打开 Windows 这扇窗

深入 Windows 编程

—Windows 加密及压缩软件编程技巧与方法

雷　军　王全国　马贤亮　著

清华大学出版社

Windows 与 DOS 仅一步之遥，

让我们一起打开 Windows 这扇窗。

致　谢

我们的力量来源于众多朋友的支持和鼓励，在此谨向他们表示诚挚谢意。

香港金山公司副总裁求伯君先生的大力支持，同事傅占华、胡卫翔、陈波、冯志宏、钱国祥等不遗余力的协助，才使本书在北京金山软件公司的机房里顺利完稿。美国 Aldus 公司的张湘辉提供了一些资料，Rosenthal Engineering 的 Doren Rosenthal 试用过 PACKWIN，这两位先生给了我们相当大的帮助。

雷　军： 心中涌动着无法言语的感激，尤其是对：带我入门的武汉大学计算机系张德向等老师，最早在自己的软件产品中选用 BITLOK 的联邦软件产业发展公司总裁苏启强，给我们提供了一个创造和表现舞台的香港金山公司副总裁求伯君，还有一起奋斗的弟兄们。

王全国： 尽管我的妻子和女儿对 Windows 并不关心，但我还是要把这本书献给她们，因为她们使我明白：Windows 外面的世界更美好！

马贤亮： 献给飞扬、迷茫的青春感觉。

目 录

引言

中国软件走向世界

0.1 令人振奋的机遇与紧迫的挑战

自从 Microsoft 于 1990 年 5 月推出第一个 Windows 的成熟版本 Windows 3.0 以来，整个计算机工业界围绕 Windows 系列(包括 Windows 3.1、Windows NT、Chicago 和 Cario 等等)进行了大规模的分化和重新组合。目前，全球有 5000 万 Windows 用户，Windows 上的应用软件已逾万套。这一汹涌澎湃的 Windows 潮流给了我们一个全面赶超世界软件水平的机遇。这是因为：

Windows 与 DOS 用户界面完全不同，直接以图形模式为基础。Windows 提供与设备无关的图形操作，比如画线、画矩形和圆以及其它更复杂的区域。由于是设备无关的，所以同一功能可以在不同的打印机和监视器上工作。例如，你不用告诉程序所联的是什么打印机，只需简单地调用 Print 命令即可。Windows 的设备驱动程序将这些图形操作转换输出到打印机、显示屏或其它的输出设备。Windows 的这种图形模式，使 Windows 环境真正摆脱了对硬件文本功能(如显示卡、打印机)的依赖；同时，Windows 中各种外设均以 Driver 形式嵌入系统，对系统而言，面对的是由 Driver 描述的“虚拟设备”。Windows 的这种设备无关性要求各种设备根据 Windows 的驱动程序接口，重新编写不同于 DOS 的驱动程序，使之能够输出各种图形和文字。所以说，Windows 的这种基于图形模式的外设无关性为多文种支持(包括中文)奠定了基础。

其次，在系统职能上，Windows 相对于 DOS 有很大的增强。Windows 不仅增加了作为一个操作系统所必须的内存管理、进程管理等基本部分，吸收了多媒体、网络、笔式输入等新技术，还扩充了资源、字体、文字处理、对话、编辑等原本由应用软件处理的功能。大量的功能由应用级转移到系统级，既保证了风格的统一，简化了应用软件的开发，又使应用软件的注意力保持在问题分析和解决方案上(What to do)，而不是在方案的实现上(How to do)。上述功能的实现与具体的应用软件无关，这是软件兼容性的重要保证，也是 Windows 作为一个新的软件平台流行的原因。

目前的 Windows 就是一个包含初步国际化功能的产品。它不仅在时间格式、货币符号和键盘分布等方面提供了国际化支持，使用 8 位的 ANSI 字符集，还保留了对双字节字符(DBCS)内码的支持能力，只是在 Windows 的西文版本中没有实现而已。此外，Windows 提倡把提示信息(如菜单、对话、字符串等)都放在可执行程序的资源段中的风格，有利于不同

语言版本的转换。目前，中文之星(新天地)、PWin(Microsoft)、金山皓月(香港金山)、Win-MATE(四通利方)等 Windows 中文环境已陆续出现。它们以中西文兼容为突破口，消除了中西文软件应用与开发的技术障碍，基本构成了实用开放的中文系统。今后的中文平台将朝着国际化的软件操作环境、完善的中文支持(包括软件中的文化因素)、任意的软硬件适配等方向发展。即将出现的下一个 Windows 版本 Chicago 能够直接处理中文，任何应用软件都不需要“中文本地化”(汉化)。Windows 国际化的特点使得语言不再是阻碍我国软件开发的主要因素。

再者，Windows 作为一个崭新的操作系统，尽管国外的开发者可以早一步拿到完整的资料、拥有众多的软件，但从总体上讲，国内和国外的开发者仍处于同一起跑线，大家都需要一段时间才能熟悉 Windows，国内外的软件同行在 Windows 上的差距是很小的。如我们开发的 Windows 执行文件压缩软件 PACKWIN，和国外同期开发的类似产品相比，效果要好得多。这说明在一些具体的应用软件开发上，我们还是有可能走在国外软件开发者前面。

综上所述，Windows 跨越语言的差异，作为一个理想的软件开发平台，它使中国的用户有可能同步享受世界最新的软件精品，更使中国的软件工程师得到了一个与世界各国同行共同开拓的契机。

Windows 为我们提供了一个舞台，但这是一个需要奋力搏击的舞台。Windows 给了中国软件产业一次机会，同时也把中国的软件市场推向了世界。因为 Windows 的开放性，使国外软件更容易突破中文处理技术的壁垒，于是，国外的优秀软件扬起 Windows 风帆，纷纷来到中国：Word, Excel, FoxPro, Lotus 1-2-3, AmiPro, Word Perfect...

这是对民族软件工业的一次挑战。面对这一个相互竞争的市场，软件开发人员应该尽快进入 Windows 世界，积极吸收国外先进的技术；同时，在实践中逐步培养“软件工程”的观念和经验，把软件开发真正作为一个工程，开发出世界级顶尖产品。

虽然我国潜在的市场很大，但目前相对国际市场来说还是很小的，估计还不到 1%。我国有着世界优秀的软件工程师，应该好好把握 Windows 这一次新的机会与挑战，创造世界名牌，让中国软件走向世界。

打开 Windows 这扇窗，外面的世界更精彩！

0.2 编写目的

两年前，香港金山公司的 WPS 文字处理系统就已风靡全国，成为装机量仅次于 DOS 的软件产品。为了拓展产品范畴，公司决定由北京金山软件公司开发 Windows 上的产品“双城电子表”。那时，我们对 Windows 的了解比较肤浅，对 Windows 核心还一无所知。

因为“双城电子表”不是一、二万行就能解决的小项目，随着程序量的增大，在调试中我们经常碰到一些莫名其妙的问题，而在参考手册中找不到这些问题的解决办法；再者，我们开发的“双城电子表”及一系列 Windows 上的应用产品都即将面市，必须解决加密问题；同时，我们又不安心于总在“Windows 黑匣子”外面逗留。实际需求加上内心的冲动，迫使我们逐步深入到 Windows 底层中去。这本书奉献给大家的就是我们这一方面两三年经验的积累。

0.3　为什么要深入 Windows 核心

Windows 是什么？对于一般的用户而言，它是菜单、按钮、对话框、多个窗口、能够同时执行的多个程序，是一个比 DOS 强大的运行环境。对于编程人员而言，首先，Windows 是一套完备的图形界面接口，编程人员只需按照 Microsoft Windows 文档中的规定传递参数，调用函数，而不用知道更进一步的细节。比如说，编程时经常要用到各种 handle(selector)，它们分别指向相应的数据结构，而编程人员并不需要知道这个数据结构的具体内容 — Windows 把这些信息隐藏了，这也正符合结构化程序设计中所强调的“信息隐藏”观念；其次，Windows 是许多生硬的定义的组合，Windows 实现了针对这些定义的操作，但这些被隐藏在背后。对 Windows 开发人员——如果他有刨根问底的兴趣的话——这些操作的实现细节象一个魔术袋，开发人员只能站在外围看魔术师表演，却没有机会把头探进魔术袋看个究竟，当然也就无法揭穿魔术师的把戏了！这种血肉分离的定义是生硬的。比如，Microsoft 的文档中给出了 NE 文件结构中每一项的说明，与 NE 文件结构相关的基本操作是：Windows 把一个磁盘上的 AP 或 DLL 运行起来需要经过载入、启动、执行等步骤，中间还涉及动态连接等其它方面，但 Windows 把这些事情全部接管了，编程人员从 Microsoft 的文档中无法找到 Windows 执行机制的细致描述，也就难以编写自己的装载和启动程序。Windows 实施这种策略的后果是：程序员对 NE 结构只有静态的感觉，没有与执行机制融合在一起的生动理解。

现在的问题是，我们必须超越“Windows 高级编程”，深入到 Windows 核心，从高处把握 Windows。比如说，我们需要知道 Module Handle、Task Handle、Instance Handle 之间的关系；还要为自己的 Windows 压缩、加密程序编写自装载代码；我们还想与 Windows 平起平坐，共同操纵执行程序以满足特别的需要，而不愿意把事情全权委托给 Windows，让它用自己缺省的方式在“暗室”里处理完毕后再交回来……

那么，如何深入到 Windows 核心呢？我们选择了“Windows 可执行文件”为出发点。因为一个操作系统的可执行文件格式从多种意义上讲就是该系统本身执行机制的反映，而且研究可执行文件格式能够积累大量的知识。本书的主要内容正是围绕 Windows 可执行文件格式展开的。

0.4　本书的结构

● **第 1 章　分析 Windows 执行文件**

本章引用了 Microsoft 的参考手册中关于 NE 文件格式的说明，并修正了原文中的一些错误，添加了一些保留或未予说明的内容。NE 文件格式的说明包括 WINSTUB，NE 信息块，段表、资源表、驻留名表、模块引用表、输入名表、入口表、非驻留名表，代码段和数据段的重定位信息，以及几种主要资源的结构。本章对以上内容都使用二

进制 dump 方式给予详细讲解，以便读者对 NE 结构有形象的理解和直观的记忆。

● **第 2 章 Windows 执行文件的分析工具**

如果你已经了解了 NE 结构，自然不会再喜欢用原始的 dump 方式去察看文件。市面上有许多流行的 NE 文件分析工具，如 TDUMP、EXEHDR、MAPWIN、EXEDUMP、NEWEXE 等，它们都能将 NE 文件中的执行信息以简洁明了的文本形式给出。这些工具有各自的侧重点，代表了不同的需要。本章的重点是介绍作者研制的 Power 系列分析工具中的两个：Power Dump 和 Power FileInfo，并通过用上述工具分析实例来强化第 1 章中讲述的概念。

● **第 3 章 分析工具的开发实例**

探索 Windows 的第一步是深入到 Windows 可执行文件中去。在前面两章中，我们对 NE 格式的了解只是概念性的。为了熟练驾驭结构复杂的 Windows 可执行文件，本章首先定义了一个文件对象 File Object，接着向读者演示如何开发一个类似于 EXEHDR 的分析工具 MSDUMP。选择这一个演示实例是基于以下的两个理由：首先，知道 NE 中的执行信息的含义和能够编写程序获取这些执行信息是不同的两回事 — “概念上理解”和“实际编程”是不同的两回事；再者，任何针对 NE 文件的操作都必须以正确地读取 NE 中的信息块、各种表以及重定位信息为基础，MSDUMP 这一个例子正好适合。在演示 MSDUMP 的开发过程之前，本章中还介绍了一个扩展工具集 ExtTools，该工具汇集了编程中经常要用到的一些操作，因而它使应用程序的主要意图清晰明了。

● **第 4 章 直接修改 Windows 执行文件**

如果没有一个执行文件的源程序，我们只有硬着头皮去直接修改。进行这种操作需要实施者对 Windows 执行机制有全面的了解。所谓执行机制，通俗地讲，就是 Windows 系统把一个执行程序从磁盘上载入到内存中，并根据 NE 信息块和文件头中各种表格的内容，在内存中为该程序维护相应的数据记录，初始化与该程序有关的堆栈、事件队列等，从重定位表中的信息准确地找到重定位的代码，以及在程序执行时处理各种问题的原则和方法的总称。本章的重点是叙述与执行机制相关的概念、数据结构以及他们之间错综复杂的动态关系。以这些知识为基点，本章演示了直接修改 Windows 执行文件中某些部分的方法，读者可以参照这个示范修改其它 NE 程序。

● **第 5 章 用汇编语言编写 Windows 应用程序**

虽然 C 语言是编写 Windows 程序的最佳语言，但至少有以下两个理由说明在有些场合还必须用汇编语言编写 Windows 程序：一是涉及到一些低级操作，如针对 BIOS 的读写；另一种情况是在那些必须追求效率的场合，如编写压缩软件。然而，即使你对 DOS 环境下的各种汇编语言相当熟练，编写 Windows 汇编程序并非一件易如反掌的小事：Windows 汇编引入了一些新的宏定义，它有自己的编程约定！

在这一章中，我们强调“用汇编语言编写 Windows 的方法”是出于以下两个目的：一是在本书第 6、8、9 章中有多个用汇编语言编写的示范程序；另外一点是，我们认为，一个能够用汇编语言编写 Windows 程序的程序员，他阅读、跟踪、调试实际运行代码不会遇到什么大的困难，同时，他又具备了深入 Windows 核心的关键手段：因为

只有汇编语言才是与机器语言直接等价的。

● **第 6 章　自装载 Windows 执行文件**

在可执行文件启动和执行以前，存在一个装载过程。对于符合 NE 格式标准的文件，装载过程一般是交由 Windows KERNEL 按缺省方式自动完成。但是，如果 NE 程序被加密或压缩过，它的执行信息就被破坏了，那么在装载时就需要首先把被变形的程序还原成标准的格式；如果程序的某个段大小超过了 64K，它就不再是符合标准的格式，Windows 不能接受，因而装载程序必须为这样的段分配内存。本章讲解了自装载 Windows 程序时用到的装载函数、数据表，并以 HELLOWIN 为例，给出了一个完整的自装载实例。自装载技术是操纵 Windows EXE 时最关键、最复杂的技术，也是笔者分析 Windows 的最大收获。

● **第 7 章　LZ 压缩算法原理与具体实现**

Windows 可执行文件通常占用很大的存储空间，良好的数据压缩技术可以有效地利用有限的存储介质。本书第 8 章将实现一个 Windows EXE 压缩软件 PACKWIN。这一章向不熟悉压缩技术的读者讲述了数据压缩概论、LZ 压缩算法原理、压缩与还原算法的实现等基础知识，最后给出了 PACKWIN 中用到的压缩函数的实现细节。

● **第 8 章　开发 Windows 执行文件压缩软件**

这里所指的“压缩软件”要求能够随着 Windows EXE 的执行，自动解压缩相关的段和资源，使之符合标准的 NE 格式。本章向读者介绍了 PACKWIN 是如何压缩 STUB，如何压缩代码段、数据段及 Windows 资源；如何在被压缩的文件中插入自装载段，又如何在自装载段中的 LoadAppSeg 函数里实现解压缩。PACKWIN 实现涉及到了第 4 章“直接修改 Windows 执行文件”、第 6 章“自装载 Windows 应用程序”、第 7 章“LZ 压缩算法原理与具体实现”以及第 3 章中的 File Object 定义，它汇集了操纵 Windows EXE 时经常使用的手段，所以“PACKWIN”是一个比较典型的主题。

● **第 9 章　开发 Windows 加密软件**

Windows 取代 DOS 成为微机上新的软件开发平台已是事实，很多的软件厂商都在开发基于 Windows 的应用软件。在软件的法律保护并不完善的今天，过去常用的通过加密手段来保护软件版权的办法，同样也遇到了一个从 DOS 到 Windows 转变的问题，这种转变的主要困难在于 NE 新格式和 Windows 特殊的执行机制等方面。BITLOK for Windows 在 BITLOK 的基础上，针对 NE 的特点，成功地实现了对 Windows EXE 的加密。本章讲述了软件加密的一般知识，并给出了 BITLOK for Windows 的实现方法。

● **第 10 章　Windows NT 及 Chicago 执行文件格式**

在不久的将来，Windows NT、Chicago 等系统将逐渐取代 Windows 3.1。Microsoft 为这些基于 Win32 的系统设计了一种不同于 NE 的新文件格式 — PE 格式。因为系统可执行文件格式最能反应这个操作系统本身的特性，所以我们仍然选择 Win32 执行文件格式来引导读者深入到这些未来的操作系统内部。本章较详细地介绍了 PE 格式。鉴于本书前面对 16 位 Windows 的描述，我们在介绍基于 Win32 的 PE 文件格式时，将它和 16 位 NE 文件格式作了比较对照。本章还将第 2 章中介绍过的 PDUMP 加以改造，使之能够分析 PE 格式的文件。

0.5 如何使用本书

如果你仅仅对于 Windows 有些兴趣，只需要翻翻即可。需要的时候，再详读重点章节。

如果你需要编写 Windows 下的压缩或加密软件，必须配合磁盘中的实例，系统地阅读本书的每个章节。

不管你是 DOS 资深专家还是 Windows 初学者，阅读本书都不会遇到什么困难。

0.6 磁盘资料

本书所附的软盘中，不仅包含了各章节中示范程序的源代码，同时还有许多可以立即使用的工具程序。盘片中的具体资料说明如下，如有修改，以盘中文件为准，不另外说明。

README	磁盘文件的说明
TPU.ZIP	编译 Pascal 程序所用到的 TPU
PKUNZIP.EXE	展包程序
\Chap1	
WINSTUB.EXE	一般的 STUB 程序
LOADSTUB.EXE	在 DOS 下能够装入 Windows 的 STUB 程序
MINISTUB.EXE	最小的 STUB 程序
MINI.BAT	生成 MINISTUB 的批文件
MINISTUB.ASM	MINISTUB 的汇编源程序
COM2EXE.EXE	COM 文件转换成 EXE 文件
COM2EXE.PAS	COM2EXE 的源程序
RESTUB.EXE	修改 STUB 的工具
GETSEG.EXE	取某段的工具
GETSEG.PAS	GETSEG 的源程序
PACKBMP.EXE	压缩 BITMAP 的工具
PACKBMP.PAS	PACKBMP 的源程序
\Chap2	
TOOLS.ZIP	包含许多著名的 Windows 分析工具
PDUMP.EXE	黄玫瑰开发的分析文件格式的工具
PDUMP.PAS	PDUMP 的源程序
PFI.EXE	黄玫瑰开发的分析文件格工的工具
PFI.PAS	PFI 的源程序
\Chap3	
EXTTOOLS.PAS	EXTTOOLS 的接口说明部分
FILEDEF.PAS	File Object 的接口说明
MSDUMP.EXE	利用 File Object 开发的类似 EXEHDR 的工具
MSDUMP.PAS	MSDUMP 的源程序
\Chap4	
ADDCODE.EXE	附加一段代码的工具
ADDCODE.PAS	ADDCODE 的源程序
RESTUB.PAS	修改 STUB 工具的源程序

\ Chap5		
	HELLOWIN.ASM	HELLOWIN 的 MASM 版
	HELLOWIN.DEF	HELLOWIN 的定义文件
	HELLOWIN.LNK	链接 HELLOWIN 的定义文件
	MAKEFILE	MAKE 文件
	HELLOWIN.EXE	
\ Chap6		
	APPLOAD.ASM	自装载的源程序
	RELOC.ASM	Windows 重定位的源程序
	HELLOWIN.C	开发一个带自装载功能的 HELLOWIN
	HELLOWIN.DEF	HELLOWIN 的定义文件
	MAKEFILE	MAKE 文件
	C.BAT	编译批处理程序
	HELLOWIN.EXE	带自装载功能的 HELLOWIN
\ Chap7		
	LZPACK.TPU	压缩程序库
	LZPACK.PAS	压缩库的源程序
	ZLITE.EXE	DOS 版执行程序压缩工具
	ZLITE.PAS	ZLITE 的源程序
\ Chap8		
	PACKWIN.EXE	DOS 和 Windows 执行文件压缩工具
	PACKWIN.PAS	PACKWIN 源程序
\ Chap9		
	BLW.EXE	BITLOK for Windows 1.0 加密主程序
	BLWINST.EXE	BITLOK for Windows 1.0 硬盘安装程序
\ TOOLS		
	RI.COM	RAMINIT DOS 的内存清理工具
	RI.DOC	RI 的使用说明
	RI.ASM	RI 的源程序

0.7　关于编程风格

一个优秀的程序员必须具备良好的编程风格，一个编程风格好的程序能极大简化调试和维护。本书提供了大量用 Turbo Pascal 编写的示范程序，如第 3 章中的 ExtTools.pas 和 FileDef.pas 等。希望读者仔细研读这些程序，提高自己的编程能力和技巧。本书选用 Borland Pascal 作为主要的示范编程语言，是因为 Pascal 描述算法比较清楚，习惯使用其它语言的读者能够很轻松地看懂或者将示范程序转换成其它语言。

0.8　神秘"黄玫瑰"

一个满脸胡须、经常嗑着葵花籽的人物，他有一把十几个枪管的手枪。此人来无影、去无踪，枪法奇准。他以勇猛、果断的方式除暴安良、杀富济贫。每次"革命"行动之后，总会留下一朵黄玫瑰再飘然离去……。这是罗马尼亚电影《神秘的黄玫瑰》的

故事梗概。

这里要讲述的是另外一个故事……

在本书示范程序的版权说明中，经常出现“Yellow Rose Software Workgroup”字样，表示这些软件由黄玫瑰组开发。本书作者都是黄玫瑰组的核心成员。近年来，随着黄玫瑰组开发的软件在国内逐渐流行，那些电脑爱好者对“黄玫瑰”的兴趣与日俱增。本文是对黄玫瑰软件制作组的介绍，同时也是本书作者小传。

黄玫瑰是一个软件俱乐部形式的小组，成立的目的是加强合作，切磋技艺，共同发展。一个人的力量是有限的，但一群人团结在一个共同旗帜下，能够制作出更好的软件，更容易被社会接受，也便于与其它团体交流。

黄玫瑰主张软件应该首先打响团体品牌。

黄玫瑰的诞生缘于雷军和王全国的一次偶然相遇，当时雷军是武汉大学计算机系 87 级学生，王全国在武汉测绘科技大学计算机系任教。黄玫瑰的第一个作品是 BITLOK V0.99。

雷　军：我和王全国相识是一个非常偶然的机会。那是 89 年的夏天，特别平静又特别闷热。我刚念完二年级，得过几次奖学金，又参与过老师们的几个课题，应该属于“品学兼优”那一档，而且正是充满幻想和自信的年纪 — 我看过许多关于硅谷发展史的书，可能是书中描述的硅谷英雄们的成功鼓舞着我 — 我经常梦想着有一天能创建一个一流的软件公司，满世界都用我们的软件。武大距东湖新技术开发区不远，我常到开发区去看一看，因为那里的软件和机器总是比学校里新一些。

遇到王全国那天是大太阳，阳光耀眼，热得很，而且是在号称“火炉”的武汉！街上没有什么行人。我拐进武汉九州公司的营业部时，看见里面的小机房里有个小伙子全神贯注地盯着计算机屏幕，在专心致志地做程序。他就是王全国。我站在他身后看了一会儿，见他指法娴熟，软件用的很“溜”，心想可能是一位“高手”。他扭过头来的时候，我问他做的是什么，他说在做一个加密软件，并演示了他做的界面部分。我不禁一阵惊喜，因为我正在做一个加密程序的加密算法和反跟踪部分。我告诉他我做的加密软件的进展情况和遇到的一些困难，相互交流了各自的一些想法，我们谈得很投机。

结识王全国后不久，我就预感到我们可以很好地合作。这不仅因为他的计算机水平，更主要是他的性格。计算机是一门技术，而不是艺术，需要的首先是毅力。王全国言语不多，但思维敏捷，对所做的事情致志不渝，有韧性，“踏实能干”是他的最大特点，也是软件开发人员最需要的。另外一点是他给我的外在感觉。他充满灵气，同时性情宁静，极少为世俗的喧嚣和杂乱所感染。我当时是二年级学生，而王全国毕业已经一年，他能平等地待我，这一点一直让我很感激。我崇尚的是“太阳每天都是新的”，富于挑战的生活正是我所渴望的。总之，结识王全国后，我开始了一种与众不同的、不平静的大学生活，后来的事情也证明了这一点。

王全国：我大学毕业后留校，在学校的九州公司做一些自己感兴趣的程序，衣食无忧。雷军那天到九州去，我给他演示了我做的加密程序，然后他谈了他自己做加密程序的情况。我当时的判断是，一个二年级学生能如此深入到问题的核心，应该是很不错的。而且他

有朝气，有激情，对技术问题都有自己的看法，还能以自己的观点去说服人，这一点就说明他与一般在校念书的学生不一样。在我们交往中他表现的热情让我觉得，他已认定计算机软件作为自己的职业！

我们认识后不久，就商定共同做一个加密系统。当时时间不多，因为那一学期开学较早，雷军还要去上课。我们并不准备做得非常完美，只是计划从 8 月 2 日到 8 月 17 日完成加密系统的各个方面，构造一个可用的软件。最后的情况是，我们基本按时完成了计划，接着又花了两天的时间作了一些细小的修改和完善。我们把这个加密系统取名为 BITLOK，意思是"锁住每一比特"；版本号为 V 0.99，因为她距商用的、可发售的版本还有一点距离。在这段合作中，我们都很辛苦，一般要从早上 8 点工作到第二天凌晨 2－3 点，晚上睡公司的沙发。半个月下来，两个人瘦了一大圈，衣服也累积了一大堆。这半个月两个人都处于极度兴奋的状态！

黄玫瑰的兴趣主要集中在工具软件。工具软件不同于一般应用程序，它的开发需要更深入的知识和更全面的考虑。黄玫瑰的早期产品除 BITLOK 外，还有"免疫 90"等。黄玫瑰的成长过程中有被社会认可的欣慰，也有天真烂漫带来的懊恼，只是他们的信念始终如一："赠人玫瑰之手，历久尤有余香"。

雷　军：我们当时有这样的认识：在中国做软件，一定要掌握中文和加密这两项最关键的技术。中文是中国软件开发的障碍，它使中国用户不容易同步享受世界最新的软件成果；同时也是中国软件产业的天然壁垒，一个不是在汉文化环境中成长起来的程序员很难突破这个壁垒，所以中文会保护中国软件产业，也是中国软件的市场和希望所在。加密技术是获利的保证，金山公司的中文字处理软件 WPS 的成功以及它遭受盗版的严重侵害也说明了加密技术的重要。我们做 BITLOK 就是基于这种考虑。

BITLOK 做完以后，我们很激动，心想该为我们的这个组合取一个响亮、上口的名字。以前看过一个电影，叫《神秘的黄玫瑰》，好象是罗马尼亚的，说的是个除暴安良的神秘人物。我们做 BITLOK 就是为了扼制盗版，所以借助这个故事，取了一个很"靓"的名字——"黄玫瑰"。

冯志宏：90 年春节期间，我和雷军一起做了一个防病毒软件"免疫 90"，也是打着黄玫瑰的招牌。那一阵病毒大流行。当时公安部门为了避免社会上将病毒越解越"毒"，禁止一般单位出售防病毒软件。我们的"免疫 90"在社会上已经扩散了一阵，而"黄玫瑰"又不是一个挂牌营业的公司，所以此后我们没有再花时间改善"免疫 90"，这个东西也就夭折了。这是一段小插曲。

马贤亮：我当时在武大空间物理系念书，最感兴趣的是什么汉语言文学、人文史哲之类特别玄虚的东西。我开始涉足一些计算机知识完全是受雷军的煽动。

黄玫瑰组经过短暂的离散又汇集在了一起，现在他们的力量更加强大。每年都有一两项产品。累计到现在有："免疫 90"，BITLOK，BITSHELL，DM（Directory Manager，DOS 下的目录管理工具），UnDup（直接展开 DUP 磁盘映象文件的工具），RI（RamInit，内存清理工具），BITLOK for Windows，PACKWIN，PDUMP，PFI 等。

雷　军：我大学毕业后被分配到北京的一个研究所，但黄玫瑰这一个兴趣小组还是被保留下来。我把大学毕业前买的 286 带到北京，利用下班后的时间完善 BITLOK，并将这个软件商品化，找到了一批大用户。BITLOK 得到了社会的认可，显示了黄玫瑰小组的技术实力，因为加密软件不同于一般的应用开发……这又增强了我们的自信心。

91 年年底，一个偶然的机会，我认识了以研制 WPS 而闻名中国软件界的求伯君。他当时是香港金山公司副总裁，负责金山公司在国内的软件开发，可以说是中国软件工程师成功的典范。他邀请我到金山公司工作，经过一段时间的考虑，我离开了研究所，加盟到金山旗下。

92 年下半年，王全国、冯志宏和一些新人陆续来到了金山公司，成立了香港金山公司北京开发部，现在更名为北京金山软件公司。金山公司为黄玫瑰组提供了一个广阔的舞台。在完成本职工作之外，黄玫瑰的业余软件产品还在不断问世，这些产品大都是免费的，希望能得到用户的青睐。

黄玫瑰的成员现在大都加盟到了北京金山软件公司，他们来到了一个更加广阔、可以自由驰骋的天地。作为一个曾经培育他们成长的沃土，黄玫瑰这一业余产品品牌将继续存在，今后还会有更多的黄玫瑰产品奉献给社会，希望用户给予更多的关注。

北京金山软件公司已经成为方正集团及香港金山公司一支重要的开发力量。两年来，完成了方正外接式汉卡、电脑好译通、双城电子表、双城电子词典、汉字自动校对系统等一系列产品，同时编写了《深入 DOS 编程》、《WPS NT 循序渐进》等优秀、畅销的书籍。

在中国，象黄玫瑰这样的制作室还有很多。他们有相当的技术实力，朝气蓬勃，对市场非常敏感，都怀有类似的目标和期望，有朝一日会成为中国软件产业的中坚。他们代表明天和希望。

祝他们好运！

第 1 章

分析 Windows 执行文件

1.1 Windows 执行文件格式与动态链接

目前微机上的操作系统主要有 DOS 和 Windows，它们定义了不同的执行文件格式。DOS上的可执行文件有 COM、EXE 两种文件格式(BAT 批处理文件是文本文件，由 COMMAND.COM 解释，我们在本书中提到的可执行文件不包含 BAT 文件)。COM 程序中的代码、数据和堆栈存储在最大 64K 的范围内，是简单的内存映像。COM 程序是 CP/M 时代的遗骸，因为那时的 RAM 很小，程序一般不会超过 64K。

与 COM 文件不同，EXE 程序可以达到 DOS 能够使用的最大内存大小甚至更大，而且 EXE 文件段不用按顺序存储。EXE 中的代码、数据和堆栈存储在不同的段中，同时各部分可以占据多个段。EXE 文件比 COM 文件灵活、优越，应用面也就广泛得多。

Windows 是一个基于 DOS 的多任务系统。为了适应新的操作环境，Microsoft 基于 DOS EXE，设计了一种新的称作 NE(New EXE)的格式。NE 格式设计思想与 DOS EXE 的最大不同是它适应 Windows 动态连接机制。

所谓 Windows 的动态链接(Dynamic Linking)是相对于 MS-DOS 的静态链接(Static Linking)而言。MS-DOS 所有的函数库都只是目标码的集合，应用程序链接函数库时，实际上是把库中的目标码拷贝一份到应用程序中，形成单个独立的文件。这一动作是在编译链接时完成。此后在应用程序执行时，DOS 只需将程序载入即可，不必考虑执行时的链接问题。这种静态链接库的优点是：所有应用程序用一套标准的例程，而不用包含这些例程的源代码。自然，如果几个应用程序都链接了相同的目标码，就会有多份目标码被复制到应用程序中。

与 DOS 的静态链接库不同，Windows 的动态链接库(Dynamically Linked Libraries，DLLs)是在运行时与应用程序链接。动态链接机制使得不同的应用模块间或不同的应用程序间可以共享代码和资源。其优点是：DLLs 仍然是一个分立的部分，我们可以修改或更新它而不用重新编译整个应用程序；内存中任何共享代码只备有一份，节约了内存空间；应用程序可以共享诸如数据和硬件之类的资源。所有的 Windows 库都是 DLL。比如 GDI.EXE，USER.EXE，KERNAL.EXE，.DLL，.DRV，.FON 等等。

Windows 可执行文件中的 API 调用是以 ACSII 字符串的形式给出 API 函数名，这样，Windows EXE 比 DOS 的 EXE 或 COM 更为结构化，我们不必去执行或反编译，只需作静态分析就能得知 Windows 文件的大部分内容。

Microsoft Windows 操作系统的可执行文件是代码和数据的组合，或代码、数据和资源的组合。文件里还包含两个首部，即 MS-DOS 首部(WINSTUB)和 Windows 首部。

1.2 WINSTUB——MS-DOS 首部

Windows 执行文件保留了 MS-DOS 的首部，以便与 DOS 兼容。这样，Windows 程序在 MS-DOS 下启动时，会直接启动 MS-DOS 的 EXE 格式的插入程序，即所谓的 WINSTUB。本节我们介绍 DOS 的 EXE 文件头结构和 Windows EXE 中 WINSTUB(MS-DOS 首部)的情况。

1.2.1 DOS EXE 的文件头格式

MS-DOS EXE 文件有一个特殊的文件头，叫 MZ HEADER(MZ 是 MS-DOS 的主要设计者 Mark Zbikowski 名字的字首)。因为 DOS EXE 经常占用好几个段的空间，并且可能有多个数据段和代码段，DOS EXEC(INT 21h，功能 4Bh，参见香港金山公司求伯君主编《深入 DOS 编程》第 84 页)就依据这个文件头的信息来载入程序。与 DOS COM 程序的绝对内存映像方式不同，EXE 程序可同时占用多个段，并且包含多个代码和数据段。DOS EXE 存储方式是可重定位的内存映像，这样，DOS 就可以在系统总空间的状态和本程序所需的内存之间进行调整。

我们先用 DEBUG 来看一个 DOS EXE 的文件头，然后再具体分析。

```
C:\DOS>ren move.exe move
C:\DOS>debug move
-d
0000  4D 5A 8F 01 24 00 05 00-20 00 34 03 34 03 8C 04 MZ..$....4.4...
0010  00 30 1E 6E 70 14 00 00-1E 00 00 00 01 00 0C 00 .0.np...........
0020  49 04 7E 14 00 00 57 15-00 00 00 00 6E 03 D5 09 I.~...W.....n..
0030  73 03 00 00 00 00 00 00-00 00 00 00 00 00 00 00 s...............
......
```

EXE 文件头信息	
偏　移	意　义
00H-01H	一般为'MZ'(04DH, 05AH)或'ZM'(05AH, 04DH)。EXE 文件标志信息，由 LINK 程序产生
02H-03H	文件长度除以 512 的余数，即文件长度 MOD 512
04H-05H	文件页数。文件以 512byte 为一页，文件页数即文件长度除以 512 的商
06H-07H	重定位表中项数
08H-09H	文件头的节数，即文件头的长度除以 16 的商
0AH-0BH	运行该程序所需的最小节数(MINALLOC)。如果可用内存小于这一数目，程序就不能运行

0CH-0DH	运行该程序希望得到的最大节数
0EH-0FH	堆栈段的段地址初始值
10H-11H	SP 寄存器的初始值
12H-13H	文件校验和，为 2 的补码
14H-15H	IP 寄存器的初始值
16H-17H	代码段的段地址的初始值
18H-19H	文件第一个重定位项的偏移量
1AH-1BH	覆盖号，如果驻留则为 0
1CH	RESERVED
	重定位表
	RESERVED

图 1.1　DOS 执行文件的头信息

我们注意到：文件中第一个重定位项的偏移量，即[18H]处的值为 1EH，小于 40H。

紧跟在文件头后面的是一段保留空间。保留空间的后面是 MZ 文件头的重定位表。MZ 文件头的长度并不固定，视重定位项的多少而异；不过其大小一定是 16 的倍数。EXE 程序的代码段、数据段和堆栈段分别存放在不同的段(segment)中。

EXE 文件格式有个缺点：它不保留程序中任何关于各个独立的段的有关信息，这样 DOS Loader 无法对程序中的单一段进行操作，也就无法为各个段分配特定的内存，使之以彼此互不影响的方式进行处理。当 LINK 建立 EXE 时，它把程序中关于段的信息收集为一个单独的全局重定位表，重定位表中记录的就是这些信息。

文件头中的内容为系统对该文件的控制信息和重定位信息。它位于文件的开始，并且由格式化区和重定位表组成。格式化区共有 28 个字节，每两个字节为一项，重定位表紧跟在格式化区之后。组成这个重定位表的重定位项的个数是可变的，在格式化区偏移值 06H-07H 指出。每个重定位项包含四个字节，它表示一个内存地址，前两个字节为偏移值，后两个字节为段地址。重定位项与装入区的数据是对应的，这个字在该模块得到控制权之前要求进行修改，一次只有一个重定位表项被读入到工作区中。重定位表中每一项的段值加上程序的初始段值就是新的段值。这个过程叫“重定位”。

1.2.2　Windows EXE 中的 MS-DOS 首部

Windows 可执行文件保留了 MS-DOS 的首部(WINSTUB)。MS-DOS 插入程序一般以 STUB.EXE 或 WINSTUB.EXE 文件名单独存放，在编译 Windows 程序时连接到 Windows EXE 的头部。一般，该插入程序的目的是：如果程序在 DOS 下启动，就提醒用户该程序需要在 Windows 环境下运行，即：

```
This program requires Microsoft Windows.
```

有的程序的 STUB 是该程序的 DOS 版本。这样，这个程序无论在 DOS 还是 Windows 环境中，完成的功能是一致的。

如果该 NE 文件在 Windows 的程序管理器 Program Manager 或文件管理器 File Manager 中被执行，Windows NE Loader 判断这一个程序是单纯的 MZ 格式还是 NE 格式，如果是 NE 格式，则 NE Loader 会跳过 WINSTUB 部分。

Microsoft 文档中说，如果 MS-DOS 首部的偏移地址[18H]处的值为 40H 或更大，则该程序为 Windows EXE，且偏移地址[3CH]处的值为 Windows 首部的偏移量。

实际上，"[18H]处的值为 40H 或更大"并不是判断一个文件是否为 Windows EXE 的必要条件：有一些典型的 Windows EXE 并不遵守这一缺省约定，如 Microsoft EXCEL 的 excel.exe的偏移[18H]处的值为 1CH。TDUMP 认为该程序不是 NE 格式的文件，这也正是 TDUMP(由 Borland International Co.开发)在判断时出错的原因。如果将该程序的[18H]处的值由 1CH 改为 40H，TDUMP 就会正确运行。事实上，Windows 系统本身也没有使用这一判断条件。

判断一个文件是不是 Windows EXE 的条件有很多，比如，[3CH]指针所指的内容是否为 Windows 特征字"NE"，并且其信息块中指示的目标操作系统是否为 Microsoft Windows 等等。

明白了 WINSTUB 的作用后，下面来介绍一组 WINSTUB 实例和处理 WINSTUB 的小工具。这些东西都放在本书磁盘的 CHAP1 目录下。

WINSTUB 普通的 STUB

在 Visual C++ 1.0 或 1.5 的 BIN 的目录下，有一个 STUB 程序叫 WINSTUB.EXE。编写 Windows 程序时一般需要三个文件，比如典型的 HELLOWIN 有 HELLOWIN.MAK、HELLOWIN.C、HELLOWIN.DEF 三个文件。在 HELLOWIN.DEF 中，我们可以定义 STUB 的文件名。下面是 HELLOWIN.DEF 的内容是：

```
;-----------------------------------------
; HELLOWIN.DEF module definition file
;-----------------------------------------

NAME            HELLOWIN
DESCRIPTION     'Hello Windows Program (c) Charles Petzold, 1992'
APPLOADER       '__MSLANGLOAD'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       8192
```

WINSTUB.EXE 的作用是在 DOS 下运行程序时，提醒用户要先运行 Windows。

STUB 文件可以定义为其它的 DOS 可执行文件，比如 PCTOOLS.EXE：

```
STUB        'PCTOOLS.EXE'
```

MINISTUB 最小的 STUB

本书所附磁盘的 CHAP1 目录下有一个 STUB 程序叫 MINISTUB.EXE，其功能和标准

的 WINSTUB.EXE 相同，用 DEBUG 或 PCTOOLS 把它 DUMP 出来：

```
C:\>dump ministub.exe
0000  4D 5A 80 00 01 00 00 00-04 00 F8 0F FF FF 00 00  MZ..............
0010  FE FF 00 00 00 01 F0 FF-40 00 00 00 53 54 55 42  ........@...STUB
0020  59 65 6C 6C 6F 77 20 52-6F 73 65 20 53 6F 66 74  Yellow Rose Soft
0030  77 61 72 65 20 43 6F 2E-20 4C 45 49 00 00 00 00  ware Co. LEI....
0040  0E 1F BA 0E 01 B4 09 CD-21 B8 00 4C CD 21 54 68  ........!..L.!Th
0050  69 73 20 70 61 63 6B 65-64 20 70 72 6F 67 72 61  is packed progra
0060  6D 20 72 65 71 75 69 72-65 73 20 4D 69 63 72 6F  m requires Micro
0070  73 6F 66 74 20 57 69 6E-64 6F 77 73 2E 0D 0A 24  soft Windows...$
```

多么精练的 STUB，就只有 128 个字节。一个真正的迷你型 STUB 程序！

编写一个 STUB 程序其实很简单，下面就是汇编语言的 MiniStub 源程序，这个程序只有 14 行。不过，该程序经过汇编后，生成的还暂时是一个 COM 文件。接下来我们要提供一个将 COM 程序转换为 EXE 的小工具 COM2EXE，经转换后就可以生成上面介绍的 MiniStub.exe。在本书的第八章“开发 PackWin 软件”中，我们用 MiniStub.exe 替换 Windows 程序中标准的 STUB，以达到压缩的目的，所以 MiniStub 提示给用户的信息中指明该程序被压缩过 — “This Packed Program”。下面是 MiniStub.asm 的源程序。（请注意：本书所列程序清单中，表示十六进制的后缀，H，一般用 h 代之）。

```
    .model  tiny
    .code
    org     100h
main:
    push    cs
    pop     ds
    lea     dx, Msg
    mov     ah, 9
    int     21h
    mov     ax, 4c00h
    int     21h
Msg db      'This packed program requires Microsoft Windows.', 0dh, 0ah, '$'
    ends
    end     main
```

LOADSTUB 能够自动装载 Windows 的 STUB

如果在 .DEF 文件中定义 STUB 文件名为 LOADSTUB.EXE，经编译链接，生成的文件在 DOS 下执行时，会首先装载 Windows，然后再去执行该文件。Microsoft Excel 就具有这种功能。

RESTUB 替换 Windows 程序中 STUB 工具

RESTUB 并不是一种 STUB，它的功能只是用新的 DOS 执行文件替换原有的 WINSTUB，并将旧的 STUB 取回来。下面是使用 RESTUB 的例子：

```
C:\>restub pbrush.exe ministub.exe
Power reStub Version 1.0 Copyright (c) 1993 KingSoft Ltd.

"PBRUSH.EXE" new stub is "MINISTUB.EXE"
```

```
Getting winstub to PSTUB.EXE ...
Erasing PBRUSH.EXE ...
Renaming PTEMP.EXE to PBRUSH.EXE ...
```

执行完毕后,原来的 WINSTUB 被取出来,放在了一个名为 PSTUB.EXE 的文件中。当然也可以再执行下面的命令来替换成原来的 WINSTUB。

```
C:\ >restub pbrush.exe pstub.exe
```

执行下面的这个命令会产生什么效果?

```
C:\ >restub pbrush.exe nc.exe
```

噢! 原来的 WINSTUB 被 Norton Utilities 中的 NC.EXE 替换了,在 DOS 提示符下运行 pbrush.exe 就相当于运行 NC! 用 RESTUB 作这种替换等价于在编译时定义 STUB 文件为 NC.EXE。RESTUB 的好处是:

(1) 可以轻松替换 Windows 程序中 STUB,而不用重新编译。

(2) 可以取回一些有意思的 STUB。

(3) 对于没有源码的程序只有采用这个工具修改 STUB。

如何用一个 DOS COM 程序来充当 WINSTUB 呢?

读者可用本书提供的 COM2EXE 先将 DOS COM 转换为 DOS EXE,再执行 NEWSTUB。市面上有很多的这种 COM2EXE 的程序,为什么要特别推荐一个新的 COM2EXE 呢? 因为 Windows 要求 WINSTUB 的 MZ 文件头长于 40h,这一点已在前面讲述过。本书所附的 COM2EXE 就是为这一特别条件编写的。

最后有一点要提醒读者注意: RESTUB 中的 DOS EXE 应该是标准的 MZ 格式。在 DOS 上可以执行的程序不一定就是标准的 MZ 格式。比如,用很流行的语言开发工具 Borland Pascal 编写的程序,如果编译时加上 DPMI 开关,生成的 EXE 就是一种被称为 BOSS 的格式(TURBO.EXE 就是这种格式),用这种格式的文件作为 STUB 文件就可能出问题。

COM2EXE 源程序

下面是用 Pascal 编写的 COM2EXE 源程序。RESTUB 的程序比较复杂,我们把它放在第四章中讲述。

```
{ * * * * * * * * * * * * * * COM2EXE * * * * * * * * * * * * * * * * * * * * * * }
Uses FileDef, ExtTools, DOS;

procedure InitHeader (var MZ: tagMzHeader; FSize: LongInt; Msg: String);
var
  I: word;
  exeSize: LongInt;
begin
  exeSize := ImageSize(FSize);
  with Mz do begin
    Sign := $5A4D;
    lenImage := dword(exeSize).loword;
    numPages := dword(exeSize).hiword;
    numRelocEntry := 0;
    sizeHeader := $4;
```

```
    numMinAlloc := $1000 - FSize shr 4;
    numMaxAlloc := $ffff;
    _SS := 0;
    _SP := $FFFE;
    CheckSum := 0;
    _IP := $100;
    _CS := $fff0;
    ofsRelocList := $40;        { $1C }
    noOverlay := 0;
    I := wMin(Length(Msg), sizeof(Reserved));
    Move(Msg[1], Reserved, I);
    ofsWinHeader := 0;
  end;
end;

procedure com2exe(fname1, fname2: string);
var
  F: File;
  P: PBytes;
  size: LongInt;
  Header: tagMzHeader;
  fileflag, numread, ReadSize: word;
begin
  assign(F, fname1);
  reset(F, 1);
  size := filesize(f);
  blockread(F, fileflag, sizeof(fileflag), numread);
  if (numread = 2) and (size < $ffc0) and
    (fileflag <> $4d5a) and (fileflag <> $5a4d) then
  begin
    ReadSize := Size;
    InitHeader(Header, ReadSize + $40, 'STUBYellow Rose Software Co. LEI');
    Getmem(P, ReadSize + $40);
    Move(Header, P^, $40);
    Seek(f, 0);
    Blockread(f, P^[$40], readsize);
    Close(f);

    assign(f, fname2);
    rewrite(f, 1);
    blockwrite(f, P^[0], readsize + $40);
    freemem(p, readsize + $40);
    close(F);
  end
  else Close(F);
end;

var
  fn1, fn2: PathStr;
begin
  Writeln('COM2EXE - To convert COM to EXE (c) KingSoft Ltd. 1993');
  Writeln;
  if ParamCount <> 2 then begin
    Writeln('Usage: COM2EXE <comfile> <exefile> ');
    Writeln;
  end
```

```
  else begin
    fn1 := ParamStr(1);
    fn2 := ParamStr(2);
    if (Pos('.', fn1) = 0) then fn1 := fn1 + '.com';
    if (Pos('.', fn2) = 0) then fn2 := fn2 + '.exe';
    if fileexists(fn1) then
      com2exe(fn1, fn2)
    else Writeln(fn1, ' not found.');
  end;
end.
```

在上面的程序使用了 FileDef 和 ExtTools 单元，这两者的源程序将在下一章给出。注意，NewStub.pas 所实现的替换过程较为复杂，因为替换 STUB 后，Windows 的 NE 文件头的内容都必须作相应的调整，有关这一方面的内容在第四章讲述，读者在这里暂不必深究。

1.2.3 WINSTUB 的数据结构和操作

根据上面对 MZ 文件头和 WINSTUB 的分析，我们这样来定义 Windows 程序中的 MZ 文件头结构 tagMZHeader 和与之相关的操作：

```
{ 定义 MZ 文件头结构 }
type
  tagMZHeader = Record
    Sign,                  { 00h 'MZ' or 'ZM' }
    lenImage,              { 02h Length of image }
    numPages,              { 04h Size of file in 512-byte pages }
    numRelocEntry,         { 06h Number of relocation-table items }
    sizeHeader,            { 08h Size of header in paragraphs }
    numMinAlloc,           { 0Ah Minimum number of paragraphs above }
    numMaxAlloc,           { 0Ch Maximum number of paragraphs above }
    _SS,                   { 0Eh Displacement of stk seg in paragraphs }
    _SP,                   { 10h offset in SP register }
    CheckSum,              { 12h word Checksum }
    _IP,                   { 14h IP register offset }
    _CS,                   { 16h code segment displacement }
    ofsRelocList,          { 18h Displacement of first relocation item }
    noOverlay: word;       { 1Ah Overlay number (Resident code = 0) }
    Reserved: array [ $ 1C.. $ 3B] of char;
    ofsWinHeader: LongInt;     { 3Ch New exe file header offset }
  end;
{ 重定位数据结构 }
  PMzRelocItem = ^tagMzRelocItem;
  tagMzRelocItem = record
    _Ofs, _Seg: Word;
    Idx: word;             { new }          { 重定位的序号 }
  end;
```

针对 MZ 格式文件，一般有如下的基本操作：

```
{ 取文件大小 }
    function GetDosFileSize: LongInt;
{ 取文件头大小 }
    function GetMzHeaderSize: LongInt;
```

```
{ 取装入部分的大小(不含文件头部分) }
    function GetLoadImageSize: LongInt;
{ 取装入部分的大小(含文件头部分) }
    function GetImageSize: LongInt;
{ 判断调试信息的类型是 Turbo Debugger 或 Code View }
    function GetSymTabType: word;
{ 取文件重定位列表到一个缓冲区 }
    function GetRelocList: PBuf;
{ 是否存在覆盖模块 }
    function ExistOverlay: Boolean;
{ 去掉调试信息 }
    procedure StripSymTab;
{ 设置装入模块的大小,即把大小记入 MZHEADER 中 }
    procedure SetImageSize (var MZ: tagMzHeader; SizeImage: LongInt);
{ 浏览文件重定位表 }
    procedure BrowseMzRelocList (Body: PBuf; Operation: ProcMzDo);
{ 显示文件头信息 }
    procedure DisplayMzHeader;
{ 显示文件重定位信息 }
    procedure DisplayMzRelocList;
{ 显示调试信息 }
    procedure DisplaySymTab;
```

1.3　Windows 执行文件首部

整个 Windows EXE 文件头的存放形式是这样的：

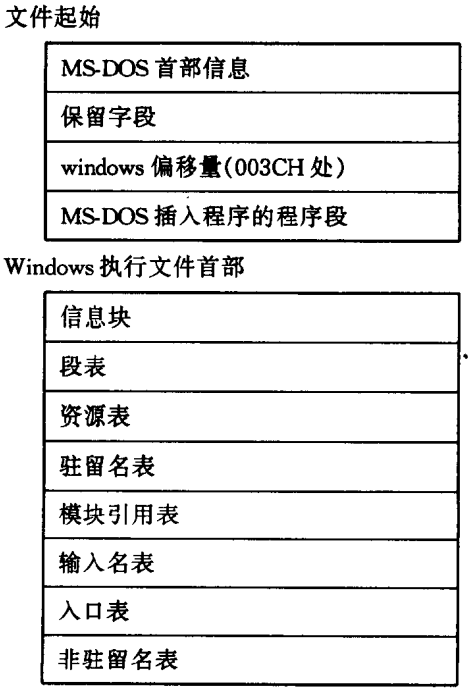

图 1.2　Windows 可执行文件头

接下来详细介绍 Windows EXE 文件首部的各个部分。Windows 文件首部中包含有大量的 Windows 执行信息，可以说是 Windows 运行机制的反映，如果读者是初级的 Windows 编程人员，可能不熟悉其中的一些概念，但这并不要紧，我们先撇下这些概念不管，如果读者有兴趣，请顺着本书的章节阅读下去，第四章阅读完后，这些概念就清楚了。

