是关于发生的事件的消息，例如，一个键按下或鼠标按钮按下，就发出了一个消息，而当松开此钮时，另