1.3.1 信息块

Windows 可执行文件首部中的信息块包含 LINK 版本和一系列用来进一步描述该可执行文件各种表的长度、从首部开始到上述表的起始的偏移量以及堆栈的大小等等。

下表概括了首部信息块的各项内容，其中“偏移”是相对于信息块的起始：

（在下表中，需要特别说明的地方均以黑体字给出）

<table>
<tr><th>偏移</th><th>说　明</th></tr>
<tr><td>00H</td><td>Windows 执行文件特征字“NE”，即 454EH</td></tr>
<tr><td>02H</td><td>LINK 版本号</td></tr>
<tr><td>03H</td><td>LINK 内部版本修订号</td></tr>
<tr><td>04H</td><td>入口表的偏移量(从首部开始)</td></tr>
<tr><td>06H</td><td>以字节为单位指定入口表的长度</td></tr>
<tr><td>08H</td><td>系统保留(装载时的变量单元)</td></tr>
<tr><td>0CH</td><td>可执行文件特征标记，该值可能是下列值中的一个或多个的组合(至于该特征标记在装载到内存后的情况，请阅读第 4 章第 4 节)：
<table>
<tr><th>位</th><th>意义</th></tr>
<tr><td>0-1</td><td>如果文件是 SINGLEDATA 格式，Linker 设置位 0 为 1，这样的文件只有一个数据段，程序可以同时被执行多次，参见第 4 章第 4.4.6 节。
如果可执行文件是 MULTIPLEDATA 格式，Linker 设置位 1，这样的执行文件包含多个数据段。程序只能同时被执行一次，参见第 4 章第 4.4.6 节。
如果位 0 和位 1 都没有置位，那么可执行文件格式是 NOAUTODATA，这种格式的可执行文件没有自动数据段。</td></tr>
<tr><td>2</td><td>系统保留(Global Initialization)</td></tr>
<tr><td>3</td><td>系统保留(Protected mode only)</td></tr>
<tr><td>8-9</td><td>系统保留(位 8 与位 9 一起说明文件中的 API 类型)
9 位　8 位　含义
0　0　Unknown
0　1　not compatible with windows API
1　0　compatiable with windows API
1　1　Windows API</td></tr>
<tr><td>11</td><td>如果该位被设置，则文件中包含自装载的代码。有关自装载的问题，请参阅第六章。</td></tr>
<tr><td>13</td><td>如果该位被设置，表示连接时出现过错误，但仍然生成了这个可执行文件</td></tr>
</table>
</td></tr>
</table>

14	如果该位被设置，则该文件是一个 DLL 库模块。
15	如果该位被设置，则 CS:IP 寄存器的值为初始化程序的入口点。

0EH	指定自动数据段的段号。如果 SINGLEDATA 和 MULTIPLEDATA 位被清 0，即该程序没有自动数据段，则该值也为 0。（Windows 程序中的段号是从 1 开始的。）
10H	以字节为单位指定局部堆的初始大小。如果没有局部堆，则该值为零
12H	以字节为单位指定堆栈的初始大小。如果 SS！ = DS，则该值为 0
14H	CS:IP
18H	SS:SP SS 是段表中的段号。如果 SS 为自动数据段的段号，并且 SP 为零，装入时 SP 为自动数据段大小加上堆栈的初始大小所得的地址。
1CH	段表中段的项数
1EH	模块引用表中的项数
20H	非驻留名表中的字节数
22H	相对于 Windows 首部的段表偏移值
24H	相对于 Windows 首部的资源表偏移值
26H	相对于 Windows 首部的驻留名表偏移值
28H	相对于 Windows 首部的模块引用表偏移量
2AH	相对于 Windows 首部起始的输入名表偏移值
2CH	相对于文件起始的非驻留名表相对偏移值 【注意】此处是相对于文件起始
30H	可移动入口项的个数
32H	标识逻辑扇区大小的移位计数值。如果该值为 4，则逻辑扇区大小为 16；如果该值为 9，则逻辑扇区的大小为 512；依此类推
34H	资源块的数目
36H	该字节的值指定了目标操作系统。 【注意】目标操作系统的类型是根据该字节的值指定的，而不是按位决定的。许多文档在这里都描述错了

数值	意　义
0	未知的操作系统
1	OS/2
2	系统是 Microsoft Windows
3	MS-DOS 4.0
4	Windows 386
5	BOSS(用 Borland Pascal 或 C 的保护模式编译生成的程序格式)
6	Invalid

37H	指定可执行文件的附加信息,可以是下列值的一个或多个的组合	
	位	意　义
	0	保留(**Support for EAs and long file names**)
	1	如果该位被设置为 1,文件包含一个 Windows 2.x 版本的应用程序,该程序以 3.x 版本保护模式运行
	2	如果该位被设置为 1,可执行文件包含一个 Windows 2.x 版本的应用程序,该程序支持可缩放字体
	3	如果该位被设置为 1,可执行文件包含一个快速装载区
38H	以扇区为单位指定从文件头到快速装载区起点的偏移量(当 not gangload, offset to return thunks)	
3AH	以扇区为单位指定快速装载区的长度(Windows 内部使用)(offset to segment return thunk)	
3CH	系统保留(最小磁盘交换区的大小 MinSwapSize)	
3EH	期望的 Windows 版本号(仅 Windows 使用该值)	

图 1.3　NE 信息块

文件信息块的 Pascal 定义本章稍后给出。读者可以将这里的说明和后面的 tagNE-Header 对照理解。

1.3.2　段表

段表包含描述可执行文件中每一段的信息,其中包括段长度、段类型和段重定位数据。下表概括了段表中的值(定位相对于每项的起始):

偏移	**说　明**	
00H	以扇区为单位指定到该段数据的偏移量(相对于文件的起始)。零值表示没有数据存在	
02H	以字节为单位指定文件中该段的长度。零值表示段长度为 64K,除非选择器偏移量也为零	
04H	可执行文件内容的标记。该值可以是下列值中的一个或多个:	
	位	意　义
	0	如果该位置 1,则该段是一个数据段;否则该段为代码段
	1	如果该位置 1,装载器分配内存给该段
	2.	如果该位置 1,表示是否现在装入该段
	3	系统保留(**Iterated**)
	4	如果该位置 1,则该段类型为 MOVEABLE;否则段类型为 FIXED
	5	如果该位置 1,则段类型为 PURE 或 SHARABLE;否则段类型为 INPURE 或 NONSHAREABLE

	6	如果该位置 1，则段类型为 PRELOAD；否则段类型为 LOADONCALL
	7	如果该位置 1 并且该段是一个代码段，则段类型为 EXECUTEONLY 如果该位置 1 并且该段是数据段，段类型为 READONLY
	8	如果该位置 1，该段包含可定位数据
	9	系统保留
	10	系统保留
	11	系统保留
	12	如果该位置 1，则该段是个可放弃段
	13	系统保留
	14	系统保留
	15	系统保留(**Large Data**)(参见第六章“自装载 Windows 程序”中的定义)
06H	以字节为单位，指定段的最小分配长度。零值表示最小分配长度为 64K	

图 1.4　段表

需要注意的是，段表中的内容在该程序执行时永久地存放内存中(Windows KERNEL 为程序维护一个 Module Database，其中含有该程序的段表。请参见第四章)，其中的“文件内容标记”随着该段的状态变化而有所不同。

1.3.3　资源表

资源表用来描述并且标志可执行文件中每个资源的位置。该表具有如下形式：

```
WORD          rscAlignShift;
TYPEINFO      rscTypes[ ];
WORD          rscEndTypes;
BYTE          rscResourceNames[ ];
BYTE          rscEndNames;
```

下面说明资源表中的各个成分：

rscAlignShift	资源数据的连接移位计数。把移位计数作为 2 的指数得到了一个以字节为单位的因子，该因子用来计算可执行文件中资源的位置。比如：rscAlignShift 为 4，则因子为 16
rscTypes	包含资源类型信息的 TYPEINFO 结构数组。可执行文件中每种类型的资源都必须有一个 TYPEINFO 结构
rscEndTypes	rscTypes[ ]结束标志。这一项必须为 0
rscResourceNames	与表中资源对应的名字(如果有的话)。每个名字被存放在连续的字节中；第一个字节指定了名字中的字符个数
rscEndNames	rscResourceNames[ ]和整个资源表结束的标志。这一项必须为 0

图 1.5　资源表

类型信息

TYPEINFO 结构具有如下形式：

```
typedef struct_TYPEINFO {
    WORD        rtTypeID;
    WORD        rtResourceCount;
    DWORD       rtReserved;
    NAMEINFO    rtNameInfo[ ];
} TYPEINFO;
```

下面说明 TYPEINFO 结构中的各项：

<table>
<tr><td rowspan="3">rtTypeID</td><td colspan="3">资源类型标记。这个整数值既可以是资源类型值也可以是一个指向资源类型名的偏移量。如果该项的高位被置位(0x8000)，则取下列资源类型值之一：</td></tr>
<tr><td colspan="3"><table>
<tr><th>值</th><th>资源类型</th><th>资源名称</th></tr>
<tr><td>9</td><td>加速键表</td><td>RT_ACCELERATOR</td></tr>
<tr><td>2</td><td>位图</td><td>RT_BITMAP</td></tr>
<tr><td>1</td><td>光标</td><td>RT_CURCOR</td></tr>
<tr><td>5</td><td>对话框</td><td>RT_DIALOG</td></tr>
<tr><td>8</td><td>字体</td><td>RT_FONT</td></tr>
<tr><td>7</td><td>字体目录</td><td>RT_FONTDIR</td></tr>
<tr><td>12</td><td>组光标</td><td>RT_GROUP_CURSOR</td></tr>
<tr><td>14</td><td>组图符</td><td>RT_GROUP_ICON</td></tr>
<tr><td>3</td><td>图符</td><td>RT_ICON</td></tr>
<tr><td>4</td><td>菜单</td><td>RT_MENU</td></tr>
<tr><td>10</td><td>资源数据</td><td>RT_RCDATA</td></tr>
<tr><td>6</td><td>字符串表</td><td>RT_STRING</td></tr>
</table></td></tr>
<tr><td colspan="3">如果该项的高位没有被置位，其值表示一个以字节为单位指向 rscResourceName 中名字的偏移量，该偏移量是相对于资源表的起始</td></tr>
<tr><td>rtResourceCount</td><td colspan="3">可执行文件中该类型资源的数目</td></tr>
<tr><td>rtReserved</td><td colspan="3">系统保留</td></tr>
<tr><td>rtNameInfo</td><td colspan="3">包含单个资源信息的 NAMEINFO 结构数组。</td></tr>
<tr><td>rtResourceCount</td><td colspan="3">指定了结构数组中的结构数量</td></tr>
</table>

图 1.6 资源信息

名字信息

NAMEINFO 结构具有如下形式：

```
typedef struct_NAMEINFO {
  WORD    rnOffset;
  WORD    rnLength;
  WORD    rnFlags;
  WORD    rnID;
```

```
    WORD    rnHandle;
    WORD    rnUsage;
} NAMEINFO;
```

下面说明 NAMEINFO 中的各个成分：

rnOffset	资源数据内容的偏移量(相对于文件的起始)	
	rnOffset 乘以 rscAlignShift 所代表的字节数即为真正的偏移量	
rnLength	以资源页为单位指定资源长度。【注意】并不是以字节为单位。	
rnFlags	指定资源是否固定，是否预装载，是否可共享。其值可以是下列值的一个或多个。	
	值	意　义
	0x0010	资源是可移动的(MOVEABLE)，否则是固定的(FIXED)
	0x0020	资源可以被共享(PURE)
	0x0040	资源是预先装载的(PRELOAD)；否则，在需要时装载(LOADONCALL)
rnID	资源标志或指向资源标志的指针。该标记可能是一个高位被置位(0x8000)的整数；	
	如果该标记高位未被置位，则这是一个指向资源字符串的偏移量，该偏移量是相对于资源表起始的	
rnHandle	系统保留	
rnUsage	系统保留	

图 1.7　名字信息

1.3.4　驻留名表

驻留名表包含标识可执行文件中的输出函数的字符串，这些字符串驻留在系统内存中并且从不被放弃。驻留名字符串区分大小字母并且不以空字符结束。下表概括了驻留名表中的值，其中“偏移”是相对于每一项的起始：

偏　移	说　明
00H	字符串的长度。如果表中再没有更多的字符串，该值为零
01H-xxH	驻留名字符串。这个字符串是区分大小写字母并且不以空字符结束的
xxH+01H	标识字符串的序数，该序数指向入口表的索引

图 1.8　驻留名表

驻留名表中的第一个字符串是模块名，这是缺省约定。

1.3.5　模块引用表

模块引用表包含模块名的偏移量，这些模块名存放在输入名表中。模块引用表中的每一项都是 2 字节长。

1.3.6 输入名表

输入名表包含可执行文件输入的模块名。每一项包含两部分:指定字符串的长度的一个字节及字符串本身。表中的字符串不以空字符终止。

1.3.7 入口表

入口表包含可执行文件的入口点集(bundles of entry points)(每个集合都由连接器产生)。入口表序数值以 1 为基数,即对应于第一入口点的序数值为 1。入口表数据通过束来组织,每束以一个 2 字节的首部开始。首部的第一字节指定束中的项数(00H 表示入口表的结束)。第二个字节指定相应的段是可移动的还是固定的。如果该字节的值为 0FFH,则相应的段是可移动的。如果该字节中的值是 0FEH,则该项不是指向一个段而是指向一个模块中定义的常数。如果这个字节的值不是 0FFH 也不是 0FEH,则该项是一个段索引。特别地,如果第二个字节为 00H,表示是 NULL Entry。比如,01 00 表示有一个 NULL Entry。放置一些空的入口表项只是为了使入口表更规整,并无多大的意义。

对于可移动的段,每一项由六个字节组成,其形式如下:

偏移	说 明	
00H	一个字节,该值可以是下列各位的组合:	
	位	意 义
	0	如果该位被置位为 1,则该项被输出
	1	如果该位被置位为 1,则段使用全局(共享的)数据段
	3-7	如果可执行文件包含执行环转移的指令,这些位指定组成栈的字数。在环转移时,这些字必须从一个环复制到另一个中去。
01H	指定 INT 3FH 指令。中断 INT 3FH 功能调用 Windows Loader,装载该段。参见第四章。	
03H	指定段号	
04H	指定段偏移量	

图 1.9 入口表(可移动段)

对于固定段,每一项由三个字节组成,其形式如下:

偏移	说 明	
00H	一个字节,该值可以是下列各位的组合:	
	位	意 义
	0	如果该位被置位为 1,则该项被输出
	1	如果该位被置位为 1,则该项使用全局(共享的)数据段(这位只有对 SINGLE-DATA 库模块才可能被置位)

	3-7	如果可执行文件包含执行环转移的指令，这些位指定组成栈的字数。在环转移时，这些字必须从一个环复制到另一个中去
01H	指定一个偏移量	

图 1.10　入口表(固定段)

1.3.8　非驻留名表

非驻留名表包含标识可执行文件中输出函数的字符串。这些字符串不是一直驻留在系统内存中，并且是可以放弃的。非驻留名字符串区分大小写字母并且不以空字符结束。下表概括了非驻留名表中的值("偏移"是相对于每一项的起始点)：

偏　移	说　明
00H	字符串的长度。如果表中再没有更多的字符串，该值为零
01H-xxH	非驻留名字符串。这个字符串是区分大小写字母并且不以空字符结束的
xxH+01H	标识字符串的序数，该序数指向入口表的索引

图 1.11　非驻留名表

非驻留名表中的第一个名字是模块描述字符串(在模块定义文件中指定)。

1.3.9　文件头分析实例——PBRUSH.EXE 的文件头

下面的例子全面分析了 PBRUSH.EXE 的 Windows 文件头。用它来直观地验证我们在前面对 Windows 文件头的文字描述。

```
0000  4D 5A 3F 00 08 00 00   00-20   00 00 00 FF FF 07 00  MZ?.............
"4D 5A"          MS-DOS 文件头标记 'MZ'
"3F 00 08 00"    文件的大小为 07 x 512 + 63 (3F) = 3647 = 0x0E3F
                 DOS 第一次读入的数据的多少是根据这里的文件大小决定的，
                 但这里的文件大小并不是实际的文件大小，实际的文件大小
                 可通过 DOS 下的系统功能调用取得。
"00 00"          重定位项数为 0
"20 00"          文件头大小为 20h 节，即 200h 字节
"07 00"          Starting SS 为 0070h
0010  00 01 65 40 00 00 00   00-40   00 00 00 01 00 00 00  ..e@....@.......
"00 01"          Starting SP 为 0100h
"65 40"          文件校验和为 4065
"00 00 00 00"    CS 为 0000，IP 为 0000
"40 00"          文件第一个重定位项的偏移为 0400h。把第一个
                 重定位项的偏移设置为 0400h，就是为了为 003Ch 处的
                 双字预留出空间
0020  00 00 00 00 00 00 00   00-00   00 00 00 00 00 00 00  ................
0030  00 00 00 00 00 00 00   00-00   00 00 00 00 04 00 00  ................
"00 04 00 00"    因为 0018h 处的值为 40h，所以此处的 0000:0400 为
                 Windows 文件头的偏移量.
0040  00 00 00 00 00 00 00   00-00   00 00 00 00 00 00 00  ................
... lots of zeros ...
```

```
01F0  00  00  00  00  00  00  00  00-00   00  00  00  00  00  00  00  ................
至此,MS-DOS 文件头结束
0200  E8  53  00  54  68  69  73  20-70   72  6F  67  72  61  6D  20  . S . This program
0210  72  65  71  75  69  72  65  73-20   4D  69  63  72  6F  73  6F  requires Microso
0220  66  74  20  57  69  6E  64  6F-77   73  2E  0D  0A  24  20  20  ft Windows . . . $
0230  20  20  20  20  20  20  20  20-20   20  20  20  20  20  20  20
0240  20  20  20  20  20  20  20  20-20   20  20  20  20  20  20  20
0250  20  20  20  20  20  20  5A  0E-1F   B4  09  CD  21  B8  01  4C        Z.....!..L
0260  CD  21  00  00  00  00  00  00-00   00  00  00  00  00  00  00  .!..............
0270  00  00  00  00  00  00  00  00-00   00  00  00  00  00  00  00  ................
... lots of zeros ...
03F0  00  00  00  00  00  00  00  00-00   00  00  00  00  00  00  00  ................
至此, WINSTUB 结束
0400  4E  45  05  14  EF  04  DD  05-20   01  4F  F4  02  03  22  00  NE.......O...".
"4E 45"          Windows 文件的特征字"NE"
"05 14"          Linker 版本为 5.20
"EF 04"          入口表起始地址相对 Windows 文件头为 0x04EF, 即 0x08EF
"DD 05"          入口表长度为 0x05DD
"02 03"          0Ch 处的 WORD 值为 0x0302,表示该可执行文件是 MULTIPLEDATA
                 格式、包含多个数据段、是 Windows AP
"22 00"          该文件有 22H(34)个自动数据段
0410  00  10  00  20  03  05  01  00-00   00  22  00  22  00  08  00  ........."."...
"00 10"          局部堆大小为 4k
"00 20"          堆栈的初始大小为 8k
"03 05 01 00"    CS:IP 为 0100:0503
"00 00 22 00"    SS:SP 为 0022:0000
"22 00"          段表共有 0x0022 项
"08 00"          模块引用表共有 0x0008 项
0420  71  02  40  00  50  01  4E  04-59   04  69  04  CC  0E  00  00  q.@.P.N.Y.i.....
"71 02"          非驻留名表长 0271h
"40 00"          段表的起始地址为 0x0040 + 0x0400 = 0x0440
"50 01"          资源表起始为 0x0550
"4E 04"          驻留名表起始为 0x084E
"59 04"          模块引用表起始为 0x0859
"69 04"          输入名表起始为 0x0869
"CC 0E"          非驻留名表从 0x0ECC 处开始
0430  F5  00  04  00  00  00  02  08-16   01  34  14  00  00  0A  03  .........4.....
"F5 00"          可移动入口点数目为 F5h
"04 00"          逻辑扇区大小为 2 * * 4 = 16
"00 00"          资源段的数目为 0
"02"             Target Operation System 为 Microsoft Windows
"08"             该可执行文件包含一个快速装载区
"16 01"          从文件头开始到快速装载区的长度为 0x1160 字节
"34 14"          快速装载区的长度为 14340H 字节
"00 00"          MinSwapSize 为 0
"0A 03"          期望的 Windows 版本为 3.1
```

段表的起始位置

```
0440  18  01  64  0D  50  1D  64  0D-04   02  27  0E  50  1D  27  0E  ..d.P.d...'.P.'.
每一个段的描述信息占 8 个字节
"18 01"          第一个段的段数据从 1180H 处开始
"64 0D"          该段长 0D64H 个字节
"50 1D"          1D50H = 0001 1101 0101 0000B
```

```
                表示该段为代码段
                Loader 并不为该段分配内存
                该段未被装载
                该段为非重叠的、MOVEABLE、NOSHAREABLE、PRELOAD
                该段不是 EXECUTEONLY
                该段包含可重定位数据
                该段为 DISCARDABLE
"64 0D"         该段的最小分配内存为 0D64H
... lots of segment info omitted ...
Total: 34 (22h) segments
0540  84  24  E0  1C  10  1D  E1  1C-0C  10  A0  4F  51  0D  A0  4F  .$.....OQ..O
"0C ... 4F"     第 34(22H)个段的描述信息
```

资源表的起始位置

```
0550  04  00  0C  80  07  00  00  00-00  00  7B  26  02  00  30  1C  .....{&..0.
"04 00"         资源数据的移位计数为 4
"0C 80"         资源类型标记为 800CH,高位被设置,资源类型为 12,即 GROUP_CURSOR
"07 00"         该资源数目为 7
"00 00-00 00"   保留
"7B 26"         资源表数据内容的位置为 267B0H(相对于文件起始)
"02 00"         资源长 02H
"30 1C"         0x1C30,该资源为可移动的、可共享的、并且在需要时装载
0560  A0  02  00  00  00  00  7D  26-02  00  30  1C  A6  02  00  00  .....}&..0.....
"A0 02"         资源字符串的偏移量为 0x02A0(相对于资源表),即 0x07F0 处
"00 00 00 00"   保留
0570  00  00  7F  26  02  00  30  1C-AD  02  00  00  00  00  81  26  ...&..0........&
0580  02  00  30  1C  B3  02  00  00-00  00  83  26  02  00  30  1C  ..0........&..0.
0590  BA  02  00  00  00  00  85  26-02  00  30  1C  C3  02  00  00  .......&..0.....
05A0  00  00  87  26  02  00  30  1C-C8  02  00  00  00  00  0E  80  ...&..0.........
05B0  01  00  00  00  00  00  89  26-03  00  30  1C  CD  02  00  00  .......&..0.....
05C0  00  00  02  80  03  00  00  00-00  00  8C  26  35  02  30  0C  ...........&5.0.
05D0  D4  02  00  00  00  00  C1  28-8E  00  30  0C  DD  02  00  00  .......(..0.....
05E0  00  00  4F  29  0D  00  30  0C-E8  02  00  00  00  00  04  80  ..O)..0.........
05F0  01  00  00  00  00  00  0E  15-38  00  70  1C  EF  02  00  00  ........8.p.....
0600  00  00  05  80  0C  00  00  00-00  00  5C  29  17  00  30  1C  ..........\)..0.
......
0690  09  00  30  1C  16  80  00  00-00  00  06  80  0E  00  00  00  ..0.............
......
0740  45  00  30  1C  44  80  00  00-00  00  09  80  01  00  00  00  E.0.D...........
0750  00  00  D3  2B  06  00  30  0C-F7  02  00  00  00  00  10  80  ...+..0.........
0760  01  00  00  00  00  00  D9  2B-1E  00  30  0C  01  80  00  00  .......+..0.....
0770  00  00  03  80  02  00  00  00-00  00  F7  2B  13  00  10  1C  ...........+....
0780  01  80  00  00  00  00  0A  2C-2F  00  10  1C  02  80  00  00  .......,/.......
0790  00  00  01  80  07  00  00  00-00  00  39  2C  14  00  10  1C  ..........9,....
......
07E0  00  00  B1  2C  14  00  10  1C-09  80  00  00  00  00  00  00  ...,............
07F0  05  46  4C  4F  4F  44  06  43-52  4F  53  53  48  05  44  55  .FLOOD.CROSSH.DU
0800  4D  4D  59  06  58  44  55  4D-4D  59  08  53  49  44  45  41  MMY.XDUMMY.SIDEA
0810  52  4F  57  04  50  49  43  4B-04  54  45  58  54  06  50  42  ROW.PICK.TEXT.PB
0820  52  55  53  48  08  50  54  4F-4F  4C  42  4F  58  0A  50  42  RUSH.PTOOLBOX.PB
0830  57  54  4F  4F  4C  42  4F  58-06  50  41  52  52  4F  57  07  WTOOLBOX.PARROW.
0840  50  42  52  55  53  48  32  06-50  42  52  55  53  48          PBRUSH2.PBRUSH
```

资源表结束

注意:这里并没有 rscEndNames

```
驻留名表开始
0840                                                    07 50                .p
"07"            驻留名字符串长度
0850  62 72 75 73 68 58 00  00-00                                    brushX...
"50 .. 58"      该驻留名表的第一个字符串是 PbrushX,即模块名
"00 00"         指向入口表的索引

模块引用表开始
0850                                    01 00 58 00 5F 00 63          ..X..c
0860  00 68 00 71 00 78 00  80-00                                    .h.q.x...
"01 00"         PBRUSH          0001 处为 PBRUSH
"58 00"         KERNAL          0058 处为 KERNAL
                GDI
                USER
                KEYBOARD
                OLESVR
                COMMDLG
                SHELL

输入名表开始
可执行文件的输入模块名:字符串的长度和字符串本身
0860                                    00 06 50 42 52 55 53          ..PBRUS
0870  48 0D 56 43 52 45 41  54-45  42 49 54 4D 41 50 0B  H.VCREATEBITMAP.
0880  47 45 54 56 43 41 43  48-45  44 43 07 56 50 41 54  GETVCACHEDC.VPAT
0890  42 4C 54 07 56 42 49  54-42  4C 54 0B 56 53 54 52  BLT.VBITBLT.VSTR
08A0  45 54 43 48 42 4C 54  0D-56  44 45 4C 45 54 45 4F  ETCHBLT.VDELETEO
08B0  42 4A 45 43 54 0B 44  49-53  43 41 52 44 42 41 4E  BJECT.DISCARDBAN
08C0  44 06 4B 45 52 4E 45  4C-03  47 44 49 04 55 53 45  D.KERNEL.GDI.USE
08D0  52 08 4B 45 59 42 4F  41-52  44 06 4F 4C 45 53 56  R.KEYBOARD.OLESV
08E0  52 07 43 4F 4D 4D 44  4C-47  05 53 48 45 4C 4C     R.COMMDLG.SHELL
注意:输入名表中的字符串只有一部分出现在模块引用表中

入口点表开始
入口点表定义了模块中使用的入口点的段、偏移和类型。
该表包含一个以上的项,每一项描述了一给定段的入口点。
入口点集由 Linker 产生,入口表按束组织,每个束包含
两个字节的束首部

08E0                                                          05             .
08F0  FF 01 CD 3F 03 C8 0C  01-CD  3F 03 A6 05 01 CD 3F  ...?.....?.....?
"05"            束中的项数为 5
"FF"            FFh,表示是可移动的
"01"            位 0 被置 1,该项被输出
"CD 3F"         "CD 3F"即 int 3Fh
0900  1F 34 13 01 CD 3F 1D  E1-01  01 CD 3F 18 37 01 01  .4...?.....?.7..
0910  00 01 FF 01 CD 3F 08  00-00  01 00 04 FF 01 CD 3F  .....?.........?
"01 00"           01 00     表示有一个 NULL Entry
                            放置一些空的入口表只是为了使入口表更规整,
                            并无多大的意义
0920  0F FE 04 01 CD 3F 14  00-00  01 CD 3F 07 00 00 01  .....?.....?....
0930  CD 3F 0A 00 00 01 00  05-FF  01 CD 3F 1B 00 00 01  .?.........?....
0940  CD 3F 11 00 00 01 CD  3F-0E  00 00 01 CD 3F 21 49  .?.....?.....?!I
0950  03 01 CD 3F 1B 36 02  05-00  05 FF 01 CD 3F 1B E0  ...?.6.......?..
```

```
0960  06 01 CD 3F 19 82 24 01-CD 3F 13 D0 00 01 CD 3F ...?..$..?.....?
0970  18 27 07 01 CD 3F 02 63-00 01 00 06 FF 01 CD 3F .'...?.c.......?
0980  16 7C 01 01 CD 3F 16 DA-01 01 CD 3F 16 38 02 01 .|...?.....?.8..
0990  CD 3F 16 B1 02 01 CD 3F-16 F7 02 01 CD 3F 16 1E .?.....?.....?..
09A0  03 04 00 07 FF 01 CD 3F-16 78 04 01 CD 3F 16 A0 .......?.x...?..
09B0  04 01 CD 3F 16 45 06 01-CD 3F 16 7D 07 01 CD 3F ...?.E...?.}...?
09C0  16 1F 05 01 CD 3F 16 D5-04 01 CD 3F 16 93 07 03 .....?.....?....
09D0  00 D4 FF 01 CD 3F 16 3F-0E 01 CD 3F 16 0A 0A 01 .....?.?...?....
09E0  CD 3F 16 C2 0B 01 CD 3F-16 2C 0A 01 CD 3F 16 4C .?.....?.,...?.L
09F0  0B 01 CD 3F 16 E3 0D 01-CD 3F 16 CC 0D 01 CD 3F ...?.....?.....?
0A00  16 08 0E 01 CD 3F 16 71-0E 01 CD 3F 16 88 0E 01 .....?.q...?....
0A10  CD 3F 0F 92 09 01 CD 3F-13 1A 1B 00 CD 3F 1F C4 .?.....?.....?..
Entry 61,即 3DH
"00"              该字节位 0 未被置 1,表示该项不是用来输出
......
0EB0  01 4B 08 00 CD 3F 01 59-08 00 CD 3F 01 38 0C 00 .K...?.Y...?.8..
0EC0  CD 3F 01 3E 0C 00 CD 3F-01 8E 0C 00             .?.>...?....
"00"              00 表示入口表结束

非驻留名表开始
  非驻留名表包含可执行文件中输出函数的字符串
  第一个字符串是模块描述字符串,该串在模块定义文件中指定
  索引
0EC0                                              0E 50 43 20              .PC
0ED0  50 61 69 6E 74 62 72 75-73 68 2B 00 00 13 4F 42  Paintbrush+....OB
0EE0  4A 45 43 54 55 50 44 41-54 45 44 4C 47 50 52 4F  JECTUPDATEDLGPRO
0EF0  43 3D 00 13 44 4F 43 53-45 54 44 4F 43 44 49 4D  C=..DOCSETDOCDIM
"3D 00"           索引为 3Dh,即入口表中的 Entry 61(在前面已经说明)
0F00  45 4E 53 49 4F 4E 53 2B-00 08 53 52 56 52 4F 50  ENSIONS+..SRVROP
......
1120  45 53 2A 00 08 45 4E 55-4D 50 45 4E 53 1C 00 0A  ES*..ENUMPENS...
1130  50 41 47 45 53 45 54 44-4C 47 18 00 00           PAGESETDLG . . .
入口索引
没有其它字符串,非驻留名表结束
```

1.3.10　NE 文件首部的数据结构和操作

根据本节对 NE 文件首部信息块和各种表的分析,我们这样来定义 NE 文件首部结构 tagNEHeader。

```
tagNEHeader = Record
  Sign: word;          { 00h 'NE'                              }
  Version,             { 02h The linker version number         }
  Revision: byte;      { 03h The linker revision number        }
  ofsEntryTab,         { 04h The offset of the entry table     }
  lenEntryTab: word;   { 06h The length of the entry table     }
  CheckSum: longint;   { 08h Reserved                          }
  Flags,               { 0Ch Flags that describe exe-file      }
  AutoDataSeg,         { 0Eh The automatic data segment        }
  HeapSize,            { 10h Initial size of local heap        }
  StackSize,           { 12h Initial size of the stack         }
  _IP,                 { 14h Segment:offset of CS:IP           }
```

```
  _CS,
  _SP,                          { 18h Segment:offset of SS:SP          }
  _SS,
  numSegmentTab,                { 1Ch Number of entries in Segment table  }
  numModuleReferenceTab,
                { 1Eh Number of entries in module-reference table }
  numNonresidentNameTab,
                { 20h Number of bytes in the nonresident-name table }
  ofsSegmentTab,                { 22h offset of segment table          }
  ofsResourceTab,               { 24h offset of resource table         }
  ofsResidentNameTab,           { 26h offset of resident-name table    }
  ofsModuleReferenceTab,        { 28h offset of module-reference table }
  ofsImportedNameTab: word;     { 2Ah offset of imported-name table    }
  ofsNonresidentNameTab:        { 2Ch offset of nonresident-name table }
    LongInt;
  numMovableEntryPoints,        { 30h Number of movable entry points   }
  SegmentAlign,                 { 32h Page size shift count            }
  numResourceSeg: word;         { 34h Number of resource segments      }
  TargetOS,                     { 36h Target operating system          }
  ExtFlags: byte;               { 37h Additional info about exe-file   }
  ofsGangLoad,                  { 38h offset of fast-load area         }
  lenGangLoad,                  { 3Ah length of fast-load area         }
  sizeMinCodeSwap,              { 3Ch Reversed.                        }
  WinVer: word;                 { 3Eh Expected version for windows     }
  { ------------------------------------------------------------------}
  { 直接保存各种表的长度,便于存取 }
  lenSegmentTab,
  lenResourceTab,
  lenResidentNameTab,
  lenModuleReferenceTab,
  lenImportedNameTab: word;
end;

{ Windows structure define ---------------------------------------------}
{ 段的类型 }
{ Segment flags }
const
  SEG_DATA          = $0001;
  SEG_MOVABLE       = $0010;SEG_SHAREABLE = $0020;SEG_PRELOAD = $0040;SEG_RELOCIN-
  FO                = $0100;SEG_DISCARD = $1000;SEG_READONLY = $0080;

type
{ Segment table entry }
{ Flags that discribe the contents of exe-file
  Bit     Meanning
  0       Data segment if sets. otherwise, code segment
  1       Loader has allocated memory for the segment
  2       Segment is loaded
  3       ..
  4       MOVABLE if sets. otherwise, FIXED
  5       PURE or SHAREABLE
  6       PRELOAD if sets. Otherwise, LOADONCALL
  7       EXECUTEONLY (code) or READONLY (data)
  8       Contains relocation data
  9       ..
```

```
  10      ..
  11      ..
  12      Discardable
  13      ..
  14      ..
  15      >= 64K
Size is 8 bytes }

{ 段表结构 }
PSegmentEntry = ^tagSegmentEntry;
tagSegmentEntry = Record
  ofs,
  lenSeg,
  Attr,
  lenMem: word;
end;

{ 入口表结构 }
{ Entry table }
PEntryBundle = ^tagEntryBundle;
tagEntryBundle = Record
  Count,
  Flags: byte; { Movable if 0ffh, otherwise, fixed }
end;
{ 可移动的入口 }
PMovableEntry = ^tagMovableEntry;
tagMovableEntry = Record
  Flags: byte;
  CD3F: word;
  SegNum: byte;
  Ofs: word;
end;
{ 固定的入口 }
PFixedEntry = ^tagFixedEntry;
tagFixedEntry = Record
  Flags: byte;
  Ofs: word;
end;
{ 一个通用的入口 }
PAEntry = ^tagAEntry;
tagAEntry = record
  Idx: Word;
  Bundle: PEntryBundle;
  MEntry: PMovableEntry;
  FEntry: PFixedEntry;
end;
{ 重定位 }
{ Reloction Information }
{ RelocAddrType:
    Value     Meaning
    0         Low byte at the specified offset
    2         16-bit selector
    3         32-bit pointer
    5         16-bit offset
    11        48-bit pointer
```

```
    13          32-bit offset
  RelocType:
    Value       Meaning
    0           Internal reference
    1           Imported ordinal
    2           Imported name
    3           OSFIXUP
  Ord:
    Imported ordinal        Index to a module's reference table
                            Ord specify a function ordinal value
    Imported name           Index to a module's reference table
                            Ord to an imported-name table
    Internal reference      Index = 0ffh
      Fixed                 Ord specify an offset to the segment
      Movable               Ord specify an ordinal value found in entry tab}
PRelocItem = ^tagRelocItem;
tagRelocItem = Record
  RelocAddrType,
  RelocType: byte;
  Ofs,
  Idx,
  Ord: Word
end;

{ Predefined Resource Types }
资源
const
  rt_Cursor         = $8001;
  rt_Bitmap         = $8002;
  rt_Icon           = $8003;
  rt_Menu           = $8004;
  rt_Dialog         = $8005;
  rt_String         = $8006;
  rt_FontDir        = $8007;
  rt_Font           = $8008;
  rt_Accelerator    = $8009;
  rt_RCData         = $800A;
  rt_NameTable      = $800F;
  rt_VerInfo        = $8010;
  rt_HF             = $806C;
  rt_DATA           = $80BF;
  rt_CCTL           = $82E2;

  rt_Group_Cursor = rt_Cursor + 11;
  rt_Group_Icon   = rt_Icon + 11;

type
{ Resource table declare }
{ word      rscAlignShift
  typeinfo rscTypes[ ];
  word      rscEndTypes;
  byte      rscResourceNames[ ];
  byte      rscEndNames;
}
PTypeInfo = ^tagTypeInfo;
```

```
tagTypeInfo = Record
  rtTypeID,
  rtResourceCount: word;
  rtReserved: LongInt;
  { rtNameInfo: array [1..rtResCount] of TNameInfo }
end;

PNameInfo = ^tagNameInfo;
tagNameInfo = Record
  rnOffset,
  rnLength,
  rnFlags,     { 10h Movable 20h Pure 40h Preload }
  rnID,
  rnHandle,
  rnUsage: word;
end;

PEachRes = ^tagEachRes;
tagEachRes = Record
  Idx: Word;
  TypeInfo: PTypeInfo;
  NameInfo: PNameInfo;
end;
```

1.4　代码段和数据段的重定位信息

1.4.1　代码段和数据段的重定位信息格式

代码和数据段跟随在 Windows 首部之后。有些代码可能包含对其它段中函数的调用，因此可能需要重定位数据来确定这些引用。这个重定位数据存放在一个重定位表中，该表紧跟着段中的代码和数据。表中字节指定了该表包含的重定位项目数。一个重定位项目是一个标识下列信息的字节集合：

地址类型(段，偏移量，段和偏移量)

重定位类型(内部引用，输入序号，输入名)

段号或序号标志(对于内部引用)

引用表索引或函数序号(对于输入序号)

引用表索引或名表偏移量(对于输入号)

每个重定位项目包含 8 个字节数据，其中第一字节指定下列重定位地址类型之一：

值	意　义
02H	16 位段选择符
03H	32 位指针
05H	16 位偏移量
0BH	48 位指针
0DH	32 位偏移量

图 1.12　重定位的地址类型

第二字节指定下列重定位类型之一：

值	意　义
0	内部引用
1	输入序号
2	输入名
3	OSFIXUP(操作系统确定使用数学协处理器还是使用仿真协处理器)

图 1.13　重定位类型

第三和第四字节指定段内重定位项目的偏移量。

如果重定位类型是输入序号，则第五和第六字节指定一个模块引用表的索引，第七和第八字节指定一个函数序号。

如果重定位类型是输入名，则第五和第六字节是一个指向模块表的索引，第七和第八字节指定一个指向名表的偏移量。

如果重定位类型是内部引用并且是固定的，则第五字节指定该段号，第六字节为零，第七和第八指定一个指向段的偏移量。

如果重定位类型是内部引用并且段是可移动的，则第五字节是 OFFh，第六字节为零，第七和第八字节指定一个段入口表的序号。

1.4.2　代码段和重定位表的实例

从 1.3.9 中的 PBRUSH.EXE 的文件头信息中可以得知：

第一个段的段数据从[1180H]处开始，长 0x0D64 个字节，即从[1180H]到[1EE3H]；该段包含可重定位数据，重定位数据紧跟在该段的后面，所以我们可以这样来取 Segment 1 的代码、数据及重定位信息：

[1180H]到[1EE3H]为代码及数据，从[1EE4H]开始为重定位表。

[1EE4H]处的字为 0028H，即总共有 28H 个重定位项，每个重定位项目包含 8 个字节，即其重定位数据长 140H。

我们把从[1180H]到[203FH]的内容 DUMP 出来，这些内容就是 SEGMENT 1 的代码数据和重定位信息的全部。这里的 DUMP 是按逻辑地址的顺序列出的，不过，读者应先察看 SEGMENT 1 重定位表的内容，再反过头来看 SEGMENT 1 的代码和数据。为了节约篇幅，我们略去了代码段的大部分内容。如果读者对重定位的机制不熟悉，应将某个段的代码和重定位表全部 dump 出来，对照上面说明的重定位地址类型和重定位类型逐个分析。

```
1180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
1190  1E 58 90 45 55 8B EC 1E-8E D8 83 EC 4A 57 56 2B .X.EU.......JWV+
... lots of code omitted ...
1660  2A CD 21 8B 5E 06 89 4F-04 32 E4 89 47 06 8A C6 *.!.^..O.2..G...
1670  89 07 8A C2 89 47 02 5F-5E 8D 66 FE 1F 5D 4D CB .....G._^.f..]M.

1680  C3 FF FF 33 ED 55 9A FF-FF 00 00 0B C0 74 65 8C ...3.U.......te.
```

```
1690  06 3E 0C 81 C1 00 01 72-5B 89 0E 10 0C 89 36 12 .>.....r[.....6.
16A0  0C 89 3E 14 0C 89 1E 16-0C 8C 06 18 0C 89 16 1A ..>.............
16B0  0C 9A FF FF 00 00 9A FF-FF 00 00 9A FF FF 00 00 ................
16C0  33 C0 50 9A FF FF 00 00-FF 36 14 0C 9A FF FF 00 3.P......6......
16D0  00 0B C0 74 1F FF 36 14-0C FF 36 12 0C FF 36 18 ...t..6...6...6.
16E0  0C FF 36 16 0C FF 36 1A-0C 9A FF FF 00 00 50 9A ..6...6.......P.
```

由重定位表可知,[16EAH]处调用的是
SEGMENT 0003:2034H 处的内容
即 Far Call 0003:2034

```
16F0  FF FF 00 00 B0 FF 50 9A-FF FF 00 00 55 8B EC 57 ......P.....U..W
1700  56 53 33 FF 8B 46 08 0B-C0 7D 11 47 8B 56 06 F7 VS3..F...}.G.V..
... lots of code omitted ...
1A20  3E 86 0C D6 D6 75 04 FF-16 8C 0C BE 86 0C BF 86 >....u..........
1A30  0C E8 80 00 BE 86 0C BF-86 0C E8 77 00 E8 27 00 ...........w..'.
```

"B4 4C ... CD 21"即 MOV ah, 4C INT 21 退出 Windows 程序
在第四章讲述"启动过程时"需要引此为证

```
1A40  58 0A E4 75 17 8B 46 06-B4 4C 2E F7 06 01 05 01 X..u..F..L.....
1A50  00 74 07 9A A9 07 00 00-EB 02 CD 21 5F 5E 83 ED .t.........!^..
1A60  02 8B E5 1F 5D 4D CB 8B-0E 7C 0C E3 07 BB 02 00 ....]M...|......
1A70  FF 1E 7A 0C 1E C5 16 2A-0C B8 00 25 2E F7 06 01 ..z....*...%....
... lots of code omitted ...
1D50  5D 4D CA 02 00 8C D8 90-45 55 8B EC 1E 8E D8 57 ]M......EU.....W
1D60  FF 76 06 0E E8 B7 FF 0B-C0 74 12 8D 0E 9C 0C 1E .v.......t......
1D70  51 9A FF FF 00 00 1E 50-9A F2 0B 00 00 5F 83 ED Q......P....._..
```

由重定位表可知,[1D79H]处调用一个远函数 KERNEL.115。

"9A"	[1D78H]处的字节为 9Ah,表示是汇编语言中的 Far Call KERNEL.115
"F2 0B 00 00"	[1D79H]处的 32 位指针值为 0x00000BF2,表示在段内的偏移量为 [00000BF2H]处,即绝对地址 1180 + 0BF2 = [1D72H]处存在一个相同的远函数调用

我们再来绝对地址[1D72H]处的内容
[1D72H]处调用的函数与[1D79H]调用同一个函数,都是 KERNEL.115。

"9A"	[1D71H]处的字节为 9AH,表示是汇编语言中的 Far Call KERNEL.115
"FF FF 00 00"	[1D72H]处的 32 位指针值为 0x0000FFFF,表示调用关系链结束。 这样,[1D79H]处和[1D72H]处调用的函数形成了一个调用关系链。

```
1D80  02 8B E5 1F 5D 4D CA 02-00 00 55 8B EC 53 06 51 ....]M....U..S.Q
1D90  B9 00 04 87 0E 76 0C 51-50 9A FF FF 00 00 5B 8F .....v.QP.....[.
... lots of code omitted ...
1EC0  7E 0A 00 74 05 8B 46 0A-EB 03 B8 01 00 2B D2 52 ~..t..F......+.R
1ED0  50 B8 62 00 50 9A FF FF-00 00 8B D0 2B C0 8B E5 P.b.P.......+...
1EE0  5D 4D CB 90                                     ]M..
```

SEGMENT 1 重定位表开始

```
1EE0              28 00 03  01-F9  0B 02 00 73 00 03 01  (.......s...
```

"28 00"	重定位表共有 0028h(40)项
"03 01"	03 表示重定位地址类型为 32 位指针
"01"	01 表示重定位类型为输入序号
"F9 0B"	重定位项在段内的偏移量为[0BF9H],即绝对地址 1180 + 0BF9 = [1D79H]处 有关[1D79H]处的内容请参考前面在[1D79H]处的解释
"02 00"	0002H 为模块引用表索引,0002 即 KERNEL

```
"73 00"          函数序号为 0073H,即 115
                 模块名与函数序号结合,
                 表示[1D79H]处调用的是 KERNEL.115(OUTPUTDEBUGSTRING)
1EF0  9A 0B 02 00 01 00 03 01-43 03 03 00 7B 00 03 01  ........C...{...
.........
1FC0  F7 02 04 00 51 00 05 01-01 05 02 00 B2 00 03 00  ....Q...........
"05"             05 表示重定位地址类型为 16 位偏移量
"01"             01 表示重定位类型为输入序号
"01 05"          重定位项在段内的偏移量为[0501H],即绝对地址 1180 + 0501 = [1681H]处
                 有关[1681H]处的内容请参考前面在[1681H]处的解释
"02 00"          0002H 为模块引用表索引,0002 即 KERNEL
"B2 00"          函数序号为 00B2H,即 178
                 模块名与函数序号结合,
                 表示[1681H]处调用的是 KERNEL.178(_WINFLAGS)
1FD0  32 05 FF 00 FD 00 03 01-5C 09 02 00 66 00 03 00  2.......\...f...
1FE0  37 05 FF 00 FE 00 03 00-3C 05 FF 00 FF 00 03 00  7.......<.......

"03"             03 表示重定位地址类型为 32 位指针
"00"             00 表示重定位类型为内部引用,而且在段表中,
                 我们已经看到 SEGMENT 1 是 MOVEABLE,
                 所以第五字节为 0FFH,第六字节为 0
"6A 05"          重定位项在段内的偏移量为[056AH],即绝对地址 1180 + 056A = [16EAH]处
                 有关[16EAH]处的内容请参考前面在[16EAH]处的解释
"00 01"          第七和第八字节为入口表序号 0100H,即 256
{查阅入口表序号为 256 的内容,可知它调用的是 SEGMENT 0003:2034h 处的内容}
1FF0  6A 05 FF 00 00 01 03 00-70 05 FF 00 01 01 03 00  j.......p.......
2000  78 05 FF 00 02 01 03 01-9E 00 03 00 63 00 03 00  x...........c...
2010  1A 0C FF 00 03 01 03 00-DD 0C FF 00 04 01 03 00  ................
2020  33 0D FF 00 05 01                                3.....
"03"             重定位地址类型均为 32 位指针
"00 或 01"       重定位类型为:01 输入序号,00 内部引用
"xx xx"          指定段内重定位项目的偏移量,即在什么地方需要重定位
"xx xx"          指向模块引用表的索引
2020                    00 00-00 00 00 00 00 00 00 00          ..........
2030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
这是一段空区域
```

上面讲述了重定位的几种类型。如果读者有意加深对“重定位”的理解,请参见第 6 章中提供的源程序 RELOC.ASM。

1.4.3 GetSeg 取某段代码数据的工具

在上一个小节中,确定 Segment 1 的起始地址和重定位数据大小的过程较为复杂。本节提供了一个新的工具 GetSeg,它能首先确定一个段的代码和重定位表的地址范围,然后将这些内容取出来,放在一个单独的文件中。在需要仔细分析一个段的内容时,GetSeg 这个工具显得比较必要。其用法如下:

```
GetSeg <WinExe> <SegNo> <TargetFileNmae>
```

如果参数 SegNo 为 0,则 GetSeg 取出的内容为 WINSTUB。

比如:

```
C:\ >getseg \windows\pbrush.exe 1 pb-seg1.bin
```

你可能已经注意到，如果 SegNo！ = 0，则取出的内容多了一个 16 字节的头，比如，上面 pb-seg1.bin 的前 16 个字节的含义为：

```
0000: 18  01  64  0D  50  1D  64  0D  AE  0E  00  00  02  00  04  00  ..d.P.d.........
段表中描述的 Segment 1
"64 0D"          该段代码和重定位表的总长度
"02 00"          KERNEL 模块在模块引用表中的序号
"04 00"          USER 模块在模块引用表中的序号
```

GetSeg 将 KERNEL、USER 模块在模块引用表中的序号加在该段代码的前面，这样处理的原因请参考 8.3.2。

下面给出 GetSeg 的源程序。

```
{ GetSeg Written by Mr. Leijun 1993.12 }
Uses FileDef, ExtTools, Dos;
{ File Struct Object }
type
  PGetSeg = ^TGetSeg;
  TGetSeg = Object (TNeHeader)
    SegIdx: Word;
    TName : PathStr;
    constructor Init(SFName: PathStr; SegNo: Word; TFName: PathStr);
    procedure Run;
  end;
{ TGetSeg Object }
constructor TGetSeg.Init;
begin
  if Pos('.', SFName) = 0 then SFName := SFName + '.EXE';
  if Pos('.', TFName) = 0 then begin
    if SegNo = 0 then
      TFName := TFname + '.EXE'
    else TFName := TFName + '.BIN';
  end;
  TName := TFName;
  SegIdx := SegNo;
  inherited Init(SFName);
  if rtError <> 0 then fail;
end;
procedure TGetSeg.Run;
var
  FT: File;
  Idx: Word;
  I: LongInt;
  P1, P2: PBuf;
  SegEntry: tagSegmentEntry;
begin
  if SegIdx = 0 then
    GetWINSTUB(TName)
  else begin
```

```
  assign(FT, TName);
  rewrite(FT, 1);
  Move(GetSegmentEntry(nil, SegIdx)^, SegEntry, 8);
  BlockWrite(FT, SegEntry, 8);
  P1 := ReadASeg(SegIdx);
  P2 := ReadARelocList(SegIdx);
  I := P1^.Len + P2^.Len + 2;
  BlockWrite(FT, I, 4);
  Writeln('Length is ', I, ' (', lhnzStr(I), 'h)');
  Idx := GetModuleIdx('KERNEL');
  BlockWrite(FT, Idx, 2);
  Idx := GetModuleIdx('USER');
  BlockWrite(FT, Idx, 2);

  WriteSeek(FT, $10);
  BlockWrite(FT, P1^.P^, P1^.Len);
  if ExistRelocItem (SegIdx) then begin
    I := P2^.Len div 8;
    BlockWrite(FT, I, 2);
    BlockWrite(FT, P2^.P^, P2^.Len);
  end;
  DisposePBuf(P2);
  DisposePBuf(P1);
  Close(FT);
  end;
end;
{ ------------------------------------------------------------------------ }
var
  PRun: PGetSeg;
  SegNo, Code: Word;

begin
  Writeln('GetSeg Version 1.0 Copyright (c) 1993 KingSoft Ltd.');

  if (ParamCount = 3) then
    Val(ParamStr(2), SegNo, Code);

  if (ParamCount <> 3) or (Code <> 0) then begin
    Writeln;
    Writeln('Usage: GetSeg <WinExe> <SegNo> <TargetFileName>');
    Writeln;
    Halt(1);
  end;

  Writeln;
  if SegNo = 0 then
    Writeln('Getting WINSTUB',
      ' from "', ParamStr(1), '" to "', ParamStr(3), '" ...')
  else
    Writeln('Getting seg', segno,
      ' from "', ParamStr(1), '" to "', ParamStr(3), '" ...');
  PRun := New(PGetSeg,
    Init(ParamStr(1), SegNo, ParamStr(3)));
  if PRun = nil then
    DisplayRtError
  else begin
    PRun^.Run;
```

```
    Dispose(PRun,Done);
  end;
end.
```

1.5 资　　源

所谓资源是一组提供给应用程序使用的二进制码。资源大致可分为两类:一是系统标准资源,这些资源在前面的资源表部分已经说明。它们的格式由 Windows 制定,Windows 也提供了许多 API 函数处理这些资源。这些资源的二进制码资料由各种工具准备好后,通过 RC 文件描述,通过资源编译器产生.res 资源文件,再与程序结合在一起。在 Microsoft Visual C++ 1.5 的 windows.h 中,有如下的一段定义:

```
/* Predefined Resource Types */
#define RT_CURSOR           MAKEINTRESOURCE(1)
#define RT_BITMAP           MAKEINTRESOURCE(2)
#define RT_ICON             MAKEINTRESOURCE(3)
#define RT_MENU             MAKEINTRESOURCE(4)
#define RT_DIALOG           MAKEINTRESOURCE(5)
#define RT_STRING           MAKEINTRESOURCE(6)
#define RT_FONTDIR          MAKEINTRESOURCE(7)
#define RT_FONT             MAKEINTRESOURCE(8)
#define RT_ACCELERATOR      MAKEINTRESOURCE(9)
#define RT_RCDATA           MAKEINTRESOURCE(10)
#define RT_GROUP_CURSOR     MAKEINTRESOURCE(12)
#define RT_GROUP_ICON       MAKEINTRESOURCE(14)
```

另一类资源是由使用者自己定义的。这一类资源是只读的信息,放在资源段,不占用代码/数据段的空间。

在前面我们已经介绍过,Windows EXE 文件头中含有资源表 Resource Table,此表格中记录着每一项资源的位置、类型、名字 ... 等等。下面我们分类介绍 Windows EXE 中的几种资源的存放格式。

1.5.1 BITMAP

BITMAP 格式

Windows 的位图文件以 DIB(Device Independent Bitmap)格式存储,使得 Windows 可以在任何设备上显示这个位图。

Windows 3.0 以上的版本中的 DIB 包含两个不同的部分:一个是 BITMAPINFO 结构,描述了位图中的尺寸和颜色,另一个是定义位图像素的字节阵列。阵列中的位被压缩在一起,但每一个扫描行必须填入 0 直到行结束的 32 位边界处。位图的起始位置是左下角。

```
typedef struct tagBITMAPINFO
{
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD          bmiColors[ ];
```

```
} BITMAPINFO;
```

BITMAPINFOHEADER 的数据结构如下：

```
typedef struct tagBITMAPINFOHEADER
{
    DWORD    biSize;             该数据结构所需的字节数
    LONG     biWidth;            位图宽度,以像素为单位
    LONG     biHeight;           位图高度,以像素为单位
    WORD     biPlanes;           目标设备的位平面,必须为 1
    WORD     biBitCount;         每一个像素的位数,必须为 1, 4, 8 或 24
    DWORD    biCompression;      为 BI_RGB, BI_RLE8 或 BI_RLE4
    DWORD    biSizeImage;        图像的大小,以字节为单位。如果位图是 BI_RGB,则该项为 0
    LONG     biXPelsPerMeter;    目标设备的水平分辨率
    LONG     biYPelsPerMeter;    目标设备的垂直分辨率
    DWORD    biClrUsed;          位图中实际使用的颜色表中的颜色变址数
    DWORD    biClrImportant;     显示位图过程中,被认为重要的颜色索引数
} BITMAPINFOHEADER;

/* constants for the biCompression field */
#define  BI_RGB       0L       位图未被压缩
#define  BI_RLE8      1L       位图的运行长度编码格式为每像素 8 位
#define  BI_RLE4      2L       位图的运行长度编码格式为每像素 4 位

typedef struct tagRGBQUAD
{
    BYTE     rgbBlue;
    BYTE     rgbGreen;
    BYTE     rgbRed;
    BYTE     rgbReserved;
} RGBQUAD;
```

在 PBRUSH.EXE 的 Windows EXE 文件头的资源表中，我们看到有三个位图：

```
Bitmap.PTOOLBOX      offset:  268C0h    length: 2350h
Bitmap.PBWTOOLBOX    offset:  28C10h    length: 08E0h
Bitmap.PARROW        offset:  294F0h    length: 00D0h
```

下面以 Bitmap.PTOOLBOX 为例来说明 BITMAP 的存放格式：

```
Bitmap.PTOOLBOX
268C0  28  00  00  00  3A  00  00   00-17   01  00  00  01  00  04  00  (...:...........
"28 00 00 00"      该 BITMAPINFOHEADER 长 28H
"3A 00 00 00"      位图像素宽度为 3AH,位图像素高度为 117H
"01 00"            目标设备的位平面数为 1
"04 00"            每一像素的位数为 4
                   每一行为 3AH 像素, 按 32 Bit 规整,每一行占 32 字节,即 20H
                   总的像素阵列长 117H * 20H = 22E0H
268D0  00  00  00  00  00  00  00   00-00   00  00  00  00  00  00  00  ................
"00 00 00 00"      biCompression 为 BI_RGB(0),位图未被压缩
"00 00 00 00"      位图是 BI_RGB 格式,该项置 0
                   biXPelsPerMeter 和 biYPelsPerMeter 均未指定

268E0  00  00  00  00  00  00  00    00                                        ........
"00 00 00 00"      biClrUsed 为 0,位图使用对应于 biBitCount 的最多颜色种数,即 16
"00 00 00 00"      biClrImportant 为 0,所有的颜色都重要
```

```
BITMAPINFOHEADER 结束

16 个 RGBQUAD 结构定义了 16 种颜色,每个 RGBQUAD 占 4 个字节
268E0                                -00  00  00  00  00  00  80  00  ........
268F0 00  80  00  00  00  80  80   00-80  00  00  00  80  00  80  00  ................
26900 80  80  00  00  80  80  80   00-C0  C0  C0  00  00  00  FF  00  ................
26910 00  FF  00  00  00  FF  FF   00-FF  00  00  00  FF  00  FF  00  ................
26920 FF  FF  00  00  FF  FF  FF  00                                  ........
RGBQUAD 结束

位图像素的字节阵列开始
26920                                -FF  FF  FF  FF  FF  FF  FF  FF        ........
26930 FF  FF  FF  FF  FF  FF  FF   FF-FF  FF  FF  FF  FF  FF  FF  FF  ................
 ... ... ... ...
28BF0 00  00  00  00  00  00  00   00-00  00  00  00  00  00  00  00  ................
28C00 00  00  00  00  00  C8  FB  C8                                  ........
位图像素的字节阵列结束
从 26928H 到 28C07H 共 22E0H 个字节
28C00                               -00  00  00  00  00  00  00  00        ........
资源按 16 个字节对齐
```

压缩 BITMAP

BITMAP 通常占用很大的空间。如果一个 BITMAP 的 biCompression 为 BI-RGB(0),我们可以对它进行压缩。本书所附的 PackBMP.EXE 可以依照标准的 BMP 压缩方法完成这一功能。PackBMP 的使用方法如下:

```
packbmp <bitmap1> <bitmap2>
```

比如,对于这个刚才描述的 Bitmap.PTOOLBOX,我们可以先用 BPW 的 Resource Workshop 将此资源取出,存放在一个名为 ptoolbx1.bmp 的文件中,然后调用 packbmp。(取出该 bitmap 的具体过程是:运行 Resource Workshop,选取菜单 File.OpenProject,打开 PBRUSH.EXE,双击 Bitmap.PTOOLBOX,选取菜单 Resource.SaveResourceAs,在对话框中指定文件类型为"BMP - device independent bitmap",输入文件名为 ptoolbx1.bmp。Resource Workshop 将此资源取出后,在前面增加了一个 BMP 头。)

```
C:\ >packbmp ptoolbx1.bmp ptoolbx2.bmp
```

这样被压缩的文件就被放在了 ptoolbx2.bmp 中。

```
Uses Dos, ExtTools;

{ Bitmap header definition }

type
  PBitmap = ^TBitmap;
  TBitmap = record
    bmType: Integer;
    bmWidth: Integer;
    bmHeight: Integer;
    bmWidthBytes: Integer;
    bmPlanes: Byte;
    bmBitsPixel: Byte;
```

```
    bmBits: Pointer;
  end;

type
  TRGBQuad = record
    rgbBlue: Byte;
    rgbGreen: Byte;
    rgbRed: Byte;
    rgbReserved: Byte;
  end;


type
  PBitmapInfoHeader = ^TBitmapInfoHeader;
  TBitmapInfoHeader = record
    biSize: Longint;
    biWidth: Longint;
    biHeight: Longint;
    biPlanes: Word;
    biBitCount: Word;
    biCompression: Longint;
    biSizeImage: Longint;
    biXPelsPerMeter: Longint;
    biYPelsPerMeter: Longint;
    biClrUsed: Longint;
    biClrImportant: Longint;
  end;

{ Constants for the biCompression field }

const
  bi_RGB = 0;
  bi_RLE8 = 1;
  bi_RLE4 = 2;

type
  PBitmapInfo = ^TBitmapInfo;
  TBitmapInfo = record
    bmiHeader: TBitmapInfoHeader;
    bmiColors: array[0..0] of TRGBQuad;
  end;

type
  PBitmapFileHeader = ^TBitmapFileHeader;
  TBitmapFileHeader = record
    bfType: Word;
    bfSize: Longint;
    bfReserved1: Word;
    bfReserved2: Word;
    bfOffBits: Longint;
  end;

function PackBitmap (InFName, OutFName: PathStr): Boolean;

  function PackLine (P1, P2: PBytes; Len, BitCount: Word): Word;
  var
  I, J, K, L, M, I1, I2: Word;
  begin
    Dec(Len);
```

```
I1 : = 0;
I2 : = 0;
if (BitCount = 4) then begin
  while (I1 < = Len) do begin                    { 16 Colors }
    J : = I1;
    repeat
      I : = 0;
      K : = I1;
      repeat
        Inc(I);
        Inc(I1);
      Until (I1 > Len) or (P1^[I1] < > P1^[I1-1]);
  Until (I > 1) or (I1 > Len);
  if (I1 > = Len) and (I = 1) then
    K : = I1;
  if (K - J = 2) then begin
    P2^[I2] : = 2;
    P2^[I2 + 1] : = P1^[J];
    P2^[I2 + 2] : = 2;
    P2^[I2 + 3] : = P1^[J + 1];
    Inc(I2, 4);
  end
  else if (K - J = 1) then begin
    P2^[I2] : = 2;
    P2^[I2 + 1] : = P1^[J];
    Inc(I2, 2);
  end
  else if (K > J) then begin
    L : = K-J;
    Repeat
      if L > $7f then
        M : = $7f
      else M : = L;
      Dec(L, M);
      P2^[I2] : = 0;
      P2^[I2 + 1] : = M * 2;
      Move(P1^[J], P2^[I2 + 2], M);
      Inc(I2, AlignSize(M + 2, 2));
    Until (L = 0);
  end;
  if (I > 1) then begin
    repeat
      if I > $7F then
        M : = $7F
      else M : = I;
      Dec(I, M);
      P2^[I2] : = M * 2;
      P2^[I2 + 1] : = P1^[I1-1];
      Inc(I2, 2);
    Until (I = 0);
    end;
  end; { while }
  P2^[I2] : = 0;
  P2^[I2 + 1] : = 0;
```

```
Inc(I2, 2);
end
else if (BitCount = 8) then begin
  while (I1 <= Len) do begin                    { 256 Colors }
    J := I1;
    repeat
      I := 0;
      K := I1;
      repeat
        Inc(I);
        Inc(I1);
      Until (I1 > Len) or (P1^[I1] <> P1^[I1-1]);
    Until (I > 1) or (I1 > Len);
    if (I1 >= Len) and (I = 1) then
      K := I1;
    if (K - J = 2) then begin
      P2^[I2] := 1;
      P2^[I2 + 1] := P1^[J];
      P2^[I2 + 2] := 1;
      P2^[I2 + 3] := P1^[J + 1];
      Inc(I2, 4);
    end
    else if (K - J = 1) then begin
      P2^[I2] := 1;
      P2^[I2 + 1] := P1^[J];
      Inc(I2, 2);
    end
    else if (K > J) then begin
      L := K-J;
      Repeat
        if L > $fe then
          M := $fe
        else M := L;
        Dec(L, M);
        P2^[I2] := 0;
        P2^[I2 + 1] := K - J;
        Move(P1^[J], P2^[I2 + 2], K-J);
        Inc(I2, AlignSize(K-J + 2, 2));
      Until (L = 0);
    end;
    if (I > 1) then begin
      Repeat
        if I > $fe then
          M := $fe
        else M := I;
        Dec(I, M);
        P2^[I2] := M;
        P2^[I2 + 1] := P1^[I1-1];
        Inc(I2, 2);
      Until (I = 0);
    end;
  end; { while }
  P2^[I2] := 0;
  P2^[I2 + 1] := 0;
```

```
        Inc(I2, 2);
      end;
      PackLine := I2;
end;

var
  InF, OutF: File;
  fHeader: TBitmapFileHeader;
  bmHeader: TBitmapInfoHeader;
  PA, PB: PBuf;
  I, J, LineWidth: Word;
  CurrImageSize, bmHeaderLen: LongInt;
  IsPacked: Boolean;
begin
  IsPacked := False;
  assign(InF, InFName);
  reset(InF, 1);
  BlockRead(InF, FHeader, SizeOf(TBitmapFileHeader));
  if (FHeader.bfType = $4D42) then begin
    BlockRead(InF, bmHeader, sizeof(TBitmapInfoHeader));
    if (bmHeader.biSize = $28) and
       (bmHeader.biCompression = bi_RGB) and
       ((bmHeader.biBitCount = 4) or (bmHeader.biBitCount = 8)) then
    begin
      assign(OutF, OutFName);
      rewrite(OutF, 1);
      Seek(InF, 0);
      CopyFileBlock(InF, OutF, fHeader.bfOffBits);
      LineWidth := AlignSize(
        (bmHeader.biWidth * bmHeader.biBitCount + 7) div 8, 4);
      CurrImageSize := 0;
      NewPBufEx(PA, LineWidth, nil);
      NewPBufEx(PB, LineWidth*2, nil);
      Seek(InF, fHeader.bfOffBits);
      for I := 1 to bmHeader.biHeight do begin
        BlockRead(InF, PA^.P^, LineWidth);
        FillChar(PB^.P^, PB^.Len, #0);
        J := PackLine (PA^.P, PB^.P, LineWidth, bmHeader.biBitCount);
        BlockWrite(OutF, PB^.P^, J);
        Inc(CurrImageSize, J);
      end;
      PB^.P^[0] := 0;
      PB^.P^[1] := 1;
      BlockWrite(OutF, PB^.P^, 2);
      Inc(CurrImageSize, 2);
      DisposePBuf(PB);
      DisposePBuf(PA);
      close(Inf);
      { if CurrImageSize < bmHeader.biSizeImage then } IsPacked := True;
      Seek(OutF, 0);
      fHeader.bfSize := fHeader.bfOffBits + CurrImageSize;
      BlockWrite(OutF, fHeader, SizeOf(TBitmapFileHeader));
      bmHeader.biSizeImage := CurrImageSize;
      if bmHeader.biBitCount = 4 then
```

```
        bmHeader.biCompression := bi_RLE4
      else bmHeader.biCompression := bi_RLE8;
      bmHeader.biClrUsed := 0;
      bmHeader.biClrImportant := 0;
      BlockWrite(OutF, bmHeader, SizeOf(TBitmapInfoHeader));
      Close(OutF);
      end;
    end;
    if not IsPacked then CopyFile(InFName, OutFName);
    PackBitmap := IsPacked;
  end;

  var
    FName1, FName2: PathStr;

  begin
    Writeln('PackBitmap (c) 1993.12 by KingSoft Ltd. ');
    Writeln;
    if ParamCount = 0 then begin
      Writeln('Usage: PackBM <bitmap1> <bitmap2>');
      Writeln;
      Halt(1);
    end;

    if (ParamCount >= 1) and (FileExists(ParamStr(1))) then
      FName1 := ParamStr(1);
    if (ParamCount > 1) then
      FName2 := ParamStr(2)
    else begin
      FName2 := '!' + FName1;
    end;

    if PackBitmap(Fname1, Fname2) then
      Writeln('Compression successful. ')
    else Writeln('This bitmap cannot be packed. ');

end.
```

1.5.2 ICON 图符图像和 CURSOR 光标图像

Windows EXE 中的图符图像资源块与该文件的 Windows EXE 文件头资源表中所指明的 ICON 资源相对应，其存放形式与 BITMAP 十分类似。

每个图符图像由图符图像首部、颜色表、XOR 掩码和 AND 掩码组成。图符图像具有如下形式：

```
BITMAPINFOHEADER icHeader;
RGBQUAD          icColors[ ];
BYTE             icXOR[ ];
BYTE             icAND[ ];
```

图符图像首部定义成 BITMAPINFOHEADER 结构，指定了图符位图的尺寸和颜色、格式。BITMAPINFOHEADER 的结构在前面已经说明。值得注意的是：icHeader 只用到了 BITMAPINFOHEADER 结构中 biSize 到 biBitCount 之间的项和 biSizeImage 项。其他项（如 biCompression 和 biClrImportant 等）都必须设置为 0。另外，biHeight 为 icXOR 和icAND 的高度之和。

颜色表定义成 RGBQUAD 结构数组，指定了 XOR 掩码用到的颜色。与位图文件中的颜色表一样，图符图象首部的 biBitCount 成分确定了颜色数组元素的数目。RGBQUAD 的结构已在前面说明。

与 BITMAP 不同的是，图符图像有两个字节数组。一个是 XOR 掩码字节数组，一个是 AND 掩码字节数组。两个掩码数组均按扫描线方式组织，其宽高相等。

为什么图符图像需要设置两个字节数组呢？

这是因为当 Windows 系统绘制一个图符时，系统采用 AND 掩码和 XOR 掩码把图符图象与已存在于显示面上的象素组合起来。Windows 首先将已存在于显示面上的像素与该 ICON的 AND 掩码进行“与”操作，然后再将所得结果与 XOR 掩码进行“异或”操作，这样就绘制成了每个象素的最终颜色。用公式来表示就是：

NEW = (ORG ⋂ icAND) ⊙ (icXOR)

这样，AND 掩码和 XOR 掩码的不同组合就会形成不同的视觉效果：

icAND	icXOR	视觉效果
0	0	黑
0	1	白
1	0	透空
1	1	反白

光标图像与图符图像的结构极为类似，其结构如下：

```
WORD              wXHotspot;
WORD              wYHotspot;
BITMAPINFOHEADER  crHeader;
RGBQUAD           crColors[ ];
BYTE              crXOR[ ];
BYTE              crAND[ ];
```

与图符图像不同的是，光标图像结构的前面有两个字的热点坐标，而且 crHeader 中的 biPlanes 和 biBitCount 项都必须为 1。

在 PBRUSH.EXE 的 Windows EXE 文件头的资源表中，我们看到有两个 ICON 资源：

```
Identifier 1    offset:  2BF70h    length:    0130h
Identifier 2    offset:  2C0A0h    length:02F0h
```

和七个 CURSOR 资源：

```
Identifier 3    offset:  2C390h    length: 0140h
... ...
Identifier 9    offset:  2CB10h    length:0140h
```

图符资源包含了 Windows 应用程序所用图符的图像数据。下面以第一个 ICON 资源为例来说明 ICON 的存放格式：

```
2BF70 28 00 00 00 20 00 00 00-40 00 00 00 01 00 01 00 (.....@.....
"28 00 00 00"     该 BITMAPINFOHEADER 长 28H
"20 00 00 00"     位图像素宽度为 20H,位图像素高度为 40H
        (注意:这里的 40H 为 icAND[]和 icXOR[]的高度之和,即它们分别均为 20H)
"01 00"           目标设备的位平面数为 1
"01 00"           每一像素的位数为 1
                  每一行为 20H 像素, 按 32 Bit 规整,每一行占 4 字节
                  icAND[ ]和 icXOR[ ]总的像素阵列长 100H
2BF80 00 00 00 00 00 01 00 00-00 00 00 00 00 00 00 00 ................
2BF90 00 00 00 00 00 00 00 00                         ........
BITMAPINFOHEADER 结束

2 个 RGBQUAD 结构定义了 2 种颜色,每个 RGBQUAD 占 4 个字节
2BF90                        -00 00 00 00 FF FF FF 00         ........
2BFA0 00 00 00 00 00 1F FF 00-00 7F FF C0 01 FE 0F F0 ................
... lots of other data omitted ...
2C090 F0 00 3F FF F8 00 7F FF-FE 01 FF FF FF FF FF FF ..?.............
ICON 1 资源块结束
```

下面以第一个 CURSOR 资源在文件中的存放格式,图符资源包含了 Windows 应用程序所用光标的图像数据。读者可参照前面对 ICON 存放格式的讲解自行分析 CURSOR 的结构。

```
"00 00"           热点的 X 坐标
"1C 00"           热点的 Y 坐标
2C390 00 00 1C 00 28 00 00 00-20 00 00 00 40 00 00 00 ....(.....@...
2C3A0 01 00 01 00 00 00 00 00-00 01 00 00 00 00 00 00 ................
2C3B0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
... lots of data omitted ...
2C4C0 FF FF FF FF 00 00 00 00-00 00 00 00 00 00 00 00 ................
资源块按 16 字节对齐
```

1.5.3 GROUP_CURSOR 组光标和 GROUP_ICON 组图符

组光标资源指定了与之相关的光标图像数目及每个图像的尺寸。在 Windows EXE 中,组光标结构具有如下形式:

```
typedef struct _CURSORDIR {
    WORD        cdReserved;     保留;必须设置为 0
    WORD        cdType;         资源块类型,必须为 2
    WORD        cdCount;        该块中光标的数目
    CURSORDIRENTRY cdEntries[]; CURSORDIRENTRY 结构数组,每个结构包含一个单独光标的信息
} CURSORDIR;

typedef struct _CURSORDIRENTRY {
    WORD        wWidth;         以像素为单位指定光标的宽度
    WORD        wHeight;        以像素为单位指定光标的高度
    WORD        wPlanes;        位图颜色的平面数;必须设置为 1
    WORD        wBitCount;      每个像素的颜色数目;必须设置为 1
    DWORD       lBytesInRes;    以字节为单位给出资源的大小
    WORD        wImageIndex;    光标图像的索引,该图像与该光标目录相对应的。索引以 1 为基
                                数,并且在所有光标目录中是唯一的,即,在一个可执行文件的光
                                标资源中,所有的 CURSORDIR.cdEntries[ ].wImageIndex 各不相同
```

```
} CURSORDIRENTRY;
```

组图符资源指定了与之相关的图符图像数目及每个图像的尺寸。在 Windows EXE 中，组图符结构与组光标结构非常类似：

```
typedef struct ICONDIR {
    WORD            idReserved;     保留;必须设置为 0
    WORD            idType;         资源块类型,必须为 1
    WORD            idCount;        该块中图符的数目
    ICONDIRENTRY    idEntries[ ];   ICONDIRENTRY 结构数组,每个结构包含一个单独图符的信
                                    息
} ICONHEADER;

typedef struct ICONDIRENTRY {
    BYTE        bWidth;             图符的宽度,必须为 16,32 或 64
    BYTE        bHeight;            图符的高度,必须为 16,32 或 64
    BYTE        bColorCount;        图符中的颜色数目,必须为 2,8 或 16
    BYTE        bReserved;          保留;必须设置为 0
    WORD        wPlanes;            图符位图颜色的平面数
    WORD        wBitCount;          每个像素的颜色数目
    DWORD       dwBytesInRes;       以字节为单位给出资源的大小
    WORD        wImageIndex;        图符图像的索引,该图像与该图符目录相对应的。索引以 1
                                    为基数,并且在所有图符目录中是唯一的,即在一个可执行文
                                    件的图符资源中,所有的 ICONHEADER. cdEntries [ ]. wIm-
                                    ageIndex 各不相同
} ;
```

在 PBRUSH.EXE 的 Windows EXE 文件头的资源表中，我们看到有七个 GROUP_CURSOR 资源：

```
Identifier FLOOD        offset:  267B0h    length: 0020h
Identifier CROSSH       offset:  267D0h    length: 0020h
Identifier DUMMY        offset:  267F0h    length: 0020h
Identifier XDUMMY       offset:  26810h    length: 0020h
Identifier SIDEAROW     offset:  26830h    length: 0020h
Identifier PICK         offset:  26850h    length: 0020h
Identifier TEXT         offset:  26870h    length: 0020h
```

和一个 GROUP

```
Identifier PBRUSH       offset:  26890h    length: 0030h
```

下面以这一组资源为例来说明 GROUP_CURSOR 和 GROUP_ICON 的存放格式：

```
GroupCursor.FLOOD
 267B0 00  00  02  00  01  00  20   00-40   00  01  00  01  00  34  01  .......@.....4.
"00 00"             保留
"02 00"             资源块类型为 2
"01 00"             光标数为 1
"20 00"             光标宽度为 20H,光标高度为 20H(AND) + 20H(XOR) = 40H
"01 00 01 00"       平面数 01,颜色数 01
"34 01 00 00"       资源大小为 00000134H
 267C0 00  00  03  00  00  00  00   00-00   00  00  00  00  00  00  00  ................
"03 00"             FLOOD.cdEntries[0].wImageIndex 为 0003
资源按 16 字节对齐
```

```
GroupCursor.CROSSH
 267D0 00 00 02 00 01 00 20 00-40 00 01 00 01 00 34 01 .......@.....4.
 267E0 00 00 04 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
CROSSH.cdEntries[0].wImageIndex 为 0004

GroupCursor.DUMMY
 267F0 00 00 02 00 01 00 20 00-40 00 01 00 01 00 34 01 .......@.....4.
 26800 00 00 05 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
  DUMMY.cdEntries[0].wImageIndex 为 0005

  ... other GROUP_CURSOR data omitted ...

  GroupCursor.TEXT
 26870 00 00 02 00 01 00 20 00-40 00 01 00 01 00 34 01 .......@.....4.
 26880 00 00 09 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
TEXT.cdEntries[0].wImageIndex 为 0009

GroupIcon.PBRUSH
"00 00"          保留
"01 00"          资源块类型为 1
"02 00"          图符数为 2
 26890 00 00 01 00 02 00 20 20-02 00 01 00 01 00 30 01 ......  ......0.
 268A0 00 00 01 00                                      ....
第 1 图符的数据
"20 20"          图符宽度为 20H,图符高度为 20H
"02"             图符中的颜色数为 02
"01 00"          图符颜色的平面数为 0001
"01 00"          每个像素的所需的 BIT 数为 0001
"30 01 00 00"    资源大小为 00000130H
"01 00"          PBRUSH.idEntries[0].wImageIndex 为 0001

 268A0              20 20 10 00-01 00 04 00 E8 02 00 00      ..........
 268B0 02 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
第 2 图符的数据
"20 20"          图符宽度为 20H,图符高度为 20H
"10"             图符中的颜色数为 10H(16)
"01 00"          图符颜色的平面数为 0001
"04 00"          每个像素所需的 BIT 数为 0004
"E8 20 00 00"    资源大小为 000002E8H
"02 00"          PBRUSH.idEntries[0].wImageIndex 为 0002
 资源按 16 字节对齐
```

在本节中,我们了解了 Windows EXE 中几种资源的存放格式,至于其它的资源格式,比如 MENU、DIALOG、STRING、ACCELERATOR、FONT、VERINFO 等,用户可以参阅相关的数据结构说明资料,按照我们前面示范的方法进行分析,在此不再详细说明。

如何获得一个资源的具体大小,可以使用 File Object 提供的 GetResLength 函数,请参见第 3 章。

第 2 章

Windows 执行文件的分析工具

在第 1 章中，我们用手工方法全面解剖了 Windows EXE 结构，包括 Windows EXE 的文件头、代码/数据段、资源段内容及杂类信息等等，并且还涉及了 Windows 执行文件中各种表之间的关系，这些关系构成了 Windows 执行机制的基础。在本章中，我们暂时不去深究 Windows 的执行机制，而是先熟悉一些 Windows EXE 的分析工具，这些分析工具有的是市面上流行的，有的是作者自己开发的。这些工具以便捷的方式展示 Windows 执行文件中的有用信息，而不必用 DEBUG 或 PCTOOLS 之类的工具去解读可执行文件中的每一个字节。

2.1 分析 Windows 文件格式的常用工具

目前流行的 Windows 文件格式分析工具有：EXEHDR、TDUMP、MAPWIN、EXEDUMP、NEWEXE 等。使用这些工具，我们可以将 Windows 可执行文件的内在组成，包括未公开的函数、未公开的数据结构，一一显示出来；另外，Windows 可执行文件中含有大量运行信息，上述工具将这些运行信息条理化地显示出来，使用户更容易理解 NE 格式中各项的具体含义，也使 Windows 的执行机制直观化。这样，即使没有被分析文件的源代码，我们也可以直接从编译后生成的模块入手，猜测其中的某些函数调用与可执行文件中某种功能之间的关系，甚至直接修改 Windows 可执行文件，以适应我们特别的需要，比如：添加一个对话框，改变一个资源等等，这些内容在第四章说明。

2.1.1 EXEHDR 和 TDUMP

Microsoft 在 MSC/C + + 7.0 中提供了一个叫 EXEHDR 的程序，专门用来观察 Windows 执行文件的内容。Borland International 在 Borland Pascal 等语言产品中也提供了一个类似的工具 Turbo Dump(TDUMP)。

TDUMP 能分析 .OBJ, .EXE, .LIB 和 .DLL 等二进制文件中的各种技术信息，包括重定位、CRC 校验、内存需求、内存堆需求、内存段及其它信息。TDUMP 还能显示与 Turbo Debugger、CodeView 有关的所有符号调试信息。

TDUMP 的一般用法如下：

```
TDUMP filename
```

其中文件名指 .OBJ、.EXE、.LIB 和 .DLL 文件。

TDUMP 完整的命令行形式如下：

```
TDUMP options inputfile outputfile options
```

options 是命令行选项。如果直接在 DOS 提示符下键入 TDUMP 然后回车，TDUMP 就会给出所有的命令行选择项及功能信息。

下面是用 TDUMP 分析 PBRUSH.EXE 的结果。

```
Turbo Dump Version 3.1 Copyright (c) 1988, 1992 Borland International
                Display of File \WINDOWS\PBRUSH.EXE

Old Executable Header

DOS File Size                                  2CC50h  (183376.  )
Load Image Size                                 C3Fh  (  3135.  )
Relocation Table entry count                   0000h  (     0.  )
Relocation Table address                       0040h  (    64.  )
Size of header record          (in paragraphs) 0020h  (    32.  )
Minimum Memory Requirement     (in paragraphs) 0000h  (     0.  )
Maximum Memory Requirement     (in paragraphs) FFFFh  ( 65535.  )
File load checksum                             4065h  ( 16485.  )
Overlay Number                                 0000h  (     0.  )

Initial Stack Segment    (SS:SP)                0007:0100
Program Entry Point      (CS:IP)                0000:0000

New Executable header
Operating system                               Windows
File Load CRC                                  0F44F0120h
Program Entry Point      (CS:IP)                0001:0503
Initial Stack Pointer    (SS:SP)                0022:0000
Auto Data Segment Index                        0022h  (    34.  )
Initial Local Heap Size                        1000h  (  4096.  )
Initial Stack Size                             2000h  (  8192.  )
Segment count                                  0022h  (    34.  )
Module reference count                         0008h  (     8.  )
Moveable Entry Point Count                     00F5h  (   245.  )
File alignment unit size                       0010h  (    16.  )
DOS File Size                                  2CC50h  (183376.  )
Linker Version                                 5.20
Program Flags
    DGROUP                  : multiple (unshared)
    Global initializaton    : No
    Protected mode only     : No
    Application type        : Uses windowing API
    Self Loading            : No
    Errors in image         : No
    Module type             : Application
Other EXE Flags
    2.X protected mode      : No
    2.X proportional font   : No
    Gangload area           : Yes
Start of Gangload Area                         1160h
Length of Gangload Area                        14340h
Miminum code swap area size                    0
Expected Windows Version                       3.10
Segment Table                  Offset: 0440h          Length: 0110h
Resource Table                 Offset: 0550h          Length: 02FEh
```

```
Resident Names Table              Offset: 084Eh       Length: 000Bh
Module Reference Table            Offset: 0859h       Length: 0010h
Imported Names Table              Offset: 0869h       Length: 0086h
Entry Table                       Offset: 08EFh       Length: 05DDh
Nonresident Names Table           Offset: 0ECCh       Length: 0271h

Segment Table                     offset: 0440h

    Segment Number: 01h
    Segment Type:   CODE          Alloc Size : 0D64h
    Sector Offset:  0118h         File length: 0D64h
    Attributes: Moveable Preloaded Relocations Discardable

    Segment Number: 02h
    Segment Type:   CODE          Alloc Size : 0E27h
    Sector Offset:  0204h         File length: 0E27h
    Attributes: Moveable Preloaded Relocations Discardable

    Segment Number: 03h
    Segment Type:   CODE          Alloc Size : 285Ch
    Sector Offset:  0310h         File length: 285Ch
    Attributes: Moveable Preloaded Relocations Discardable

    ... lots of "segment table" omitted ...

    Segment Number: 21h
    Segment Type:   CODE          Alloc Size : 1CE1h
    Sector Offset:  2484h         File length: 1CE0h
    Attributes: Moveable Relocations Discardable

    Segment Number: 22h
    Segment Type:   DATA          Alloc Size : 4FA0h
    Sector Offset:  100Ch      File length: 4FA0h
    Attributes: Moveable Preloaded Relocations

Resource Table                    offset: 0550h

  Sector size: 0010h

  type: Group Cursor
       Identifier: FLOOD
           offset: 267B0h         length: 0020h
       Attributes: Moveable           Shareable

       ... other "Resources" omitted ...
Resident Name Table               offset: 084Eh

    Module Name: 'PbrushX'

Module Reference Table            offset: 0859h
    Module  1: PBRUSH
    Module  2: KERNEL
    Module  3: GDI
    Module  4: USER
    Module  5: KEYBOARD
    Module  6: OLESVR
    Module  7: COMMDLG
    Module  8: SHELL

Imported Names Table              offset: 0869h
    name                                 offset
    PBRUSH                                0001h
    VCREATEBITMAP                         0008h
```

```
    GETVCACHEDC                         0016h
    VPATBLT                             0022h
    VBITBLT                             002Ah
    VSTRETCHBLT                         0032h
    VDELETEOBJECT                       003Eh
    DISCARDBAND                         004Ch
    KERNEL                              0058h
    GDI                                 005Fh
    USER                                0063h
    KEYBOARD                            0068h
    OLESVR                              0071h
    COMMDLG                             0078h
    SHELL                               0080h

Entry Table                         offset: 08EFh
  Moveable Segment Records          ( 5 Entries)
    Entry    1: Segment: 03h  Offset: 0CC8h    Exported
    Entry    2: Segment: 03h  Offset: 05A6h    Exported
    Entry    3: Segment: 1Fh  Offset: 1334h    Exported
    Entry    4: Segment: 1Dh  Offset: 01E1h    Exported
    Entry    5: Segment: 18h  Offset: 0137h    Exported

  Null Entries: 1

  Moveable Segment Records          ( 1 Entries)
    Entry    7: Segment: 08h  Offset: 0000h    Exported

  Null Entries: 1

  ... lots of "Entry Table" omitted ...

    Entry    60: Segment: 0Fh  Offset: 0992h    Exported
    Entry    61: Segment: 13h  Offset: 1B1Ah    Exported
    Entry    62: Segment: 1Fh  Offset: 06C4h

  ... lots of "Entry Table" omitted ...

    Entry    261: Segment: 01h  Offset: 0C8Eh

Non-Resident Name Table             offset: 0ECCh
     Module Description: 'PC Paintbrush + '
     Name: OBJECTUPDATEDLGPROC          Entry:    61
     Name: DOCSETDOCDIMENSIONS          Entry:    43
     Name: SRVROPEN                     Entry:    30
     Name: PALWNDPROC                   Entry:    27
     Name: FULLWP                       Entry:    16
     Name: ITEMSETBOUNDS                Entry:    56
     Name: NULLWP                       Entry:    26
     Name: PARENTWP                     Entry:    1

     ... other "Non-Resident Name Table" omitted ...

Segment Relocation Records

    Segment 0001h relocations

    type        offset      target
    PTR         0BF9h       KERNEL.115
    PTR         0B9Ah       KERNEL.1
    PTR         0343h       GDI.123
    PTR         0C63h       KERNEL.5
    PTR         0D07h       KERNEL.6
```

```
PTR     0C84h     KERNEL.7
PTR     0B91h     KERNEL.137
PTR     006Ch     GDI.3
PTR     0CACh     KERNEL.15
PTR     0D56h     KERNEL.16
PTR     0CC2h     KERNEL.17
PTR     0030h     KERNEL.18
PTR     0059h     KERNEL.19
PTR     054Dh     USER.5
PTR     0334h     GDI.14
PTR     0C4Ah     KERNEL.23
PTR     0C6Fh     KERNEL.24
PTR     0544h     KERNEL.30
PTR     0061h     GDI.30
PTR     02A0h     GDI.39
PTR     0120h     0001h:057Ch
PTR     017Fh     0001h:0616h
PTR     02E7h     USER.180
PTR     02EEh     GDI.66
PTR     02FFh     GDI.69
PTR     00C5h     GDI.80
PTR     0507h     KERNEL.91
PTR     02F7h     USER.81
OFFS    0501h     KERNEL.178
PTR     0532h     0001h:076Ch
PTR     095Ch     KERNEL.102
PTR     0537h     0001h:0948h
PTR     053Ch     0001h:0AEAh
PTR     056Ah     0003h:2034h
PTR     0570h     0001h:084Bh
PTR     0578h     0001h:0859h
PTR     009Eh     GDI.99
PTR     0C1Ah     0001h:0C38h
PTR     0CDDh     0001h:0C3Eh
PTR     0D33h     0001h:0C8Eh
Relocations: 40

... some "Segment Relocation Records" omitted ...

Segment 0022h relocations

type    offset    target
PTR     0A2Eh     000Ch:02F3h
PTR     0A2Ah     001Eh:0598h
PTR     0A5Eh     001Ah:018Dh
PTR     0A56h     0017h:007Eh
PTR     0A4Eh     001Ch:02D5h
PTR     0A3Ah     0006h:0539h
PTR     0A46h     001Ch:007Eh
PTR     0A42h     0012h:01CDh
PTR     0A3Eh     000Bh:049Dh
PTR     0A36h     000Dh:0000h
PTR     0A26h     0005h:0083h
PTR     0A1Eh     0019h:1063h
PTR     0A32h     000Ch:0684h
Relocations: 13
```

TDUMP 本身还有一些缺陷，比如第 1 章中提及的对 Windows EXE 的识别错误：如果我们用 DEBUG 或 PCTOOLS 将 PBRUSH.EXE 文件偏移[18H]的 40H 改为 39H 或更小，PBRUSH.EXE 仍然可以在 Windows 下正确运行，即 Windows 并不以[18H]处的值来判断一个文件是不是 Windows EXE；而 TDUMP 把修改过的 PBRUSH.EXE 当成普通的 DOS EXE，没有分析其中与 Windows 有关的内容。

用命令

```
C:>exehdr /v pbrush.exe
```

观察 PBBRUSH.EXE 所得到的信息与用 TDUMP 所得到的信息大致相同，读者可自己作一个比较。下面是用 EXEHDR 观察 PBRUSH.DLL 时所输出的结果。

```
C:>exehdr /v pbrush.dll

Microsoft (R) EXE File Header Utility Version 3.00
Copyright (C) Microsoft Corp 1985-1992. All rights reserved.

... some details omitted ...

Library:                    PBRUSH
Description:                Virtual bitmap manager
Operating system:           Microsoft Windows - version 3.0
Data:                       SHARED
Initialization:             Per-Process
Initial CS:IP:              seg  1 offset 0010
Initial SS:SP:              seg  0 offset 0000
DGROUP:                     seg  2
Linker version:             5.20
32-bit Checksum:            9f517ae5
Segment Table:              000000c0 length 0010 (16)
Resource Table:             000000d0 length 0000 (0)
Resident Names Table:       000000d0 length 000a (10)
Module Reference Table:     000000da length 0004 (4)
Imported Names Table:       000000de length 000c (12)
Entry Table:                000000ea length 001b (27)
Non-resident Names Table:   00000105 length 007e (126)
Movable entry points:       0
Segment sector size:        16
Heap allocation:            0400 bytes

no. type address file mem flags
  1 CODE 00000190 016b9 016ba EXECUTEREAD, LOADONCALL, NONCONFORMING,
                  NOIOPL, relocs, (fixed), (nondiscardable), (nonshared)
  2 DATA 000019a0 000ce 00180 READWRITE, SHARED, PRELOAD, NOEXPANDDOWN,
                  NOIOPL, (movable), (nondiscardable)
Exports:
ord seg offset name
  6   1   0541  VSTRETCHBLT exported, shared data
  1   1   0075  WEP exported, shared data
  8   1   0d3c  DISCARDBAND exported, shared data
  7   1   0770  VDELETEOBJECT exported, shared data
  4   1   0220  VPATBLT exported, shared data
  5   1   0341  VBITBLT exported, shared data
  3   1   0208  GETVCACHEDC exported, shared data
  2   1   009e  VCREATEBITMAP exported, shared data
```

```
1  type     offset  target
   PTR      0061    imp KERNEL.4
   PTR      0831    imp KERNEL.132
   PTR      0b8d    imp KERNEL.5
   PTR      0138    imp KERNEL.6
   PTR      0bf6    imp KERNEL.7
   PTR      0522    imp GDI.1
   PTR      1053    imp KERNEL.15
... more details omitted ...
   PTR      05d1    imp GDI.80
   PTR      0022    imp KERNEL.91, additive
   PTR      0c14    imp KERNEL.92
   PTR      0c27    imp KERNEL.97
   PTR      008e    imp KERNEL.102
   PTR      0f9c    imp GDI.106
   PTR      16af    imp KERNEL.115
41 relocations
```

由上面列表,我们可看到 PBRUSH.DLL 输出(export)8 个函数。表中"exports"之后的部分其实就是由 NE 文件格式中的 Entry Table、Resident Name Table 和 Nonresident Name Table 所综合而成的。TDUMP 是将上述表格分别列出的,这是两种分析工具之间比较明显的差别。

2.1.2 MAPWIN

MAPWIN 是 Phar Lap 公司开发的。此程序可以很容易地看出.EXE、.DLL、.DRV 等程序中引入(imports)和释出(exports)的 API 函数名,而不是象 EXEHDR、TDUMP 那样仅给出函数代号,正因为它能给出函数名,所以它常被用来发掘 Windows 未公开的 API 函数。MAPWIN 所以能给出函数名,是因为它内部存放有函数名称和对应的"模块名.序号"的表格,例如,如果程序中以序号"KERNEL.15"的形式给出了它调用的函数,MAPWIN 就从对应表格中找到函数名"GlobalAlloc"。在 MAPWIN 中,这个表格存放在文件 WINFUNC.IMP 中,用户可以参照该文件的格式,扩充这种对应关系的描述。

MAPWIN 有两个命令行参数:

```
-impmake     生成一个输入名文件(.IMP);
@filename    指定选用的输入名文件(.IMP),用户可以同时指定数个不同的.IMP 文件
```

下面是用 MAPWIN 观察 PBRUSH.EXE 所得的结果:

```
C:\BIN>MAPWIN \WINDOWS\PBRUSH.EXE
MAPWIN: 4.0 — Copyright (C) 1986-91 Phar Lap Software, Inc.

Dump of the .EXE file — \windows\pbrush.exe

Header information

Target operating system.....Windows
Initial CS:IP.....#0001:0503
Initial SS:SP.....#0022:0000
Initial DS.....#0022
Initial heap size.....4096 bytes (0x1000)
Initial stack size.....8192 bytes (0x8B5E)
```

```
Automatic data segment.....................Multiple
DLLs called by this program                模块引用表(Module Reference Table)的内容
   PBRUSH
   KERNEL
   GDI
   USER
   KEYBOARD
   OLESVR
   COMMDLG
   WINNLS
   SHELL
Exported entry points                      列出程序所 export 的函数
                                        也就是:Entry Table 中所有属性为 Exported 的入口点
                                           NonResident Name Table 的内容
   OBJECTUPDATEDLGPROC (pbrush.61)
   DOCSETDOCDIMENSIONS (pbrush.43)
   SRVROPEN (pbrush.30)
   PALWNDPROC (pbrush.27)
   FULLWP (pbrush.16)
   ITEMSETBOUNDS (pbrush.56)
   NULLWP (pbrush.26)
   PARENTWP (pbrush.1)
   ... other "Exported entry points" omitted ...
Imported references                        根据代码/数据段中重定位信息整理出的函数名和序号
  __AHSHIFT (KERNEL.113)
  _WINFLAGS (KERNEL.178)
  _LCLOSE (KERNEL.81)
  _LLSEEK (KERNEL.84)
  _LOPEN (KERNEL.85)
  _LREAD (KERNEL.82)
  _LWRITE (KERNEL.86)
  _WSPRINTF (USER.420)
  ADDATOM (KERNEL.70)
  ANSILOWER (USER.432)
  ANSINEXT (USER.472)
  ANSIPREV (USER.473)
  ANSIUPPER (USER.431)
  BEGINPAINT (USER.39)
  BITBLT (GDI.34)
  BRINGWINDOWTOTOP (USER.45)
  CHECKDLGBUTTON (USER.97)
  CHECKMENUITEM (USER.154)
  CHECKRADIOBUTTON (USER.96)
  ... other "Imported references" omitted ...
```

和 TDUMP、EXEHDR 相比，MAPWIN 有两个不同点：一是 MAPWIN 给出了函数名，而不象前者那样只给出了序号；二是“Imported Reference”将各段的重定位信息经过整理，按字母顺序给出程序所调用的函数，去掉了重复的函数名，而不是象前者那样依据重定位表，按代码/数据段索引的递增顺序给出其中各处调用的函数。这一点使 MAPWIN 更适合于分析、罗列 Windows 的所有 API 调用，包括未公开的 API。

2.1.3 EXEDUMP

EXEDUMP 由《Undocumented Windows》(Addison-Wesley, 1992)的作者 Andrew Schulman 开发。

EXEDUMP 可分析 Windows 或 OS/2 1.x 可执行文件(EXE、DLL、DRV 等)的细节信息,包括 Entry Table(exports)、Relocation Table(imports)、Segment Table、NE Header、Module Reference Table 以及可能有的 CodeView 调试信息。

显示 Windows 或 OS/2 NE 可执行文件中的信息的命令格式是:

```
EXEDUMP exe_file
```

EXEDUMP 还有如下几个命令行参数:

-CVBLANKS 就大部分纠错版的 Windows 而言,它们并没有将内部函数的符号信息放在里面,而是用空白字符串代替。使用该参数,指定 EXEDUMP 将 CodeView 的空白符号调试信息显示出来。有关 CodeView 符号调试信息的格式可在 MSC Developer's Toolkit Reference (Microsoft Part No. 18161)或 Microsoft 的 Open Tools 中找到。

-EXPORTS 只显示执行文件 export 的函数。这个参数与 MAPWIN -IMPMAKE 的功能相同。使用下面的命令可以建立一个包含 Windows API 函数的数据库:

```
C:\>for %f in (\windows\*.*) do exedump -exports %f >> winfunc.dat
```

-MAGIC 用该选择项可迅速得到可执行文件的特征码(magic number)。

```
MZ - DOS EXE
NE - new exectuable(Windows 或 OS/2 1.x 可执行文件)
LE - linear exectuable(Windows VxD)
LX - linear exectuable(OS/2 2.x 可执行文件)
W3 - WIN386.EXE,为 LE 可执行文件的集合
PE - portable exectuable(Win/32 NT 可执行文件)
```

-DESC 显示"NE"可执行文件的文件名和描述字符串。

-NORELOC 不显示重定位信息。

下面是用 EXEDUMP 观察 HELLOWIN.EXE 所输出的结果:

```
name \windows\pbrush.exe
program                          可执行文件类型:"program"或"library"。
target WINDOWS 3.10              目标操作系统和版本。版本号由"linker"设定。
80286 instructions
dgroup 34                        第 34 个段是"自动数据段"。
start_csip 1:0503                程序进入点
description "PC Paintbrush +."   描述字符串
modname PbrushX                  模块名

begin entry                      入口表:段号、偏移值、序号、名称(如果有的话)。
    3       0ccc     1       PARENTWP
    3       05a2     2       PAINTWP
    31      1334     3       TOOLWP
    ... some "entry"s omitted ...          该程序所 export 的全部函数。
    22      0e88     59      ITEMDOVERB
    15      09a2     60      FILEDIALOG
```

```
    19      1bd8    61      OBJECTUPDATEDLGPROC
    31      06c4    62
    31      1ddb    63
    31      16a0    64
    ... some "entry"s omitted ...
    1       0c38    260
    1       0c3e    261
    1       0c8e    262

begin segtab                                段表:段号、段的起始地址和长度、
                                                段的类型(code、data)、USE16 或 USE32
    1    1180  0d64  CODE  USE16
    2    2040  0e29  CODE  USE16
    ... some "segtab"s omitted ...
    32   24550 0e26  CODE  USE16
    33   25550 1ce0  CODE  USE16
    34   102e0 4ff8  DATA  USE16

begin modref                        DLLs Called by this program
    PBRUSH
    KERNEL
    GDI
    USER
    KEYBOARD
    OLESVR
    COMMDLG
    WINNLS
    SHELL

begin reloc
    1    0503  [e]  start                    [e]表示可执行文件的进入点。
    1    0bf9  [m]  KERNEL  115      PTR [m]表示列出的模块名和函数序号。
    1    0bf2  [m]  KERNEL  115      PTR
    ... some "reloc"s omitted ...
    1    0d33  [m]  PbrushX  262     PTR
    2    0d38  [n]  SIZEWP / * PbrushX 4 * /SEG      [n]表示此模块内部
    2    0d75  [n]  COLORWP / * PbrushX 5 * /    SEG 也使用到了此 export 项。
    2    0523  [n]  MOUSEDLG / * PbrushX 10 * /   SEG
    2    0543  [n]  COLORDLG / * PbrushX 12 * /   SEG
    2    0dcd  [n]  FULLWP / * PbrushX 16 * /       SEG
    2    0e0b  [n]  ZOOMOTWP / * PbrushX 17 * /   SEG
    2    09c6  [m]  USER 110  PTR
    2    09b5  [m]  USER 111  PTR
    ... some "reloc"s omitted ...
    16   1278  [m]  PbrushX  184     PTR
    16   1554  [i]  PBRUSH VBITBLT   PTR       表示列出的模块名和函数名称
    16   0dd3  [i]  PBRUSH VBITBLT   PTR
    16   0e5f  [i]  PBRUSH VBITBLT   PTR
    16   07f2  [i]  PBRUSH VBITBLT   PTR
    16   0840  [i]  PBRUSH VBITBLT   PTR
    16   05bd  [i]  PBRUSH VBITBLT   PTR
    16   0631  [i]  PBRUSH VBITBLT   PTR
    16   1220  [m]  USER  282  PTR
    ... some "reloc"s omitted ...
    33   19d8  [m]  PbrushX  243     SEG
    33   0923  [m]  PbrushX  121     PTR
```

```
... some "reloc"s omitted ...
34    0a26  [m]  PbrushX  235      PTR
34    0a36  [m]  PbrushX  109      PTR
```

我们对 EXEDUMP 输出个项的含义都有了比较详细的说明，读者应该能理解。值得说明的是，EXEDUMP 根据被分析程序中“Relocation Table”的重定位类别列出了各种辅助信息：

[i] Imported by name:列出的是模块名和函数名称
[m] Imported by ordinal:列出的是模块名和函数序号
[n] 模块内部也使用此 export 项
[s] 修正模块内某处的 segment:offset 地址
[e] 执行文件的进入点，名称固定为“start”
[g] 来自__GP 表的 GP fault 的处理程序地址

PTR 被移位修正的是一个 4-byte 的远程指针
SEG 被移位修正的是一个 2-byte 的段地址
OFF 被移位修正的是一个 2-byte 的偏移

2.1.4 NEWEXE

NEWEXE V0.01 是一个文件头分析工具，由 Thuan-Tit Ewe 开发。它的用法如下：

```
Usage: NEWEXE [options] <file-list>
```

缺省情况下，NEWEXE 将显示文件头和各种表格的信息，如果指定 option，NEWEXE 就只给出指定部分的信息。options 的有效值有：

```
/t[types]
Valid types are:
        'h' Header
        'r' Resource
        'e' Entry table
        's' Segment table
        'l' Relocation tables
        'b' Resident table
        'n' Non-resident table
        'a' All (default)
/rv     Resource verbose
/i      Suppress display of internal relocations
/c      Outputs C tables
/heap:(0-ffffh)
/stack:(0-ffffh)
/verbose
/help
/l      Displays license information.
```

下面是用 NEWEXE 分析 PBRUSH.EXE 时所得的输出：

```
Thuan's NEWEXE File Header Utility Version 0.01
Copyright (C) Thuan-Tit Ewe 1991. All rights reserved.

Header:
Name:                           PbrushX
Description:                    PC Paintbrush +
... some boring details omitted ...
```

```
Resource table:
Type:800c, offset:160960, size: 32, flags:1c30, id:02a0 Unknown
... some "Resource"s omitted ...
Type:800e, offset:161184, size: 48, flags:1c30, id:02cd Icon header
Type:8002, offset:161232, size:9040, flags:0c30, id:02d4 Bitmap
Type:8002, offset:170272, size:2272, flags:0c30, id:02dd Bitmap
Type:8002, offset:172544, size:208, flags:0c30, id:02e8 Bitmap
Type:8004, offset: 86880, size:944, flags:1c70, id:02ef Menu template
Type:8005, offset:172752, size:352, flags:1c30, id:8002 Dialog-box template
... some "Resource"s omitted ...
Type:8006, offset:176096, size: 32, flags:1c30, id:8007 String table
... some "Resource"s omitted ...
type:8009, offset:180496, size: 96, flags:0c30, id:02f7 Accelerator table
Type:8010, offset:180592, size:480, flags:0c30, id:8001 Unknown
Type:8003, offset:181072, size:304, flags:1c10, id:8001 Icon
Type:8003, offset:181376, size:752, flags:1c10, id:8002 Icon
Type:8001, offset:182128, size:320, flags:1c10, id:8003 Cursor
... some "Resource"s omitted ...

Entry table:
[  1] Exprt:1, Dat:0, Words:0, Sgm:3, Offset:3276[0ccc]
[  2] Exprt:1, Dat:0, Words:0, Sgm:3, Offset:1442[05a2]
[  3] Exprt:1, Dat:0, Words:0, Sgm:31, Offset:4916[1334]
[  4] Exprt:1, Dat:0, Words:0, Sgm:29, Offset:481[01e1]
[  5] Exprt:1, Dat:0, Words:0, Sgm:24, Offset:311[0137]
[  6] Zero type.
[  7] Exprt:1, Dat:0, Words:0, Sgm:8, Offset:0[0000]
[  8] Zero type.
.... other "Entry" omitted ...

Segments:
Segment:    Offset:      Sector:      Size:  Flags:MinAlloc: Relocs:
1           4480  280    3428  1d50   3428   41 CODE (movable)
            NONSHARED PRELOAD relocs NONCONFORMING NOEXPANDDOWN IOPL
            (discardable)
... some "Segment" omitted ...
34          66272  4142  20472  0d51  20472  13  DATA (movable)
            NONSHARED PRELOAD relocs NONCONFORMING NOEXPANDDOWN IOPL
            (nondiscardable)

Relocation(s) for segment 1:
Module:KERNEL.115       OUTPUTDEBUGSTRING
Module:KERNEL.1     FATALEXIT
Module:GDI.123      PLAYMETAFILE
... some "Relocation entry" omitted ...
Moveable: Ordinal 82:
Moveable: Ordinal 88:
... some "Relocation entry" omitted ...
Moveable: Ordinal 262:

... Relocation(s) for segment 2 to 34 omitted ...

Resident name table:
PbrushX(0)

Non resident name table:
PC Paintbrush + (0)
OBJECTUPDATEDLGPROC(61)
```

```
DOCSETDOCDIMENSIONS(43)
SRVROPEN(30)
... some "Non resident name" omitted ...
ENUMPENS(28)
PAGESETDLG(24)
```

除了在"Relocation(s) for Segment #x"栏中没有给出重定位的地址外，和其它文件分析工具相比，NEWEXE 给出的分析结果是最为详细的。

2.2　Power 系列分析工具

这一节要介绍的两个新的分析工具，Power Dump(PDUMP)、Power FileInfo(PFI)。它们与第 1 章中已经介绍的 GetSeg、Power Strip(PSTRIP)和第 3 章中将要介绍的 MSDUMP 等工具一起被称为"Power 系列分析工具"，这些工具由"黄玫瑰软件开发组"(Yellow Rose Software Workgroup)研制。

既然已经有了上一节中介绍的好几个文件分析工具，为什么还要开发新的类似工具？我们是基于以下的几点考虑：一是前面介绍的工具有一些小的 Bug；二是每个工具都有不同的侧重点，如 MAPWIN 和 EXEDUMP 侧重于分析可执行文件中的函数调用，以比较出未公开的函数和数据结构，而 TDUMP 和 NEWEXE 侧重于按文件的逻辑地址逐个分析其中的执行信息，我们的工具也有自己的特点；三是在我们的开发工作中有一些特殊的需求，现有的工具无法完成，我们必须自己开发，所谓"磨刀不误砍柴功"；最后的一个考虑是，自己编写 Windows 可执行文件分析工具本身就是一项极富挑战意义的工作，它能促使编程人员真正了解 Windows 最核心的东西。一个对 Windows 内部的一切都喜欢刨根问底、又需要编写一些深入 Windows 核心的程序的程序员确实需要这种编程经历。

2.2.1　Power Dump (PDUMP)

PDUMP 是一个综合性的 DUMP 工具，它能够分析 DOS EXE、Windows 可执行文件(EXE 和 DLL)中的各种相关信息，并能够以二进制的方式 dump 文件内容。下面分项列出 PDUMP 的各种使用情况。

用 PDUMP 分析 DOS 文件

(1) 用 PDUMP 观察一个 text 文件 — 等价于 DOS 命令"TYPE"

```
C:\BIN>pdump mouse.ini
Power Dump Version 1.0 Copyright (c) 1993 KingSoft Ltd. Mr. Leijun
                Display of File MOUSE.INI

* * * * * * mouse.ini * * * * * *

[mouse]
MouseType = Serial1

Total lines is 2
```

(2) 用 PDUMP 观察一个.COM 文件 — 等价于 PCTOOLS 的"View"(16 进制方式)

RamInit(RI.COM)是一个 DOS 环境下的内存清理工具,它被执行时就驻留内存(一般把它放在 AUTOEXEC.BAT 的最后一行),并接管键盘中段,如果按热键激活它,则凡是在 RI 驻留以后进入内存的内容全部被清除。这个程序在某些死机的场合,特别是键盘死锁的情况下显得特别有用。我们用 PDUMP 来观察它:

```
C:\BIN>pdump ri.com
Power Dump Version 1.0 Copyright (c) 1993 KingSoft Ltd. Mr. Leijun
                    Display of File RI.COM

0000: E9 95 06 54 0D 56 32 2E 31 32 20 30 34 2F 31 32  ...T.V2.1204/12
... some contents omitted ...
0630: CD 10 EB F1 2E A0 3F 07 2E A2 40 07 C3 00 00 17  ......?...@.....
0640: 17 20 52 41 4D 69 6E 69 74 20 20 56 65 72 73 69  .  RAMinit  Versi
0650: 6F 6E 20 32 2E 31 32 20 20 28 63 29 20 31 39 38  on 2.12 (c) 198
0660: 39 2D 31 39 39 34 20 62 79 20 4B 69 6E 67 53 6F  9-1994 by KingSo
0670: 66 74 20 4C 74 64 2E 20 20 4D 72 2E 20 4C 65 69  ft Ltd. Mr. Lei
0680: 6A 75 6E 0D 0A 20 5B 2A 5D 20 41 63 74 69 76 61  jun.. [*] Activa
0690: 74 65 2E 2E 2E 0A 0D 24 E9 45 05 52 41 4D 69 6E  te.....$.E.RAMin
06A0: 69 74 20 20 56 65 72 73 69 6F 6E 20 32 2E 31 32  it Version 2.12
06B0: 20 43 6F 70 79 72 69 67 68 74 20 28 63 29 20 31   Copyright (c) 1
06C0: 39 38 39 2D 31 39 39 34 20 20 62 79 20 4B 69 6E  989-1994  by Kin
06D0: 67 53 6F 66 74 20 4C 74 64 2E 20 0D 0A 24 41 6C  gSoft Ltd... $ Al
06E0: 72 65 61 64 79 20 69 6E 73 74 61 6C 6C 65 64 20  ready installed
06F0: 21 0D 0A 0A 46 6F 72 20 48 65 6C 70 2C 20 74 79  !...For Help, ty
0700: 70 65 20 22 52 49 20 2F 3F 22 2E 20 0D 0A 24 0A  pe"RI /?"... $.
0710: 52 65 73 69 64 65 6E 74 73 20 61 20 6E 65 77 20  Residents a new
0720: 52 41 4D 69 6E 69 74 20 63 6F 70 79 20 5B 79 2F  RAMinit copy [y/
... other content omitted ...
```

(3)用 PDUMP 观察一个 DOS EXE 文件 — 显示 MZ 文件头信息,包括重定位表虽然 PDUM 可以观察 Windows 可执行文件,但它本身只是 DOS 下的应用程序。下面用 PDUMP 来观察 pdump.exe。

```
C:\BIN>pdump pdump.exe
Power Dump Version 1.0 Copyright (c) 1993 KingSoft Ltd. Mr. Leijun
                Display of File PDUMP.EXE

DOS File Size                                   553Ah  ( 21818. )
Load Image Size                                 54CAh  ( 21706. )
Relocation Table entry count                    0001h  (     1. )
Relocation Table address                        0052h  (    82. )
Size of header record          (in paragraphs)  0007h  (     7. )
Minimum Memory Requirement     (in paragraphs)  08DAh  ( 2266. )
Maximum Memory Requirement     (in paragraphs)  FFFFh  ( 65535. )
File load checksum                              0000h  (     0. )
Overlay Number                                  0000h  (     0. )

Initial Stack Segment  (SS:SP)                  0554:0200
Program Entry Point    (CS:IP)                  FFF0:0100

Relocation Locations (1 Entry)
    0000:0007
```

用 PDUMP 观察一个 Windows 可执行文件

PDUMP 的主要功能是分析 Windows 可执行文件中的信息。PDUMP 的用法如下：

```
Power Dump Version 1.0 Copyright (c) 1993 KingSoft Ltd. Mr. Leijun
Usage: PDUMP <ExeFile> [/P] [/B] [/EXPORTS]
```

PDUMP 与 TDUMP 功能一致，但是加/P 参数后，修改了 TDUMP 中存在的不足，功能强大得多。PDUMP 不仅能够显示程序调用的系统函数名，而且能够显示调用的用户 DLL 库的函数名(如果在函数名前面有“*”，则说明这个函数是用户 DLL)。PDUMP 还能够显示重定位的链表。当然，PDUMP 能够分析 PACKWIN 压缩过的程序。

1. 用 pdump /exports 观察一个 Windows 可执行文件 — 只列出输出的函数名

```
C:\BIN>pdump /exports \windows\pbrush.exe

Power Dump Version 1.0 Copyright (c) 1993 KingSoft Ltd. Mr. Leijun
            Display of File \WINDOWS\PBRUSH.EXE

;
; \windows\pbrush.exe
; "PC Paintbrush + "
;
PbrushX . 1         PARENTWP
PbrushX . 2         PAINTWP
PbrushX . 3         TOOLWP
PbrushX . 4         SIZEWP
PbrushX . 5         COLORWP
PbrushX . 7         CLEARDLG
PbrushX . 9         INFODLG
PbrushX . 10        MOUSEDLG
PbrushX . 11        BRUSHDLG
PbrushX . 12        COLORDLG
PbrushX . 14        ABORTDLG
PbrushX . 15        ABORTPRT
PbrushX . 16        FULLWP
PbrushX . 17        ZOOMOTWP
PbrushX . 18        PRINTFILEDLG
PbrushX . 24        PAGESETDLG
PbrushX . 25        TILTBLT
PbrushX . 26        NULLWP
PbrushX . 27        PALWNDPROC
PbrushX . 28        ENUMPENS
PbrushX . 30        SRVROPEN
PbrushX . 31        SRVRCREATE
PbrushX . 32        SRVRCREATEFROMTEMPLATE
PbrushX . 33        SRVREDIT
PbrushX . 34        SRVREXIT
PbrushX . 35        SRVRRELEASE
PbrushX . 40        DOCSAVE
PbrushX . 41        DOCCLOSE
PbrushX . 42        DOCSETHOSTNAMES
PbrushX . 43        DOCSETDOCDIMENSIONS
PbrushX . 44        DOCGETOBJECT
```

```
PbrushX . 45        DOCRELEASE
PbrushX . 46        DOCSETCOLORSCHEME
PbrushX . 50        ITEMQUERYPROTOCOL
PbrushX . 51        ITEMDELETE
PbrushX . 52        ITEMSHOW
PbrushX . 53        ITEMGETDATA
PbrushX . 54        ITEMSETDATA
PbrushX . 55        ITEMSETTARGETDEVICE
PbrushX . 56        ITEMSETBOUNDS
PbrushX . 57        ITEMENUMFORMATS
PbrushX . 58        ITEMSETCOLORSCHEME
PbrushX . 59        ITEMDOVERB
PbrushX . 60        FILEDIALOG
PbrushX . 61        OBJECTUPDATEDLGPROC
```

我们发现,此时 PDUMP 给出的是该程序所"Exported Entry Points"的入口索引和其在非驻留名表中对应的函数名。值得注意的是,它是按入口索引的大小顺序来组织的,从 1 到 61(中间不连续的序号为空的入口点)。前面讲述的很多工具也能分析对"Exported Entry Points"作特别的分析。

2. 用 pdump /b 观察一个 Windows 可执行文件 — 强迫 PDUMP 以二进制方式 dump

一般是和 GetSeg 联合使用。如果要具体分析一个代码段的内容,我们可以先用 GetSeg 取出该段的内容(GetSeg 的用法已在第一章中说明),然后用 pdump /p 来作进一步分析(不要忘了 GetSeg 为所取出的内容加上了一个 16 个字节的头)。比如:

```
C:\BIN>getseg \windows\pbrush.exe 1 \pb_seg1.bin
C:\BIN>pdump /b \pb_seg1.bin
```

实际输出内容省略。

3. 用 pdump /p 观察一个 Windows 可执行文件 — 最完整的 dump 形式

一般情况下,我们使用 PDUMP 侦查 Windows 可执行文件时是不加命令行参数 "/p" 的。加上这一选项后,PDUMP 能够给出程序中所调用函数的函数名。我们知道,重定位的类型可以是"输入序号",这个时候函数名不在文件中。MAPWIN 等工具所以能够给出函数名,是因为它内建有这种对应关系的表格。与其它工具的不同之处是,PDUMP 是通过自动寻找 Windows 系统所在的目录来建立这种对应关系的。下面是用 PDUMP /P 分析 PBRUSH.EXE 所得的结果。

```
C:\BIN>pdump /p \windows\pbrush.exe

Power Dump Version 1.0 Copyright (c) 1993 KingSoft Ltd. Mr. Leijun
              Display of File \WINDOWS\PBRUSH.EXE

Old Executable Header

DOS File Size                                  2D030h  (184368.  )
Load Image Size                                 C4Eh  (  3150.  )
Relocation Table entry count                   0000h  (     0.  )
Relocation Table address                       0040h  (    64.  )
Size of header record          (in paragraphs) 0020h  (    32.  )
Minimum Memory Requirement     (in paragraphs) 0000h  (     0.  )
Maximum Memory Requirement     (in paragraphs) FFFFh  ( 65535.  )
File load checksum                             4065h  ( 16485.  )
```

```
Overlay Number                                  0000h (     0. )

Initial Stack Segment   (SS:SP)                 0007:0100
Program Entry Point     (CS:IP)                 0000:0000

New Executable header
Operating system                                Windows
File Load CRC                                   084823BE5h
Program Entry Point     (CS:IP)                 0001:0503
Initial Stack Pointer   (SS:SP)                 0022:0000
Auto Data Segment Index                         0022h (    34. )
Initial Local Heap Size                         1000h (  4096. )
Initial Stack Size                              2000h (  8192. )
Segment count                                   0022h (    34. )
Module reference count                          0009h (     9. )
Moveable Entry Point Count                      00F6h (   246. )
File alignment unit size                        0010h (    16. )
DOS File Size                                   2D030h (184368. )
Linker Version                                  5.20
Program Flags
    DGROUP              : multiple (unshared)
    Global initializaton : No
    Protected mode only : No
    Application type    : Uses windowing API
    Self Loading        : No
    Errors in image     : No
    Module type         : Application
Other EXE Flags
    2.X protected mode  : No
    2.X proportional font : No
    Gangload area       : Yes
Start of Gangload Area                                  1160h
Length of Gangload Area                                 145E0h
Mininum code swap area size                             0
Expected Windows Version                                3.10
Segment Table                   Offset:  440h  Length:  0110h
Resource Table                  Offset:  550h  Length:  02FEh
Resident Names Table            Offset:  84Eh  Length:  000Bh
Module Reference Table          Offset:  859h  Length:  0012h
Imported Names Table            Offset:  86Bh  Length:  008Dh
Entry Table                     Offset:  8F8h  Length:  05E3h
Nonresident Names Table         Offset:  EDBh  Length:  0271h

Segment Table       offset:     440h

    Segment Number:  01h
    Segment Type:    CODE       Alloc Size : 0D64h
    Sector Offset:   0118h      File length: 0D64h
    Attributes: Moveable Preloaded Relocations Discardable

    ... some "Segment Table" omitted ...

    Segment Number: 21h
    Segment Type:    CODE       Alloc Size : 1CE1h
    Sector Offset:   2555h      File length: 1CE0h
    Attributes: Moveable Loadoncall Relocations Discardable

    Segment Number:  22h
```

```
    Segment Type:    DATA       Alloc Size : 4FF8h
    Sector Offset:   102Eh      File length: 4FF8h
    Attributes: Moveable Preloaded Relocations

Resource Table      offset:      550h

    Sector size: 0010h

    type: Group Cursor
          Identifier: FLOOD
              offset: 274C0h  length:    0020h
          Attributes: Moveable Shareable

          ... some "Group Cursor"s omitted ...

          Identifier: TEXT
              offset: 27580h  length:    0020h
          Attributes: Moveable Shareable
    type: Group Icon
          Identifier: PBRUSH
              offset: 275A0h  length:    0030h
          Attributes: Moveable Shareable

    type: Bitmap
          Identifier: PTOOLBOX
              offset: 275D0h  length:    2350h
          Attributes: Moveable Shareable

          Identifier: PBWTOOLBOX
              offset: 29920h  length:    08E0h
          Attributes: Moveable Shareable

          Identifier: PARROW
              offset: 2A200h  length:    00D0h
          Attributes: Moveable Shareable

    type: Menu
          Identifier: PBRUSH2
              offset: 15360h  length:    03B0h
          Attributes: Moveable Shareable      Preload

    type: Dialog
          Identifier: 2
              offset: 2A2D0h  length:    0160h
          Attributes: Moveable Shareable

          ... Other "resources" omitted ...

Resident Name Table             offset:    84Eh
    Module Name: 'PbrushX'

Module Reference Table          offset:    859h
    Module  1: PBRUSH
    Module  2: KERNEL
    Module  3: GDI
    Module  4: USER
    Module  5: KEYBOARD
    Module  6: OLESVR
    Module  7: COMMDLG
    Module  8: WINNLS
    Module  9: SHELL

Imported Names Table            offset:    86Bh
```

```
    name                offset
    PBRUSH              0001h
    VCREATEBITMAP       0008h
    GETVCACHEDC         0016h
    VPATBLT             0022h
    VBITBLT             002Ah
    VSTRETCHBLT         0032h
    VDELETEOBJECT       003Eh
    DISCARDBAND         004Ch
    KERNEL              0058h
    GDI                 005Fh
    USER                0063h
    KEYBOARD            0068h
    OLESVR              0071h
    COMMDLG             0078h
    WINNLS              0080h
    SHELL               0087h
Entry Table          offset: 8F8h = = > all "exported" entry must
  Moveable Segment Records ( 5 Entries) have crossponding entry in
    Entry    1:  Segment: 03h  Offset: 0CCCh    Exported    "nonresident..."
    Entry    2:  Segment: 03h  Offset: 05A2h    Exported
    Entry    3:  Segment: 1Fh  Offset: 1334h    Exported
    Entry    4:  Segment: 1Dh  Offset: 01E1h    Exported
    Entry    5:  Segment: 18h  Offset: 0137h    Exported
  Null Entry: 1

  Moveable Segment Records    (  1 Entry)
    Entry    7:  Segment: 08h  Offset: 0000h    Exported
  Null Entry: 1

... part of "Entry Table" omitted ...

  Moveable Segment Records    (213 Entries)
    Entry   50: Segment: 16h  Offset: 0E3Fh    Exported
    Entry   51: Segment: 16h  Offset: 0A0Ah    Exported
    Entry   52: Segment: 16h  Offset: 0BC2h    Exported
    Entry   53: Segment: 16h  Offset: 0A2Ch    Exported
    Entry   54: Segment: 16h  Offset: 0B4Ch    Exported
    Entry   55: Segment: 16h  Offset: 0DE3h    Exported
    Entry   56: Segment: 16h  Offset: 0DCCh    Exported
    Entry   57: Segment: 16h  Offset: 0E08h    Exported
    Entry   58: Segment: 16h  Offset: 0E71h    Exported
    Entry   59: Segment: 16h  Offset: 0E88h    Exported
    Entry   60: Segment: 0Fh  Offset: 09A2h    Exported
    Entry   61: Segment: 13h  Offset: 1BD8h    Exported
    Entry   62: Segment: 1Fh  Offset: 06C4h
    Entry   63: Segment: 1Fh  Offset: 1DDBh

    ... part of "Entry table" omitted ...

    Entry  254:Segment: 01h  Offset: 076Ch
    Entry  255:Segment: 01h  Offset: 0948h
    Entry  256:Segment: 01h  Offset: 0AEAh
    Entry  257:Segment: 03h  Offset: 206Dh
    Entry  258:Segment: 01h  Offset: 084Bh
    Entry  259:Segment: 01h  Offset: 0859h
    Entry  260:Segment: 01h  Offset: 0C38h
```

```
    Entry    261:Segment: 01h  Offset: 0C3Eh
    Entry    262:Segment: 01h  Offset: 0C8Eh

Non-Resident Name Table       offset:  EDBh
    Module Description: 'PC Paintbrush + '
    Name: OBJECTUPDATEDLGPROC          Entry:      61
    Name: DOCSETDOCDIMENSIONS          Entry:      43
    Name: SRVROPEN                     Entry:      30
    Name: PALWNDPROC                   Entry:      27
    Name: FULLWP                       Entry:      16
    Name: ITEMSETBOUNDS                Entry:      56
    Name: NULLWP                       Entry:      26
    Name: PARENTWP                     Entry:      1
    ... part of "Non-Resident name Table" omitted ...
    Name: ENUMPENS                     Entry:      28
    Name: PAGESETDLG                   Entry:      24

Segment Relocation Records

    Segment 0001h relocations

    type    offset        target
    PTR     0BF9h         KERNEL.115        OUTPUTDEBUGSTRING
            Linking...  0BF2h
    PTR     0B9Ah         KERNEL.1          FATALEXIT
    PTR     0341h         GDI.123           PLAYMETAFILE
    PTR     0C63h         KERNEL.5          LOCALALLOC
    PTR     0D07h         KERNEL.6          LOCALREALLOC
    PTR     0C84h         KERNEL.7          LOCALFREE
    PTR     011Fh         GDI.128           MULDIV
            Linking...  013Ah
    PTR     0B91h         KERNEL.137        FATALAPPEXIT
    PTR     006Ch         GDI.3             SETMAPMODE
            Linking...  0294h 030Bh
    PTR     0CACh         KERNEL.15         GLOBALALLOC
    PTR     0D56h         KERNEL.16         GLOBALREALLOC
    PTR     0CC2h         KERNEL.17         GLOBALFREE
    PTR     0030h         KERNEL.18         GLOBALLOCK
            Linking...  02C0h
    PTR     0059h         KERNEL.19         GLOBALUNLOCK
            Linking...  0349h
    PTR     054Dh         USER.5            INITAPP
    PTR     0332h         GDI.14            SETVIEWPORTEXT
    PTR     0C4Ah         KERNEL.23         LOCKSEGMENT
            Linking...  0CEBh
    PTR     0C6Fh         KERNEL.24         UNLOCKSEGMENT
            Linking...  0D13h
    PTR     0544h         KERNEL.30         WAITEVENT
    PTR     0061h         GDI.30            SAVEDC
            Linking...  02D7h
    PTR     029Eh         GDI.39            RESTOREDC
            Linking...  0353h
    PTR     018Bh         0001h:057Ch
            Linking...  0192h 01E5h 01ECh 023Bh 0242h
                        0272h  0279h
    PTR     017Dh         0001h:0616h
            Linking...  0184h 01D7h 01DEh 022Dh 0234h
```

```
                    0264h  026Bh
PTR     02E5h         USER.180          GETSYSCOLOR
PTR     02ECh         GDI.66            CREATESOLIDBRUSH
PTR     02FDh         GDI.69            DELETEOBJECT
PTR     00C5h         GDI.80            GETDEVICECAPS
        Linking...  00D2h 00DFh 00ECh 0116h 0131h
PTR     0507h         KERNEL.91         INITTASK
PTR     02F5h         USER.81           FILLRECT
OFFS    0501h         KERNEL.178        __WINFLAGS
PTR     095Ch         KERNEL.102        DOS3CALL
        Linking...  092Ah 0906h 08D4h 07A9h 0787h
                    0480h 046Ah 0432h 040Eh 03EAh
PTR     0532h         0001h:076Ch
PTR     0537h         0001h:0948h
PTR     053Ch         0001h:0AEAh
PTR     056Ah         0003h:206Dh
PTR     0570h         0001h:084Bh
PTR     009Eh         GDI.99            LPTODP
PTR     0578h         0001h:0859h
PTR     0C1Ah         0001h:0C38h
PTR     0CDDh         0001h:0C3Eh
        Linking...  0C39h
PTR     0D33h         0001h:0C8Eh
Relocations: 41
```

和 TDUMP 最为接近的 Power 系列分析工具从不同的角度入手，解剖了 NE 文件的方方面面。我们来细致地看一下 PDUMP 所显示的文件信息，如下所示：

☐ Old Executable Header：DOS 文件头。

☐ Operating System：目标操作系统，为 Windows 或 OS/2。如果是 Windows 执行文件，则还会显示 Windows 的版本号。这个版本号是由 Windows Linker 产生的。

☐ Program Entry Point (CS:IP)：Windows AP 或 DLL 的进入点。这个地址的格式并不是指向 WinMain()或 LibMain()，而是指向真正的程序起始码，这段代码通常是由连接器加上的，KERNEL 执行这段代码后才去调用 WinMain()或 LibMain()。有关程序启动过程的问题，请参阅第四章。

☐ Auto Data Segment Index：程序中预定义数据段的段号。Windows 中的段号从 1 开始。有关自动数据段的描述，请参阅第 4 章。

☐ DOS File Size：实际上就是这个可执行文件的真实大小。

☐ Program Flags：DGROUP 可以为 multiple，也可以为 single。Self Loading 用于确定程序是否包含自己的装载器。有些 Windows 程序必须自己装载自己，有关内容请参阅第 6 章。Module type 可以为 Application(AP)或 Library(DLL)。

☐ Gangload area：程序中指定的快速装载区。为了提高程序的装载速度，为 Windows 设计的 Compiler 会根据需要设定一个快速装载区。在一般的情况下，一个程序中所有需要预装载的内容都会连续地存放在文件的前面，包含这一部分内容的区域被设为快速装载区。Start of Gangload Area 和 Length of Gangload Area 指定了这一块区域的位置。

☐ Segment Table：

```
Segment Number: 01h
```

```
Segment Type:    CODE       Alloc Size : 0D64h
Sector Offset:   0118h      File length: 0D64h
Attributes: Moveable Preloaded Relocations Discardable
```

Segment Table 包含可执行文件中每个段的信息。PDUMP 列出了每个段的逻辑段号 LSN(Logical Segement Number)(10 进制)、在执行文件中的偏移、长度、类型(CODE 或 DATA)、段的属性(是否可移动、是否可抛弃、是否预装载、是否含有重定位项)。

☐ Resource Table:资源表

```
type: Group Cursor
     Identifier: FLOOD
        offset: 274C0h       length: 0020h
     Attributes: Moveable Shareable
```

指出了资源的类型、资源的标识符、位置和属性。从 PDUMP 所列出的资源表中,我们可以看到,只有 Menu 资源 PBRUSH2 需要预装载。

☐ Resident Name Table:驻留名表。驻留名表的第一项是模块名。因为 PBRUSH 是一个应用程序,所以没有更多的驻留名。

☐ Module Reference Table:表示该应用程序需要使用到哪些模块。在上面的例子中我们看到,PBRUSH 不仅使用到了 USER、KERNEL 和 GDI 模块,还使用了 OLESVR 模块,所以它能够作为 OLE Server 使用。

☐ Imported Names Table:给出该程序所使用到的函数名和模块名。在 Windows 的代码段后的重定位表中,重定位类型可以是函数序号,也可以是函数名。在 PBRUSHX 中使用 USER 等模块时给出的是函数序号,所以不必在 Imported Names Table 中给出其中每个函数的函数名;而使用 PBRUSH.DLL 时给出的是函数名,所以在 Imported Names Table 中给出了该 DLL 中被使用函数的函数名。

☐ Entry Table:该 AP 或 DLL 提供的全部 export(无论是命名或未命名的)。PDUMP 会显示每个 export 所属的 LSN(段号)、在该段中的偏移和序号。如果该 export 的属性为标有"Exported",则在 Non-Resident Table 还可以找到对应的函数名。

☐ Non-Resident Name Table:给出 Entry Table 中所有命名的 export 的名称字符串。

☐ Segment Relocation Records:在 Windows 执行文件中,每个段之后可能会跟着一个 Relocation Table,所谓重定位是指程序在执行前需要"修正"的地方,Relocation Table 就是存放这些数据的表格。对于 PDUMP 来说,加不加命令行参数 /P 的区别就在于如何显示这些重定位数据。上例中是加上该参数时的情况,它显示了具体的函数名。如果不加上这一个参数,PDUMP 就不显示某个序号对应的函数名。其中的 Linking...表示在该段中的其它某个(些)地方要调用相同的函数。

4. 如何利用 PDUMP 反汇编程序

有时我们需要反汇编某个程序的一个段,方法是这样的:

Step 1. 用 PDUMP 分析这个段的信息。

如:PDUMP SAMPLE.EXE /Sn (其中 n 表示段号)

Step 2. 用 GETSEG 把这个段取出来。

Step 3. 用 DEBUG 或 Sourcer 把这段数据反汇编出来,再结合 Step 1 中得到的重定位

信息，就可以把这个段完整地反汇编出来。

2.2.2　Power FileInfo(PFI)

PFI 是另外一个比较有特色的 Windows 文件分析工具，它的设计目标是按程序的逻辑地址递增的顺序，逐个列出其中每一个区段的内容；其最明显的特点是简洁明了。

下面是用 PFI 分析 PBRUSH.EXE 的结果。

```
C:\BIN>pfi \windows\pbrush.exe

Power FileInfo Version 1.0 Copyright (c) 1993 Yellow Rose Workgroup

File name: \windows\pbrush.exe
Module name: PbrushX
Description: PC Paintbrush +
Page size: 16     Resource page size: 16

    Position          Length          Owner
    ----------        --------        ----------------
    0000h-03FFh       0400h           WINSTUB
    0400h-114Bh       0D4Ch           New Executable file header
    0400h-043Fh       0040h             NE header record
    0440h-054Fh       0110h             Segment table
    0550h-084Dh       02FEh             Resource table
    084Eh-0858h       000Bh             Resident name table
    0859h-086Ah       0012h             Module reference table
    086Bh-08F7h       008Dh             Imported names table
    08F8h-0EDAh       05E3h             Entry table
    0EDBh-114Bh       0271h             Non-resident names table
                      0034h                       ;多余的填充数据
F   1180h-1EE3h       0D64h           seg1 code movable preload
F   1EE4h-202Dh       014Ah           seg1 Relocation: 41 entries
                      0012h
F   2040h-2E68h       0E29h           seg2 code movable preload
F   2E69h-30D2h       026Ah           seg2 Relocation: 77 entries
                      002Dh
F   3100h-5A89h       298Ah           seg3 code movable preload
F   5A8Ah-5E6Bh       03E2h           seg3 Relocation: 124 entries
                      0014h
F   5E80h-7669h       17EAh           seg4 code movable preload
F   766Ah-788Bh       0222h           seg4 Relocation: 68 entries
                      0014h
F   78A0h-8260h       09C1h           seg6 code movable preload
F   8261h-834Ah       00EAh           seg6 Relocation: 29 entries
                      0015h
F   8360h-90B0h       0D51h           seg15 code movable preload
F   90B1h-927Ah       01CAh           seg15 Relocation: 57 entries
                      0025h
F   92A0h-A6F9h       145Ah           seg17 code movable preload
F   A6FAh-A90Bh       0212h           seg17 Relocation: 66 entries
                      0014h
F   A920h-C587h       1C68h           seg19 code movable preload
F   C588h-C8D1h       034Ah           seg19 Relocation: 105 entries
                      002Eh
F   C900h-CD95h       0496h           seg21 code movable preload
```

```
F   CD96h-CF77h         01E2h          seg21 Relocation: 60 entries
                        0028h
F   CFA0h-D6DDh         073Eh          seg24 code movable preload
F   D6DEh-D79Fh         00C2h          seg24 Relocation: 24 entries
                        0020h
F   D7C0h-DAADh         02EEh          seg29 code movable preload
F   DAAEh-DB7Fh         00D2h          seg29 Relocation: 26 entries
                        0020h
F   DBA0h-FF3Fh         23A0h          seg31 code movable preload
F   FF40h-102B1h         0372h          seg31 Relocation: 110 entries
                        002Eh
F   102E0h-152D7h       4FF8h          seg34 data movable preload
F   152D8h-15341h       006Ah          seg34 Relocation: 13 entries
                        001Eh
F   15360h-15700h       03A1h          Menu.PBRUSH2 moveable Shareable Preload
                        003Fh
    15740h-159CDh       028Eh          seg5 code movable
    159CEh-15A77h       00AAh          seg5 Relocation: 21 entries
    15A80h-15D53h       02D4h          seg7 code movable
    15D54h-15DD5h       0082h          seg7 Relocation: 16 entries
    ...
    25550h-2722Fh       1CE0h          seg33 code movable
    27230h-274B1h       0282h          seg33 Relocation: 80 entries
    274C0h-274D5h       0016h          GroupCursor.FLOOD moveable Shareable
    ...
    2CEF0h-2D023h       0134h          Cursor.9 moveable
File size is 184368  [ Stub: 0.5%  Code: 66.0%  Data: 11.1%  Resource: 12.9% ]
; 文件大小              STUB 大小     代码大小      数据大小      资源大小
```

因为在这一章中，PBRUSH.EXE 已经被分析好多次了，所以在此就不再作过多的讲述。有两点读者可能已经注意到了：一是如果该行前面有字母“F”，表示该代码段或资源处于快速装载区；二是所有应该快速装载的内容都连续存放，而并不是按逻辑段号的顺序排列。

如果被分析的文件用 PACKWIN 压缩过，PFI 会在压缩段前显示一个“*”。用 PFI 也可以分析 DOS 执行文件的段信息。读者可以自己试一下。

我们开发 PFI 的目的是用来辅助开发 PACKWIN(见第 8 章)。PFI 能够检查出一个 Windows 执行文件中隐含的结构错误。如第 j 段的起始地址在第 i 段中，PFI 就会鸣笛报错。PFI 还会核查重定位、资源等信息。

PFI 的另外一个应用是帮助修改 Windows 执行文件。比如插入一个代码段后，可以用 PFI 分析后把结果存放在文件中，然后用比较工具对照，可以有效地检查修改的结果是否正确。

第 3 章

文件格式分析工具的开发实例

在第 2 章中，我们介绍了 Power 系列分析工具的功能和用法。但是，如果读者只是从使用的角度了解了这些工具的功能，还谈不上熟练驾驭结构复杂的 Windows 可执行文件。

在实际应用过程中，只有在透彻地了解 Windows EXE 结构规则的基础上，才能达到操纵 Windows 可执行文件的目的。比如，在一个指定段后面附加一段代码，修改段中的一部分代码，添加或删除一个重定位项，插入新的入口点或段表信息，压缩或变形（加密）Windows 可执行文件，编写被压缩或被加密程序的自装载过程，甚至在 Windows 系统已加载的情况下，在内存中动态修改或重定向 Windows 的系统功能调用。这些操作都需要对 Windows EXE 中的执行信息和 Windows 的执行机制有透彻的理解。本书中将要讨论的 Windows 压缩工具 PACKWIN 和加密工具 BITLOK for Windows 就是通过对 Windows EXE 中的执行信息进行各种复杂处理来完成的，另外，著名的 Windows 中文平台产品中文之星和金山皓月也是通过重定向 Windows 系统的输入输出功能调用来构筑中文支持环境的。

所以，在本章里我们引入一个完整的文件对象 File Object，它定义了 DOS 及 Windows 等操作系统中主要文件类型的结构数据和相应的操作，其中包括结构最复杂的 NE 格式文件。File Object 中较全面定义了针对 NE 格式的各种操作，其中运用的处理手段和技巧具有很高的通用性，在与 Windows EXE 操作有关的程序中可方便地使用。作为 File Object 的应用，我们还开发了一个类似 Microsoft EXEHDR 的分析工具——MSDUMP，读者通过分析这一个示范程序可以初步了解 File Object 中定义的操作，这是继续阅读后续章节的基础。在实现 File Object 时有大量针对文件的各种操作并经常用到一些重要的 Pascal 子程序，为了简化使用，我们将它们提取出来，独立成为一个文件操作扩展工具 EXTTOOLS — 这一方面的内容构成了本章的第一节。

3.1 一个 DOS 文件操作功能的扩展工具——EXTTOOLS

在第一章的示范程序中，我们直接引用了一些较高级的文件操作功能，如 FileExists 和 CopyFileBlock 等等。这些功能并不是由 DOS 或 Turbo Pascal 编译器直接提供的，而是笔者自行定义的。我们认为，在编程实践中，有意识地将一些经常需要而又不能直接从操作系统或语言函数库提供的函数中调用的功能分别予以实现，然后分类整理，制作成一个小工具库，并不断改进使其更加通用。这是一种良好的编程习惯，可以极大地提高编程效率和程序的稳定性，简化程序维护。

在这里，我们给出了一个 DOS 文件操作功能的扩展工具 EXTTOOLS，它包括字符串操作和转换、布尔变量和与其相应的字符串表示、数学移位计算、较大较小值运算、起始时间和终止时间的获取、绝对磁盘访问、判断文件是否存在、复制文件块、申请缓冲区和释放缓冲区等等。这一个扩展工具中的函数在前面章节的示范程序中已多次出现，在本书后面的章节中会被更加频繁地调用，它屏蔽了许多琐碎的操作，使示范程序的主要意图清晰简明。读者应该反复阅读这个程序直到熟练掌握其中各个函数功能。

```
{ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * }
{ *     EXTTOOLS.TPU - - Extended subprocedure library                          * }
{ *     Copyright (c) 1993 by Yellow Rose workgroup                             * }
{ *     Copyright (c) 1993 by KingSoft Beijing Co.                              * }
{ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * }

unit ExtTools;

interface

uses DOS;

type
  PStr       = ^String;
  ByteArray  = Array [0.. $ []ff00] of byte;
  PBytes     = ^ByteArray;
  WordArray  = Array [0.. $ 7f00] of word;
  PWords     = ^WordArray;
  DWordArray = Array [0.. $ 3f00] of LongInt;
  PDWords    = ^DWordArray;
  PByte      = ^Byte;
  PWord      = ^Word;
  PDWord     = ^LongInt;

  String10   = String[10];
  String20   = String[20];
  String40   = String[40];
  String80   = String[80];

{ 在 Intel 8086 体系中,双字的存放顺序是低字节在前,高字节在后 }
  DWord = Record
    LoWord, HiWord: Word
  end;

const
{ fPrompt:提示标志。若 fPrompt 为 TRUE,表明程序中的信息都可以显示出来 }
```

```
  fPrompt: Boolean = True;

{ 定义命令行选项中前导字符 }
{ 如:PDUMP SAMPLE.EXE /B  或 PDUMP SAMPLE.EXE -B }
  OptionChars = ['-','/'];

  ShowChars = [# $ 20..# $ 7E];
  TextChars = [#9, #10, #13] + ShowChars;

{ ------ NEW ------------------------------------------------------------ }
{ 取对数。一般用于计算文件的大小 }
function Log2 (I: word): Word;
{ 按页面大小将以长整数给定的大小对齐,返回对齐后的大小 }
function AlignSize (InSize: LongInt; PageSize: Word): LongInt;
{ 显示两个大小之间的比值 }
procedure WriteRatio(SourceSize, NewSize: LongInt);
{ 删除文件 }
function DelFile (InFName: PathStr): Boolean;
{ 取文件大小 }
function GetFileSize (InFName: PathStr): LongInt;
{ 更换一个文件的名字 }
function RenameFile (InFName, OutFName: PathStr): Boolean;

{ ------ CRT ------------------------------------------------------------ }
{ 从键盘上读入一个字符 }
{ Turbo Pascal 的 CRT 库中有该函数。因为 CRT 库中有很多函数, }
{ 如果应用程序只使用该函数,就不必使用 CRT 库了 }
function ReadKey: Char;

{ ------ String ---------------------------------------------------------- }
{ 各种数值转换到字符串的操作函数 }
{ Turbo Pascal 的数值转换和字符串显示功能远不如 C 语言方便、灵活, }
{ 这里给出一组经常使用的函数 }
{ 字节型数转换到 16 进制字符串 }
function bhStr(I: Byte): string10;
{ 字型数转换到 16 进制字符串 }
function whStr(I: Word): string10;
{ 长整型数转换到 16 进制字符串 }
function lhStr(I: LongInt): string10;
{ 字节型数转换到 16 进制字符串,去掉前导 0 }
function bhNzStr(I: byte): string10;
{ 字型数转换到 16 进制字符串,去掉前导 0 }
function whNzStr(I: Word): string10;
{ 长整型数转换到 16 进制字符串,去掉前导 0 }
function lhNzStr(I: LongInt): string10;
```

```
{ 字节型数转换到 2 进制字符串 }
function bbStr(I: Byte): String10;
{ 字节型数转换到 16 进制字符串 }
function HexByte(b: Byte): String10;
{ 字型数转换到 16 进制字符串 }
function HexWord(w: Word): String10;
{ 长整型数转换到 16 进制字符串 }
function HexLongInt(l: LongInt): String10;
{ 将一个字符串的前四个字符转换为 10 进制字型整数。比如:
S 为"2049624TELEPHONE"则返回值 2049;
S 为"BP2561155-1997LEIJUN"则返回值 25;
S 为"FAX:2049621KINGSOFT"则返回 0 }
function MyVal (S: String10): Word;
{ 整型数转换到 10 进制字符串 }
function strNum(I: word): string10;
{ 长整型数转换到 16 进制字符串,如果高字为 0,则只转换低字部分 }
function lwhStr(I: LongInt): string10;
{ PChar 型字符串转换到 10 进制字型整数 }
function wPChar(S: PChar): word;
{ PChar 型字符串转换到 10 进制长整数 }
function lPChar(S: PChar): longint;
{ 显示一个 16 进制字型整数 }
procedure WriteHexWord(w : Word);
{ 合并版本号:V1 为主版本号,V2 为次版本号 }
function VerStr(V1, V2: Byte): String20;
{ 字符转换为小写字符 }
function DownCase(Ch: Char): Char;
{ 字符串转换为大写字符 }
function UpcaseStr(Str: String): String;
{ 字符串转换为小写 }
function DowncaseStr(Str: String): String;
{ 去掉字符串前导"0" }
function DelLeadingZero(S: String10): String10;
{ 在行中间位置显示字符串,行宽为 L }
procedure WriteCenterStr (L: Byte; S: String);
{ ---- BooleanStr ---------------------------------------------------------- }
{ 将一个 Boolean 变量的值以字符串"Yes"或"No "表示 }
function BooleanStr (B: Boolean): String10;
{ 判断一个 Word 的值。如果其值为 0,返回"No ";否则返回"Yes" }
function wBooleanStr (W: Word): String10;
{ ------ Math ---------------------------------------------------------------- }
```

```
{ 移位函数 }
function ROR( N: word; Count: Byte): word;
function ROL( N: word; Count: Byte): word;
function bROR( N: byte; Count: Byte): byte;
function bROL( N: byte; Count: Byte): byte;

{ 较大值较小值函数(2 个数比较) }
function wMin(N1, N2: word): word;
function wMax(N1, N2: word): word;
function lMin(N1, N2: LongInt): LongInt;
function lMax(N1, N2: LongInt): LongInt;

{ ------ Clock ------------------------------------------------------------------------ }
{ 取起始时间和终止时间,用以测试某个操作的执行时间 }
procedure StartTimer;
function EndTimer: String;

{ ------ Abosulte Disk Access ---------------------------------------------------------}
{ 绝对磁盘访问 }

{ 从指定驱动器中将指定扇区开始的若干扇区读到指定缓冲区中 }
function AbsDiskRead(Drive:Byte;StartSec: LongInt;
  Secs:Word;var Buf): boolean;
{ 将指定缓冲区中的内容写到指定驱动器中从某个扇区开始的一段扇区中 }
function AbsDiskWrite(Drive:Byte;StartSec: LongInt;
  Secs:Word;var Buf): boolean;

{ ------ File access ------------------------------------------------------------------ }
{ 文件访问 }

{ 判断文件是否存在 }
function FileExists(FileName: PathStr): Boolean;
{ 判断可执行文件是否存在 }
function RunFileExists(FileName: PathStr): boolean;
{ 将源文件中给定长度的内容复制到目标文件中 }
function CopyFileBlock(var sf, tf: File; Len: LongInt): boolean;
{ 将源文件中从某一位置开始的一定长度的内容复制到目标文件的某一位置 }
function CopyFileBlockEx(var sf: File; sPos: LongInt;
  var tf: File; tPos: LongInt;
  Len: LongInt): boolean;
{ 复制整个文件 }
procedure CopyFile(Str1, Str2: PathStr);
{ 将一个文件按给定的页面大小对齐 }
procedure AlignFile(var f: File; PageSize: Word);
{ 文件定位。如果文件长度小于文件指针所指的长度,
则加大文件长度使之等于文件指针所指的长度 }
procedure WriteSeek (var F: File; Position: LongInt);
```

```
{ ------------------------------------------------------------------------ }
{ 定义缓冲区的数据结构。
数据结构有两项:缓冲区长度和指向缓冲区当前位置的指针。
Pascal 的缓冲区指针不包含缓冲区的大小,定义 PBuf 这样的数据类型提高了程序的封装性 }
type
  PBuf = ^TBuf;
  TBuf = record
    Len: Word;
    P: PBytes;
  end;

{ 申请缓冲区和释放缓冲区 }
procedure NewPBuf (var P: PBuf);
procedure NewPBufEx (var P: PBuf; Length: LongInt; T: Pointer);
procedure ResizePBuf (var P: PBuf; NewLen: word);
procedure DisposePBuf (P: PBuf);

implementation

{ ------ String ---------------------------------------------------------- }
{ 字符串操作中用到的常量串 }
const
  hexChars : array [0.. $F] of Char = '0123456789ABCDEF';

procedure WriteCenterStr (L: byte; S: String);
begin
  Writeln('': ((L - Length(S)) div 2), S);
end;

function wPChar(S: PChar): word;
begin
  wPChar := PWord(@S^)^;
end;

function lPChar(S: PChar): longint;
begin
  lPChar := PDWord(@S^)^;
end;

function BooleanStr (B: Boolean): String10;
begin
  if B then BooleanStr := 'Yes' else BooleanStr := 'No ';
end;
```

```
function wBooleanStr (W: Word): String10;
begin
  if W <> 0 then wBooleanStr := 'Yes' else wBooleanStr := 'No ';
end;
function MyVal (S: String10): Word;
var
  I, J: Word;
begin
  I := 1;
  J := 0;
  While ((I <= Length(S)) and (I < 5) and
    (S[I] in ['0'..'9'])) do begin
    J := J * 10 + Ord(S[I]) - Ord('0');
    Inc(I);
  end;
  MyVal := J;
end;

procedure WriteHexWord(w : Word);
begin
  Write(hexChars[Hi(w) shr 4],
        hexChars[Hi(w) and $F],
        hexChars[Lo(w) shr 4],
        hexChars[Lo(w) and $F]);
end;

function HexWord(w : Word): string10;
begin
  HexWord := hexChars[Hi(w) shr 4]  +
             hexChars[Hi(w) and $F] +
             hexChars[Lo(w) shr 4]  +
             hexChars[Lo(w) and $F];
end;

function HexByte(b: Byte): string10;
begin
  HexByte := hexChars[b shr 4] +
             hexChars[b and $F];
end;

function HexLongInt(l: LongInt): string10;
begin
  HexLongInt := HexWord(dword(l).HiWord) + HexWord(dword(l).LoWord);
```

```
end;

function bhStr(I: byte): string10;
begin
  bhStr := hexChars[I shr 4] + hexChars[I and $f];
end;

function whStr(I: Word): string10;
begin
  whStr := bhStr(Hi(I)) + bhStr(Lo(I));
end;

function lhStr(I: LongInt): string10;
begin
  lhStr := whStr(I shr 16) + whStr(I and $ffff);
end;

function DelLeadingZero(S: String10): String10;
var
  I: Byte;
  SS: String10;
begin
  I := 0;
  while (I < Length(S)) and
    (S[I+1] in ['0', ' ', #9]) do Inc(I);
  if I > 0 then
    SS := copy(S, I+1, Length(S) - I)
  else SS := S;
  if SS = '' then SS := '0';
  DelLeadingZero := SS;
end;

function bhNzStr(I: byte): string10;
begin
  bhNzStr := DelLeadingZero(hexChars[I shr 4] + hexChars[I and $f]);
end;

function whNzStr(I: Word): string10;
begin
  whNzStr := DelLeadingZero(bhStr(Hi(I)) + bhStr(Lo(I)));
end;

function lhNzStr(I: LongInt): string10;
```

```
begin
  lhNzStr := DelLeadingZero(whStr(I shr 16) + whStr(I and $ffff));
end;

function bbStr(I: Byte): String10;
var
  j: Byte;
  S: String10;
begin
  S := '';
  for J := 1 to 8 do
  begin
    if (I and $80) = $80
      then S := S + '1'
      else S := S + '0';
    I := I shl 1;
  end;
  bbStr := S;
end;

function lwhStr(I: LongInt): string10;
var
  S: String10;
begin
  S  = lhNzStr(I);
  if Length(S) > 4 then
    lwhStr := S
  else lwhStr := whStr(I);
end;

{ ---- Math ---------------------------------------------------------- }

function wMin(N1, N2: word): word;
begin
  if N1 < N2 then wMin := N1 else wMin := N2;
end;

function wMax(N1, N2: word): word;
begin
  if N1 > N2 then wMax := N1 else wMax := N2;
end;

function lMin(N1, N2: LongInt): LongInt;
```

```
begin
  if N1 < N2 then lMin := N1 else lMin := N2;
end;

function lMax(N1, N2: LongInt): LongInt;
begin
  if N1 > N2 then lMax := N1 else lMax := N2;
end;

function ROR( N: word; Count: Byte): word; assembler;
asm
  mov   ax, N
  mov   cl, Count
  ror   ax, cl
end;

function ROL( N: word; Count: Byte): word; assembler;
asm
  mov   ax, N
  mov   cl, Count
  rol   ax, cl
end;

function bROR(N, Count: Byte): byte; assembler;
asm
  mov   al, N
  mov   cl, Count
  ror   al, cl
end;

function bROL(N, Count: Byte): byte; assembler;
asm
  mov   al, N
  mov   cl, Count
  rol   al, cl
end;

function UpcaseStr(Str: String): String;
var
  I: byte;
begin
  UpcaseStr[0] := Str[0];
  for I := 1 to Length(Str) do
```

```
    UpcaseStr[I] := Upcase(Str[I]);
end;

function DownCase(Ch: Char): Char;
begin
  if (Ch in ['A'..'Z']) then
    DownCase := Chr(Ord(Ch) + $20)
  else DownCase := Ch;
end;

function DownCaseStr(Str: String): String;
var
  I: byte;
begin
  DownCaseStr[0] := Str[0];
  for I := 1 to Length(Str) do
    DownCaseStr[I] := DownCase(Str[I]);
end;

{ ---------- Absolute Disk Access ---------- }

type
  LongRec = record
    Lo, Hi: Word;
  end;

{ 磁盘访问时的错误 }
var
  AbsDiskError: word;
{       -- AH Register Error Codes --
  80h   Attachment failed to respond
  40h   Seek operation failed
  20h   Controller failed
  10h   Data error (bad CRC)
  08h   DMA failure
  04h   Requested sector not found
  03h   Write-protect fault
  02h   Bad address mark
  01h   Bad command
        -- AL Register Error Codes --
  00h   Write-protect error
  01h   Unknown unit
  02h   Drive not ready
```

```
  03h   Unknown command
  04h   Data error (bad CRC)
  05h   Bad request structure length
  06h   Seek error
  07h   Unknown media type
  08h   Sector not found
  09h   Printer out of paper
  0Ah   Write fault
  0Bh   Read fault
  0Ch   General failure
}
  AbsDiskBuffer: array [0..4] of word;
{
  Offset  Length   Comments
  ------  -------  ----------------------------
   00h     dd      Logical sector number, zero-based
   04h     dw      Number of sectors to transfer
   06h     dd      Pointer to data buffer
}

function AbsDiskRead(Drive:Byte;StartSec: LongInt;
  Secs:Word;var Buf): boolean;  assembler;
{ Drive   Drive number (0 = A, 1 = B) }
asm
   MOV  AL, Drive
   MOV  CX, Secs
   MOV  DX, LongRec(StartSec).Lo
   LES  BX, Buf
   CMP  AL, 2              { floppy disk }
   JB   @@2
   PUSH AX
   PUSH BX
   PUSH CX
   MOV  AH, 30H
   INT  21H
   CMP  AX, 1F03H
   JZ   @@1
   CMP  AL, 4
@@1:
   POP  CX
   POP  BX
   POP  AX
   JB   @@2
```

```
    MOV  WORD PTR AbsDiskBuffer[0], DX
    MOV  DX, LongRec(StartSec).Hi
    MOV  WORD PTR AbsDiskBuffer[2], DX
    MOV  WORD PTR AbsDiskBuffer[4], CX
    MOV  WORD PTR AbsDiskBuffer[6], BX
    MOV  WORD PTR AbsDiskBuffer[8], DS
    LEA  BX, AbsDiskBuffer
    MOV  CX, 0FFFFH
@@2:
    INT  25H
    ADD  SP, 2
    MOV  AbsDiskError, AX
    MOV  AX, True
    JNC  @@3
    MOV  AX, False
@@3:
end;

function AbsDiskWrite(Drive:Byte;StartSec: LongInt;
  Secs:Word;var Buf): boolean;  assembler;
asm
    MOV  AL, Drive
    MOV  CX, Secs
    MOV  DX, LongRec(StartSec).Lo
    LES  BX, Buf
    CMP  AL, 2              { floppy disk }
    JB   @@2
    PUSH AX
    PUSH BX
    PUSH CX
    MOV  AH, 30H
    INT  21H
    CMP  AX, 1F03H
    JZ   @@1
    CMP  AL, 4
@@1:
    POP  CX
    POP  BX
    POP  AX
    JB   @@2
    MOV  WORD PTR AbsDiskBuffer[0], DX
    MOV  DX, LongRec(StartSec).Hi
    MOV  WORD PTR AbsDiskBuffer[2], DX
```

```
    MOV   WORD PTR AbsDiskBuffer[4], CX
    MOV   WORD PTR AbsDiskBuffer[6], BX
    MOV   WORD PTR AbsDiskBuffer[8], DS
    LEA   BX, AbsDiskBuffer
    MOV   CX, 0FFFFH
@@2:
    INT   26H
    ADD   SP, 2
    MOV   AbsDiskError, AX
    MOV   AX, True
    JNC   @@3
    MOV   AX, False
@@3:
end;

{ ------------------------------------------------------------------------ }

var
  BegHour, BegMinute, BegSecond, BegSec100: Word;
procedure StartTimer;
begin
  GetTime(BegHour, BegMinute, BegSecond, BegSec100);
end;
function EndTimer: String;
var
  S1, S2: String[20];
  TimeUsed: Longint;
  EndHour, EndMinute, EndSecond, EndSec100: Word;
begin
  GetTime(EndHour, EndMinute, EndSecond, EndSec100);
  TimeUsed := (((EndHour * 60 + EndMinute) * 60 + EndSecond) * 100 + EndSec100)
            - (((BegHour * 60 + BegMinute) * 60 + BegSecond) * 100 + BegSec100);
  Str((TimeUsed div 100):1, S1);
  Str((TimeUsed mod 100):2, S2);
  if S2[1] = ' ' then S2[1] := '0';
  if S2[2] = ' ' then S2[2] := '0';
  EndTimer := S1 + '.' + S2 + 's';
end;
{ -------- CRT ------------------------------------------------------------ }
function ReadKey: Char; assembler;
asm
  mov  ah, 0
```

```
  int   $16
end;
{ ———— File Access ———————————————————————————————————— }
procedure AlignFile(var f: File; PageSize: Word);
var
  I: Word;
  P: PBytes;
begin
  I := PageSize - (FileSize(f) mod PageSize);
  GetMem(P, I);
  FillChar(P^, I, '*');
  BlockWrite(F, P^, I);
  FreeMem(P, I);
end;

function strNum(I: word): string10;
var
  S: String10;
begin
  str(I, S);
  strNum := S;
end;

function VerStr(V1, V2: Byte): String20;
var
  S1, S2: String[3];
begin
  Str(V2:2, S1);
  if S1[1] = ' ' then S1[1] := '0';
  Str(V1, S2);
  VerStr := S2 + '.' + S1;
end;

procedure WriteSeek(var F: File; Position: LongInt);
var
  P: PBytes;
  FS: LongInt;
begin
  FS := FileSize(F);
  if Position <= FS then
    Seek(F, Position)
  else begin
```

```
      GetMem(P, Position-FS);
      FillChar(P^, (Position-FS), #0);
      Seek(F, FS);
      BlockWrite (F, P^, (Position-FS));
      FreeMem(P, Position-FS);
    end;
  end;

{ 判断文件是否存在的操作函数 }
function FileExists(FileName: PathStr): Boolean;
var f: file;
begin
  Assign(f, FileName);
  { $I- }
  Reset(f);
  Close(f); { $I+ }
  FileExists := (IOResult = 0) and (FileName <> '');
end;  { FileExists }

{ 执行文件以 COM、EXE 为括展文件名 }
function RunFileExists(FileName: PathStr): boolean;
begin
  RunFileExists := ((Pos('.COM', FileName) >0 ) or
                    (Pos('.com', FileName) >0 ) or
                    (Pos('.EXE', FileName) >0 ) or
                    (Pos('.exe', FileName) >0 )) and
                    FileExists(FileName);
end;

{ 突破 64K 限制的文件块拷贝。给定源文件、目标文件起始地址 }
function CopyFileBlockEx(var sf: File; sPos: LongInt;
  var tf: File; tPos: LongInt;
  Len: LongInt): boolean;
begin
  Seek(sf, sPos);
  WriteSeek(tf, tPos);
  CopyFileBlockEx := CopyFileBlock(sf, tf, Len);
end;

{ 突破 64K 限制的文件块拷贝。给定源文件、目标文件起始地址 }
function CopyFileBlock(var sf, tf: File; Len: LongInt): boolean;
const
  DefMem = $8000;
```

```
var
  P: PBytes;
  Error: Boolean;
  MemRead, NumRead, NumWritten: Word;
begin
  GetMem(P, DefMem);
  Error := False;
  while (Len <> 0) and (not Error)do begin
    if Len < DefMem then
      MemRead := Len
    else MemRead := DefMem;
    BlockRead(sF, P^, MemRead, NumRead);
    BlockWrite(tF, P^, NumRead, NumWritten);
    if NumRead <> NumWritten then Error := True;
    Dec(Len, MemRead);
  end;
  FreeMem(P, DefMem);
  CopyFileBlock := not Error;
end;

procedure CopyFile(Str1, Str2: PathStr);
var
  FromF, ToF: file;
  NumRead, NumWritten: Word;
begin
  if Str1 <> Str2 then begin
    {$I-}
    Assign(FromF, Str1);
    Reset(FromF, 1);
    Assign(ToF, Str2);
    Rewrite(ToF, 1);
    CopyFileBlock(FromF, ToF, FileSize(FromF));
    Close(FromF);
    Close(ToF);
    {$I+}
    if (IOresult <> 0) then Halt(38);
  end;
end;

{ ---------- New ---------- }
function AlignSize( InSize: LongInt; PageSize: Word): LongInt;
begin
```

```
  AlignSize := (InSize + PageSize-1) div PageSize * PageSize;
end;

procedure NewPBuf(var P: PBuf);
begin
  New(P);
  P^.len := 0;
  P^.P := nil;
end;

procedure NewPBufEx(var P: PBuf; Length: LongInt; T: Pointer);
begin
  New(P);
  Length := lMin(Length, $fff0);
  P^.Len := Length;
  if Length = 0 then
    P^.P := nil
  else begin
    GetMem(P^.P, P^.Len);
    if T <> nil then Move(T^, P^.P^, Length);
  end;
end;

procedure DisposePBuf(P: PBuf);
begin
  if (P <> nil) then begin
    if (P^.P <> nil) and (P^.Len <> 0) then
      FreeMem(P^.P, P^.Len);
    Dispose(P);
  end;
end;

procedure ResizePBuf(var P: PBuf; NewLen: word);
var
  A: PBuf;
begin
  NewPBufEx(A, NewLen, nil);
  Move(P^.P^, A^.P^, wMin(NewLen, P^.Len));
  DisposePBuf(P);
  P := A;
end;

{ ---------- File Operations ---------- }
```

```
function RenameFile(InFName, OutFName: PathStr): Boolean;
var F: File;
begin
  {$I-}
  assign(F, OutFName);
  erase(F);
  assign(F, InFName);
  rename(F, OutFName);
  {$I+}
  RenameFile := (IOResult = 0);
end;

function DelFile(InFName: PathStr): Boolean;
var
F: File of byte;
begin
  {$I-}
  assign(F, InFName);
  erase(F);
  {$I+}
  DelFile := (IOResult = 0);
end;

function Log2(I: word): word;
var
  L: Word;
begin
  L := 15;
  while (I and $8000 = 0) do begin
    Dec(L);
    I := I shl 1;
  end;
  Log2 := L;
end;

procedure WriteRatio(SourceSize, NewSize: LongInt);
var
  I: Word;
  Result: LongInt;
begin
  Result := NewSize * 10;
  for I := 1 to 4 do begin
    Write(Ressult div SourceSize);
```

```
    if I = 2 then Write('.');
    Result := (Result mod SourceSize) * 10;
  end;
end;

function GetFileSize (InFName: PathStr): LongInt;
var
  F: File of byte;
begin
  if FileExists(InFName) then begin
    assign(F, InFName);
    FileMode := 0;
    Reset(F);
    GetFileSize := FileSize(F);
    FileMode := 2;
    close(F);
  end
  else GetFileSize := 0;
end;

end.

{ ---------- End of ExtTools.pas ---------- }
```

3.2 一个通用的文件对象——FILE OBJECT

本书中所有程序是围绕 Windows 可执行文件展开的，这里给出的 File Object 定义是全书示范程序的基础。源程序放在本书所附磁盘的 CHAP2 目录下的 filedef.pas 中。在本节中，我们首先简单说明一下面向对象技术，然后说明 File Object 的层次关系，最后给出 File Object 的源程序定义。

3.2.1 面向对象技术

“面向对象”是近年来计算机领域中一个非常时髦的词汇，在软件工程的实践者看来，它常常与如下的一些概念密切相关：数据抽象、封装性、继承性、多态性、可扩充性、类型程序设计、信息隐藏、代码重用、模块化等等，甚至还可以列出更多的相关概念。人们往往把“面向对象”与面向对象的程序设计语言联系起来，但这是不正确的，因为面向对象的程序设计语言有许许多多，而且风格迥异，所以从语言的角度出发来理解“面向对象”是片面的。

“面向对象”这一概念是：根据不同对象之间的相互作用和联系，以对象为单位，从结构组织上去模拟客观世界。从系统实现的角度看，“面向对象”的实质也许可以看成是支持“数据抽象”，而且是“分层的数据抽象”。所谓“数据抽象”，粗略地讲，就是把数据对象和内部表

示和它所允许的操作汇集起来，并予以封装，使得对于数据对象的访问只能通过引用其接口中的所规定的外部可见操作来访问。

3.2.2　File Object 的层次关系图

在 PC 环境编程中，我们经常要对 DOS 和 Windows 操作系统中的各种类型文件进行操作。我们通常所说的文件类型有：TEXT、BIN、COM、EXE(DOS)、NE(Windows)、PE(NT)、LE(OS/2)等。根据上述文件类型的文件格式本身的特点，在 File Object 中，我们把它们组织成如下的层次关系：

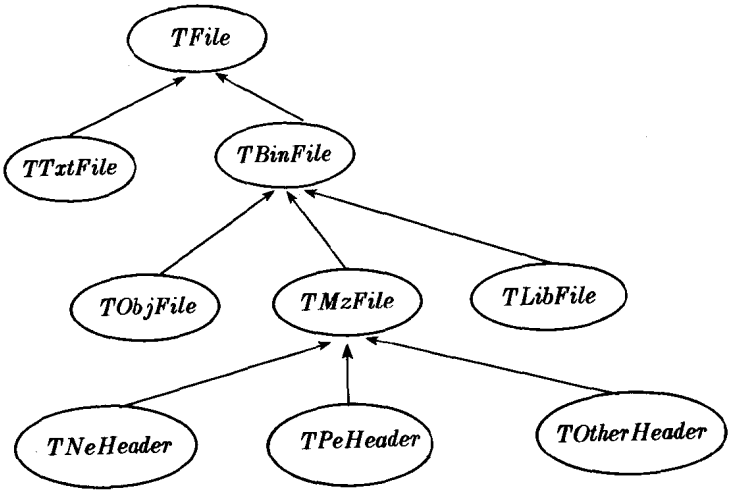

图 3.1　文件对象的层次结构

下面给出对象 File Object 层次结构的详细描述，包括每个类的成员。

TFile	
成员变量	成员函数
F	Init [V]
wSign	Done [V]
InitDir	Run [V]
FileName	Display [V]
FSize	DisplayFileInfo [V]
OfsNe	

TTxtFile	
成员变量	成员函数
TF	

TBinFile	
成员变量	成员函数
None	*None*

TMzHeader	
成员变量	成员函数
MzHeader	Init
MzHeader	GetDosFileSize
MzHeader	GetMzHeaderSize
MzHeader	GetloadImageSize
MzHeader	GetImageSize
MzHeader	GetSymTabtype
MzHeader	GetRelocList
MzHeader	StripSymTab
MzHeader	SetImageSize
MzHeader	BrowseMzRelocList
MzHeader	DisplayMzHeader
MzHeader	DisplayMzRelocList
MzHeader	DisplaySymTab
MzHeader	ExistOverlay

TLeHeader	
成员变量	成员函数
None	Init

TOtherHeader	
成员变量	成员函数
None	Init

TNeHeader	
成员变量	成员函数
neHeader	Init
Pseg	Done [V]
PResource	isSelfLoadingAP
PResName	ShiftSize
PModRef	GetPageSize
PImpName	GetResPageSize
PEntry	GetGeneralPageSize
PNonresName	GetEntryCount
ModName	CopyStr
ModDescription	GetAEntry
Root	GetImportedName
GangLoad	GetEntryName
BrowseStop	GetmoduleName
PCurrInfo	GetModuleIdx
PCurrInfo	IsExportedFunction
PCurrInfo	GetSegmentEntry
PCurrInfo	GetRtTypeIDName
PCurrInfo	ReadAseg
PCurrInfo	ExistRelocItem
PCurrInfo	ReadARelocItem
PCurrInfo	GetRelocItemCount
PCurrInfo	GetOfsEndOfHeader
PCurrInfo	GetResLength
PCurrInfo	BuildTableArray
PCurrInfo	RecalcTableOfs
PCurrInfo	GetWinStub
PCurrInfo	BuildInfoTree
PCurrInfo	DisposeInfoTree
PCurrInfo	SetInfo
PCurrInfo	BrowseInfoTree
PCurrInfo	BrowseSegmentTab
PCurrInfo	BrowseResourceTab
PCurrInfo	BrowseEntryTab
PCurrInfo	BrowseAsegRelocList
PCurrInfo	LoadTables [Private]

3.2.3　File Object 的具体实现

```
Unit FILEDEF;

interface

uses Dos, ExtTools;

const
{ PackWin 显示方式/是否压缩资源的选项 }
  fPower   : boolean = False;
  fPackRes : Boolean = True;

{ 定义错误码 }
  rtError: word = 0;
  { ErrorCode      Meanning
      0EE00H         File not found        { 文件没有找到 }
      0EE01H         File type error       { 文件类型错 }
      0EE02H         File access failure   { 文件存取失败 }
      0EE10H         Self-Loading AP       { 文件是自装载程序 }
  }

{ 定义各种文件格式的标记值。常见的文件可分为二进制文件和正文文件。
可执行文件属于二进制文件。DOS 可执行文件的标记为"MZ"或"ZM",
Windows 可执行文件的标记为"NE"... }

  BinSign     = $0000;
  TextSign    = $5454;
  EXEAppSign  = $5A4D;
  EXEAppSign1 = $5A4D;
  EXEAppSign2 = $4D5A;
  WinAppSign  = $454E;
  OS2AppSign  = $454C;
  Win386Sign  = $3357;
  OS2NewSign  = $584C;
  Win32Sign   = $4550;

{ 可执行文件文件格式的说明性字符串 }
  SignStr: array[1..6] of PChar = (
    'MZ - - not a new executable',
    'NE - - New (segmented) Executable',
    'LE - - OS/2 1.0 Linear Executable (Windows VxD)',
    'W3 - - Win386 (Collection of LE files)',
    'LX - - OS/2 2.0 Linear Executable',
    'PE - - Win32 NT Portable Executable');
```

```
{ Base file object ---------------------------------------------------------- }

{ 定义 File Object 的基类 }
type
  PFile = ^TFile;
  TFile = object
{ 定义文件句柄 }
    F: File;
{ 定义文件类型 }
    wSign: Word;
{ 定义文件的启动目录 }
    InitDir: DirStr;
{ 定义文件名 }
    FileName: PathStr;
{ 定义文件大小 }
    FSize,
{ 如果存在 NE、PE、LE,这个变量用来记录首部在文件中的位置 }
    ofsNe: LongInt;
    constructor Init(FName: PathStr);
    destructor Done; virtual;
    procedure Run; virtual;
{ Display 用于显示文件结构 }
    procedure Display; virtual;
{ DiaplayFileInfo 用于显示文件中的段结构 }
    procedure DisplayFileInfo; virtual;
  end;

{ Text file object ---------------------------------------------------------- }

  PTxtFile = ^TTxtFile;
  TTxtFile = object(TFile)
{ 定义文本文件的句柄 }
    TF: Text;
    constructor Init(FName: PathStr);
    destructor Done; virtual;
  end;

{ Bin file object ----------------------------------------------------------- }

  PBinFile = ^TBinFile;
  TBinFile = object(TFile)
  end;
```

```
{ MS-DOS EXE File Header define declare ----------------------------------------------------------}

{ 定义 MZ 文件头结构 }
type
  tagMZHeader = Record
    Sign,              { 00h  'MZ' or 'ZM' }
    lenImage,          { 02h  Length of image }
    numPages,          { 04h  Size of file in 512 - byte pages }
    numRelocEntry,     { 06h  Number of relocation-table items }
    sizeHeader,        { 08h  Size of header in paragraphs }
    numMinAlloc,       { 0Ah  Minimum number of paragraphs above }
    numMaxAlloc,       { 0Ch  Maximum number of paragraphs above }
    -SS,               { 0Eh  Displacement of stack segment in paragraphs }
    -SP,               { 10h  offset in SP register }
    CheckSum,          { 12h  word Checksum }
    -IP,               { 14h  IP register offset }
    -CS,               { 16h  code segment displacement }
    ofsRelocList,      { 18h  Displacement of first relocation item }
    noOverlay: word;   { 1Ah  Overlay number (Resident code = 0) }
    Reserved: array [ $ 1C.. $ 3B] of char;
    ofsWinHeader: LongInt;   { 3Ch  New exe file header offset }
  end;

{ 重定位数据结构,
重定位结构并中没有索引,在这里加入索引是为了便于识别 }
  PMzRelocItem = ^tagMzRelocItem;
  tagMzRelocItem = record
    -Ofs, -Seg: Word;
{ 定义重定位的索引 }
    Idx: word;              { new }
  end;

{ MS-DOS EXE File object ------------------------------------------------------------------------}

  PMzHeader = ^TMzHeader;
  ProcMzDo  = procedure (PSelf: PMzHeader; P: Pointer);
  TMzHeader = object(TBinFile)
    MzHeader: tagMZHeader;
    constructor Init (FName: PathStr);
{ 取文件大小
文件大小并不是由 MZ 文件头中的 lenImage 和 numPages 指定的 }
    function GetDosFileSize: LongInt;
{ 取文件头大小 }
```

```
    function GetMzHeaderSize: LongInt;
  { 取装入部分的大小(不含文件头部分) }
    function GetLoadImageSize: LongInt;
  { 取装入部分的大小(含文件头部分) }
    function GetImageSize: LongInt;
  { 判断程序中调试信息的类型是 TurboDebugger 或 CodeView }
    function GetSymTabType: word;
  { 取文件重定位部分到一个缓冲区 }
    function GetRelocList: PBuf;
  { 是否存在覆盖模块 }
    function ExistOverlay: Boolean;
  { 去掉调试信息 }
    procedure StripSymTab;
  { 设置装入模块的大小,即把大小记入 MZHEADER 中 }
    procedure SetImageSize (var MZ: tagMzHeader; SizeImage: LongInt);
  { 浏览文件重定位表 }
    procedure BrowseMzRelocList (Body: PBuf; Operation: ProcMzDo);
  { 显示文件头信息 }
    procedure DisplayMzHeader;
  { 显示文件重定位信息 }
    procedure DisplayMzRelocList;
  { 显示调试信息 }
    procedure DisplaySymTab;
  end;

{ OS/2 EXE File Struct Object ------------------------------------------------ }
  PLEHeader = ^TLEHeader;
  TLEHeader = object (TMzHeader)
    constructor Init(FName: PathStr);
  end;

{ Other EXE File Struct Object ---- ------------------------------------------- }
  POtherHeader = ^TOtherHeader;
  TOtherHeader = object (TMzHeader)
    constructor Init(FName: PathStr);
  end;

{ New executable file header define declare ---------------------------------- }
  tagNEHeader = Record
    Sign: word;                 { 00h  'NE'                          }
    Version,                    { 02h  The linker version number     }
    Revision: byte;             { 03h  The linker revision number    }
    ofsEntryTab,                { 04h  The offset of the entry table }
    lenEntryTab: word;          { 06h  The length of the entry table }
```

```
    CheckSum: longint;          { 08h  Reserved                          }
    Flags,                      { 0Ch  Flags that describe exe-file      }
    AutoDataSeg,                { 0Eh  The automatic data segment        }
    HeapSize,                   { 10h  Initial size of local heap        }
    StackSize,                  { 12h  Initial size of the stack         }
    _IP,                        { 14h  Segment:offset of CS:IP           }
    _CS,
    _SP,                        { 18h  Segment:offset of SS:SP           }
    _SS,
    numSegmentTab,              { 1Ch  Number of entries in Segment table }
    numModuleReferenceTab,
                { 1Eh  Number of entries in module-reference table }
    numNonresidentNameTab,
                { 20h  Number of bytes in the nonresident-name table }
    ofsSegmentTab,              { 22h  offset of seg ent table           }
    ofsResourceTab,             { 24h  offset of resource table          }
    ofsResidentNameTab,         { 26h  offset of resident-name table     }
    ofsModuleReferenceTab,      { 28h  offset of module-reference table  }
    ofsImportedNameTab: word;   { 2Ah  offset of imported-name table     }
    ofsNonresidentNameTab:      { 2Ch  offset of nonresident-name table  }
      LongInt;
    numMovableEntryPoints,      { 30h  Number of movable entry points    }
    SegmentAlign,               { 32h  Page size shift count             }
    numResourceSeg: word;       { 34h  Number of resource segments       }
    TargetOS,                   { 36h  Target operating system           }
    ExtFlags: byte;             { 37h  Additional info about exe-file    }
    ofsGangLoad,                { 38h  offset of fast-load area          }
    lenGangLoad,                { 3Ah  length of fast-load area          }
    sizeMinCodeSwap,            { 3Ch  Reversed.                         }
    WinVer: word;               { 3Eh  Expected version for windows      }
    { ------------------------------------------------------------------ }
{ 直接保存各种表的长度,便于存取 }
    lenSegmentTab,
    lenResourceTab,
    lenResidentNameTab,
    lenModuleReferenceTab,
    lenImportedNameTab: word;
  end;

{ Windows structure define ------------------------------------------------}
{ 段的类型常量定义 }
{ Segment flags }
const
```

```
  SEG_DATA       = $0001;
  SEG_MOVABLE    = $0010;
  SEG_SHAREABLE  = $0020;
  SEG_PRELOAD    = $0040;
  SEG_RELOCINFO  = $0100;
  SEG_DISCARD    = $1000;
  SEG_READONLY   = $0080;

type
  {段表中段项的属性定义}
  {Segment table entry}
  {Flags that discribe the contents of exe-file
    Bit   Meanning
     0    Data segment if sets. otherwise, code segment
     1    Loader has allocated memory for the segment
     2    Segment is loaded
     3    ..
     4    MOVABLE if sets. otherwise, FIXED
     5    PURE or SHAREABLE
     6    PRELOAD if sets. Otherwise, LOADONCALL
     7    EXECUTEONLY (code) or READONLY (data)
     8    Contains relocation data
     9    ..
     10   ..
     11   ..
     12   Discardable
     13   ..
     14   ..
     15   >= 64K
  Size is 8 bytes}
{段表结构}
  PSegmentEntry = ^tagSegmentEntry;
  tagSegmentEntry = Record
    ofs,
    lenSeg,
    Attr,
    lenMem: word;
  end;
{入口表结构}
  {Entry table}
  PEntryBundle = ^tagEntryBundle;
  tagEntryBundle = Record
```

```
    Count,
    Flags: byte;    { Movable if 0ffh, otherwise, fixed }
  end;
{ 可移动的入口 }
  PMovableEntry = ^tagMovableEntry;
  tagMovableEntry = Record
    Flags: byte;
    CD3F: word;
    SegNum: byte;
    Ofs: word;
  end;
{ 固定的入口 }
  PFixedEntry = ^tagFixedEntry;
  tagFixedEntry = Record
    Flags: byte;
    Ofs: word;
  end;
{ 一个通用的入口,便于可移动入口和固定入口使用 }
  PAEntry = ^tagAEntry;
  tagAEntry = record
    Idx: Word;
    Bundle: PEntryBundle;
    MEntry: PMovableEntry;
    FEntry: PFixedEntry;
  end;
{ 重定位 }
  { Reloction Information }
  { RelocAddrType:
      Value  Meaning
       0      Low byte at the specified offset
       2      16 - bit selector
       3      32 - bit pointer
       5      16 - bit offset
       11     48 - bit pointer
       13     32 - bit offset
    RelocType:
      Value  Meaning
       0      Internal reference
       1      Imported ordinal
       2      Imported name
       3      OSFIXUP
    Ord:
      Imported ordinal      Index to a module's reference table
```

```
                                Ord specify a function ordinal value
        Imported name           Index to a module's reference table
                                Ord to an imported-name table
        Internal reference      Index = 0ffh
          Fixed                 Ord specify an offset to the segment
          Movable               Ord specify an ordinal value found in entry table
    }
    PRelocItem = ^tagRelocItem;
    tagRelocItem = Record
      RelocAddrType,
      RelocType: byte;
      Ofs,
      Idx,
      Ord: Word
    end;

  { Predefined Resource Types }
  { 资源类型常量定义 }
  const
    rt_Cursor       = $8001;
    rt_Bitmap       = $8002;
    rt_Icon         = $8003;
    rt_Menu         = $8004;
    rt_Dialog       = $8005;
    rt_String       = $8006;
    rt_FontDir      = $8007;
    rt_Font         = $8008;
    rt_Accelerator  = $8009;
    rt_RCData       = $800A;
    rt_NameTable    = $800F;
    rt_VerInfo      = $8010;
    rt_HF           = $806C;
    rt_DATA         = $80BF;
    rt_CCTL         = $82E2;

    rt_Group_Cursor = rt_Cursor + 11;
    rt_Group_Icon   = rt_Icon + 11;

  type
    { Resource table declare }
    { word      rscAlignShift
      typeinfo rscTypes[];
      word      rscEndTypes;
```

```
    byte    rscResourceNames[];
    byte    rscEndNames;
  }
  PTypeInfo = ^tagTypeInfo;
  tagTypeInfo = Record
    rtTypeID,
    rtResourceCount: word;
    rtReserved: LongInt;
    { rtNameInfo: array [1..rtResCount] of TNameInfo }
  end;

  PNameInfo = ^tagNameInfo;
  tagNameInfo = Record
    rnOffset,
    rnLength,
    rnFlags,        { 10h Movable   20h Pure   40h Preload }
    rnID,
    rnHandle,
    rnUsage: word;
  end;

  PEachRes = ^tagEachRes;
  tagEachRes = Record
    Idx: Word;
    TypeInfo: PTypeInfo;
    NameInfo: PNameInfo;
  end;

{ Tree define ---------------------------------------------------------- }
{ PACKWIN 装入段的类型 }
{ BlkInfo flags const define }
const
  { flags }
  TheEnd      = 0;
  Stub        = 1;
  Sys         = 2;
  cSeg        = 3;
  dSeg        = 4;
  Reloc       = 5;
  Res         = 6;
  Null        = $fe;
  Unknown     = $ff;
```

```
    { sys.Idx }
    sysAll      = 0;
    sysNe       = 1;
    sysSeg      = 2;
    sysRes      = 3;
    sysResName  = 4;
    sysModRef   = 5;
    sysImpName  = 6;
    sysEntry    = 7;
    sysNonres   = 8;

  { 在 PackWin 中,每块或每段数据用一个 BlkInfo 结构来描述 }
  { File information block define }
  const
    GangLoadFlag = 1;
    PackedFlag   = 2;
  { 段的信息头 }
  type
    PBlkinfo = ^tagBlkInfo;
    tagBlkInfo = record
      flags: byte;              { Info Node type                }
      OtherFlag: byte;          { GangLoad flag or Packed flag  }
      Idx: word;                { Index                         }
      Ofs,                      { offset                        }
      Len,                      { Length                        }
      NewOfs,                   { new offset                    }
      NewLen: LongInt;          { Real Length                   }
      Owner: PChar;             { Declare                       }
      Left,                     { Left pointer                  }
      Right: PblkInfo;          { Right pointer                 }
    end;

  { New executable file header object ------------------------------ }
  { * * * }

  type
    PNeHeader   = ^TNEHeader;
    TableArray = array [sysNe..sysNonres] of TBuf;
    ProcDo = procedure (PSelf: PNeHeader; P: Pointer);
    ProcOperation = procedure (PSelf: PNeHeader; P: Pointer; Index: word);
    ProcDoRelocItem = procedure (PSelf: PNeHeader; P: PRelocItem; Body:
PBytes);
```

```
  TNeHeader = object (TMzHeader)
    neHeader: tagNeHeader;              { New exe header record  }
    PSeg,                               { Segment Table          }
    PResource,                          { Resource Table         }
    PResName: Pbytes;                   { Resident Name Table    }
    PModRef: PWords;                    { Module Reference Table }
    PImpName,                           { Imported Name Table    }
    PEntry,                             { Entry Table            }
    PNonresName: PBytes;                { Nonresident Name Table }

    ModName,                            { The Module Name        }
    ModDescription: String[80];         { Module Description     }
{ 第一个块 }
    Root: PBlkInfo;
{ 是否包含快装区域 }
    GangLoad: Boolean;
{ 停止浏览 }
    BrowseStop: Boolean;
{ 当前块 }
    PCurrInfo: PBlkInfo;

    constructor Init (FName: PathStr);
    destructor Done; virtual;
{ 判断是否是自装入的执行程序 }
    function isSelfLoadingAP (AHeader: tagNeHeader): boolean;
{ 计算 2 的幂数 }
    function ShiftSize(Secs: Word): LongInt;
{ 取逻辑页大小 }
    function GetPageSize: word;
{ 取资源逻辑页大小 }
    function GetResPageSize: word;
{ 取通用页的大小 }
    function GetGeneralPageSize: word;
{ 取入口表的入口项个数 }
    function GetEntryCount: Word;
{ 串复制 }
    function CopyStr(var Buf): String;
{ 取某一个入口项的内容 }
    function GetAEntry(EntryNo: Word): PAEntry;
{ 取某个输入函数的名字 }
    function GetImportedName (Idx: word): String;
{ 取某个入口点的名字 }
    function GetEntryName(EntryNo: Word): String;
```

```
{ 取模块名 }
  function GetModuleName (Module: Word): string;
{ 取某个模块名的序号 }
  function GetModuleIdx (ModuleName: String): word;
{ 判断是否输出函数 }
  function isExportedFunction(EntryNo: Word): boolean;
{ 取段表项 }
  function GetSegmentEntry (P_Seg: PBytes; Idx: word): PSegmentEntry;
{ 取资源名 }
  function GetRtTypeIDName (rtTypeID: word): String20;
{ 取一个段的数据 }
  function ReadAseg (segNo: word): PBuf;
{ 是否存在重定位表 }
  function ExistRelocItem (segNo: word): Boolean;
{ 取重定位表 }
  function ReadARelocList (segNo: word): PBuf;
{ 取重定位表的计数 }
  function GetRelocItemCount (segNo: word): word;
{ 返回首部实际大小 }
  function GetOfsEndOfHeader (var AHeader: tagNeHeader): LongInt;
{ 取某个资源的具体大小 }
  function GetResLength (S: String; ResTypeID: Word;
                         ResOfs, ResLen: LongInt): LongInt;
{ 读入各种表格 }
  procedure BuildTableArray(var T: TableArray);
{ 重新计算表的位置,这一函数经常是在表修改后被调用 }
  procedure RecalcTablesOfs (var AHeader: tagNeHeader);
{ 把 stub 存入某个文件 }
  procedure GetWinStub (StubName: PathStr);
{ 建立信息块排序树 }
  procedure BuildInfoTree;
{ 释放信息块排序树 }
  procedure DisposeInfoTree;
{ 建立信息块 }
  procedure SetInfo(myType: byte; myIndex: word;
                    myOfs, myLen: LongInt; myOwner: String);
{ 浏览信息块排序树 }
  procedure BrowseInfoTree (NRoot: PblkInfo; DoNode: ProcDo);
{ 浏览段表 }
  procedure BrowseSegmentTab (Body: PBuf; Operation: ProcOperation);
{ 浏览资源表 }
  procedure BrowseResourceTab (Body: PBuf; DoRes: ProcDo);
```

```
{ 浏览入口表 }
    procedure BrowseEntryTab (Body: PBuf; DoEntry: ProcDo);
{ 浏览重定位表 }
    procedure BrowseAsegRelocList (Body: PBuf; segNo: word;
                                   DoProc: ProcDoRelocItem);

  private
    procedure LoadTables;
  end;

{ Function ------------------------------------------------------------}
{ 计算装入模块的大小/MZ }
  function ImageSize (size: LongInt): LongInt;
{ 取文件类型标志 }
  function GetFileSign(FName: PathStr): Word;
{ 取输出函数 }
  function GetExportedName (FName, APath: PathStr; FuncIdx: Word): String;
{ 显示返回错误 }
  procedure DisplayRtError;

implementation
{ 在下面的具体实现过程中,不再给出注释,请读者自行分析 }
Uses PackUnit, Strings;
{ ------------------------------------------------------------ }
function ImageSize (size: LongInt): LongInt;
var
  exeSize: LongInt;
  wPages, lenImage, numPages: word;
begin
  wPages := size shr 9;
  lenImage := size and $1FF;
  if lenImage = 0 then
    numPages := wPages
  else numPages := wPages + 1;
  DWord(exeSize).HiWord := numPages;
  DWord(exeSize).LoWord := lenImage;
  ImageSize := exeSize;
end;
function GetFileSign(FName: PathStr): Word;
var
  F: File;
  I, J, fSign: Word;
  Header: tagMzHeader;
```

```
  Buf: array [1.. $ 40] of char;
begin
  fSign : = 0;
  if FileExists(FName) then begin
    assign(F, FName);
    reset(F, 1);
    if FileSize(F) > $ 40 then begin
      BlockRead(F, Header, sizeof(tagMZHeader));
      if (Header.Sign = (EXEAppSign1)) or                    { 'MZ' }
        (Header.Sign = (ExeAppSign2)) then begin             { 'ZM' }
        Header.Sign : = ExeAppSign;
        fSign : = (ExeAppSign1);
        with Header do
         if ( (ofsRelocList > = $ 40) or
             (ofsRelocList + numRelocEntry * 4 < $ 3C)) and
           ( ofsWinHeader + 2 < FileSize(F) ) and
           ( ofsWinHeader > = $ 40 ) then begin
           Seek(F, ofsWinHeader);
           BlockRead(F, fSign, 2);
        end;
      end;
    end;  { file size }
    if fSign = 0 then begin
      I : = wMin(FileSize(F), $ 40);
      Seek(F, 0);
      BlockRead(F, buf, I);
      J : = 0;
      repeat
        Inc(J);
      until (J = I) or not (buf[J] in TextChars);
      if buf[J] in TextChars then fSign : = (TextSign);
    end;
    close(F);
  end;  { file exist }
  GetFileSign : = fSign;
end;

procedure DisplayRtError;
begin
  if rtError < > 0 then begin
    Write('ERROR: ');
    case rtError of
      $ EE00: Write('File not found. ');
```

```
        $EE01: Write('File format error. ');
        $EE02: Write('File access failure. ');
       else Write('Unknown error #', whStr(rtError), 'h');
     end;
     Writeln;
   end;
end;

function GetExportedName(FName, APath: PathStr; FuncIdx: Word): String;
var
  S, S1, SS, PATH: PathStr;
  P: PNeHeader;
begin
  if FName = '' then begin
    GetExportedName := '';
    Exit;
  end;

  S := '';
  PATH := GetEnv('PATH');

  if (APath <> '') then begin
    S := FSearch(FName, APath);
    if (Pos('.', FName) = 0) and (S = '') then begin
      S := FSearch(FName + '.EXE', APath);
      if (S = '') then S := FSearch(FName + '.DLL', APath);
    end;
  end;

  if Pos('.', FName) = 0 then S1 := FName + '.DLL' else S1 :=
FName;{ +.DLL }
  if (S = '') then S := FSearch(S1, PATH);
{ Path  }

  if (S = '') and (Pos('.',Fname) = 0) then begin
    S1 := FName + '.EXE';
{ +.EXE }
    if (APath <> '') and FileExists(APath + S1) then S := APath + S1;
{ APath }
    if (S = '') then S := FSearch(S1, PATH);
{ Path  }
  end;

  if (S <> '') then begin
    S := FExpand(S);
    P := New(PNeHeader, Init(S));
    if (P <> nil) then begin
```

```
      if (P^.wSign = (WinAppSign)) then
        GetExportedName := P^.GetEntryName(FuncIdx);
      Dispose(P,Done)
    end
    else GetExportedName := '';
  end
  else GetExportedName := '';
end;

{ Base file format object ---------------------------------------- }

constructor TFile.Init(FName: PathStr);
var
  Name: NameStr;
  Ext: ExtStr;
begin
  rtError := 0;
  wSign := 0;                          { Sign   }
  ofsNe := 0;                          { offset }
  FileName := FName;
  FSplit(FName, InitDir, Name, Ext);

  if not FileExists(FName) then begin
    rtError := $EE00;
    fail;
  end;
  {$I-}
  assign(F, FName);
  reset(F, 1);
  FSize := FileSize(F);
  {$I+}
  if (IOResult <> 0) then begin
    rtError := $EE02;
    Done;
    fail;
  end;
end;

destructor TFile.Done;
begin
  close(F);
end;

procedure TFile.Display;
begin
end;
```

```
procedure TFile.DisplayFileInfo;
begin
  Writeln;
  Writeln('Only display New Executable file information');
  Writeln;
end;

procedure TFile.Run;
begin
end;

{ Text File Format object ------------------------------------------ }

constructor TTxtFile.Init(FName: PathStr);
begin
  inherited Init(FName);
  if rtError <> 0 then fail;
  inherited Done;
  {$I-}
  assign(TF, FName);
  reset(TF);
  {$I+}
  if (IOResult <> 0) then begin
    rtError := $EE02;
    Done;
    fail;
  end;
end;

destructor TTxtFile.Done;
begin
  close(TF);
end;

{ Bin File Format object ------------------------------------------- }

{ MS-DOS Header object --------------------------------------------- }

constructor TMZHeader.Init(FName: PathStr);
begin
  inherited Init(FName);
  if rtError <> 0 then fail;
  {$I-}
  FillChar(mzHeader, sizeOf(tagMZHeader), #0);
  if FSize <= $1C then begin
    rtError := $EE01;
    Done;
```

```
    Fail;
  end
  else if FSize < sizeof(tagMZHeader) then
    BlockRead(F, mzHeader, $ 1C)
  else BlockRead(F, mzHeader, sizeof(tagMZHeader));
  {$I+}
  if (MZHeader.Sign <> $ 4D5A) and                    {'MZ'}
     (MZHeader.Sign <> $ 5A4D) then begin             {'ZM'}
    rtError := $ EE01;
    Done;
    Fail;
  end;
  with MZHeader do begin
    if ( (ofsRelocList >= $ 40) or
       (ofsRelocList + numRelocEntry * 4 < $ 3C)) and
       ( ofsWinHeader + 2 < FSize ) and
       ( ofsWinHeader >= $ 40 ) then begin
       Seek(F, ofsWinHeader);
       BlockRead(F, wSign, 2);
       ofsNE := MZHeader.ofsWinHeader
    end;
    {
    if ofsRelocList < $ 40 then
      FillChar(PBytes(@mzHeader)^[ofsRelocList], ( $ 40 - ofsRelocList), #0);
    }
  end;
  if (IOResult <> 0) then begin
    rtError := $ EE02;
    Done;
    fail;
  end;
end;

function TMZHeader.GetDosFileSize: LongInt;
begin
  GetDosFileSize := FSize;
end;

function TMZHeader.GetLoadImageSize: LongInt;
var
  lrPages: LongInt;
begin
  with MZHeader do begin
    if lenImage <> 0 then
```

```
      lrPages := numPages - 1
    else lrPages := numPages;
    GetLoadImageSize := lrPages * $200 + lenImage-sizeHeader * $10;
  end;
end;

function TMZHeader.GetImageSize: LongInt;
var
  lrPages: LongInt;
begin
  with MZHeader do begin
    if lenImage <> 0 then
      lrPages := numPages - 1
    else lrPages := numPages;
    GetImageSize := lrPages * $200 + lenImage;
  end;
end;

procedure TMZHeader.SetImageSize;
var
  wrPages: word;
begin
  with MZ do begin
    wrPages := sizeImage div $200;
    lenImage := sizeImage mod $200;
    if lenImage = 0 then
      numPages := wrPages
    else numPages := wrPages + 1;
  end;
end;

function TMZHeader.GetMZHeaderSize: LongInt;
begin
  GetMZHeaderSize := MZHeader.sizeHeader shl 4;
end;

function TMzHeader.getSymTabType: word;
const
  TDSign: PChar = #$FB#$52;
  CVSign: PChar = 'NB02';
var
  Sign: LongInt;
begin
  getSymTabType := 0;
  if ((GetMzHeaderSize + GetLoadImageSize + 2) < FSize) then begin
    seek(F, getMzHeaderSize + getLoadImageSize);
```

```
    Sign : = 0;
    BlockRead(F, Sign, 2);
    if Sign = wPChar(TDSign) then
      getSymTabType : = 1;
  end;
  seek(F, FSize - 8);
  BlockRead(F, Sign, 4);
  if Sign = lPChar(CVSign) then
    getSymTabType : = 2;
end;

function TMzHeader.ExistOverlay: Boolean;
begin
  ExistOverlay : = (FSize > GetImageSize);
end;

procedure TMzHeader.StripSymTab;
var
  FPos: LongInt;
begin
  case getSymTabType of
    1: begin      { Borland }
         seek(F, getMzHeaderSize + getLoadImageSize);
         truncate(F);
       end;
    2: begin      { Microsoft }
         seek(F, FSize - 4);
         BlockRead(F, FPos, 4);
         if fPower then
           seek(F, FSize-FPos)
         else seek(F, FSize-FPos + $ 10);      { TDSTRIP! }
         truncate(F);
       end;
  end;
end;

function TMzHeader.GetRelocList: PBuf;
  procedure QuickSort(A: PDWords; Lo, Hi: Integer);
    function RSize(K: LongInt): LongInt;
    var
      I: LongInt;
    begin
      I : = DWord(K).HiWord shl 4;
      RSize : = DWord(K).LoWord + I ;
```

```
    end;
    procedure Sort(l, r: Integer);
    var
      x, y: LongInt;
      i, j: integer;
    begin
      i := l; j := r; x := RSize(a^[(l + r) DIV 2]);
      repeat
        while RSize(a^[i]) < x do i := i + 1;
        while x < RSize(a^[j]) do j := j - 1;
        if i <= j then
        begin
          y := a^[i]; a^[i] := a^[j]; a^[j] := y;
          i := i + 1; j := j - 1;
        end;
      until i > j;
      if l < j then Sort(l, j);
      if i < r then Sort(i, r);
    end;
  begin {QuickSort};
    Sort(Lo,Hi);
  end;
var
  A: PBuf;
begin
  NewPBuf(A);
  if mzHeader.numRelocEntry > $3f00 then begin
    Write(#7);
    mzHeader.numRelocEntry := $3f00;
  end;
  A^.Len := mzHeader.numRelocEntry * 4;
  if A^.Len <> 0 then begin
    GetMem(A^.P, A^.Len);
    Seek(F, mzHeader.ofsRelocList);
    BlockRead(F, A^.P^, A^.Len);
    QuickSort( PDWords(A^.P), 0, mzHeader.numRelocEntry - 1);
  end;
  GetRelocList := A;
end;
procedure TMzHeader.BrowseMzRelocList (Body: PBuf; Operation: ProcMzDo);
var
```

```
  I, N: word;
  SelfGet: Boolean;
  A: PBuf;
  B: tagMzRelocItem;
begin
  SelfGet := False;
  if (Body = nil) then begin
    A := GetRelocList;
    if A^.Len <> 0 then SelfGet := True;
    Body := A;
  end;
  if (Body^.Len <> 0) and (Body^.P <> nil) then begin
    N := Body^.Len div 4;
    for I := 1 to N do begin
      B._Ofs := PWord(@Body^.P^[(I-1)*4])^;
      B._Seg := PWord(@Body^.P^[(I-1)*4+2])^;
      B.Idx  := I;
      Operation (@Self, @B);
    end;   { for }
  end;   { if }
  if SelfGet then DisposePBuf(A);
end;

procedure TMzHeader.DisplayMzHeader;
begin
  With MZHeader do begin
    if (wSign <> 0) and (wSign <> $4D5A) then
    begin
      Writeln('Old Executable Header');
      Writeln;
    end;
    Writeln('DOS File Size', lhNzStr(GetDosFileSize): 43,
          'h  (', GetDosFileSize:6, '. )');
    Writeln('Load Image Size', lhNzStr(GetLoadImageSize): 41,
          'h  (', GetLoadImageSize:6, '. )');
    Writeln('Relocation Table entry count', whStr(numRelocEntry):28,
          'h  (', numRelocEntry:6, '. )');
    Writeln('Relocation Table address', whStr(ofsRelocList):32,
          'h  (', ofsRelocList:6, '. )');
    Writeln('Size of header record       (in paragraphs)',
          whStr(sizeHeader):14, 'h  (', sizeHeader:6, '. )');
    Writeln('Minimum Memory Requirement (in paragraphs)',
          whStr(numMinAlloc):14, 'h  (', numMinAlloc:6, '. )');
```

```
      Writeln('Maximum Memory Requirement (in paragraphs)',
            whStr(numMaxAlloc):14, 'h  (', numMaxAlloc:6, '. )');
      Writeln('File load checksum', whStr(CheckSum):38,
            'h  (', CheckSum:6, '. )');
      Writeln('Overlay Number', whStr(noOverlay):42,
            'h  (', noOverlay:6, '. )');
      Writeln;
      if (PWord(@mzHeader.Reserved[$20])^ = $726a) then begin
        Writeln('Borland TLINK Version ',
          VerStr(ord(mzHeader.Reserved[$1f]) shr 4,
                ord(mzHeader.Reserved[$1f]) and $f));
        Writeln;
      end;
      Writeln('Initial Stack Segment  (SS:SP)', #9#9#9' ',
            whStr(_SS), ':', whStr(_SP));
      Writeln('Program Entry Point    (CS:IP)', #9#9#9' ',
            whStr(_CS), ':', whStr(_IP));
      Writeln;
    end;
  end;

  { Browse MS-DOS Execute file relocation item list }
  procedure ShowMzRelocItem (PSelf: PMzHeader; P: Pointer); far;
  begin
    with PSelf^, PMzRelocItem(P)^ do begin
      Write(':4, whStr(_Seg), ':', whStr(_Ofs));
      if (Idx mod 5) = 0 then Writeln;
    end;
  end;

  procedure TMzHeader.DisplayMzRelocList;
  var
    I, J: Word;
    A: TBuf;
  begin
    if mzHeader.numRelocEntry <> 0 then begin
      Write('Relocation Locations (', mzHeader.numRelocEntry);
      if mzHeader.numRelocEntry <= 1 then
        Writeln(' Entry)  ')
      else Writeln(' Entries)');
      BrowseMzRelocList (nil, ShowMzRelocItem);
      Writeln;
    end;
  end;
```

```
procedure TMzHeader.DisplaySymTab;
begin
  case getSymTabType of
    1: begin
         Writeln('Borland TLINK Symbol Table Present');
       end;
    2: begin
         Writeln('Microsoft CodeView Symbol Table Present');
       end;
  end;
  Writeln;
end;

{ OS/2 EXE File Struct Object ---------------------------------------- }

constructor TLEHeader.Init (FName: PathStr);
begin
  inherited Init(FName);
  if rtError <> 0 then fail;
  if (wSign <> (OS2AppSign)) then begin
    rtError := $EE01;
    fail;
  end
end;

{ Other unknown format object ---------------------------------------- }

constructor TOtherHeader.Init (FName: PathStr);
begin
  inherited Init(FName);
  if rtError <> 0 then fail;
end;

{ New executable header ---------------------------------------------- }

constructor TNeHeader.Init (FName: PathStr);
begin
  inherited Init(FName);
  if rtError <> 0 then fail;
  if (wSign <> (WinAppSign)) then begin
    rtError := $EE01;
    fail;
  end
  else begin
    LoadTables;
    Root := nil;
```

```
    GangLoad := (neHeader.ExtFlags and 8) <> 0;
    if (IOResult <> 0) then begin
      rtError := $EE02;
      Done;
      fail;
    end;
  end;
  PCurrInfo := nil;
end;
procedure TNeHeader.LoadTables;
  procedure LoadTab(var P: PBytes; ofsTab: LongInt; lenTab: Word);
  begin
    if (LenTab <> 0) and (ofsTab + lenTab <= FSize) then begin
      GetMem(P, lenTab);
      seek(F, ofsTab);
      blockRead(F, P^, lenTab);
    end
    else P := nil;
  end;  { LoadTable }
var
  P: PBytes;
begin
  {$I-}
  seek(F, ofsNe);
  blockread(F, neHeader, sizeof(tagNEHeader));
  with neHeader do begin
    lenSegmentTab := ofsResourceTab - ofsSegmentTab;
    lenResourceTab := ofsResidentNameTab - ofsResourceTab;
    lenResidentNameTab := ofsModuleReferenceTab - ofsResidentNameTab;
    lenModuleReferenceTab := ofsImportedNameTab - ofsModuleReferenceTab;
    lenImportedNameTab := ofsEntryTab - ofsImportedNameTab;
    { Read segment table }
    LoadTab(PSeg, ofsNe + ofsSegmentTab, lenSegmentTab);
    { Read Resource table }
    LoadTab(PResource, ofsNe + ofsResourceTab, lenResourceTab);
    { Read Resident Name table }
    LoadTab(PResName, ofsNe + ofsResidentNameTab, lenResidentNameTab);
    { Read module reference table }
    LoadTab(PBytes(PModRef), ofsNe + ofsModuleReferenceTab,
            lenModuleReferenceTab);
```

```
    { Read Imported Name table }
    LoadTab(PImpName, ofsNe + ofsImportedNameTab, lenImportedNameTab);
    { Read Entry table }
    LoadTab(PEntry, ofsNe + ofsEntryTab, lenEntryTab);
    { Read NonresidentName table }
    LoadTab(PNonresName, ofsNonresidentNameTab, numNonresidentNameTab);
    if (PResName <> nil) then
      ModName := CopyStr(PResName^)
    else ModName := '';
    if (PNonresName <> nil) then
      ModDescription := CopyStr(PNonresName^)
    else ModDescription := '';
  end;
  {$I-}
  if IOResult <> 0 then rtError := $EE02;
end;

destructor TNeHeader.Done;
  procedure FreeTab(P: PBytes; lenTab: Word);
  begin
    if (P<>nil) and (lenTab <> 0) then
      FreeMem(P, lenTab);
  end;
begin
  with neHeader do begin
    FreeTab(PNonresName, numNonresidentNameTab);
    FreeTab(PEntry, lenEntryTab);
    FreeTab(PImpName, lenImportedNameTab);
    FreeTab(PBytes(PModRef), lenModuleReferenceTab);
    FreeTab(PResName, lenResidentNameTab);
    FreeTab(PResource, lenResourceTab);
    FreeTab(PSeg, lenSegmentTab);
  end;
  inherited Done;
end;

procedure TNeHeader.BuildTableArray (var T: TableArray);
begin
  with neHeader do begin
    T[sysNe].Len := $40;
    T[sysNe].P := @neHeader;
    T[sysSeg].Len := lenSegmentTab;
```

```
    T[sysSeg].P := PSeg;
    T[sysRes].Len := lenResourceTab;
    T[sysRes].P := PResource;
    T[sysResName].Len := lenResidentNameTab;
    T[sysResName].P := PResName;
    T[sysModRef].Len := lenModuleReferenceTab;
    T[sysModRef].P := PBytes(PModRef);
    T[sysImpName].Len := lenImportedNameTab;
    T[sysImpName].P := PImpName;
    T[sysEntry].Len := lenEntryTab;
    T[sysEntry].P := PEntry;
    T[sysNonres].Len := numNonresidentNameTab;
    T[sysNonres].P := PNonresName;
  end;   { with }
end;

procedure TNeHeader.RecalcTablesOfs(var AHeader: tagNeHeader);
begin
  with AHeader do begin
    ofsResourceTab := ofsSegmentTab + lenSegmentTab;
    ofsResidentNameTab := ofsResourceTab + lenResourceTab;
    ofsModuleReferenceTab := ofsResidentNameTab + lenResidentNameTab;
    ofsImportedNameTab := ofsModuleReferenceTab + lenModuleReferenceTab;
    ofsEntryTab := ofsImportedNameTab + lenImportedNameTab;
    ofsNonresidentNameTab  := ofsNE + ofsEntryTab + lenEntryTab;
  end;
end;

function TNeHeader.GetOfsEndOfHeader (var AHeader: tagNeHeader): LongInt;
begin
  GetOfsEndOfHeader :=
    AHeader.ofsNonresidentNameTab + AHeader.numNonresidentNameTab;
end;

function TNeHeader.GetPageSize: word;
begin
  getPageSize := 1 shl neHeader.SegmentAlign;
end;

function TNeHeader.ShiftSize(secs: word): longint;
begin
  shiftsize := LongInt(secs) shl neHeader.SegmentAlign;
end;

function TNeHeader.GetGeneralPageSize: word;
begin
```

```
  GetGeneralPageSize := 
    wMax(GetPageSize, GetResPageSize);
end;

function TNeHeader.isSelfLoadingAP (AHeader: tagNeHeader): boolean;
begin
  isSelfLoadingAP := (AHeader.Flags and $800 <> 0);
end;

function TNeHeader.GetSegmentEntry(P_Seg: PBytes; Idx: word):
PSegmentEntry;
var
  P: PSegmentEntry;
begin
  if (P_Seg = nil) then P_Seg := PSeg;
  if Idx > 0 then
    P := PSegmentEntry(@(P_seg^[(Idx-1) * 8]))
  else P := nil;
  GetSegmentEntry := P;
end;

function TNeHeader.GetImportedName (Idx: word): String;
begin
  GetImportedName := CopyStr(PImpName^[Idx]);
end;

function TNeHeader.GetModuleName (Module: Word): string;
begin
  if (Module <= neHeader.numModuleReferenceTab) and
     (Module > 0) then
    GetModuleName := CopyStr(PImpName^[PModRef^[Module-1]])
  else GetModuleName := '';
end;

function TNeHeader.GetModuleIdx (ModuleName: String): word;
var I: word;
begin
  I := 0;
  ModuleName := UpcaseStr(ModuleName);
  repeat
    Inc(I);
  Until (ModuleName = GetModuleName(I)) or
    (I > neHeader.numModuleReferenceTab);
  if (I > neHeader.numModuleReferenceTab) then
    GetModuleIdx := 0
  else GetModuleIdx := I;
```

```
end;

function TNeHeader.GetAEntry(EntryNo: Word): PAEntry;
var
  P: PAEntry;
  A: PEntryBundle;
  B: PMovableEntry;
  C: PFixedEntry;
  I, J, K: Word;
begin
  if NEHeader.lenEntryTab = 0 then
    P := nil
  else begin
    I := 0;
    K := 0;
    while (PEntry^[I] <> 0) and (I < neHeader.lenEntryTab) and
      (K < EntryNo) do begin
      A := PEntryBundle(@PEntry^[I]);
      Inc(I, 2);
      if (A^.flags = $ff) then begin
        J := 0;
        Repeat
          B := PMovableEntry(@PEntry^[I]);
          Inc(K);
          Inc(I, sizeof(tagMovableEntry));
          Inc(J);
        Until (K >= EntryNo) or (J = A^.Count);
      end
      else if A^.Flags = 0 then begin
        Inc(K, A^.Count);
        if K >= EntryNo then K := EntryNo;
      end
      else begin
        J := 0;
        Repeat
          Inc(K);
          C := PFixedEntry(@PEntry^[I]);
          Inc(I, sizeOf(tagFixedEntry));
          Inc(J);
        Until (K >= EntryNo) or (J = A^.Count);
      end;
      if (K = EntryNo) then begin
        New(P);
```

```
          P^.Bundle := A;
          P^.MEntry := B;
          P^.FEntry := C;
          P^.Idx := K;
        end
        else P := nil;
      end;
    end;   { null }
    GetAEntry := P;
  end;

  function TNeHeader.CopyStr(var Buf): String;
  var
    I: byte;
    S: String;
  begin
    Move(Buf, I, 1);
    if I <> 0 then begin
      Move(Buf, S[0], I+1);
      CopyStr := S;
    end
    else CopyStr := '';
  end;

  function TNeHeader.isExportedFunction(EntryNo: Word): boolean;
  var
    P: PAEntry;
    Flags: Byte;
  begin
    P := GetAEntry(EntryNo);
    if P <> nil then begin
      if P^.Bundle^.Flags = $ff then
        Flags := P^.MEntry^.Flags
      else Flags := P^.FEntry^.Flags;
      Dispose(P);
      isExportedFunction := (Flags and 1 <> 0);
    end
    else isExportedFunction := false;
  end;

  function TNeHeader.GetEntryName(EntryNo: Word): String;
  var
    I: Word;
    GetEntryName: String;
  begin
```

```
  _GetEntryName := '';
  if isExportedFunction(EntryNo) then begin
    if PNonresName <> nil then begin
      I := PNonresName^[0] + 1 + 2;
      while (PNonresName^[I] <> 0) and
        (PWord(@PNonresName^[I + PNonresName^[I] + 1])^ <> EntryNo) do
        Inc(I, PNonresName^[I] + 3);
      if (PWord(@PNonresName^[I + PNonresName^[I] + 1])^ = EntryNo) then
        _GetEntryName := CopyStr(PNonresName^[I]);
    end;  { NonresidentName }
    if (_GetEntryName = '') and (PResName <> nil) then begin
      I := PResName^[0] + 1 + 2;
      while (PResName^[I] <> 0) and
        (PWord(@PResName^[I + PResName^[I] + 1])^ <> EntryNo) do
        Inc(I, PResName^[I] + 3);
      if (PWord(@PResName^[I + PResName^[I] + 1])^ = EntryNo) then
        _GetEntryName := CopyStr(PResName^[I]);
    end;  { ResidentName }
  end;  { isImportedName }
  GetEntryName := _GetEntryName;
end;

function TNeHeader.GetEntryCount: Word;
var
  P: PAEntry;
  A: tagEntryBundle;
  I, K: Word;
begin
  if neHeader.lenEntryTab = 0 then
    GetEntryCount := 0
  else begin
    I := 0;   K := 0;
    while (PEntry^[I] <> 0) and (I <= neHeader.lenEntryTab) do begin
      A := PEntryBundle(@PEntry^[I])^;
      Inc(I, 2);
      if (A.flags = $ff) then begin
        Inc(K, A.Count);
        Inc(I, A.Count * SizeOf(tagMovableEntry));
      end
      else if A.Flags = 0 then begin
        Inc(K, A.Count);
      end
      else begin
```

```
        Inc(K, A.Count);
        Inc(I, A.Count * SizeOf(tagFixedEntry));
      end;
    end;
    GetEntryCount := K;
  end;
end;

function TNeHeader.getRtTypeIDName(rtTypeID: word): string20;
var
  SS: String20;
begin
  case rtTypeID of
    RT_ACCELERATOR:  SS := 'Acc';
    RT_BITMAP:       SS := 'Bitmap';
    RT_CURSOR:       SS := 'Cursor';
    RT_DIALOG:       SS := 'Dialog';
    RT_FONT:         SS := 'Font';
    RT_FONTDIR:      SS := 'FontDir';
    RT_GROUP_CURSOR: SS := 'GroupCursor';
    RT_GROUP_ICON:   SS := 'GroupIcon';
    RT_ICON:         SS := 'Icon';
    RT_MENU:         SS := 'Menu';
    RT_RCDATA:       SS := 'RC';
    RT_STRING:       SS := 'StrTab';
    RT_VERINFO:      SS := 'Verinfo';
    RT_NameTable:    SS := 'NameTab';
    RT_HF:           SS := 'HF';
    RT_DATA:         SS := 'DATA';
    RT_CCTL:         SS := 'CCTL';
    else SS := 'res #' + strNum(rtTypeID);
  end;  { case }
  getRtTypeIDName :=  SS;
end;

{ Get the resource length, in bytes }
function TNeHeader.GetResLength (S: String; ResTypeID: Word;
  ResOfs, ResLen: LongInt): LongInt;

  procedure CheckResLength (ResLen: LongInt; var ReturnLength: LongInt);
  begin
    if (ReturnLength > ResLen)
      { or ((ReturnLength + GetResPageSize) < ResLen)} then begin
      ReturnLength := ResLen;
      Writeln;
```

```
    Writeln('Error: ', S);
  end; { if }
end;

type
  PNewBitmapInfoHeader = ^TNewBitmapInfoHeader;
  TNewBitmapInfoHeader = record
    biSize: LongInt;
    biWidth: word;
    biHeight: word;
    biPlanes: Word;
    biBitCount: Word;
  end;

  PBitmapInfoHeader = ^TBitmapInfoHeader;
  TBitmapInfoHeader = record
    biSize: Longint;
    biWidth: Longint;
    biHeight: Longint;
    biPlanes: Word;
    biBitCount: Word;
    biCompression: Longint;
    biSizeImage: Longint;
    biXPelsPerMeter: Longint;
    biYPelsPerMeter: Longint;
    biClrUsed: Longint;
    biClrImportant: Longint;
  end;

{ Constants for the biCompression field }
const
  bi_RGB  = 0;
  bi_RLE8 = 1;
  bi_RLE4 = 2;

function GetBitmapLength (ResOfs, ResLen: LongInt): LongInt;
var
  A: TBitmapInfoHeader;
  B: TNewBitmapInfoHeader absolute A;
  LenRead: Word;
  ReturnLength: LongInt;
begin
  ReturnLength := ResLen;
  if ResLen >= 4 then begin
    {$I-}
    BlockRead (F, A, 4, LenRead);
```

```
    {$I+}
    if (IOResult = 0) and (LenRead = 4) and
       ((A.biSize = $28) or (A.biSize = $c)) then begin
      Seek(F, ResOfs);
      {$I-}
      BlockRead (F, A, SizeOf(TBitmapInfoHeader), LenRead);
      {$I+}
      if (IOResult = 0) and
         (LenRead = SizeOf(TBitmapInfoHeader)) then begin
        if A.biSize <> $28 then begin
          ReturnLength :=
            AlignSize((B.biWidth * B.biBitCount + 7) div 8, 4)
            * B.biHeight + B.biSize + 1 shl B.biBitCount * 3;
        end
        else begin
          if (A.biCompression = bi_RGB) then
            ReturnLength :=
              AlignSize((A.biWidth * A.biBitCount + 7) div 8, 4)
              * A.biHeight
          else ReturnLength := A.biSizeImage;
          Inc(ReturnLength, A.biSize + 1 shl A.biBitCount * 4);
        end;
        CheckResLength(ResLen, ReturnLength);
      end; { if }
    end; { if }
  end; { if }
  GetBitmapLength := ReturnLength;
end;

function GetIconLength(ResOfs, ResLen: LongInt): LongInt;
var
  LenRead: Word;
  ReturnLength: LongInt;
  A: TBitmapInfoHeader;
begin
  ReturnLength := ResLen;
  if ResLen >= 4 then begin
    {$I-}
    BlockRead (F, A, 4, LenRead);
    {$I+}
    if (IOResult = 0) and (LenRead = 4) and (A.biSize = $28) then begin
      Seek(F, ResOfs);
      {$I-}
```

```
      BlockRead (F, A, SizeOf(TBitmapInfoHeader), LenRead);
      {$I+}
      if (IOResult = 0) and
         (LenRead = SizeOf(TBitmapInfoHeader)) then begin
        ReturnLength :=
          AlignSize(A.biWidth * A.biBitCount div 8, 4)
          * A.biHeight div 2 + {XOR} AlignSize(A.biWidth div 8, 4)
          * A.biHeight div 2 + {AND} A.biSize + 1 shl A.biBitCount * 4;
      end;

      CheckResLength(ResLen, ReturnLength);
    end
    else Write('I');
  end; {if}
  GetIconLength := ReturnLength;
end;

function GetGroupIconLength(ResOfs, ResLen: LongInt): LongInt;
type
  IconDir = record
    idReserved: Word;
    idType: Word;
    idCount: Word;
    {idEntries: array [0..0] of IconDirEntry;}
  end;

  IconDirEntry = record
    bWidth,
    bHeight,
    bColorCount,
    bReserved: Byte;
    wPlanes,
    wBitCount: Word;
    dwBytesInRes: LongInt;
    wIdx: word;
    {dwImageOffset: DWord;}
  end;

var
  A: IconDir;
  ReadLen: Word;
  ReturnLength: LongInt;
begin
  ReturnLength := ResLen;
  if ResOfs > SizeOf(IconDir) then begin
```

```
    {$I-}

    BlockRead(F, A, SizeOf(IconDir), ReadLen);
    {$I+}
    if ((IOResult = 0) and (ReadLen = SizeOf(IconDir))) then
      ReturnLength := SizeOf(IconDir) + A.idCount * SizeOf(IconDirEntry);
  end;
  CheckResLength(ResLen, ReturnLength);
  GetGroupIconLength := ReturnLength;
end;

function GetCursorLength(ResOfs, ResLen: LongInt): LongInt;
var
  LenRead: Word;
  ReturnLength: LongInt;
  A: TBitmapInfoHeader;
begin
  ReturnLength := ResLen;
  if ResLen >= 8 then begin
    {$I-}
    Seek(F, ResOfs + 4);
    BlockRead (F, A, 4, LenRead);
    {$I+}
    if (IOResult = 0) and (LenRead = 4) and (A.biSize = $28) then begin
      Seek(F, ResOfs + 4);
      {$I-}
      BlockRead (F, A, SizeOf(TBitmapInfoHeader), LenRead);
      {$I+}
      if (IOResult = 0) and
        (LenRead = SizeOf(TBitmapInfoHeader)) then begin
        ReturnLength :=
          AlignSize(A.biWidth * A.biBitCount div 8, 4)
          * A.biHeight div 2 + {XOR}
          AlignSize(A.biWidth div 8, 4) * A.biHeight div 2 + {AND}
          A.biSize + 1 shl A.biBitCount * 4 + 4;
      end;
      CheckResLength(ResLen, ReturnLength);
    end
    else Write('C');
  end; {if}
  GetCursorLength := ReturnLength;
end;

function GetGroupCursorLength(ResOfs, ResLen: LongInt): LongInt;
```

```
type
  CursorDir = record
    cdReserved: Word;
    cdType: Word;
    cdCount: Word;
    { cdEntries: array [0..0] of CursorDirEntry;}
  end;

  CursorDirEntry = record
    bWidth,
    bHeight,
    bColorCount,
    bReserved: Byte;
    wXHotspot,
    wYHotspot: Word;
    dwBytesInRes,
    dwImageOffset: DWord;
  end;

var
  A: CursorDir;
  ReadLen: Word;
  ReturnLength: LongInt;
begin
  ReturnLength := ResLen;
  if ResOfs > SizeOf(CursorDir) then begin
    {$I-}
    BlockRead(F, A, SizeOf(CursorDir), ReadLen);
    {$I+}
    if ((IOResult = 0) and (ReadLen = SizeOf(CursorDir))) then
      ReturnLength := SizeOf(CursorDir) +
        A.cdCount * SizeOf(CursorDirEntry);
  end;
  CheckResLength(ResLen, ReturnLength);
  GetGroupCursorLength := ReturnLength;
end;

function GetMenuLength(ResOfs, ResLen: LongInt): LongInt;
const
  mf_Grayed          = $0001;
  mf_Disabled        = $0002;
  mf_Checked         = $0008;
  mf_Popup           = $0010;
  mf_MenuBarBreak    = $0020;
  mf_MenuBreak       = $0040;
```

```
    mf.End              = $0080;
  var
    PA: PBuf;
    CurrP, wVersion, ReadLen: Word;
    ReturnLength: LongInt;
    procedure SkipSubMenu;
    var
      fItemFlags: Word;
    begin
      if CurrP > ResLen then Exit;
      Repeat
        fItemFlags := PWord(@PA^.P^[CurrP])^;
        if (fItemFlags and mf.Popup <> 0) then
          Inc(CurrP, 2)
        else Inc(CurrP, 4);
        while (PA^.P^[CurrP] <> 0) do Inc(CurrP);   { PChar }
        Inc(CurrP);
        if (fItemFlags and mf.Popup <> 0) then
          SkipSubMenu;
      Until (fItemFlags and mf.End <> 0);
    end;
  begin
    ReturnLength := ResLen;
    NewPBufEx(PA, ResLen, nil);
    {$I-} BlockRead(F, PA^.P^, PA^.Len, ReadLen); {$I+}
    if (IOresult = 0) and (ReadLen = ResLen) then begin
      wVersion := PWord(PA^.P)^;
      if (wVersion = 0) then begin
        CurrP := 4;
        SkipSubMenu;
        ReturnLength := CurrP;
        CheckResLength(ResLen, ReturnLength);
      end;
    end;
    DisposePBuf(PA);
    GetMenuLength := ReturnLength;
  end;
  function GetDialogLength(ResOfs, ResLen: LongInt): LongInt;
  const
    DS_SETFONT = $00000040;
  type
    PDialogBoxHeader = ^TDialogBoxHeader;
```

```
  TDialogBoxHeader = record
    lStyle: LongInt;
    bNumberOfItems: Byte;
    X, Y, CX, CY: Word;
  end;

  PControlData = ^TControlData;
  TControlData = record
    X, Y, CX, CY, wID: Word;
    lStyle: LongInt;
  end;

var
  PA: PBuf;
  PHeader: PDialogBoxHeader;
  PControl: PControlData;
  CurrP, ReadLen, I: Word;
  ReturnLength: LongInt;

  procedure IncPCharLength;
  begin
    while (PA^.P^[CurrP] <> 0) and (CurrP < PA^.Len) do
      Inc(CurrP);
    Inc(CurrP);
  end;

begin
  ReturnLength := ResLen;
  NewPBufEx(PA, ResLen, nil);
  {$I-} BlockRead(F, PA^.P^, PA^.Len, ReadLen); {$I+}
  if (IOresult = 0) and (ReadLen = ResLen) then begin
    CurrP := SizeOf(TDialogBoxHeader);
    PHeader := Pointer(PA^.P);
    IncPCharLength;                          { szMenuName    }
    IncPCharLength;                          { szClassName   }
    IncPCharLength;                          { szCaption     }
    if (PHeader^.lStyle and DS_SetFont <> 0) then begin
      Inc(CurrP, 2);                         { wPointSize    }
      IncPCharLength;                        { szFaceName    }
    end;
    if (PHeader^.bNumberOfItems <> 0) then
      for I := 1 to PHeader^.bNumberOfItems do begin
        PControl := @PA^.P^[CurrP];
        Inc(CurrP, SizeOf(TControlData));
        if (PA^.P^[CurrP] and $80 = $ff) then
          Inc(CurrP, 3)
```

```
        else if (PA^.P^[CurrP] and $80 <> 0) then
          Inc(CurrP)
        else IncPCharLength;                    { szClass      }
        IncPCharLength;                         { szText       }
        Inc(CurrP);
      end;
    ReturnLength := CurrP;
    CheckResLength(ResLen, ReturnLength);
  end;
  DisposePBuf(PA);
  GetDialogLength := ReturnLength;
end;

function GetStrTabLength(ResOfs, ResLen: LongInt): LongInt;
var
  PA: PBuf;
  I, J, ReadLen: Word;
  ReturnLength: LongInt;
begin
  ReturnLength := ResLen;
  NewPBufEx(PA, ResLen, nil);
  {$I-} BlockRead(F, PA^.P^, PA^.Len, ReadLen); {$I+}
  if (IOresult = 0) and (ReadLen = ResLen) then begin
    I := 0;
    J := 0;
    Repeat
      Inc(I, PA^.P^[I] + 1);
      Inc(J);
    Until (I > ResLen) or (J = 16);
    if (J = 16) then ReturnLength := I;
    CheckResLength(ResLen, ReturnLength);
  end;
  DisposePBuf(PA);
  GetStrTabLength := ReturnLength;
end;

function GetAcceleratorLength(ResOfs, ResLen: LongInt): LongInt;
type
  AccelTableEntry = record
    fFlags: byte;
    wEvent: word;
    wId: word;
  end;
var
```

```
    A: AccelTableEntry;
    ReadLen: Word;
    ReturnLength, I: LongInt;
  begin
    ReturnLength := ResLen;
    I := 0;
    Repeat
      {$I-}
      BlockRead(F, A, SizeOf(AccelTableEntry), ReadLen);
      {$I+}
      Inc(I, SizeOf(AccelTableEntry));
    Until (A.fFlags and $80 <> 0) or
      (IOResult <> 0) or (Readlen <> SizeOf(AccelTableEntry)) or
      (I > ResLen);
    if (A.fFlags and $80 <> 0) then
      ReturnLength := I;
    CheckResLength(ResLen, ReturnLength);
    GetAcceleratorLength := ReturnLength;
  end;

  function GetVerInfoLength(ResOfs, ResLen: LongInt): LongInt;
  var
    ReturnLength: LongInt;
    cbBlock, ReadLen: word;
  begin
    ReturnLength := ResLen;
    if (ResLen > 2) then begin
      {$I-} BlockRead(F, cbBlock, 2, ReadLen); {$I+}
      if (IOResult = 0) and (ReadLen = 2) then
        ReturnLength := cbBlock;
      CheckResLength (ResLen, ReturnLength);
    end;
    GetVerInfoLength := ReturnLength;
  end;

  function GetFontLength(ResOfs, ResLen: LongInt): LongInt;
  begin
    GetFontLength := ResLen;
  end;

  function GetFontDirLength(ResOfs, ResLen: LongInt): LongInt;
  begin
    GetFontDirLength := ResLen;
  end;

var
```

```
  CurrLength: LongInt;
begin  { GetResLength }
  Seek(F, ResOfs);
  CurrLength := ResLen;
  case ResTypeID of
    RT_ICON:          CurrLength := GetIconLength(ResOfs, ResLen);
    RT_GROUP_ICON:    CurrLength := GetGroupIconLength(ResOfs, ResLen);
    RT_CURSOR:        CurrLength := GetCursorLength(ResOfs, ResLen);
    RT_GROUP_CURSOR:  CurrLength := GetGroupCursorLength(ResOfs, ResLen);
    RT_MENU:          CurrLength := GetMenuLength(ResOfs, ResLen);
    RT_DIALOG:        CurrLength := GetDialogLength(ResOfs, ResLen);
    RT_BITMAP:        CurrLength := GetBitmapLength(ResOfs, ResLen);
    RT_FONT:          CurrLength := GetFontLength(ResOfs, ResLen);
    RT_FONTDIR:       CurrLength := GetFontDirLength(ResOfs, ResLen);
    RT_STRING:        CurrLength := GetStrTabLength(ResOfs, ResLen);
    RT_ACCELERATOR:   CurrLength := GetAcceleratorLength(ResOfs, ResLen);
    RT_VERINFO:       CurrLength := GetVerInfoLength(ResOfs, ResLen);
  end;  { case }
  GetResLength := CurrLength;
end;   { GetResLength }

{ TNeHeader.Display ... ------------------------------------------ }
procedure TNeHeader.SetInfo;

  procedure InsertNode(var Root: PBlkInfo;Pbi: PBlkInfo);
  begin
    if Root = nil then
    begin
      New(Root);
      Root^ := Pbi^;
      PCurrInfo := Root;
    end
    else begin
      if (Pbi^.newofs < Root^.newofs) then
        InsertNode(Root^.Left, Pbi)
      else if (Pbi^.newofs = Root^.newofs) then begin
        if (Pbi^.Flags < Root^.Flags) or
          ((Pbi^.Flags = Root^.Flags) and (Pbi^.Idx < Root^.Idx)) then
          InsertNode(Root^.Left, Pbi)
        else InsertNode(Root^.Right, Pbi)
      end
      else InsertNode(Root^.Right, Pbi);
    end;
```

```
    end;  { InsertNode }
  var
    P: PBlkInfo;
    I: LongInt;
    A: array [0..80] of char;
  begin
    New(P);
    StrPCopy(A, myOwner);
    with P^ do begin
      flags  := myType;
      Idx    := myIndex;
      ofs    := myOfs;
      len    := myLen;
      Owner  := StrNew(A);
      Left   := nil;
      Right  := nil;
      newofs := ofs;
      newlen := len;
      OtherFlag := 0;
      if GangLoad and
         (ofs >= ShiftSize(neHeader.ofsGangLoad)) and (ofs + Len <=
          ShiftSize(neHeader.ofsGangLoad + neHeader.lenGangLoad)) then
        OtherFlag := OtherFlag or GangLoadFlag;
      if (flags in [dseg, cseg, reloc]) and (Len > 4) then begin
        if flags = reloc then I := Ofs + 2 else I := Ofs;
        if IsPackedSeg(F, I, Len, NewLen) then
          OtherFlag := OtherFlag or PackedFlag;
      end;
    end;
    InsertNode(Root, P);
  end;

  procedure TNeHeader.BuildInfoTree;
    function SegAttrName (Attr: Word): String40;
    var
      S: String40;
    begin
      S := '';
      if Attr and Seg_DATA <> 0 then
        S := S + 'data '
      else S := S + 'code ';
      if Attr and Seg_Movable <> 0 then   S := S + 'movable ';
      if Attr and Seg_Shareable <> 0 then S := S + 'sharable ';
```

```
    if Attr and Seg_Preload <> 0 then   S := S+'preload';
    if Attr and Seg_Readonly <> 0 then
    if Attr and seg_data <> 0 then
      S := S + 'readonly'
    else S := S + 'exeonly';
    if fPower then begin
      if Attr and $0008 <> 0 then S := S+'iterated';
      if Attr and Seg_RelocInfo <> 0 then S := S+'reloc';
      if Attr and $0200 <> 0 then S := S + 'conform';
      if Attr and Seg_Discard <> 0 then S := S + 'discardable';
    end;
    SegAttrName := S;
  end;  { SegAttrName }
var
  S, SS: String;
  P: PSegmentEntry;
  A: PTypeInfo;
  B: PNameInfo;
  CurrResLen: LongInt;
  I, J, K, RecNum, ResPageSize: Word;
begin  { BuildInfoTree }
  with neHeader do begin
    SetInfo(Stub,0,
      0, ofsNe, 'WINSTUB');
    SetInfo(sys, sysAll,
      ofsNe, (ofsNonresidentNameTab + numNonresidentNameTab-ofsNe),
      'New Executable file header');
    SetInfo(sys, sysNe,
      ofsNe, $40,
      '  NE header record');
    SetInfo(sys, sysSeg,
      ofsNe + ofsSegmentTab, lenSegmentTab,
      '  Segment table');
    SetInfo(sys, sysRes,
      ofsNe + ofsResourceTab, lenResourceTab,
      '  Resource table');
    SetInfo(sys, sysResName, ofsNe + ofsResidentNameTab,
      lenResidentNameTab,
      '  Resident name table');
    SetInfo(sys, sysModRef,
      ofsNe + ofsModuleReferenceTab, lenModuleReferenceTab,
      '  Module reference table');
```

```
SetInfo(sys, sysImpName,
  ofsNe + ofsImportedNameTab, lenImportedNameTab,
  ' Imported names table');
SetInfo(sys, sysEntry,
  ofsNe + ofsEntryTab, lenEntryTab,
  ' Entry table');
SetInfo(sys, sysNonres,
  ofsNonresidentNameTab, numNonresidentNameTab,
  ' Non-resident names table');

if (Pseg <> nil) then begin
  for I := 1 to numSegmentTab do begin
    P := GetSegmentEntry(PSeg, I);
    S := 'seg' + strNum(I) + ' ' + SegAttrName(P^.Attr);
    if P^.ofs <> 0 then begin
      if P^.Attr and Seg_DATA <> 0 then
        SetInfo(dSeg, I, ShiftSize(P^.ofs), P^.lenSeg, S)
      else SetInfo(cSeg, I, ShiftSize(P^.ofs), P^.lenSeg, S);
      if P^.Attr and Seg_RelocInfo <> 0 then begin
        Seek(F, ShiftSize(P^.ofs) + P^.lenSeg);
        BlockRead(F, RecNum, 2);
        if RecNum <> 0 then begin
          S := 'seg' + strNum(I) + ' Relocation: ' + strNum(RecNum) + ' ';
          if RecNum > 1 then
            S := S + 'entries'
          else S := S + 'entry';
          SetInfo(reloc, I,
            ShiftSize(P^.ofs) + P^.lenSeg,
            RecNum * sizeof(tagRelocItem) + 2, S);
        end;   { RecNum <> 0 }
      end;   { show relocinfo }
    end;
  end;

  if (PResource <> nil) then begin
    ResPageSize := 1 shl PWord(@PResource^[0])^;
    K := 0;
    I := 2;
    while (PWord(@PResource^[I])^ <> 0) do begin
      A := PTypeInfo(@PResource^[I]);
      SS := getRtTypeIDName(A^.rtTypeID);
      Inc(I, 8);
      if (A^.rtResourceCount > 0) then
        for J := 1 to A^.rtResourceCount do begin
```

```
                S := '  ' + SS;
                B := PNameInfo(@PResource^[I]);
                if B^.rnID >= $8000 then
                  S := S + '.' + strNum(B^.rnID and $7fff) + '  '
                else S := S + '.' + CopyStr(PResource^[B^.rnID]) + ' ';
                if (B^.rnFlags and $0010 <> 0) then S := S + 'moveable ';
                if (B^.rnFlags and $0020 <> 0) then S := S + 'Shareable ';
                if (B^.rnFlags and $0040 <> 0) then S := S + 'Preload';
                Inc(K);
                if fPackRes then
                  CurrResLen := GetResLength(S, A^.rtTypeID,
                    LongInt(B^.rnOffset) * ResPageSize,
                    LongInt(B^.rnLength) * ResPageSize)
                else CurrResLen := LongInt(B^.rnLength) * ResPageSize;
                SetInfo(Res,
                  K,
                  LongInt(B^.rnOffset) * ResPageSize,
                  CurrResLen,
                  S);
                Inc(I, sizeof(tagNameInfo));
              end; { for }
            end;  { while }
          end;  { PResource <> nil }
          SetInfo(TheEnd, 0, FSize, 0, 'EOF');
        end;
      end;
    end;  { BuildInfoTree }

    procedure TNeHeader.DisposeInfoTree;
      procedure DisposeTree(Root: PblkInfo);
      begin
        if (Root <> nil) then begin
          if Root^.Left <> nil then DisposeTree(Root^.Left);
          if Root^.Right <> nil then DisposeTree(Root^.Right);
          StrDispose(Root^.Owner);
          Dispose(Root);
        end;
      end;
    begin  { DisposeInfoTree }
      DisposeTree(Root);
    end;  { DisposeInfoTree }

    procedure TNeHeader.BrowseInfoTree(NRoot: PblkInfo;
      DoNode: ProcDo);
```

```
  procedure Trace(P: PblkInfo);
  begin
    if P <> nil then begin
      Trace(P^.Left);
      DoNode(@Self, P);
      Trace(P^.Right);
    end;
  end;  { Trace }
begin
  Trace(NRoot);
end;
function TNeHeader.ReadAseg (segNo: word): PBuf;
var
  A: PBuf;
  FPos: LongInt;
  PA: PSegmentEntry;
begin
  NewPBuf(A);
  PA := GetSegmentEntry(PSeg, SegNo);
  if (SegNo <= neHeader.numSegmentTab) and
     (PA <> nil) and (PA^.ofs <> 0) and (PA^.lenSeg <> 0) then begin
    FPos := ShiftSize(PA^.ofs);
    if (FPos + PA^.LenSeg <= FSize) then begin
      A^.Len := PA^.LenSeg;
      GetMem(A^.P, A^.Len);
      Seek(F, FPos);
      BlockRead(F, A^.P^, A^.Len);
      Unpack (A, A^.Len);
    end;
  end;
  ReadAseg := A;
end;
function TNeHeader.ExistRelocItem (segNo: word): Boolean;
var
  PA: PSegmentEntry;
begin
  PA := GetSegmentEntry(PSeg, SegNo);
  ExistRelocItem := (PA^.Attr and $100 <> 0);
end;
function TNeHeader.ReadARelocList (segNo: word): PBuf;
var
  B: PBuf;
```

```
  PP: PBytes;
  PA: PSegmentEntry;
  ReadLen: Word;
  MemSize, RelocListSize: LongInt;
  FPos: LongInt;
begin
  NewPBuf(B);
  PA := GetSegmentEntry(PSeg, SegNo);
  if (SegNo <= neHeader.numSegmentTab) and
    (PA <> nil) and (PA^.ofs <> 0) and (PA^.lenSeg <> 0) then begin
    FPos := ShiftSize(PA^.ofs) + PA^.lenSeg;
    if (FPos <= FSize - 2) then begin
      Seek(F, FPos);
      BlockRead(F, readLen, 2);
      if readLen <> 0 then begin
        MemSize := readLen * SizeOf(tagRelocItem);
        IsPackedSeg(F, FPos + 2, MemSize, RelocListSize);
        Seek(F, FPos + 2);
        GetMem(PP, memSize);
        BlockRead(F, PP^, memSize);
        if ((readLen = 1) and (PWord(@PP^[0])^ = 0)) then
          FreeMem(PP, memSize)
        else begin
          B^.len := memSize;
          B^.P := PP;
          Unpack(B, RelocListSize);
        end;
      end; { reloc item number <> 0 }
    end; { filesize }
  end;
  readARelocList := B;
end;

function TNeHeader.GetRelocItemCount (segNo: word): word;
var
  FPos: LongInt;
  RelocNum: Word;
  PA: PSegmentEntry;
begin
  RelocNum := 0;
  PA := GetSegmentEntry(PSeg, SegNo);
  if (SegNo <= neHeader.numSegmentTab) and
    (PA <> nil) and (PA^.ofs <> 0) and (PA^.lenSeg <> 0) then begin
```

```
    FPos := ShiftSize(PA^.ofs) + PA^.lenSeg;
    if (FPos <= FSize - 2) then begin
      Seek(F, FPos);
      BlockRead(F, RelocNum, 2);
    end; { filesize }
  end;
  GetRelocItemCount := RelocNum;
end;
procedure TNeHeader.BrowseAsegRelocList (Body: PBuf; segNo: word;
  DoProc: ProcDoRelocItem);
const
  SelfGet: Boolean = false;
var
  A: PBuf;
  PB: PRelocItem;
  I, numRelocItem: Word;
begin
  if Body = nil then begin
    SelfGet := True;
    Body := readARelocList(segNo);
  end;
  if Body^.len <> 0 then begin
    if fPower then A := readAseg(segNo) else NewPBuf(A);
    numRelocItem := Body^.len div sizeOf(tagRelocItem);
    for I := 0 to numRelocItem - 1 do begin
      PB := PRelocItem(@(Body^.P)^[I * sizeOf(tagRelocItem)]);
      DoProc(@self, PB, A^.P);
    end;
    if fPower then DisposePBuf(A);
  end;
  if SelfGet then DisposePBuf(Body);
end;
procedure TNeHeader.BrowseSegmentTab;
var
  I: Word;
  P: PSegmentEntry;
begin
  if (Body <> nil) and (Body^.len <> 0) then begin
    for I := 1 to neHeader.numSegmentTab do begin
      P := PSegmentEntry(@Body^.P^[(I - 1) * sizeof(tagSegmentEntry)]);
      Operation (@self, P, I);
    end; { for }
```

```
  end; { if }
end;

function TNeHeader.GetResPag Size: word;
begin
  if PResource <> nil then
    GetResPageSize := 1 shl PWord(@PResource^[0])^
  else GetResPageSize := 1;
end;

procedure TNeHeader.BrowseResourceTab;
var
  I, J, K: Word;
  A: PTypeInfo;
  B: PNameInfo;
  C: tagEachRes;
begin
  if (Body^.len <> 0) then begin
    I := 2;
    K := 0;
    BrowseStop := False;
    while (PWord(@(Body^.P)^[I])^ <> 0) do begin
      A := PTypeInfo(@(Body^.P)^[I]);
      Inc(I, 8);
      if (A^.rtResourceCount > 0) then
        for J := 1 to A^.rtResourceCount do begin
          B := PNameInfo(@(Body^.P)^[I]);
          Inc(K);
          C.TypeInfo := A;
          C.NameInfo := B;
          C.Idx := K;
          DoRes (@Self, @C);
          Inc(I, sizeof(tagNameInfo));
          if BrowseStop then Exit;
        end;   { for }
    end;   { while }
  end;   { if }
end;

procedure TNeHeader.BrowseEntryTab (Body: PBuf; DoEntry: ProcDo);
var
  Entry: tagAEntry;
  A: PEntryBundle;
  B: PMovableEntry;
  C: PFixedEntry;
```

```
  I, J, K: Word;
begin
  if Body^.len <> 0 then begin
    I := 0;
    K := 0;
    while ((Body^.P)^[I] <> 0) and (I < neHeader.lenEntryTab) do begin
      A := PEntryBundle(@(Body^.P)^[I]);
      Entry.Bundle := A;
      Entry.Idx := 0;
      DoEntry(@Self, @Entry);
      Inc(I, 2);
      if (A^.flags = $ff) then begin
        for J := 1 to A^.Count do begin
          Inc(K);
          B := PMovableEntry(@(Body^.P)^[I]);
          Inc(I, sizeof(tagMovableEntry));
          Entry.MEntry := B;
          Entry.Idx := K;
          DoEntry(@Self, @Entry);
        end  { for }
      end
      else if A^.Flags = 0 then
        Inc(K, A^.Count)
      else begin
        for J := 1 to A^.Count do begin
          Inc(K);
          C := PFixedEntry(@(Body^.P)^[I]);
          Inc(I, sizeOf(tagFixedEntry));
          Entry.FEntry := C;
          Entry.Idx := K;
          DoEntry(@Self, @Entry);
        end;  { for }
      end; { if }
    end; { while }
  end; { if }
end;

procedure TNeHeader.GetWinStub (StubName: PathStr);
var
  MyMz: tagMzHeader;
  FStub: File;
begin
  assign(FStub, StubName);
```

```
    rewrite(FStub, 1);
    MyMZ := mzHeader;
    CopyFileBlockEx(F, 0, FStub, 0, OfsNe);
    Seek(FStub, 0);
    SetImageSize(MyMz, MyMz.ofsWinHeader);
    MyMZ.ofsWinHeader := 0;
    MyMZ.noOverlay    := 0;
    BlockWrite(FStub, MyMZ, $40);
    close(FStub);
  end;
{-------------------------------------------------------------}

end.
{ The End! }
```

3.3 开发 MSDUMP——一个类似 EXEHDR 的工具

本节演示一个文件分析工具 MSDUMP 的开发过程。为了便于读者阅读开发 MSDUMP 的源程序，我们先给出用 MSDUMP 分析一个 Windows EXE 时的显示输出。

用 MSDUMP 分析 \ windows \ calc.exe 时得到如下的结果：

```
MSDUMP — Microsoft Windows EXE File Header Utility  Version 3.10
Copyright (C) Yellow Rose Software Co. 1993.  All rights reserved.

Module:                    Calc
Description:          Microsoft Calculator: Developed by Kraig Brockschmidt
Data:                      UNSHARED
Initial CS:IP:             seg   4 offset 0013
Initial SS:SP:             seg   5 offset 0000
Extra stack allocation:    1800 bytes
DGROUP:                    seg   5
Heap allocation:           0800 bytes
Application type:          WINDOWAPI
Other module flags:        Contains gangload area; start: 0x860; size 0x76e0

no. type address  file  mem   flags
  1 CODE 00000880 024f4 024f4 PRELOAD, (movable), (discardable)
  2 CODE 00007f40 00c82 00c82 (movable), (discardable)
  3 CODE 00008d70 00fbc 00fbd (movable), (discardable)
  4 CODE 00003100 02ed6 02ed6 PRELOAD, (movable), (discardable)
  5 DATA 00006c40 01104 01104 PRELOAD, (movable)
```

```
Exports:
ord seg offset name
  1   1  0526  CALCWNDPROC exported
  2   4  2da0  _ _ _EXPORTEDSTUB exported
  3   2  0404  STATBOXPROC exported
  4   3  0f40  _SIGNALHANDLER exported
```

MSDUMP 根据 NE 文件头的信息，将一些主要的 Windows 执行信息显示出来。其中包括段表信息和输出函数名等。读者在阅读下面的程序时，请留意 MSDUMP 是如何取得这些细枝末节的信息的。有不清楚的地方请参看第 1 章的说明。

```
Uses FileDef, ExtTools, Dos;

{ New executable file header object ------------------------------ }
type
  PDumpNE = ^TDumpNE;
  TDumpNE = object (TNeHeader)
{ 显示 NE 文件信息 }
    procedure DisplayNewExeMessage;
{ 显示段表信息，取代 TNeHeader.DisplaySegmentTab }
    procedure DisplaySegmentTab;
{ 显示输出函数名 }
    procedure DisplayExportedFuncs;
{ 续承 Display 虚方法 }
    procedure Display; virtual;
  end;

procedure TDumpNE.DisplayNewExeMessage;
begin
  with neHeader do begin
    Writeln('Module:', '':19, ModName);
    Writeln('Description: ', '':13, ModDescription);
    Write('Data:', '':21);
    if (Flags and 1) <> 0 then Writeln('SHARED');
    if (Flags and 2) <> 0 then Writeln('UNSHARED');
    Writeln('Initial CS:IP:', '':12, 'seg', _cs:4,
            ' offset ', DownCaseStr(whStr(_ip)));
    Writeln('Initial SS:SP:', '':12, 'seg', _ss:4,
            ' offset ', DownCaseStr(whStr(_sp)));
    Writeln('Extra stack allocation:  ',
            DownCaseStr(whstr(StackSize)), ' bytes');
    Writeln('DGROUP:','':19, 'seg  ', AutoDataSeg);
    Writeln('Heap allocation:','':10,
             DownCaseStr(whstr(heapSize)), ' bytes');
```

```
    Write('Application type:          ');
    case ((Flags and $300) shr 8) of
      0: Writeln('Unknown');
      1: Writeln('NOTWINDOWCOMPAT');
      2: Writeln('WINDOWCOMPAT');
      3: Writeln('WINDOWAPI');
    end;
    Write('Other module flags:        ');
    if ExtFlags and 1 <> 0 then begin
      Writeln('Support for EAs and long file names');
      Write('':26);
    end;
    if GangLoad then
      Writeln('Contains gangload area;start: 0x',
        DownCaseStr(lhNzStr(LongInt(ofsGangLoad) * GetPageSize)),
        '; size 0x',
        DownCaseStr(lhNzStr(LongInt(lenGangLoad) * GetPageSize)));
    Writeln;
  end;
end;

{ DisplaySegmentTab.ShowSegmentTab }
procedure ShowSegmentTab (PSelf: PNeHeader; P: Pointer; Index: word);far;
var
  S: String80;
begin
  with PSelf^, PSegmentEntry(P)^ do begin
    Write(Index:3,' ');
    if Attr and Seg_DATA <> 0 then
      Write('DATA')
    else Write('CODE');
    Write(' ',DownCaseStr(lhStr(LongInt(ofs) * GetPageSize)));
    Write(' 0', DownCaseStr(whStr(lenSeg)));
    Write(' 0', DownCaseStr(whStr(lenMem)),' ');
    S := '';
    if Attr and Seg_Preload <> 0 then
      S := 'PRELOAD';
    if Attr and Seg_Movable <> 0 then begin
      if S <> '' then S := S + ', ';
      S := S + '(movable)';
    end;
    if Attr and Seg_Discard <> 0 then begin
      if S <> '' then S := S + ', ';
```

```
      S := S + '(discardable)';
    end;
    Writeln(S);
  end;  { with }
end;

procedure TDumpNE.DisplaySegmentTab;
var
  I: Word;
  P: PSegmentEntry;
  B: TBuf;
begin
  With neHeader do
    if (Pseg <> nil) then begin
      Writeln('no. type address  file  mem  flags');
      B.len := lenSegmentTab;
      B.P := PSeg;
      BrowseSegmentTab (@B, ShowSegmentTab);
      Writeln;
    end; { if }
end;

procedure TDumpNE.DisplayExportedFuncs;
var
  P: PAEntry;
  I, J: Word;
  FuncName: String;
begin
  J := GetEntryCount;
  if J > 0 then begin
    Writeln;
    Writeln('Exports:');
    Writeln('ord seg offset name');

    for I := 1 to J do begin
      FuncName := GetEntryName(I);
      if FuncName <> '' then begin
        Write(I:3);
        P := GetAEntry(I);
        if P^.Bundle^.flags = $ff then
          Write((P^.MEntry^.SegNum):4,
                '  ',DownCaseStr(whStr(P^.MEntry^.Ofs)),'  ')
        else Write((P^.Bundle^.Flags):4,
                   '  ',DownCaseStr(whStr(P^.FEntry^.Ofs)),'  ');
        Dispose(P);
```

```
        Writeln(FuncName, ' exported');
      end;
    end;
  end;
end;

procedure TDumpNE.Display;
begin
  DisplayNewExeMessage;
  DisplaySegmentTab;
  DisplayExportedFuncs;
end;

{ File Struct Object ------------------------------------------------ }
{ 文件操作对象 }
type
  PDumpFile = ^TDumpFile;
  TDumpFile = Object
    P: PFile;
    constructor Init(FName: PathStr);
    procedure Run;
    destructor Done; virtual;
  end;

{ TDumpFile Object }
constructor TDumpFile.Init (FName: PathStr);
begin
  if Pos('.', FName) = 0 then FName := FName + '.EXE';
  if (FileExists(FName)) and (GetFileSign(FName) = WinAppSign) then
  begin
    P := New(PDumpNE, Init(FName));
    if rtError <> 0 then fail;
  end
  else begin
    rtError := $EE00;
    fail;
  end;
end;

procedure TDumpFile.Run;
begin
  P^.Display;
end;

destructor TDumpFile.Done;
```

```
begin
  Dispose(P, Done)
end;
{ ---------------------------------------------------------------- }
var
  I: Word;
  FName, S: String;
  PDump: PDumpFile;
begin
  FName := '';
{ 命令行参数分析 }
  if ParamCount > 0 then begin
    I := 0;
    repeat
      Inc(I);
      S := ParamStr(I);
    until (not (S[1] in OptionChars)) or (I = ParamCount);
    if not (S[I] in OptionChars) then FName := S;
    for I := 1 to ParamCount do begin
      S := UpcaseStr(ParamStr(I));
      if S[1] in OptionChars then begin
        Delete(S, 1, 1);
        if S = 'P' then
          fPower := True
      end;
    end;
  end;
  Writeln;
  Writeln('MSDUMP -- Microsoft Windows EXE File Header Utility  Version 3.10');
  Writeln('Copyright (C) Yellow Rose Software Co. 1993.  All rights reserved.');
  if FName = '' then begin
   Writeln;
   Writeln('Usage: MSDUMP <FileName>');
   Halt(1);
  end;
  Writeln;
{ 执行部分 }
  PDump := New(PDumpFile, Init(FName));
  if PDump = nil then
    DisplayRtError
  else begin
    PDump^.Run;
```

```
    Dispose(PDump,Done)
  end;
end.
```

| ———————— The End ———————— |

第 4 章

直接修改 Windows 执行文件

在前面三章中，我们围绕静态的 Windows 执行程序进行了各种分析，本章中我们把论题转到“直接修改 Windows 执行文件”上。为什么会提出这一论题？我想大部分读者都想到了这个问题的答案：“我很想修理这个程序，可惜又没有源程序……”。

“直接修改 Windows 执行文件”是一个很有吸引力的话题，但它涉及的内容太多，最主要的是 Windows 系统的执行机制。在修改 Windows 执行程序时，头脑中随时都要有清晰的执行机制“拓朴图”，稍一糊涂，就会前功尽弃。

可是，Windows 系统的执行机制实在太复杂。在前面的章节中，我们已经接触到了一些与执行机制紧密相关的概念，如 Moveable, Fixed, Discardable, Preload, LoadonCall, Multiple Data, Single Data 等，可能有一部分读者似懂非懂；与其相关的概念还有 Module, Task, Instance, Thunk 等等。这些东西搅合在一起，很多人都会感到迷惑不解。所以在这一章中，我们首先要把执行机制讲述清楚。当然，这绝不是一件简单的事，就连 Microsoft 的出版物对该问题的描述也是令人头晕眼花。经过反复权衡，我们最终选择了的方式是“讲述一个实例”，试图通过一个逐步推进的实例，结合 Heap Walker 的观察结果，让读者直观地理解执行的各个方面。这里选中的实例还是我们在前面章节中多次使用的 Paint Brush。

在 Windows 程序执行以前，还有装载和启动过程，装载过程是一个相对独立的过程，我们把它放在第 6 章说明；本章就从启动过程开始讲解，然后演示 Paint Brush 的执行过程。有了这些知识作基础，我们再去一步一步地修改 Windows 执行程序。

4.1 Windows 执行机制与动态链接

Windows 的执行机制与动态链接紧密相连，在 Windows 下动态链接扮演了十分重要的角色。那么，到底什么是动态链接呢？

动态链接有两个方面上的含义。首先，它是一种内存管理技术。Windows 内存管理系统的作用主要有两个：一是处理应用程序从全局堆或局部堆中分配内存块的请求，一是有效地利用所有处于执行状态的程序的代码段和数据。

在 Windows 系统中，分配的内存可能是可移动的，甚至是可弃的。对于可移动内存对象，Windows 可以根据需要来挪动它们以获得连续的自由内存空间。可弃内存对象不仅可以被 Windows 移动，而且有时为了满足内存分配需要，Windows 可以将其删除。内存对象被删除后，其中的信息会全部丢失，但应用程序随时都可以重载该内存对象的代码数据。

当 Windows 启动一个应用程序时，系统首先在全局堆中分配一段内存空间，然后从可执行文件中把部分代码段和数据段拷贝到内存中。代码段可以是固定的、可移动的或可弃的；数据段包含应用程序的堆栈、局部堆、静态数据和全局数据。数据段可以是固定的或可移动的，但不可被删除，因为删除后的数据无法恢复。代码段和数据段的属性在应用程序的模块定义文件中指定。应用程序的内存对象在需要时才从内存中分配。同样，代码、数据对象都是在需要时装入内存的，也可以在内存不够用时予以搬移或抛弃，这就是动态链接与内存管理的关系。

其次，动态链接提供了不同应用程序间共享数据和代码的机制。在系统中装入多份相同的程序，那么每一个程序都包含这许多相同的代码，就会形成内存的极大浪费！ Windows 系统提供了代码的共享。

Windows 系统还采用了一种动态链接的方案，就是程序需要调用的函数并不是在链接时就和应用程序结合，而是当程序执行时，Windows 先检查内存中有没有所调用函数的代码，如果有就直接调用，如果没有就到磁盘中找出来相应的文件并把它装入内存，再进行链接，这样在内存中就不会出现相同的代码。在以前编写多套界面一致的软件，每套软件中都包含类似的程序，执行时同样的代码在内存中有多份，这样内存和磁盘都形成巨大浪费。这就是为什么要使用动态链接的原因。

大致明白了 Windows 的执行机制与动态链接的关系后，我们再结合具体程序来看一下一般程序的执行过程。一个程序从磁盘文件到在 Windows 下运行要经过以下三个步骤：装载、启动和执行。Windows 程序的装载过程内容庞杂，在本书中我们把装载技术另辟为一个单独的部分，放在第 6 章予以专门讲述。在接下来的两节中，我们集中讨论 Windows NE 文件头和一些相关代码段、资源段装入内存时的情况，也就是 Windows 的启动过程。启动过程分 Windows AP 和 Windows DLL 两种情况讲述，这是因为它们之间明显的不同。比如，Windows AP 的入口和起始函数为 WinMain 函数，该函数的主体为一个消息循环；DLL 的入口和起始函数为 LibMain 函数，它没有消息回路；还有，DLL 不会主动去获得 CPU 服务，没有自己的堆栈。这些原因造成两者的启动过程不同。

一个 Windows 应用程序和动态链接库(DLL)分别对应入口函数 WinMain 和 LibMain，它们有不同的启动要求，即在调用入口函数 WinMain 或 LibMain 之前，分别着对应不同的初始化步骤。这里要说明的启动过程包含应用程序和动态链接库两种情况。

4.2 Windows 应用程序的启动过程

4.2.1 应用程序的启动

启动过程放在哪里？

您应该很熟悉以下 Windows 应用程序的基本结构：

```
int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
```

```
HINSTANCE  hInstance;          /* handle of current instance    */
HINSTANCE  hPrevInstance;      /* handle of previous instance   */
LPSTR  lpCmdLine;              /* address of command line       */
int  nCmdShow;                 /* show-window type (open/icon) */
{
  MSG msg;

  if (! hPrevInstance)
    if (! InitApplication(hInstance))
      return FALSE;

  if (! InitInstance(hinstance, nCmdShow))
    return FALSE;
  while (GetMessage(&msg, NULL, NULL, NULL)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    }

  return(msg.wParam);
}
```

Windows 启动一个应用程序，就是去直接调用应用程序的 WinMain 函数吗？不是！而是调用一个与应用程序一起提供的启动过程。启动过程完成一些必要的前置工作，然后调用 WinMain，并且当 WinMain 返回控制时，退出应用程序。

既然启动过程是与应用程序是一起提供的。那么，启动过程在哪里？它的执行步骤是什么样的？

启动过程就放在应用程序中，它由编译器提供。顾名思义，启动过程最先开始执行。在第 1 章中我们已经介绍过，Windows EXE 的 NE 首部信息块的偏移[14H]处存放有 CS:IP 寄存器的值，它代表了 Windows 程序的入口点，程序就从这里开始执行。比如，PBRUSH.EXE 的 CS:IP 为 0001:0010，CALC.EXE 的 CS:IP 为 0004:0013，即程序分别从它们的 Segment 1 和 Segment 4 开始执行，启动过程正是放在这个段中。如果您用前面介绍的文件分析工具 PDUMP 加上参数/p 分析 Windows 系统目录下的各个 Windows AP(DLL 除外)，可以看到它们都调用了 Windows 系统函数 InitTask、WaitEvent 和 InitApp，这三个函数就是 Windows 系统的启动函数。PBRUSH.EXE 和 CALC.EXE 的 InitTask、WaitEvent 和 InitApp 这三个函数分别在 Segment 1 和 Segment 4 中。另外启动过程还调用了 MS-DOS 的中止程序功能 INT 21H 功能 4CH，用于当 WinMain 返回控制时，退出应用过程。

启动步骤描述

那么，由程序起始到调用 WinMain 函数这段期间，程序做了哪些事情呢？启动过程初始化和退出 Windows 应用程序一般遵循什么样的步骤呢？

Step 0.　在 Windows 程序即将进入启动过程(地址由 CS:IP 指定)之前，以下的 CPU 寄存器被设置为如下的值：

BX	stack size(堆栈的大小)
CX	heap size(Local Heap 的大小)
DI	hInstance(当前的 Instance Handle)
SI	hPrevInstance(前一个 Instance Handle)
ES	PSP

图 4.1 启动前 CPU 寄存器的内容

Step 1. 用 InitTask 函数初始化任务。若调用成功，InitTask 函数会将寄存器的值设置为 WinMain 函数所需要的参数，同时，寄存器被设置为如下的值：

AX	1
CX	stack size(堆栈的大小)
DX	nCmdShow(和传入 WinExec 的 nCmdShow 相同)
ES:BX	lpCmdLine(ES = PSP, BX = 80H)
SI	hPrevInstance
DI	hInstance

图 4.2 执行 InitTask 后 CPU 寄存器的内容

WinMain 函数需要四个参数：hInstance、hPrevInstance、lpCmdLine 和 nCmdShow，调用 InitTask 函数后，这四个参数分别存放在寄存器 DI、SI、EX:BX 和 DX 中。

另外，InitTask 还设置 Instance Data 中的 pStackTop、pStackMin 和 pStackBottom。有关 Instance Data 的说明见后。

Step 2. 用 WaitEvent 函数清除上述 Task 的事件。WaitEvent 首先检查是否有属于该 Task 的事件存在。如果有的话，WaitEvent 清除已存在的事件，并将控制交还给启动程序，而启动程序并不去处理与该 Task 有关的事件；如果没有，则 WaitEvent 会停止程序的执行，转而调用 KERNEL 的调度程序 Scheduler。不管 Windows Scheduler 是将控制权转交给启动过程或是其它的程序，都是达到了清除上述 Task 的事件的目的。如果 Scheduler 最终将控制权交给了其它程序，WaitEvent 返回非 0 值；否则返回 0。

有一点应该指出，由于程序并没有开始运行，这里清除的事件和 Windows 的消息(Message)没有关系，而是指的通知信号(Notify)。这些通知事件记录在 Task Database 中偏移[06H]的地方。有关 Task Database 的说明见后。

Step 3. 用 InitApp 函数建立属于该 Task 的消息队列(Message Queue)。具体来讲，InitApp 会针对刚才被载入的 Windows Task 作一些检查和初始化的操作。这其中包括：载入资源处理过程(调用 KERNEL 的 SetResourceHandler)、安装属于该 Task 的信号处理过程、产生 Task Queue(Task Queue 用于存放属于该 Task 的消息，每一个 Task 对应一个 Task Queue)、将 task 加入到 task list 中等。InitApp 函数需要实例句柄参数 hInstance，该参数已由 InitTask 函数返回到寄存器 DI 中。有一点应该指出：Windows 系统为 Windows AP 自动

初始化了局部堆,所以应用程序的启动过程不用包含局部堆初始化函数。

Step 4. 调用应用程序入口函数 WinMain。WinMain 函数所需的参数已由 InitTask 函数返回。

Step 5. 当 WinMain 返回时,调用 MS－DOS 终止程序功能(中断 21H 功能 4CH)退出应用过程。

Windows 首次调用某个 AP 的启动过程以后,也就是说在进入 WinMain 函数以前,CPU 寄存器的值如下:

寄存器	值
AX	0
BX	以字节为单位,表示栈的大小
CX	以字节为单位,表示堆的大小
DI	标识新应用程序实例的句柄,即 hInstance
SI	标识前一个应用程序实例的句柄,即 hPrevInstance
BP	0
ES	程序段前缀(PSP)的段地址
DS	应用程序自动数据段的地址
SS	和 DS 一样
SP	应用程序堆栈第一字节的偏移量

图 4.3　启动完毕后 CPU 寄存器的内容

应用程序启动过程示范

启动过程的作用是初始化和退出 Windows 应用程序。对于适用于 Windows 的某种语言编译器所编译的程序,启动过程还包括该语言的初始化过程。比如说,一个 C 的编译器,其起始码程序必须将 C 的 run-time library 加以初始化;C＋＋的起始码必须调用所有静态(static)的建构函数(constructor)。在 Borland C＋＋ 4.0 中,该程序的路径是:

```
\BC4\LIB\STARTUP\COW.ASM
```

COW.ASM 不仅有 Windows 的启动过程,还有 C＋＋的起始码。

下面给出的启动过程并不通用,因为它不包含某种具体语言的初始化过程。读者只需从执行机制的全局来了解启动的大致步骤即可,并没有深入钻研的必要,因为实际中很少需要编写启动过程 — 编译器已经帮我们准备好了! 有关用汇编语言编写 Windows 程序的技术问题,请参阅本书第 5 章。

```
.xlist
mems = 1        ;small memory model.
? DF = 1        ;Do not generate default segment definitions.
? PLM = 1
? WIN = 1
```

```
include cmacros.inc
.list

STACKSLOP = 256

createseg    -TEXT, CODE, PARA, PUBLIC, CODE
createseg    NULL,  NULL, PARA,  PUBLIC,  BEGDATA, DGROUP
createseg    -DATA, DATA, PARA,  PUBLIC,  DATA,    DGROUP
defGrp       DGROUP, DATA

assumes DS, DATA

sBegin       NULL                ;NULL段
             DD   0
labelw       <PUBLIC, rsrvptrs>
maxRsrvPtrs = 5
             DW   maxRsrvPtrs
             DW   maxRsrvPtrs  DUP(0)
sEnd         NULL

sBegin   DATA
staticw  hPrev, 0            ;save WinMain parameters.
staticw  hInstance, 0
staticD  lpszCmdline, 0
staticw  cmdShow, 0
sEnd     DATA

externFP   <INITTASK>
externFP   <WAITEVENT>
externFP   <INITAPP>
externFP   <DOS3CALL>
externP    <WINMAIN>

sBegin   CODE
assumes  CS, CODE

labelNP  <PUBLIC, astart>
xor      bp, bp              ;zero bp
push     bp

cCall    INITTASK            ;Initialize the task.
or       ax, ax
jz       noinit

add      cx, STACKSLOP       ;Add in stack slop space.
jc       noinit              ;If overflow, return error.

mov      hPrev, si
mov      hInstance, di
mov      word ptr lpszCmdlime, bx
```

```
        mov     word ptr lpszCmdlime + 2, es
        mov     cmdShow, dx

        xor     ax, ax                  ;Clear initial event that
        cCall   WAITEVENT, <ax>          ;started this task.
        cCall   INITAPP, <hInstance>    ;Initialize the queue.
        or      ax, ax
        jz      noinit

        cCall   WINMAIN, <hInstance, hPrev, lpszCmdline, cmdShow>
ix:
        mov     ah, 4ch
        cCall   DOS3CALL                ;Exit with return code from app.
noinit:
        mov     al, 0FFh                ;Exit with error code.
        jmp  short ix
sEnd    CODE
        end --astart                    ;start address
```

我们注意到，在上面的程序中有一个 NULL 段，其中定义了一个数组，叫 rsrvptrs。这个数组好象没有被使用过，为什么要定义它呢？原来，InitTask 函数要将堆栈的顶、最小值和基地址的偏移量拷贝到该数组的第 3、4、5 个位置上，所以必须预留相应的空间。其实，这个数组的内容是相当重要的，比如说，应用程序需要使用这些偏移量来检查堆栈中的可用空间；Windows 的调试版本也用这些偏移量来检查堆栈。应用程序绝不可以改变该数组中的内容，否则将出现系统调试错误(RIP)。

4.2.2　应用程序启动函数说明

这里说明前面提到的三个应用程序启动函数：InitApp, InitTask 和 WaitEvent。

InitApp

说明	externFP　InitApp push　　hInstance　　;instance handle call　　InitApp or　　ax, ax　　;zero if error jz　　error - handler InitApp 函数产生应用程序队列，并安装应用程序支持的过程，如信号过程、特定资源装入过程和被零除中断过程。
参数	hInstance 标识要初始化的任务。该参数必须由 Windows 事先提供。
返回值	如果成功，则 AX 非零；否则，AX 为零，表示有错。

InitTask

<table>
<tr><td>说明</td><td colspan="2">externFP InitTask
call InitTask ;Initialize a task
InitTask 函数设置寄存器、取命令行参数和初始化堆栈。它应该是应用程序启动过程调用的第一个函数。</td></tr>
<tr><td>返回值</td><td colspan="2">如果成功，则 AX 为 1，并为新任务设置 CX、DX、EX:BX、SI 和 DI 寄存器；否则，AX 为 0，表示有错。</td></tr>
<tr><td>注释</td><td colspan="2">成功时，其它寄存器的值为：</td></tr>
<tr><td></td><td>寄存器</td><td>值</td></tr>
<tr><td></td><td>CX</td><td>以字节为单位，表示堆栈界限。启动过程应该检查该界限，以保证堆栈中至少有 100 个字节。</td></tr>
<tr><td></td><td>DI</td><td>新任务的实例句柄，即 hinstCurrent；启动过程将它传递给 WinMain 函数。</td></tr>
<tr><td></td><td>DX</td><td>nCmdShow 参数；启动过程将该参数传递给 WinMain 函数。用于 CreateWindow 函数。</td></tr>
<tr><td></td><td>ES</td><td>新任务的程序段前缀(PSP)的段地址。</td></tr>
<tr><td></td><td>EX:BX</td><td>命令行的 32 位地址(MS-DOS 格式)，即 lpszCmdLine；启动过程将它传递给 WinMain 函数。</td></tr>
<tr><td></td><td>SI</td><td>如果有的话，是应用程序前一实例的实例句柄，即 hinstPrevious；启动过程将它传递给 WinMain 函数。</td></tr>
<tr><td></td><td colspan="2">InitTask 函数还将堆栈的顶、最小值和基地址偏移量拷贝到应用程序自动数据段的前 16 字节的保留存储区，保留存储区具有以下格式：

DW 0
globalW oOldSP, 0
globalW hOldSS, 5
globalW pLocalHeap, 0
globalW pAtomTable, 0
globalW pStackTop, 0
globalW pStackMin, 0
globalW pStackBot, 0
pStackTop 和 pStackBot 指定了堆栈的范围，这个范围是固定的；pStackMin 代表堆栈的动态使用情况，它随着程序的执行而不断地变化。自动数据段的内容可用 Heap Walker 察看，在本章第四节我们将看到这 16 字节的保留存储区的具体内容。</td></tr>
</table>

WaitEvent

说明	externFP　WaitEvent push　taskID　　;task identifier call　WaitEvent or　　ax, ax jnz　　resched　　;nonzero if rescheduled
	WaitEvent 函数检查传递过来的事件。如果有事件则清除它，并交由应用程序处理；
	如果没有，则通过调用 Windows 调度程序来挂起应用程序。
参数	taskID 需要检查事件的任务。如果该参数是 0，则函数检查当前任务的事件。
返回值	如果 Windows 调度程序调度了另一个应用程序，则返回一非 0 值；否则返回 0。

4.3　动态链接库的初始化

4.3.1　Windows 应用程序如何使用 DLL

动态链接库就是指到了执行时才与应用程序链接在一起的函数库。Windows 的三大模块：KERNEL、GDI 和 USER 实际上都是动态链接库。比如，在 Microsoft Visual C 中，PBRUSH 的链接过程可能是这样的：

```
link /NOD pbrush,,, libw.lib mlibcew.lib, pbrush.def
```

其中，mlibcew.lib 是静态的 C 函数库。libw.lib 既不是一个静态链接库，也不是 DLL，Windows 把它称作 import 函数库，这个 import 函数库里并没有任何程序码，它里面包含的是将来与 Windows API 三大模块“动态链接”所需要的信息，这些信息包括函数的名称、所属的模块及在模块中的序号，即“某个函数是在某个 DLL 的某个位置”。每一个 DLL 可以有一个 import 函数库与之对应，但不一定要有，也不一定是一个 DLL 只能有一个对应的 import 函数库。需要记住的是：import 函数库中并无真正的代码，真正的代码是在 DLL 中。

由以上对函数库类型的辨析可知：在开发 Windows 应用程序的编译链接过程中一般碰到的函数库有三类：object 函数库、DLL 和 import 函数库。

所有在 DLL 中要提供给其它程序调用的 proc 都应该有特别的声明，也就是说在模块定义文件的 EXPORTS 描述中说明；相应地，如果其他可执行程序要用到这些函数时，应声明为 IMPORTS。

比如，当 Windows 将 USER.EXE 载入时，Windows 会把 USER 模块的几百个输出函数的入口点(entry point)构筑为一个 table，放在 USER 的 Module Database 中。如果有某个应用程序要使用 USER 模块的输出函数，那么当它被载入时，Windows 会把该程序中调用 USER 输出函数的地方重定位到相应的入口点。

每一个 DLL 都包含启动码和初始化函数。Windows 在某一个 DLL Module 被首次装

载时调用一次该函数，其后如果有应用程序要使用该 DLL 时，Windows 不再调用初始化函数，而只是增加 DLL 的使用计数。DLL 一直被存放在内存中，直到它的引用计数小于等于零。如果引用计数减至零，DLL 从内存中删除。当应用程序重载该 DLL 时，Windows 再调用初始化函数。这就是 Windows 系统使用 DLL 的粗略描述，具体细节请阅读后面的叙述。

4.3.2 DLL 与 Windows 应用程序的区别

DLL 与一般的 Windows AP 相比，区别很多，值得强调的有以下两点：

(1) DLL 不主动请求 CPU 服务，也不接受消息，它是被动执行的，这是 Windows 系统在设计动态链接机制时就决定了的。DLL 的作用是输出(exports)回调函数供其它程序使用，当程序需要时，DLL 中的函数被执行；执行完毕以后，控制交还给应用程序。所以 DLL 绝对不会成为一个 Task，也就不会成为 Windows Scheduler 调度的对象。在 DLL 的启动过程中没有 InitTask 。

(2) DLL 没有堆栈，它使用调用程序提供的堆栈，因为它自己根本用不着堆栈。与此相关的一个变化是：在大多数情况下，当应用程序调用 某个函数时，编译器会设置寄存器为 DS= =SS；但是对于 DLL，因为它没有堆栈，所以 DS! = SS，即数据段是 DLL 自己的数据段，而堆栈段是调用函数的堆栈段，这样，我们在 DLL 中调用这些函数时必须有另外一套处理办法：在 Microsoft Visual C 的库中看到的 sdllcew.lib，mdllcew.lib 等函数库就是针对这种要求而特别设计的 C 函数库。

当 Windows 首次进入一个 DLL 的入口时，也就是说当装载第一个需要使用该 DLL 的程序段时，CPU 寄存器被设置成如下的值：

寄存器	值
DI	指示库实例句柄
DS	如果有，则标识库的数据段
CX	堆大小。在库的.DEF 文件中指定
ES:SI	指向命令行(LoadModule 函数的 lpvParameterBlock 参数的 lpCmdLine 成员)

图 4.4 进入 DLL 前 CPU 寄存器的内容

有兴趣的读者将这里的表与图 4.1 相比较，看看启动 Windows DLL 和 Windows AP 前寄存器内容的差别。我们可以看到两点：一是 DLL 没有包含栈大小的寄存器，因为 DLL 自身没有堆栈；一是没有寄存器包含 hPrevInstance，因为 DLL 不可能有多个执行实例。

4.3.3 DLL 的制作过程和 DLL 的启动码

为了了解 DLL 的启动过程，我们先来看一下一个 DLL 的制作过程。任何动态链接库都必须有一个进入和退出函数，其入口(进入)函数是 LibMain 。下面是动态链接库入口函数的写法：

```
int FAR PASCAL LibMain(HINSTANCE hinst,
    WORD    wDataSeg,
    WORD    cbHeapSize,
```

```
    LPSTR  lpszCmdLine)
{
  ... ...                    /* 如果在第一次装载该 DLL 时还有其它的初始化工作, */
  ... ...                    /* 可以放在这里完成 */
  if (cbHeapSize ! = 0)      /* if DLL data seg is MOVEABLE */
    unlockData(0);
return 1;  /* successful installation; otherwise, return 0 */
}
```

函数 LibMain 有四个参数：hinst、wDataSeg、cbHeapSize 和 lpszCmdLine。hinst 是 DLL 的实例句柄；wDataSeg 参数是数据段(DS)寄存器的值；cbHeapSize 参数是模块定义文件定义的堆大小，例程 LibEntry 根据该值初始化局部堆；lpszCmdLine 参数包含命令行信息，但在动态链接库中很少使用该参数。

我们在制作动态链接库时，经常在链接阶段用到目标文件 LIBENTRY.OBJ，使用的命令如下：

```
link mindll.obj  libentry.obj, mindll.dll, mindll.map /map,
   mdllcew.lib libw.lib/noe/nod, mindll.def
```

那么，例程 LibEntry 和目标文件 LIBENTRY.OBJ 是用来干什么的呢？

DLL 的启动码是由文件 LIBENTRY.OBJ(或 C0DS.OBJ)提供的，在链接时启动码被加入到 DLL 中。在该 LIBENTRY.OBJ 中提供了例程 LibEntry，当 Windows 加载一个 DLL 时，由例程 LibEntry 调用 LibMain 函数。

LibEntry 的作用是在调用 LibMain 函数之前用函数 LocalInit 初始化 DLL 堆栈，如果 HEAPSIZE 的值已在 DLL 的模块定义文件中指定了的话。用 PDUMP /P 分析 Windows 系统提供的一些动态链接库，我们可以从中找到函数 LocalInit 。LocalInit 的作用是将一个指定段中的局部堆初始化，下面是该函数参数的一个简短说明。

```
BOOL LocalInit(uSegment, uStartAddr, uEndAddr)

UINT uSegment;   /* segment to contain local heap */
UINT uStartAddr; /* starting address for heap    */
UINT uEndAddr;   /* ending address for heap    */
```

其中，uSegment 为动态链接库的数据段，它的前 16 个字节保留，作为系统使用。

4.3.4　DLL 启动过程示范程序

下面是 Microsoft Windows 3.1 SDK 中以汇编语言编写的 libEntry 程序的代码，读者可以参照注释阅读这一个代码片段。有关用汇编语言编写 Windows 程序的问题，请参看下一章。

```
include cmacros.inc
externFP   <LibMain>                    ;the C routine to be called
createSeg  INIT_TEXT, INIT_TEXT, BYTE, PUBLIC, CODE
sBegin     INIT_TEXT
assume     CS, INIT_TEXT
```

```
? PLM = 0                          ;'C'naming
externA   <-acrtused>              ;ensures that Win DLL startup
                                   ;code is linked
? PLM = 1                          ;'PASCAL'naming
externFP  <LocalInit>              ;Windows heap initialization routine

cProc     LibEntry, <PUBLIC, FAR>  ;entry point into DLL

include CONVDLL.INC

cBegin
  push    di                       ;handle of the module instance
  push    ds                       ;library data segment
  push    cx                       ;heap size
  push    es                       ;always NULL; may remove
  push    si                       ;always NULL; may remove

  ; If we have some heap, then initialize it.
  jcxz    callc                    ;Jump if heap specified

  ; Call the Windows function LocalInit to set up the heap
  ; LocalInit((LPSTR)start, WORD cbHeap);

  xor     ax, ax
  cCall   LocalInit<ds, ax, cx>
  or      ax, ax                   ;Did it do it OK ?
  jz      error                    ;Quit if it failed

  ; Invoke the C routine to do any special initialization.

callc:
  call    LibMain                  ;Invoke the 'C' routine(result in AX)
  jmp  short  exit                 ;LibMain is responsible for stack cleanup
error:
  pop     si                       ;Clean up stack on a LocalInit error
  pop     es
  pop     cx
  pop     ds
  pop     di
exit:

cEnd

sEnd      INIT_TEXT

end   LibEntry
```

在上面的 LibEntry.asm 中调用了函数 LocalInit，因为 LocalInit 函数会将 Segment(数据段)锁住，所以在前面的入口函数 LibMain 中调用 UnlockData 函数，将当前数据段恢复到正常状态。

有一点应该提醒读者注意：在 Borland 的系列语言产品中，与 LIBENTRY.OBJ 对应的文件是 C0DS.OBJ，它的功能与前者相同，都要去初始化堆，并调用入口函数 LibMain 。

了解了 LibEntry、LibMain 和 LocalInit 这三者的关系后，我们就知道了一个 DLL 的启动和初始化步骤。

4.4　PBRUSH.EXE 的执行过程

4.4.1　Windows 执行 AP 或 DLL 时维护的数据

一个 Windows AP 中有代码段、资源段和数据段。程序执行时根据需要将它们加载到内存中。根据代码段和数据段的数目，Windows AP 分为小模式、中模式、压缩模式、大模式和巨模式。这几种模式的区别请参看第 5 章 3 节的说明。一般我们常见的程序为小模式和中模式，包括象 WinWord 和 Excel 这类很“庞大”的软件。不管一个 Windows AP 为什么模式，它的数据段中一定只有一个自动数据段。自动数据段的内容请参看本章在后面的解释。

Windows 不仅能同时执行数个不同的 Windows AP，也能同时执行数个相同 AP。每一个相同的程序称为该应用程序的执行实例(Instance)。一个程序的多个执行实例所使用的代码是完全一致的，只是它们的数据段必须明确分开。

Windows 是通过什么样的组织管理来实现上述功能的呢？原来，当 Windows Loader 加载一个应用程序并准备执行时，Windows 核心 KERNEL 为每一个应用程序维护一个 Module Database，为每一个 Instance(即对应的 Task)维护一个 Task Database。由于一个 AP 可以被同时执行多份，所以可能多个 Task 对应一个 Module。下图是一个应用程序执行多份时，Module、Task 及 Segment 之间的关系图。

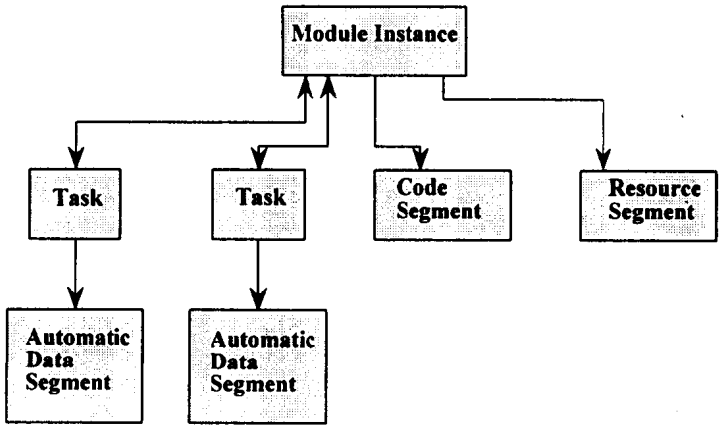

图 4.5　运行 AP 时 Module、Task 及 Segnen 之间的关系图

对于 DLL，由于它没有 Task，而且只存在一个执行实例，所以它的 Module 和 Segment 的关系图变成如下这样：

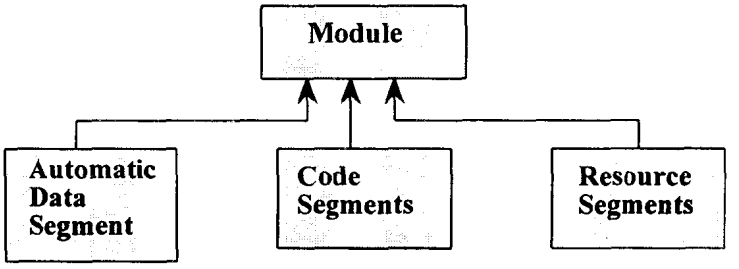

图 4.6 运行 DLI 时 Module 及 Segnen 之间的关系图

下面用 Windows SDK 提供的内存观察工具 Heap Walker(HEAPWALK.EXE)来验证 Windows 系统执行应用程序和 DLL 维护的数据，有关 Heap Walker 的说明见本章第 6 节。还是以执行 PBRUSH.EXE 为例。

首先启动 Windows，接着运行 SDK 中的 HEAPWALK.EXE，Heap Walker 显示 Windows 系统所有内存的使用情况；选择菜单 Sort.Module，指示 Heap Walker 以模块名称的字母顺序显示内存的使用情况，下图是运行 Heap Walker 时的情况。

HeapWalker-[Main Heap]

File Walk Sort Object Alloc Add!

ADDRESS	HANDLE	SIZE	LOCK	FLG	HEAP	OWNER	TYPE
8065FF00	16E6	4096		D		GDI	Private
80656BE0	008E	768				GDI	Private Bitmap
0009F300	0336	576				GDI	Private Bitmap
00095F40	0356	192				GDI	Private Bitmap
8067A300	12BE	6720		D		GDI	Private Bitmap
806492C0	03BE	3360				GDI	Private Bitmap
0002E3C0	1666	64	L1,P1			GDI	Private Bitmap
805927C0	135E	16192				GDI	Private Bitmap
00049500	04D7	64	P1	F		GDI	Private Bitmap
80649FE0	166E	512				GDI	Private Bitmap
0008E460	06DE	672		D		GDI	Resource String
806AFD20	11CE	2848				HEAPWALK	Code 1
806B0840	11C6	3136				HEAPWALK	Code 2
805FD1C0	11BE	2336		D		HEAPWALK	Code 3
805FCA20	11B6	1952		D		HEAPWALK	Code 4
805FC440	11AE	1504		D		HEAPWALK	Code 5
805FC200	11A6	576		D		HEAPWALK	Code 6
8069E780	119E	1248		D		HEAPWALK	Code 7
8069F2C0	117E	1760		D		HEAPWALK	Code 11
8069EC60	116E	1632		D		HEAPWALK	Code 13
806A0A20	1166	2624		D		HEAPWALK	Code 14
806AA000	115E	23296			Y	HEAPWALK	DGroup
80695940	11FE	896				HEAPWALK	Module Database
806BA260	10FE	17376				HEAPWALK	Private
000300E0	1136	64		D		HEAPWALK	Resource Group_Icon
0008E700	1126	704		D		HEAPWALK	Resource Menu
00031100	114E	224		D		HEAPWALK	Resource String
0002FBA0	11EF	512	P1	F		HEAPWALK	Task
00023460	0117	44896	P1	F		KERNEL	Code 1
805E45A0	011E	10848		D		KERNEL	Code 2

图 4.7 没有执行 PBRUSH.EXE 时的情况

执行 PBRUSH.EXE，再用 Heap Walker 观察，我们看到在 HeapWalk 中增加了如下有关 PBRUSH 的内容：

Global Heap Data - sorted by Module

ADDRESS	HANDLE	SIZE	LOCK	FLG	HEAP	OWNER	TYPE
0005FBA0	1C17	5824	P1	F		PBRUSH	Code 1
0006B360	1C06	3296			Y	PBRUSH	DGroup

00061260	1C26	256			*PBRUSH*	*Module Database*
00061DE0	09C6	608			PBRUSH	Private
80592500	1FBE	3456		D	PBRUSHX	Code 1
805916C0	1F9E	3648		D	PBRUSHX	Code 2
805EE920	1F96	10656		D	PBRUSHX	Code 3
805ED120	1F86	6144		D	PBRUSHX	Code 4
805EC740	1EE6	2528		D	PBRUSHX	Code 6
8064F780	1E7E	2112		D	PBRUSHX	Code 10
805E9280	1DE6	384		D	PBRUSHX	Code 14
805EB9E0	1DD6	3424		D	PBRUSHX	Code 15
805EA580	1D96	5216		D	PBRUSHX	Code 17
8064A8E0	1D36	7296		D	PBRUSHX	Code 19
8064FFC0	1D26	704		D	PBRUSHX	Code 20
805EA0E0	1CF6	1184		D	PBRUSHX	Code 21
80696160	1CEE	8608		D	PBRUSHX	Code 22
805E99A0	1CB6	1856		D	PBRUSHX	Code 24
805E96A0	1C5E	768		D	PBRUSHX	Code 29
8069E2C0	1C4E	9152		D	PBRUSHX	Code 31
80698300	1C36	7424		D	PBRUSHX	Code 33
80AFA920	1C2E	32768		Y	PBRUSHX	DGroup
0006A7C0	26C6	2976			*PBRUSHX*	*Module Database*
00070440	093E	160			PBRUSHX	Private
00031740	11EE	32			PBRUSHX	Private
0006C040	096E	96			PBRUSHX	Private
000315A0	11F7	64	P1	F	PBRUSHX	Private
00061C60	0A16	32			PBRUSHX	Private
00030180	1207	64	P1	F	PBRUSHX	Private
00030140	120F	64	P1	F	PBRUSHX	Private
00061500	1226	96			PBRUSHX	Resource Accelerators
000579E0	1AD6	288		D	PBRUSHX	Resource Cursor
000578C0	1FF6	288		D	PBRUSHX	Resource Cursor
00031600	1FEE	288	L1	D	PBRUSHX	Resource Cursor
000315E0	2776	32		D	PBRUSHX	Resource Group_Cursor
00062220	1ABE	32		D	PBRUSHX	Resource Group_Cursor
00061D20	1AE6	32		D	PBRUSHX	Resource Group_Cursor
00057B00	0876	32		D	PBRUSHX	Resource Group_Cursor
000615A0	1B1E	64		D	PBRUSHX	Resource Group_Icon
00057620	1AFE	672		D	PBRUSHX	Resource Icon
0006ECC0	1B8E	960		D	*PBRUSHX*	*Resource Menu*
00031720	0886	32		D	PBRUSHX	Resource String
000622E0	088E	32		D	PBRUSHX	Resource String
00061C40	0A2E	32		D	PBRUSHX	Resource String
00061A40	0A36	512		D	PBRUSHX	Resource String

```
000617A0 10BE      672          D       PBRUSHX  Resource String
00061620 118E      384          D       PBRUSHX  Resource String
00058B20 11E6       32          D       PBRUSHX  Resource String
00031100 2057      512 P1       F       PBRUSHX  Task

Info on Free Blocks in the Global Heap
      Number of Free Blocks = 156
      Size of Largest Free Block = 153248 Bytes
      Total Size of Free Blocks = 1761472 Bytes

MODULE          COUNT     DISCARD     NON - DISCARD     TOTAL
PBRUSH              0           0              9984      9984
PBRUSHX            34       78944             36864    115808
```

图 4.8 执行一次 PBRUSH.EXE 时的情况
(一个 Module Database 和一个 Instance Database)

Windows 系统为 AP 和 DLL 维护的数据都可以在图 4.8 中找到。

在 \ windows 目录下有三个与 PBRUSH 相关的文件:PBRUSH. EXE, PBRUSH. DLL 和 PBRUSH. HLP。当执行 PBRUSH. EXE 时要调用 PBRUSH. DLL,它是一个虚拟位图管理程序库(Virtual bitmap manager)。PBRUSH. EXE 的模块名为 PBRUSHX, PBRUSH. DLL 的模块名为 PBRUSH。所以执行 PBRUSH. EXE 时生成了两个 Module Database: PBRUSH Module Database 和 PBRUSHX Module Database, 这就是上图有两个 Module Database 的原因。

在启动过程完毕后,应用程序到底有些什么内容在内存中? 有哪些与该应用程序相关的数据记录已建立在系统中? 我们还是结合图 4.8 来回答这个问题。

在第 1 章和第 2 章中我们就已经知道:PBRUSH. EXE 有 34 个段,其中 Segment 1、2、3、4、6、15、17、19、21、24、29、31 和 34 为预装入段(Preload),这些段好象都已在内存中存在,分别对应图 4 - 2 中的 Code 1、Code 2 ... ... Code 33。但 Segment 34 似乎还没有装入? 不对! Segment 34 确实已经装入! 不要忘了 Segment 34 是自动数据段,而不是一般的代码段。PBRUSHX DGroup 对应的就是 Segment 34。除了上述预装载段之外,另外我们还发现,尽管 Paint Brush 还没有真正开始画一点什么,但 Segment 10、14、20 和 22 这些非预装入段(non-Preload, LoadonCall)也被装入内存,这又是为什么? 我们的解释是:Paint Brush 虽然还没有画一点看得见的东西,但它已作好了准备。

还有一个概念也应该说明一下,那就是快速装载区(Gangload Area)。Windows EXE 首部信息块偏移[37h]、[38h]和[3Ah]处的内容指示该程序是否包含快速装载区、快速装载区的位置(见第 1 章)。Windows Linker 一般将所有需要预装载的段和资源连续存放在文件的开头,并把这一区域设置为快速装载区。比如,用文件分析工具 Power FileInfo(PFI)可以看到,PBRUSH. EXE 的快速装载区从地址[1160H]开始,长 145E0H, 其中包含所有上述 Preload 的段和资源 Menu. PBRUSH2, 该资源对应在上面的 Heap Walker 显示结果中的"PBRUSHX Resource Menu"那一行(要在 Heap Walker 中查看已装载的资源,双击鼠标即可)。把"快速装载区"和"预装载"指向相同的内容,直接根据 Gangload Area 来 Load 所有应

该预装载的部分，达到“快速装载”的目的。当然，Linker 并不是一定要按这种方法来处理，比如，Lotus 的 Freelance for Windows V1.0 有许多需要预装载的数据段，但就是没有快速装载区。Windows 系统在加载一个应用程序的时候，会检查该程序是否有“快速装载区”，如果有，直接把该区的代码数据全部读入缓存，而“预装载”段是 Windows 在加载的过程中装入内存的。

4.4.2 Module Database

接下来我们再看看 Module Database 的内容。

当 Windows Loader 首次加载一个可执行文件并准备执行时，Windows 根据可执行文件的文件头信息在内存中建立起一块属于该程序的内存块，这一区域称为 Module Database，其中包括 segment table, resource table, entry table ... 等。Module Database 可以理解成把 NE 首部装入内存后，又作了一些细小的改动。代表该内存块的 handle 称为 Module Handle。

用鼠标双击 HeapWalk 窗口中 PBRUSH Module Database 所在的行，就会显示该 Module Database 的内容：

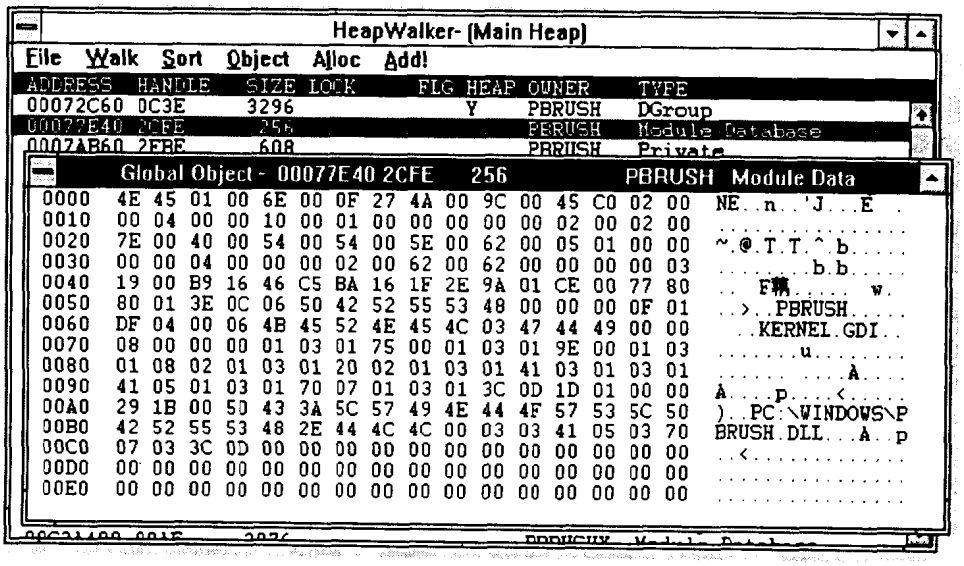

图 4.9　一个 Module Database 的内容

用 PDUMP /B 分析 PBRUSH.DLL 的内容，找出其中的 NE 文件头数据，列于图 4.10 中。读者可以将它与上面图 4.9 中 Module Database 内容对照，找出两者之间的关系。在这里，我们没有选 PBRUSH.EXE 来比较 Module Database 与 NE 首部间的关系，只是因为它的 Module Database 内容太长。

```
Power Dump  Version 1.0 Copyright (c) 1993 KingSoft Ltd.  Mr. Leijun
            Display of File \WINDOWS\PBRUSH.DLL

0080:  4E 45 05 14 6A 00 1B 00  E5 7A 51 9F 05 80 02 00  NE..j....zQ.....
```

```
0090:  00 04 00 00 10 00 01 00  00 00 00 00 02 00 02 00  ................
00A0:  7E 00 40 00 50 00 50 00  5A 00 5E 00 05 01 00 00  ~.@.P.P.Z.^.....
00B0:  00 00 04 00 00 00 02 00  00 00 00 00 00 00 00 03  ................
00C0:  19 00 B9 16 00 0D BA 16  9A 01 CE 00 71 0C 80 01  ............q...
00D0:  06 50 42 52 55 53 48 00  00 00 01 00 08 00 00 06  .PBRUSH.........
00E0:  4B 45 52 4E 45 4C 03 47  44 49 08 01 03 75 00 03  KERNEL.GDI...u..
00F0:  9E 00 03 08 02 03 20 02  03 41 03 03 41 05 03 70  ...... ..A..A..p
0100:  07 03 3C 0D 00 16 56 69  72 74 75 61 6C 20 62 69  ..<...Virtual bi
0110:  74 6D 61 70 20 6D 61 6E  61 67 65 72 00 00 0B 56  tmap manager...V
0120:  53 54 52 45 54 43 48 42  4C 54 06 00 03 57 45 50  STRETCHBLT...WEP
0130:  01 00 0B 44 49 53 43 41  52 44 42 41 4E 44 08 00  ...DISCARDBAND..
0140:  0D 56 44 45 4C 45 54 45  4F 42 4A 45 43 54 07 00  .VDELETEOBJECT..
0150:  07 56 50 41 54 42 4C 54  04 00 07 56 42 49 54 42  .VPATBLT...VBITB
0160:  4C 54 05 00 0B 47 45 54  56 43 41 43 48 45 44 43  LT...GETVCACHEDC
0170:  03 00 0D 56 43 52 45 41  54 45 42 49 54 4D 41 50  ...VCREATEBITMAP
0180:  02 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

图 4.10 与 Module Database 作对比的 NE 文件头内容

任何有关.EXE 和.DLL 在内存中的信息(如:代码段、资源段、资源、模块名等)都被集中保管在 Module Database 中。对于 Windows 应用程序,Module Database 中包含了可被同一程序的所有 Instance 共享的信息(虽然 DLL 只有一个 Instance):而属于个别 Instance 的信息,则存放在 Task Database 中(注意:DLL 不是 Task,所以没有 Task Instance)。

对于 Windows AP 和大部分的 DLL 而言,Module Database 最主要的作用是用来跟踪代码段和数据段载入、释放等操作,以及记录分配给这些段的选择子(Selector)。当 Windows KERNEL 载入一个程序时,如果该程序需要和一个已被载入内存的模块中的某个函数作动态链接时,这时 KERNEL 就可以从已经在内存中的 Module Database 内找到和该函数所在的段对应的 selector,接着 KERNEL 中的段重定位器(segment relocator)会将所找到的 selector 写入刚被载入程序的代码中,所以动态链接就成了一个直接进入其它模块代码段中执行的远程调用。DOS 使用中断和寄存器达成程序和操作系统间链接,相比较而言,Windows 的接口要简明。

对于纯粹提供资源的一些 DLL,如.FON,由于其中没有任何代码段和数据段,它们的 Module Database 主要是用来寻找文件中资源的位置及其它一些数据。

Microsoft 并没有提供 Module Database 的结构,根据 Andrew Schulman 等在《Undocmented Windows》一书中的分析,Module Database 的数据结构如下:

偏移值	大 小	意 义
00H	WORD	"NE"(454Eh)
02H	WORD	此模块被其它模块引用的次数
04H	WORD	指向此模块的 entry table 的近指针
06H	WORD	Module List 中的下一个 Module Table 的 selector

08H	WORD	DGROUP(自动数据段)在段表(segment table)中的近指针
0AH	WORD	指向模块文件信息的近指针
0CH	WORD	模块类型标记
0EH	WORD	DGROUP 的逻辑段号
10H	WORD	起始的堆栈大小
12H	WORD	起始的 CS:IP
14H	DWORD	起始的 SS:SP
18H	DWORD	段表(segment table)中段的总数目
1CH	WORD	模块引用表(module reference table)中模块的总数目
1EH	WORD	在磁盘中的非驻留名表(nonresident name table)的大小
20H	WORD	指向段表的近指针
22H	WORD	指向资源表的近指针
24H	WORD	指向驻留名表的近指针
26H	WORD	指向模块引用表的近指针
28H	WORD	指向输入名表的近指针
2AH	WORD	非驻留名表在 NE 文件中的位置
2CH	DWORD	可移动入口点(movable entries)的总数
30H	WORD	段的左移位因子

图 4.11　Module Database 的数据结构

其中偏移[0CH]处为模块的类型标记，大致上，此项和在 NE 文件头中的“可执行文件特征标记”是基本上一样的，但包含了更多的内容：

标　记	意　义
8000H	为一 DLL 模块
0800H	自装载(self-loading)的可执行文件
0300H	使用 Windows API 函数的模块
0200H	与 Windows API 函数兼容的模块
0100H	与 Windows API 函数不兼容的模块
0080H	包含非驻留(nonresident)代码段的模块
0040H	自己管理内存配置的模块
0010H	使用 LIM 3.2 API 函数的模块
0008H	只执行于保护模式的模块
0004H	Pre-process Library Initialization
0002H	每个 Instance 私有的 DGROUP(task)
0001H	多个 Instance 公用一个 DGROUP(DLL)

图 4.12　内存中的文件特征标记每个 BIT 位含义

偏移[04H]处的值是指向此模块的 entry table 的近指针，要注意的是：内存版本和磁盘版本的 entry table 的不大相同。所有空的入口点都被去掉，而且序号连续的进入点被集中起来管理，形成一个单向序列；在每个单向序列的最前端是一个作为管理用的 header 结构，随后跟着在序列中各个入口点的结构。在文件中，entry table 内针对应用程序中的每一个可移动段中的远函数调用(far call)都有 6 个字节的相关信息；但在内存中，这些信息变成了 8 个字节，剩余的两个字节的信息是由 Windows 在执行期间补上的。各个入口点的数据结构是：

BYTE	段的类型 若为 0FFH，则表示为可移动的段 否则为固定段号
	BYTE 标记 若为 1，则表示为 exported 的进入点 若为 2，则表示为 shared data 的进入点
BYTE	进入点所在的段的逻辑段号
WORD	进入点在段中的偏移量

图 4.13 内存中的“入口点数据结构”

4.4.3 Task Database

知道了 Module Database 的内容和作用后，我们再来看一下 Task Database。

Windows 为每一个 Task 维护一个 Task Database，Task Database 为每一个 Task 所独有的信息，其中包含很多重要的数据，如：该 Task 切换成 nonactive task 时的 SS:SP、有关 DOS 文件 I/O 的状态、和一个保护模式的 DOS PSP(在 Windows 中，亦称为 Program Database 或 PDB)等。因为 Windows 是一个多任务的系统，所以每一个 Task 都有自己的数据区。比如该 Task 的当前工作目录。

对于 Windows DLL，Module Database 没有对应的 Task Database，因为它只是 Windows Tasks 在执行时调用的代码。举例来说，如果在 DLL 中要打开一个文件，此时如果一个 TaskA 中要调用这一功能，那么被打开文件的句柄(handle)就被放置在 TaskA Database 中；如果 TaskB 也调用了这一功能，那么一个新的句柄被存放在 TaskB Database 中。

Task Database 的结构在 Windows DDK 所附的 TDB.INC 文件中有定义，其内容如下：

偏移	大 小	意 义
00H	WORD	下一个 TDB 的 handle；若为 0，则表示链表结束
02H	DWORD	上次被切换成 non - current task 时的 SS:SP
06H	WORD	函数 postEvent/WaitEvent 用来链接的事件计数器
08H	WORD	表示此 Task 优先级
0AH	WORD	不详

0CH	WORD	此 TDB 的 handle
0EH	BYTE[06H]	不详
14H	WORD	80x87 控制字
16H	WORD	task 标记。WINOLDAP = 1, OS2APP = 8, Win32s = 10H(Win32s 为 Win32 支持 3.1 的部分)
18H	WORD	错误模式(error mode)标记,通常为 0
1AH	WORD	此 task 所期望的 Windows 版本
1CH	WORD	此 task 的 instance handle
1EH	WORD	此 task 的 module handle
20H	WORD	此 task 的任务队列句柄(Task Queue handle)
22H	WORD	父 task 的 TDB 句柄
24H	WORD	与函数 SetSigHandler 有关的标记
26H	DWORD	SetSigHandler 所安装的处理程序地址
2AH	DWORD	SetTaskSignalProc 所安装的处理程序
2EH	DWORD	GlobalNotify 所安装的处理程序
32H	DWORD	INT 00H 中断处理程序地址
36H	DWORD	INT 02H 中断处理程序地址
3AH	DWORD	INT 04H 中断处理程序地址
3EH	DWORD	INT 06H 中断处理程序地址
42H	DWORD	INT 07H 中断处理程序地址
46H	DWORD	INT 3EH 中断处理程序地址
4AH	DWORD	INT 75H 中断处理程序地址
4EH	DWORD	GetProfileInt(modName, "Compatibility", 0)的传回值
52H	BYTE[0EH]	不详
60H	WORD	此 Task 的 PSP 句柄
62H	DWORD	指向 PSP 的数据转换区(DTA)或命令行的远指针
66H	BYTE	目前的磁盘 + 80H(80H = A、81H = B、...)
67H	BYTE[43H]	目前工作目录
AAH	WORD	经函数 DirectedYield 切换至 task 的 hTask
ACH	WORD	一个段选择子;该段内含有此 task 所需参考到的 DLL 清单,此项只在调用函数 InitTask 前存在
AEH	WORD	DLL 清单在上述段中的偏移值
B0H	WORD	此 TDB 的 CS alias
B2H	WORD	存放更多的 instance-thunk 的段选择子;若为 0,则表示不需要其它的段。所有存放 instance-thunk 的段,它们在 B2H 至 F1H 间的内容相同

B4H	WORD	instance-tHunk 的代号
B6H	WORD	不详，为 0
B8H	WORD	将此项减 6，可得到存放下一个 instance－thunk 的偏移值；若所得的值为 0，则表示已经没有空间存放 instance-thunk，这时，在[B2]处，就成为另一个存放 instance-thunk 的段选择子
BAh	BYTE[38H]	最多可容纳 7 个 instance-thunk 的空间。每个 instance-thunk 的大小为 8 个 byte，且含有原来传给函数 MakeProcInstance 的参数： mov ax, hInstance jmp lpProc
F2H	BYTE[08H]	此 Task 的模块名。若模块名长为 8 个字节，则没有节尾字节'\0'
FAH	WORD	TDB 的代号'TD'(4454h)
100H	BYTE[100H]	此 Task 的 PSP 的开始。存放在偏移[60H]的段选择子的基地址就是指向这里

图 4.14 Task Database

4.4.4 Instance Database

一个 Instance Database 实际上就是一个 DGROUP(自动数据段)。对于一个 DLL，不管有多少个 Task 调用它，总是只有一个 DGROUP。对于应用程序的一个 task，它的 instance handle 会被存放在 Task Database 中；对于 DLL，它没有 Task Database，它的 instance handle 是放在 Module Database 中。

在 HeapWalk 中，用鼠标双击"PBRUSHX DGroup"所在的行，可以看到如下的内容：

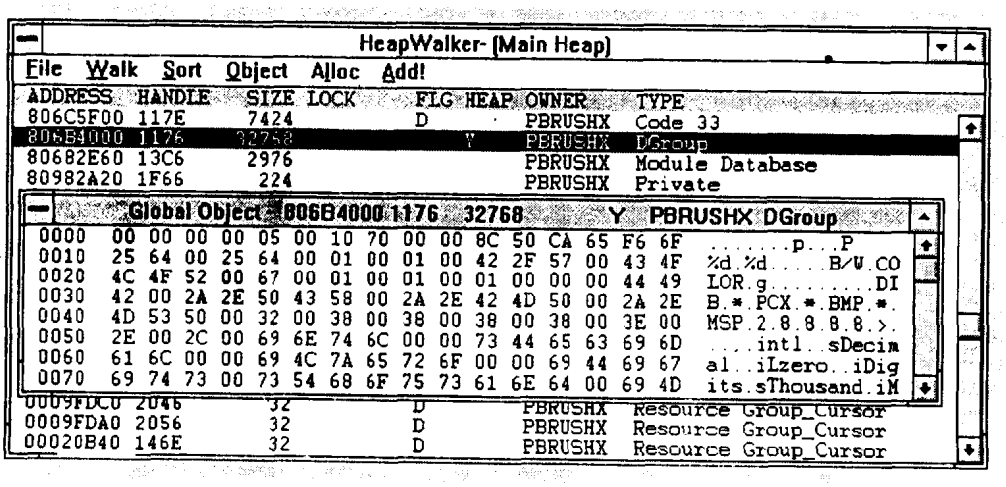

图 4.15 Instance Database

在每个 task 或 DLL 的 DGROUP 中，包含了该 instance 的常数和静态变量、预定 local heap 和堆栈。位于 DGROUP 最前面的 16 字节是保留存储区，此保留存储区又称为 In-

stance Data。在讲述 Windows AP 的启动过程时，我们知道启动函数 InitTask 函数将堆栈的顶、最小值和基地址偏移量拷贝到应用程序自动数据段的前 16 字节的保留存储区。从图 4.15 中我们可以看到执行 PBRUSH.EXE 后，启动函数存放在保留存储区的具体内容。这 16 个字节的内容是 oOldSP、hOldSS、pLocalHeap、pAtomTable、pStackTop、pStackMin、pStackBot，对应于上图中 DGROUP 数据的第一行。

在 Windows 中，并不是每一个 DGROUP 都会有自己 Instance Data 这样的数据结构。如果偏移[00H]的值不是 0，或者[06H]的值不是一个指向 local heap 的指针，则表示该 DGROUP 没有 Instance Data。例如，Windows 的一些设备驱动程序的 DGROUP 就没有 Instance Data。应用程序和 DLL 的自动数据段完整的数据结构就不再列出，有兴趣的读者可以用 ToolHelp 中提供的函数编写一个程序来观察。

顺便说明一下：执行程序的资源信息是放在 Module Database 中。但在 Windows 编程时，我们是通过给出参数 instance handle 来访问资源，而 DGROUP 或 Task Database 中并没有存放任何有关资源的信息，Windows 的内部函数是将 instance handle 转换成对应的 module handle 后，才取得所需资源的。

4.4.5　应用程序执行多份时的情况

明白了上述几个关键的数据结构后，我们再来观察一个应用程序执行多份时的情况。下图是同时执行三份 Paint Brush 时用 Heap Walker 观察所得的结果。

Global Heap Data - sorted by Module

ADDRESS	HANDLE	SIZE	LOCK	FLG	HEAP	OWNER	TYPE
00070E20	1FE7	5824	P1	F		PBRUSH	Code 1
8077AA80	1FEE	3296			Y	PBRUSH	DGroup
00078220	1FFE	256				PBRUSH	Module Database
00077AA0	1F56	608				PBRUSH	Private
8068E340	11F6	3456		D		PBRUSHX	Code 1
8068C500	11E6	3648		D		PBRUSHX	Code 2
80685C60	11BE	10656		D		PBRUSHX	Code 3
80684460	11A6	6144		D		PBRUSHX	Code 4
80683A80	1196	2528		D		PBRUSHX	Code 6
806B4240	11CE	2112		D		PBRUSHX	Code 10
8062C080	1256	384		D		PBRUSHX	Code 14
80682D20	1246	3424		D		PBRUSHX	Code 15
806818C0	118E	5216		D		PBRUSHX	Code 17
8067FC40	11AE	7296		D		PBRUSHX	Code 19
806B4A80	146E	704		D		PBRUSHX	Code 20
8068DEA0	11EE	1184		D		PBRUSHX	Code 21
80679000	11DE	8608		D		PBRUSHX	Code 22
8067F500	116E	1856		D		PBRUSHX	Code 24
8068DBA0	2026	768		D		PBRUSHX	Code 29
8067D140	2056	9152		D		PBRUSHX	Code 31

```
8067B1A0 11D6    7424          D        PBRUSHX  Code 33
80772000 1F06   32768                Y  PBRUSHX  DGroup
80793EE0 132E   32768                Y  PBRUSHX  DGroup
807865E0 0866   32768                Y  PBRUSHX  DGroup
00076380 1236    2976                   PBRUSHX  Module Database
00076320 2006      96                   PBRUSHX  Private
00021100 098E      32                   PBRUSHX  Private
00020460 10F7      64 P1         F      PBRUSHX  Private
00020420 10D7      64 P1         F      PBRUSHX  Private
00020DA0 0997      64 P1         F      PBRUSHX  Private
00020D60 12CF      64 P1         F      PBRUSHX  Private
00020D20 1327      64 P1         F      PBRUSHX  Private
00049920 113E      32                   PBRUSHX  Private
000209C0 0967      64 P1         F      PBRUSHX  Private
00020980 1E6F      64 P1         F      PBRUSHX  Private
00020940 1E8F      64 P1         F      PBRUSHX  Private
0007C800 1FC6     160                   PBRUSHX  Private
00048500 1EBE      96                   PBRUSHX  Private
0007AF00 096E      32                   PBRUSHX  Private
0007AEC0 0946      32                   PBRUSHX  Private
0007AEA0 095E      32                   PBRUSHX  Private
0007AE80 2016      32                   PBRUSHX  Private
000781C0 138E      96                   PBRUSHX  Private
000775A0 0906     160                   PBRUSHX  Private
000204A0 10FF      64 P1         F      PBRUSHX  Private
00077460 090E     160                   PBRUSHX  Private
0007AE20 10DE      96                   PBRUSHX  Resource Accelerators
0007CAA0 124E     288 L1         D      PBRUSHX  Resource Cursor
0007C980 10E6     288            D      PBRUSHX  Resource Cursor
00020500 1386     288 L2         D      PBRUSHX  Resource Cursor
00049900 1106      32            D      PBRUSHX  Resource Group_Cursor
00049240 10CE      32            D      PBRUSHX  Resource Group_Cursor
00022F00 127E      32            D      PBRUSHX  Resource Group_Cursor
000204E0 123E      32            D      PBRUSHX  Resource Group_Cursor
00049280 1136      64            D      PBRUSHX  Resource Group_Icon
000771C0 1116     672            D      PBRUSHX  Resource Icon
80A356C0 13A6     960            D      PBRUSHX  Resource Menu
00049B00 1F86      32            D      PBRUSHX  Resource String
00023180 087E      32            D      PBRUSHX  Resource String
00049220 0876      32            D      PBRUSHX  Resource String
0007B380 1F7E     512            D      PBRUSHX  Resource String
0007B0E0 1F6E     672            D      PBRUSHX  Resource String
0007AF60 1F76     384            D      PBRUSHX  Resource String
```

```
00022300 10C6      32          D       PBRUSHX  Resource String
00020A00 133F    512 P1        F       PBRUSHX  Task
0001FF20 122F    512 P1        F       PBRUSHX  Task
00020620 097F    512 P1        F       PBRUSHX  Task

Info on Free Blocks in the Global Heap
      Number of Free Blocks = 66
      Size of Largest Free Block = 258880 Bytes
      Total Size of Free Blocks = 563968 Bytes

MODULE          COUNT     DISCARD     NON-DISCARD     TOTAL

PBRUSH              0           0            9984      9984
PBRUSHX            34       78944          104448    183392

Totals            312      911424         3934336   4845760
```

图 4.16　同时执行三次 PBRUSH.EXE 时的情况
(一个 Module Database 和三个 Instance Database)

和图 4.16 比较,我们看到:对于 DLL,PBRUSH.DLL 的内容和只执行一次时相同。对于 Windows AP,由于代码段和资源都是共享的,所以和只执行一次时相同;而"PBRUSHX DGroup"、"PBRUSHX Private","PBRUSHX Task"的数量都变成了原来的三倍。DGroup 和 Task 的意思读者已经明白,那么,"PBRUSHX Private"又是什么呢?为什么也变成了原来的 3 倍?"PBRUSHX Private"是指程序以 GlobalAlloc 函数分配的全局内存,变成原来的 3 倍也就不难理解了。

读者可能会问,PBRUSH 可以同时执行多份,但有些程序不可以,比如 Freelance。这是因为任何 NE 执行程序,不管它有多少个数据段,但自动数据段最多只能有一个。试图运行有多个数据段的程序时,Windows Loader 只会加载自动数据段,而不去顾及其它的数据段,这种加载是肯定要失败的。只有单个数据段的程序才有可能被同时执行多份。Freelance 有多个数据段,用 PDUMP 可以看到这一点。还有一种情况是,该程序在内部设置了不允许执行多份。

4.4.6　Thunk 与重定位

先回忆一下第一章中讲述过 Windows NE 文件头的中的重定位表和其它表的内容。

(1) Segment Table:重定位有三种类型:内部引用、输入序号和输入名。其中,所谓内部引用是相对于该 Module 而言,即调用了该 Module 中其它段的函数。比如,PBRUSH.EXE 的 seg #1 有一个重定位项的数据如下:

```
03 00 6A 05 FF 00 00 01
```

表示在[056A]处调用了其它段中的一个函数,FF 表示被调用函数所在的段是一可移动段,该函数在入口表 0100h(256)项说明。而入口表 256 项指出:该函数在 seg #3, offset

2034h 处。即在 seg #1 的[056A]处调用了 seg #3, offset 2034h 处的函数。

(2) Resident Name Table:其中的字符串记录着该可执行文件内的 export 函数名。所谓 export 函数就是能够被其它程序调用的函数。不管该表中所指的函数所在的段是可移动的或者是可抛弃的,该 Resident Name Table 总是存在于内存中。在前面的 module Database 中就存放着 Resident Name Table 的内容(偏移 xx 处)。比如,PBRUSH.EXE 只 export 一个函数,叫"PbrushX"。

(3) Module Reference Table:凡是被应用程序调用的 API 函数所属的模块(Module)都列于此表中。比如,PBRUSH.EXE 用到了以下 8 个模块中的函数:PBRUSH, KERNEL, GDI, USER, KEYBOARD, OLESVR, COMMDLG 和 SHELL。这些具体的模块名的字符串是存储在 Import Name Table 中。

(4) Import Name Table:其中含有执行文件中 import 进来的模块名称。如果程序中的重定位类型为输入序号,只需给出模块名即可;如果重定位类型为输入名,那么,Import Name Table 要给出具体的函数名。用 PDUMP 可以看到,PBRUSH.EXE 调用的 PBRUSH.DLL 中的函数,给出的都是函数名;而它调用的 USER、GDI 等都是给的序号,所以在 Import Name Table 中只用给出模块名。

(5) Nonresident-Name Table:内部记录着该可执行程序 export 函数的名称。Entry Table 中每个入口的具体名称都在 Nonresident - Name Table 中说明。

(6) Entry Table:所谓入口表,就是指可执行文件中要调用其它地方或其他程序中函数的地址信息,以及可供其它程序调用的函数的地址信息。入口表中的地址信息根据入口点所在段的属性(可移动的或固定的)的不同而有所差异。

我们来看一下 Windows 执行一个程序时如何调用一个函数。

在前面我们已经讲过,entry table 内针可移动段的 6 字节信息在内存中变成了 8 个字节,两个新的字节是由 Windows 在执行期间补上的。这些 8 个字节的信息组成一份数据,称为"loader thunk"(载入微码)或"call thunk"(调用微码),存放在 Module Database 中,专门负责"代码段载入"事宜。

loader thunk 的内容因时而异。假设当前正在装载某个 Windows 应用程序中的段 SegA, SegA 的重定位表中要调用本程序中或另外某个 DLL 中的一个函数 funcX, funcX 在 SegB 中。第一种情况是:由于某种原因,SegB 不在内存中。比如说,当前正在装载 PBRUSH.EXE 的 seg #3,在 seg #3 中要调用 Windows API 中的函数 updatecolors,该函数的代号是 GDI.366,它在 GDI 模块的 Seg #16 中。Windows KERNEL 要在 GDI Module Database 中寻找函数 updatecolors 的当前位置。由于 Seg #16 的段属性为 Movable、Discardable,我们假设 Seg #16 未被装入,或者曾经被装入,但当前已被抛弃,这时,GDI Module Database 中的 entry table 被设置成如下的内容,告诉所有想调用函数 updatecolors 的应用程序,若调用将引起 KERNEL 加载 GDI 的 Seg #16:

```
SAR CS:[xxxx], 1
INT 3Fh seg:off
```

这两条指令的意思是加载一个段。第二种情况是:SegB 已经在内存中。对于上述例子,表示 Seg #16 已在内存中,这时,KERNEL 会将函数 updatecolors 的 loader thunk 设值

为如下的值：

```
SAR CS:[xxxx], 1
JMP ssss:oooo
```

其中第二条指令所给出的地址就是函数 updatecolors 的进入点。

如果 PBRUSH.EXE 在装载时遇到的是第一种情况，那么当 PBRUSH.EXE 执行到 Seg #3 的这个位置，或者其它已被加载的程序执行到 updatecolors 时，KERNEL 就会加载 GDI 的 Seg #16，这时，GDI Module Database 中有关函数 updatecolors 的 loader thunk 就变成了第二种情况。

如果 Windows 系统根据内存分配的实际需要，移动了函数 updatecolors 在内存中的位置，这时 KERNEL 只需要修改上述"JMP ssss:oooo"语句中的地址即可。所以不论 Windows 移动那些"会发出远调用"代码段（比如 PBRUSH 的 Seg #3），或移动那些"内含 export 函数"代码段（比如 GDI 的 Seg #16），都不需要对程序代码作任何改变：因为要完成一个远调用，并不是去调用该远函数的真正地址，而只是执行 loader thunk 即可。

反过来，如果 GDI 的 Seg #16 被抛弃，则函数 updatecolors 的 loader thunk 就又变回了第一种情况。这就是 thunk 的优越性。

有一点应该指出，在上面所讲述的 loader thunk 两种状态的转换中，并不是只有函数 updatecolors 参与这种转换，所有在 GDI Seg #16 中的 export 函数都会有同样的转换动作。这是因为 Windows 是按段来载入或抛弃的。

还是上面的这个问题，如果 funcX 所在的段 SegB 为一固定段，那么动态链接又是如何实现的呢？

与上面的情况不同，这时并不需要透过微码（thunk）调用，取而代之的是直接将 SegA 具体地址链接到被调用的函数处。

上面考虑的调用问题是以实模式为例的。实际上，在保护模式下运行的 Windows 系统，它并没有为 Module Database 维护什么 loader thunk。因为硬件提供的内存管理功能可以在不改变逻辑地址的情况下变换内存的物理地址，所以在保护模式下，这种针对可移动的链接可以视为和固定段时的情况相同。如果是针对可抛弃段的链接，由于没有 loader thunk，就可能调用一个不存在的段，这样将会造成段错误（segment fault），段错误将形成一个软件中断，通知内存管理程序必须载入这一个段。这样也同样达到了目的。

4.5　直接修改 Windows 执行文件的方法

在没有源程序的情况下，直接修改 Windows 执行程序是一件非常困难的事，但在国内也是相当必要的事。比如，有的应用软件不能在 Windows 中文环境中执行，在没有原来开发厂家支持的情况下又必须解决这个问题，就需要掌握修改的方法。

4.5.1　修改 STUB

如果你使用的是 Borland Pascal for Windows（BPW），BPW 编译的应用程序只能使用默

认的 STUB,修改 STUB 也许是你目前最迫切的需要(我们在第一章中提到的 RESTUB 就是这样的工具)。这样的工作是如何完成的?

下面我们讲述 RESTUB 的工作过程。

Step 1. 判断将要作为 STUB 的 DOS 执行文件(EXE)是否可以作为 STUB。比如 DOS 执行文件的重定位表的偏移小于 40H,就必须修改这个文件,使之满足要求。这里 DOS 执行文件是指 DOS 上 MZ 格式的文件。虽然有的程序可以在 DOS 上直接执行,但并不表示该格式就是 MZ 格式。如果用 RESTUB 修改 STUB 后不能正常执行,可以检查一下这个 STUB 是不是标准的 DOS 上 MZ 格式的文件。

Step 2. 写入新的 STUB 程序后,用段逻辑页和资源逻辑页对齐后,计算 NE 首部的新的起始位置,同时计算增加或减少的段逻辑页、资源逻辑页的数量。

Step 3. 计算新的段表。

Step 4. 计算新的资源表。

Step 5. 复制文件 NE 首部及代码段、数据段和资源等。

Step 6. 把原来的 STUB 存放在 TEMP.EXE 中。

RESTUB 能够取出原来程序的 STUB。有的软件 STUB 功能很强,比如 Microsoft EXCEL 中的 EXCEL.EXE,在 DOS 下执行时不是提示"不能在 DOS 下执行",而是直接启动 Windows 再执行。我们需要这样的 STUB 时,不用自己编写,只需要用 RESTUB 将这个 STUB 取出来即可。

```
{ RESTUB.PAS 替换 Windows 应用程序原有的 STUB 的程序 }
Uses FileDef, ExtTools, Dos;
const
  TmpFileName = 'PTEMP.EXE';
  TmpStubName = 'PSTUB.EXE';

{ MS-DOS EXE object  定义一个新的文件对象,把一个 DOS 的执行文件改成 STUB }
type
  PStubMz = ^TStubMz;
  TStubMz = object (TMzHeader)
    FT: File;
    constructor Init(FName, TFName: PathStr);
    destructor Done; virtual;
    procedure Run; virtual;
  end;

constructor TStubMz.Init;
begin
  inherited Init(FName);
  if rtError <> 0 then fail;
  assign(FT, TFName);
  rewrite(FT, 1);
end;
```

```
destructor TStubMz.Done;
begin
  close(FT);
  inherited Done;
end;
procedure TStubMz.Run;
var
  A: PBuf;
  Mz: tagMzHeader;
begin
  Mz := mzHeader;
  if mzHeader.ofsRelocList > $40 then
    {如果 DOS EXE 文件重定位表的偏移大于 $40,可以直接作为 STUB}
    CopyFileBlock(F, FT, FSize)
  else begin
    {如果 DOS EXE 文件重定位表的偏移小于 $40,必须修改这个文件的头部,
     把重定位表移动 $40 处 }
    mz.ofsRelocList := $40;
    if (mz.numRelocEntry * 4 + mz.OfsRelocList) > GetMzHeaderSize then
      Inc(mz.sizeHeader, 4);
    BlockWrite(FT, mz, $40);
    A := GetRelocList;
    BlockWrite(FT, A^.P^, A^.Len);
    DisposePBuf(A);
    CopyFileBlockEx(F, GetMzHeaderSize,
      FT, LongInt(mz.sizeHeader) shl 4,
      GetLoadImageSize);
  end;
end;
{ Windows EXE object 定义一个 Windows 执行文件的对象,修改 STUB }
type
  PStubNe = ^TStubNe;
  TStubNe = object (TNeHeader)
    FT: File;                          {目标文件的句柄}
    NewExeOfs: LongInt;                {新的 NE 头开始的位置}
    MyPageSize: Word;                  {页的大小}
    secSegAdd, secResAdd: integer;     {新增段的逻辑页数和新增资源的逻辑页数}
    constructor Init(FName, TFName: PathStr);
    destructor Done; virtual;
    procedure Run; virtual;
  end;
constructor TStubNe.Init;
```

```
begin
  inherited Init(FName);
  if rtError <> 0 then fail;
  assign(FT, TFName);
  reset(FT, 1);
  MyPageSize := GetGeneralPageSize;
end;

destructor TStubNe.Done;
begin
  close(FT);
  inherited Done;
end;

{ Browse Segment Table    修改段表中每段的起始位置 }
procedure IncSegEntry (PSelf: PNeHeader;P: Pointer;Index: word); far;
begin
  with PStubNe(PSelf)^, PSegmentEntry(P)^ do begin
    if (ofs > 0) then Inc(ofs, secSegAdd);
  end;
end;

{ Browse resource table   修改资源表中每个资源的起始位置}
procedure IncResOffset (PSelf: PNeHeader; P: Pointer); far;
begin
  with PStubNe(PSelf)^, PEachRes(P)^ do
    if GetResPageSize <> 0 then
      Inc(NameInfo^.rnOffset, secResAdd);
end;

procedure TStubNe.Run;
var
  I: Word;
  TA: TableArray;
  P1, P2: PBuf;
  NE: tagNeHeader;
{修改段表}
  function GetNewSegTab: PBuf;
  var
    A: PBuf;
  begin
    NewPBuf(A);
    A^.len := neHeader.lenSegmentTab;
    if (A^.Len <> 0) then begin
      GetMem(A^.P, A^.Len);
```

```
      Move(PSeg^, A^.P^, A^.Len);
      BrowseSegmentTab(A, IncSegEntry);
    end;
    GetNewSegTab := A;
  end;
{修改资源表}
  function GetNewResTab: PBuf;
  var
    A: PBuf;
  begin
    NewPBuf(A);
    A^.Len := neHeader.lenResourceTab;
    if A^.Len <> 0 then begin
      GetMem(A^.P, A^.Len);
      Move(PResource^, A^.P^, A^.Len);
      BrowseResourceTab (A, IncResOffset);
    end;
    GetNewResTab := A;
  end;
{修改 DOS 文件头}
  procedure SetNewMZ;
  var
    I: Word;
    MZ: tagMzHeader;
  begin
    Seek(FT, 0);
    BlockRead(FT, MZ, $ 40);
    { MZ.numMinAlloc := 0; }
    MZ.ofsWinHeader := NewExeOfs;
    I := GetOfsEndOfHeader(NE) div $ 200;
    SetImageSize (MZ, GetOfsEndOfHeader(NE));
    Seek(FT, 0);
    BlockWrite(FT, MZ, $ 40);
  end;
begin    { Run }
  NewExeOfs := AlignSize (FileSize(FT), MyPageSize);
  secSegAdd := (NewExeOfs - ofsNe) div GetPageSize;
  secResAdd := (NewExeOfs - ofsNe) div GetResPageSize;
  WriteSeek(FT, NewExeOfs);
  BuildTableArray(TA);
  P1 := GetNewSegTab;
  P2 := GetNewResTab;
```

```
    TA[sysSeg] := P1^;
    TA[sysRes] := P2^;
    NE := neHeader;
    NE.ofsNonresidentNameTab :=
      NE.ofsNonresidentNameTab + NewExeOfs - ofsNe;
    if GangLoad then Inc(NE.ofsGangLoad, secSegAdd);
    TA[sysNe].P := @NE;
    for I := sysNe to sysEntry do
      BlockWrite(FT, TA[I].P^, TA[I].Len);
    DisposePBuf(P1);
    DisposePBuf(P2);
    Seek(F, neHeader.ofsNonresidentNameTab);
    CopyFileBlock(F, FT, FSize-neHeader.ofsNonresidentNameTab);
    SetNewMZ;
    Writeln('Get winstub to ', TmpStubName, ' ...');
    GetWINSTUB(TmpStubName);
  end;  { Run }
{ Application Program Object ------------------------------------------ }
type
  PNewStub = ^TNewStub;
  TNewStub = Object
    PMz: PStubMz;
    PNe: PStubNe;
    Name, Name2, TFName: PathStr;
    constructor Init(FName, FName2: PathStr);
    procedure Run;
    destructor Done;virtual;
  end;
constructor TNewStub.Init;
begin
  if Pos('.', FName) = 0 then FName := FName + '.EXE';
  if Pos('.', FName2) = 0 then FName2 := FName2 + '.EXE';
  Name := UpcaseStr(FName);
  Name2 := UpcaseStr(FName2);
  if ((FileExists(FName)) and FileExists(FName2)) then begin
    if (GetFileSign(FName) = WinAppSign) and
      (GetFileSign(FName2) = ExeAppSign) then begin
      TFName := TmpFileName;
    end
    else rtError := $EE11;
    if rtError <> 0 then fail;
```

```
  end
  else begin
    rtError := $EE00;
    fail;
  end;
end;

procedure TNewStub.Run;
var
  FT: File;
begin
  Writeln('"', Name, '" new stub is "', Name2, '"');
  PMz := New(PStubMz, Init(Name2, TFName));
  PMz^.Run;
  Dispose(PMz,Done)
  PNe := New(PStubNe, Init(Name, TFName));
  PNe^.Run;
  Dispose(PNe,Done);
  Writeln('Erase ', Name,' ... ');
  assign(FT, Name);
  Erase(FT);
  Writeln('Rename ', TFName, ' to ', Name, ' ...');
  assign(FT, TFName);
  Rename(FT, Name);
  Writeln;
end;

destructor TNewStub.Done;
begin
end;

const
  FirstSetName: boolean = true;

var
  I: Word;
  FName, FName2, S: String;
  PTest: PNewStub;

begin
  FName := '';
  FName2 := '';
  if ParamCount > 0 then begin
    I := 0;
    repeat
      Inc(I);
```

```
      S := ParamStr(I);
      if (not (S[1] in OptionChars)) then
        if FirstSetName then begin
          FName := S;
        FirstSetName := False;
        end
        else FName2 := S;
    until (I = ParamCount);
  Writeln('Power RESTUB  Version 1.0   Copyright (c) 1993 KingSoft Ltd.');
  Writeln;
  if (FName = '') or (FName2 = '') then begin
    Writeln('Usage: NewStub <WinAp> <DosAp>');
    Writeln;
    Halt(1);
  end;
  PTest := New(PNewStub, Init(FName, FName2));
  if PTest = nil then
    DisplayRtError
  else begin
    PTest^.Run;
    Dispose(PTest,Done)
  end;
end.
```

4.5.2 在某段后面附加一段代码

修改一个程序,一般都需要打一个"补丁",即插入一段代码。在一个指定段后附加一段代码这样的工作是修改 Windows 执行程序最常用的操作。

我们重点讲述修改的过程。

Step 1. 修改文件首部:如果文件存在快装区域,需要重新计算快装区的开始地址和长度;一个段加长了,还需要修改段表中该段的长度及以后段的起始偏移,同时校正资源表中每个资源新的偏移(即修改该段之后资源的起始位置)。

Step 2. 复制首部和该段之间的内容。

Step 3. 先写该段的代码或数据,再写插入代码,最后写该段的重定位的部分。

Step 4. 把文件其它部分复制过来即可。

下面所附例子的功能是:在启动段的后面附加一段代码,程序执行时,经过该附加代码后再转入启动段。

```
{ APPENDCODE — Copyright (c) 1994.5 by Yellow Rose Software Workgroup }
{ Writen by Mr. Leijun                                                 }

Uses FileDef, ExtTools, Dos;
```

```
type
{ File Struct Object   文件对象 }
PAppendFile = ^TAppendFile;
TAppendFile = Object(TNeHeader)
  F2: File;
  CurrOfs: LongInt;
  AddPages: Word;
  constructor Init(FName, FName2: PathStr);
  destructor Done; virtual;
  procedure Run; virtual;
end;
{ TAppendFile Object }
constructor TAppendFile.Init (FName, FName2: PathStr);
begin
  { 处理文件名 }
  if Pos('.', FName) = 0 then FName := FName + '.EXE';
  { Default name is TEMP.EXE }
  { 如果没有目标文件名,则假定目标文件名为 TEMP.EXE }
  if FName2 = '' then FName2 := 'TEMP.EXE';
  inherited Init(FName);
  Assign(F2, FName2);
  Rewrite(F2, 1);
end;
destructor TAppendFile.Done;
begin
  Close(F2);
  Inherited Done;
end;
{ 利用 BrowseSegmentTab 函数遍历段表中的每一项,
然后修改位置在启动段之后的段的偏移 }
procedure IncSegOfs(PSelf: PNeHeader; P: Pointer;Index: word); far;
begin
  with PAppendFile(PSelf)^, PSegmentEntry(P)^ do
    if ofs > CurrOfs then Inc(Ofs, AddPages)
end;
{ 利用 BrowseResourceTab 函数遍历资源表中的每一项,
然后修改位置在启动段之后的资源的资源项中的偏移 }
procedure IncResOfs (PSelf: PNeHeader; P: Pointer); far;
begin
  with PAppendFile(PSelf)^, PEachRes(P)^ do
```

```
    if (CurrOfs * GetPageSize < LongInt(NameInfo^.rnOffset) * GetResPageSize) then
      Inc(NameInfo^.rnOffset, AddPages * GetPageSize div GetResPageSize);
end;

procedure TAppendFile.Run;
const
  AddLen = $100;
var
  N, oldIP, RelocNum: Word;
  Len: LongInt;
  P: PSegmentEntry;
  PMySeg, PBSeg, PBRes: PBuf;
  PCode: PBytes;
begin
  N := neHeader._CS;                    { Get boot segment number }
  P := GetSegmentEntry(PSeg, N);        { Get boot segment entry  }
  CurrOfs  := P^.Ofs;
  { Get boot segment relocation items entries }
  RelocNum := GetRelocItemCount(N);
  { 计算需要增加的逻辑页数量 }
  Len := AlignSize((P^.lenSeg + 2 + RelocNum * 8 + AddLen),
                   GetGeneralPageSize)
       - AlignSize((P^.lenSeg + 2 + RelocNum * 8), GetGeneralPageSize);
  AddPages := Len div GetPageSize;

  oldIP := neHeader._IP - P^.lenSeg - 3;
  neHeader._IP := P^.lenSeg;

  { 计算快装区域 }
  with neHeader do
    if GangLoad then begin
      if P^.Ofs < ofsGangLoad then Inc(ofsGangLoad, AddPages);
      if (P^.Ofs >= ofsGangLoad) and
          (P^.Ofs <= OfsGangLoad + lenGangLoad) then
        Inc(lenGangLoad, AddPages);
  end;

  { 修改资源表中的资源项 }
  if neHeader.lenResourceTab <> 0 then begin
    NewPBufEx(PBRes, neHeader.lenResourceTab, PResource);
    BrowseResourceTab (PBRes, IncResOfs);
  end;

  { Modify current segment entry }
  Inc(P^.lenSeg, AddLen);
  Inc(P^.lenMem, AddLen);
```

```
{ 修改段表 }
NewPBufEx(PBSeg, neHeader.lenSegmentTab, PSeg);
BrowseSegmentTab(PBSeg, IncSegOfs);

{ Copy WINSTUB }
CopyFileBlockEx(F, 0, F2, 0, ofsNE);

{ Write header }
BlockWrite(F2, neHeader, $40);

{ Write segment table }
BlockWrite(F2, PBSeg^.P^, PBSeg^.Len);
DisposePBuf(PBSeg);

{ Write resource table }
if neHeader.lenResourceTab <> 0 then begin
  BlockWrite(F2, PBRes^.P^, PBRes^.Len);
  DisposePBuf(PBRes);
end;

{ Copy some codes between resource table and boot segment }
Seek(F, FilePos(F2));
CopyFileBlock(F, F2, LongInt(CurrOfs) * GetPageSize-FilePos(F2) );

{ Restore current segment entry }
Dec(P^.lenSeg, AddLen);
Dec(P^.lenMem, AddLen);
PMySeg := ReadASeg(N);
BlockWrite(F2, PMySeg^.P^, PMySeg^.Len);
DisposePBuf(PMySeg);

{ * * * Write new append codes * * * }
GetMem(PCode, AddLen);
FillChar(PCode^, AddLen, ' ');
PCode^[0] := $E9;
PWord(@PCode^[1])^ := oldIP;
BlockWrite(F2, PCode^, AddLen);
FreeMem(PCode, AddLen);

{ Copy relocation list }
PBSeg := ReadARelocList(N);
BlockWrite(F2, RelocNum, 2);
BlockWrite(F2, PBSeg^.P^, PBSeg^.Len);

{ Align two files }
Seek(F, AlignSize(FilePos(F), GetPageSize));
WriteSeek(F2, AlignSize(FilePos(F2), GetPageSize));

{ Copy left codes }
CopyFileBlock(F, F2, FSize-FilePos(F));
```

```
end;

const
  FirstSetName: boolean = true;
var
  I: Word;
  FName, FName2, S: String;
  PAppend: PAppendFile;
begin
  FName := '';
  FName2 := '';
  if ParamCount > 0 then begin
    { Get file name }
    I := 0;
    repeat
      Inc(I);
      S := ParamStr(I);
      if (not (S[1] in OptionChars)) then
        if FirstSetName then begin
          FName := S;
          FirstSetName := False;
        end
        else FName2 := S;
    until (I = ParamCount);
  end;

  Writeln('Power Append Version 1.0 Copyright (c) 1993 KingSoft Ltd.');
  if FName = '' then begin
   Writeln('Usage: PAppend <ExeFile>');
   Writeln('       PAppend <SourceExe> <TargetExe>');
   Halt(1);
  end;
  PAppend := New(PAppendFile, Init(FName, FName2));
  if PAppend = nil then
    DisplayRtError
  else begin
    PAppend^.Run;
    Dispose(PAppend, Done)
  end;
end.
```

4.5.3 增加一个新的重定位项

如果需要在某一段中增加一个重定位项，只需要把该段重定位项计数器加一，再把这个重定位项写入在该段的重定位表后面，最后调整段表和资源表中相应的内容。

4.5.4　插入一个新的段

在很多时候，我们需要在原来的程序中插入一个新的段。比如，自装载段必须放在第一段，如果需要把一个不是自装载的程序变成自装载的程序，就必须把自装载代码插入到第一段的前面。

插入一个新的段，操作比较复杂。

Step 1.　段表：增加新段的段表项，修改相应的段表项，首部段表的项数加一；

Step 2.　自动段段号：如果这个段在自动数据段的前面，这自动数据段的段号应该加一；

Step 3.　启动段段号：如果这个段在启动段的前面，则将入口段的段号加一；

Step 4.　堆栈段段号：如果这个段在堆栈段的前面，则将堆栈段的段号加一；

Step 5.　模块引用表：新增加的段需要增加新的模块引用表，修改模块引用表的项数；

Step 6.　入口表：新增加的段需要增加新的入口项，修改入口项的个数；

Step 7.　快装区域：计算快装区域；

Step 8.　资源表：修改资源表的每项的值；

Step 9.　表的入口偏移：调整入口表、段表、资源表、模块引用表、输入名表、非驻留名表等各种不同的表的入口偏移值；

Step 10.　插入新的段：根据给定的位置插入新的段。

以上我们讲了几种经常要作的修改，“增加一个新的入口项”等其他的操作与上述的操作大体一致，读者可参考以上的小节，我们就不详细讲了。

4.6　调试 Windows 程序的技巧

深入 Windows 核心，必须依靠强大的调试器及各种调试工具。在修改 Windows 执行文件的过程中，会引入一些错误，为了解决这些问题，必须熟练使用调试工具。同时，不少读者特别关心如何调试 Windows 程序，尤其是 Windows 核心程序。本节的重点是讲解调试技巧。

调试高级语言编写的程序，可以使用 Turbo Debugger for Windows 及 CodeView for Windows 等调试器调试。调试 Windows 的核心代码，需要使用 Windows SDK 提供的 Kernel Debugger，即 WDEB386.EXE。但麻烦的是 WDEB386 显示的内容只能输出到一个终端上，必须配备一个终端。在我们开发 PACKWIN 的初期，就是使用 WDEB386 来调试的（关于 WDEB386 的使用方法可以参见清华大学出版社出版的《Microsoft Windows 3.1 程序员参考手册》）。现在调试 Windows 程序，相对来说，比较容易，可以使用 SoftICE for Windows 软件，可以很方便地调试。如果现有的工具还不能满足要求，读者还可以自己开发。我们为了开发 PACKWIN，就同时编写了 PDUMP 和 PFI 等一组工具。

下面介绍 Windows 调试软件及有关的工具。

4.6.1 SDK 中的几个工具

首先我们看一下现有 SDK 中的工具。尽管 SDK 中的工具已不再是开发 Windows 应用程序的必备工具，但也是每一个 Windows 程序员不应该忽视的，如 HEAPWALK、SPY、Code View for Windows、WSEB386、和 Windows 调试版。

HEAPWALK

HEAPWALK 能列出所有 Windows Global Heap 所包含的内存空间，同时它还会显示每个 Global 内存块的 Handle、线性基地址、大小(以 byte 为单位)、配置、拥有者的模块名、和(在某些情况下)该 Global 块的类别(如："Module Database"、"Task"、"Code"、和"DGROUP"等)。如果该 Global 内存块包含有 Local Heap，那么你还可以对该 Global 内存块做 LocalWalk，这时会有另一个窗口被生来显示该 Local Heap 的数据。在 HEAPWALK 的主窗口中，你可以在任何一个 Global 内存块上双点鼠标标左键，这时 HEAPWALK 会产生另一个窗口，显示 Global 内存块的原始数据，可惜同样的动作在 LocalWalk 的窗口中不会有作用。例如：如果你在一个标示为"Task"的 Global 内存块上双点鼠标左键，那么 Task Database 的原始数据会出现在第二个窗口中：即使你对 Task Database 很陌生，但是你仍可以在原始数据中找到档案路径、该 TASK 所属的模块名、"TD"代表号、和"PT"代表号。HEAPWALK 在标示其所认识的 Global 内存块表现很称职，它甚至会显示在 Windows 调试版中包含有调试符号表格(.SYM 文件)的模块(如：KERNEL、USER、和 GDI 模块等)所拥有内存块的名称。然而，它会将 Task Queue 数据结构标示成"USER Private"，同时在销售版 Windows 中，它也不认识 WND 窗口结构，更不用说那些不属于 Windows Global Heap 中的选择子(如：指向 PSP 的选择子)。虽然 HEAPWALK 提供"GDI LocalWalk"和"USER LocalWalk"选项，但它们只能"游走"于 GDI 和 USER 模块的预定的数据区(DGROUP)，而无法"游走"在 Windows 3.1 中为了减缓系统可用数据问题所新增的 Local Heap。

SPY

工具程序 SPY 会显示出某个窗口的数据：如果 SPY 的菜单条中选择"Windows"选项，那么出现一个对话窗，其中显示着目前鼠标所指的所在窗口的数据(如：窗口的 Handle、拥有窗口的模块名、父窗口、窗口样式、和窗口别名称等)。

例如，SPY 会显示 WinWord 1.x 版的主窗口的窗口类别名称为"OpusApp"、Visual Basic 的窗口类别名称为"ThunderForm"、和 CorelDRAW 的模块名为"waldo"等，而这些数据对于需要和其它程序联系的 Windows 程序而言是非常重要的。

除了显示有关窗口的数据以外，SPY 最主要的目的是被用来跟踪 WM_消息，我们可以利用它来视察因种种不同连作而产生的消息。然而，SDK 的 SPY 无法显示未公开的 WM_消息，如：WM_SYNCPAINT (0x022C)等。

程序员有时会利用 SPY 或类似的程序来判别某个程序(如：Program Manager)所使用的 Menu ID，如此就可以 subclass 那个程序；例如，经由视察菜单中每个选项所引发的 WM_COMMAND 消息的 wParam 值，你可以得到某个程序每个选项的 Menu ID。然而，以类似 SPY 的工具程序来做这个工作似乎有点小题大作；因为通常这些 Menu ID 都是程序菜单资

源的一部分，所以我们可以直接用 Resource Workshop 工具程序来得到相同的数据。

CodeView for Windows

CodeView for Windows 是属于源程序级的 Windows 调试器。尽管 CodeView 在使用上不是很方便，但如果在没有其它调试器可用的情况下，那么它还可以被用来探索 Windows 的内部运行。如果在程序中调用未公开函数，那么你可以按下 F8 键进入到该未公开函数中跟踪。如果你知道未公开函数的地址，那么你可以利用指令 U 来反汇编该函数；同样的，如果你知道未公开函数的地址，那么你可以在该函数上设定一个中断点。要如何找出 API 函数的地址呢?! 你可以写一个简单的 Windows 程序，由此调用函数 GetProcAddress 来得到任何你感兴趣的函数地址。

WDEB386

如果你对 Windows 内部机制(尤其是增强模式)感到很有兴趣，那么 WDEB386 是一个比 CVW 功能强得多的调试器，所有好的东西都在其中！ 例如，下面就是几个 WDEB386 支持的指令：

.dm　显示模块的列表
.dq　显示 Task Queue 列表
.dg　显示全域块描述表(GDT)
.di　显示中断块描述表(IDT)
.dl　显示区域块描述表(LDT)
.dp　显示 386 的分页目录和分页表格数据
.dt　显示 386 的 task 状态节区(TSS)
.dx　显示 LOADALL 缓行区
.?　显示 WIN386.EXE 调试版所支持的指令

但不幸的是，WDEB386 需要调试终端，而且在使用上很困难；而这些问题在 NuMega 的 Soft-ICE/Windows 中都不会出现。

调试版 Windows

Windows SDK 最重要的组成部分可以说是 Windows 调试版，其中的 KRNL286、KRNL386、USER、GDI、和 MMSYSTEM 等模块都包含有经由 CodeView 建立的调试符号。但不幸的是，有关于未公开函数和内部数据结构的 CodeView 调试符号，在 SDK 中都被有意地去掉了。

在某些情况下，Windows 调试版的内容数据结构会比销售版中的来得多一点。有关这些特殊的情况，在书中介绍到该结构体时会加以提醒。在程序中我们可以利用 GetSystem-Metrics(SM_DEBUG)来判断程序是否在调试版中执行。

如果是在 Windows 调试版中执行。那么 ToolHelp 函数库能够识别存放在 USER 模组预定数据区中的窗口、菜单等；同样的，HEAPWALK 也能够显示 KERNEL、USER 和 GDI 等模块的名称，并认得 USER 模块预定数据区中的对象类别。同时，某些 API 函数只能够在 Windows 调试版中使用，并不支持销售版。

其它的工具程序

如果你拥有 Borland C++，那么你并不需要 SDK 来开发或调试 Windows 程序，因为 Borland C++ 包含所有开发 Windows 程序所必须具备的工具。然而，Borland C++ 并未提供 3.1 SDK 所包含的完整文件，也没有调试版 Windows 和一些 SDK 所提供的工具程序（如：HEAPWALK）。也许目前开发 Windows 程序最好的方法是搭配 SDK 与 Borland C++ 一起使用。

Borland C++ 提供给使用者调试 Windows 程序的工具是 TDW（Turbo Debugger for Windows）。和 CVW 一样，你可以利用 TDW 来反汇编、设定中断点、和进入 API 函数中单步执行（包含已公开和未公开）；当然，通常你必须先得到 API 函数的地址。

Borland C++ 所提供的 Resource Workshop 是一个非常好用的工具，通过它我们可以反汇编、改变和产生有关交谈窗（Dialog）、Menu、图标（Icon）、鼠标光标（Cursor）、位元映射图（Bitmap）、和字体（Font）等资源。

Borland C++ 3.1 所提供的 WinSpector 是一个类似 Dr. Watson 的调试程序，但功能上 WinSpector 远比 Dr. Watson 来得强。一个可视为 WinSpector 附属功能的应用程序 BUILDSYM 可被用来为没有调试符号数据的执行档产生 .SYS 档案；而 BUILDSYM 之所以能够『无中生有』的为某个执行档产生调试符号，主要是靠 NE 档案格式中所提供的数据。

此外，为了观察 WM_ 消息，Borland C++ 选提供 WinSight，这个程序的功能也远比 SPY 来得强。WinSight 认得大约 25 个未公开的 WM_ 消息，而且它会用不同于已公开消息的方式来显示这些未公开消息，以期让这些未公开消息能够突现出来；同时，它也认得那些经由 RegisterWindowMessage 加以注册过的新消息。WinSight 会利用输出的层次效果来显示消息处理的巢状情况。

通过 WinSight 的"Messages"|"Options"|"Other"选项，我们可以观察未公开的消息。WinSight 除了会将认得的未公开消息名称显示出来以外，对于那些不认得的未公开消息，它会显示出该消息的常数；例如：它会显示消息 WM_SETHOTEKEY 为 WM_0x0032。有趣的是，WinSight 会将消息 WM_OTHERWINDOWCREATED 和 WM_OTHERWINDOWDESTROYED 加上双引号并显示成"OTHERWINDOWCREATED"和"OTHERWINDOWDESTROYED"，以表示这些是经由 RegisterWindowMessage 注册过的消息。

4.6.2 Soft-ICE/Windows

NuMega Technologies 发行的 Soft-ICE/Windows（WINICE）可说是每个 Windows 程序员必须具备的工具，它同时提供调试 Windows 程序和探索 Windows 内部结构的功能，足以取代所有前面介绍过的工具。

WINICE 本身就是一个完整的调试环境，一旦拥有了它，你就不再需要 CVW、TDW、WDEB386、HEAPWALK 或 SPY，同时也可以省下调试终端。在这里我们主要讲述用 WINICE 探索 Windows 核心机制的功能。WINICE 利用了内建在 386 和 486 CPU 中的调试功能；在其它低档 CPU 上同样的功能必须借助 ICE（in-circuit emulator，一种模拟硬体的设备）才能做到。例如，WINICE 利用分页功能和 386 调试寄存器来完成实时内存存取的中

断点(有其它几个调试器也提供相同的功能),利用 386 虚拟 I/O(Virtual I/O)来完成设定在 I/O 端口上的中断点,和其他等等。

WINICE 并不是一个 Windows 程序;当你在 DOS 下执行 WINICE 之后,它会自动将 Windows 以增强模式载入(WINICE 不支持标准模式),而后随时按下设定的热键(预设为 Ctrl + D)就可以进入 WINICE 中。这一点和其他的调试器不同(例如,只有在调试某个程序的情况下才能使用 CVW 或 TDW),我们不一定要调试某个程序才能使用 WINICE 的。

利用 WINICE 来反汇编

和 CVW 及 TDW 一样,WINICE 可以用来反汇编未公开的 API 函数,但是其中的差别在于 WINICE 认得 KERNEL、USER、GDI 模块的函数(已公开或未公开)所包含的调试符号。

因为 Windows 能在 WINICE 之上执行,所以我们可以观察那些在反汇编码中出现的参数。例如,在上面 GetCurrentTask 的反汇编码中,我们看到此函数有一个未公开的传回值在 DX 中。我们推测这个值应该是 task 列表中的第一个 Task Handle,现在就让我们来实际测试一番吧! 首先 GetCurrentTask 会将存放在 CS 中位移 30H 的值载入到 ES 中(是的! 即使中保护模式的 Windows 也将数据存放在数据区中):

```
:dw 11f:30
011F:00000030 0137…
```

接下来我们就可以观察位于 ES:[0226]的值,并判断该值是否是一个 task handle(接下来我们已经知道一个有效的 task handle 必须在位移 F2H 的地方存放着一个模块名,而且后面紧接着代表号"TD",再后面是一个 PSP):

```
:dw 137:226
0137:000000226 0807 ...      ;可能的 task handle
:db 807:f2
0807:000000F2 57 49 4E 46 49 4C 45 00-54 44 00 00 00 00 CD 20     WINFILE. TD...
```

果然没错! ES:[0226]的确是一个有效的 task handle,接着我们判断存放在它最前面的 WORD 值是否为另一个 task handle,以决定它是否为 task 列表中的第一个 task handle:

```
:dw 807:0
0807:00000000 14AF
:db 14af:f2
14AF:000000F2 57 49 4E 4F 4C 44 41 50-54 44 00 00 00 00 CD 20     WINDLDAPTD...
```

的确是! 我们可以重复这个动作直到我们到达任务列表的尾端:

```
:dw 11bf:0
11BF:00000000 0000...
```

透过 WINICE,你可以载入任何 DLL 或设备驱动程序(例如:SYSTEM、DISPLAY、KEYBOARD、MOUSE、和 COMM 等)的调试符号,而不是需要特别的调试版本,因为 WINICE 会自动从 NE 首部取得数据(这一切都要归功于"NE"格式)。例如,下面反汇编函

数 EnableSystemTimers 的结果透露出：这个掌管 Windows 定时器运行的基本函数也只不过是透过 DOS INT 21H AH = 25H(设定中断向量)系统调用来安装的 INT 8 中断处理常式：

```
:u enablesystemtimers
SYSTEM! ENABLESYSTEMTIMERS
0157:000002E2   PUSH   DS
0157:000002E3   MOV    DS,CS:[0000]
0157:000002E8   CMP    BYTE PTR [005C],00
0157:000002ED   JNZ    030B
0157:000002EF   MOV    BTYE PTR [005C],01
0157:000002F4   MOV    AX,3508
0157:000002F7   INT    21
0157:000002F9   MOV    [005D],BX
0157:000002FD   MOV    [005F],ES
0157:00000301   MOV    AX,2508
0157:00000304   PUSH   CS
0157:00000305   POP    DS
0157:00000306   MOV    DX,0238
0157:00000309   INT    21
0157:0000030B   POP    DS
0157:0000030C   REF
```

以同样的方法，我们可以反汇编另一种 Windows API 函数：VxD 函数。VxD 函数调用，尤其是那些 Windows VMM(Virtual Machine Manager，虚拟机管理器)所提供的函数。对于 WINICE 而言，这些 VxD 函数调用和一般的 Windows API 函数调用并没有什么不同：使用 SDK 和程序设计师可以直接测试这些原本只有在 DDK 模式下才能使用的函数调用：

```
:u get_cur_vm_handle
0028:800081AC   MOV    EEX,[80012944]
0028:800081B2   RET

:u get_sys_vm_handle
0028:800081bc   MOV    EBX,[80012948]
0028:800081C2   RET
```

不知道你是否已经注意到 WINICE 可以处理 32bit 的程序码(EBX!!)。顺带提一下，如果你已经听到许多未来 32bit Windows 程序设计的传说，那么你或许想知道：事实上现在你就已经开始练习 32bit 的程序设计，VxD(虚拟设备驱动程序)就是 32bit 的程序。

如果你透过 WINICE 来调试某个程序，那么你一定会需要该程序所有的调试符号；没问题！ WINICE 认得 CodeView 和 Turbo Debugger 的调试符号格式。而被调试的程序可以是一个 Windows 程序、DLL、设备驱动程序、VxD、执行于 DOS 窗中的 DOS 程序、或是一个 16bit 或 32bit 保护模式的 DOS 程序。

WINICE 的中断点

到目前，我们只使用到 WINICE 的反汇编功能；然而，WINICE 最强的一面并不在于反

汇编代码，上面所提到的大部分工作也可以由其它的工具来完成(如：Windows Sourcer)，而真正让 WINICE 胜过其它工具程序的是它支持多种中断点的设定：

BPX	设定执行中断点
BMSG	设定 Windows WM - 消息中断点
BPINT	设定中断的中断点
BPR	设定内存存取的中断点
BPRW	设定 Windows 模块或选择子中断点
BPMB, BPMW, BPMO	设定内存存取中断点
BPIO	设定 I/O 端口存取中断点
CSIP	指定断点的范围

例如，下面的动作会使得所有调用 USER 模块 TáskQueueES 的函数调用都被拦截下来：

```
:csip user
:bpx gettaskqueuees
:x
```

每当中断点被引发后，你可以透过命令 STACK 来观察函数是怎么被调用的(如果当时有一个有效的堆栈区，那么在其中会有参数和返回值等数据)，或是浏览此函数的代码。透过这种方式，你可以在一个正常执行的 Windows 环境中，观察一个或多个 Windows API 函数被使用频率的高低。

再举个例子，透过 WINICE 要知道 GetProcAddress 取哪一个函数的地址是轻而易举的事：

```
:bpx getprocaddress
```

更棒的是，你可以将设定在 GetProcAddress 第一指令上的中断点往下挪到 GetProcAddress 将参数装到寄存器中的地方，再加上命令 DEX(disply expression)；如此一来，WINICE 会在每次中断点被引发的时候，将传入的函数名称显示出来。

就研究未公开函数调用而言，WINICE 也扮演着很重要的角色。尽管一个反汇编器(如：Sourcer)能给我们很详细的程序码，但这些只不过是静态分析的结果。想要完全了解某个函数的运行，有时还是必须在执行时对该函数的行为加以分析才行。而 WINICE 正是可以被用来对某个函数做动态方面的分析，特别是当它和一个函数调用直译器(《Undocumented Windows》一书中介绍的 CALLFUNC)搭配在一起使用时，效率会更高。

如果要印证上面所说的，那么研究未公开函数 SysErrorBox 的运行会是一个很好的例子：因为通常只有在系统发生严重错误时，SysErrorBox 才会被使用到。因此，为了要分析它的操作，你可能会利用 CVW 或 TDW 来调试一个刻意产生严重错误的程序，然后进入该错误中跟踪；然而，这两个调试器会将发生的严重错误拦截下来，并结束被调试程序的执行，根本不让你继续调试下去。另一个办法是，你直接在程序中调用 SysErrorProc(但这时候你并不知道参数正确的型别，所以只好随意假设一番)，并将中断点设定在调用此函数的地方，然后一次又一次的尝试不同的参数类型。可是你会发现到，传递给 SysErrorBox 不正确的引数会造成系统处于不稳定的状态，而且很有可能造成系统当掉。

现在来看看我们的作法。首先,自 Sourcer 反汇编的程序码中,我们发现到 SysErrorBox 最后是以指令 RETF 0Eh 返回,表示清除掉堆栈中 7 个 WORD 的数据;这 7 个 WORD 就是调用 sysErrorBox 的引数(因为 Pascal 调用协定是由被调用函数负责清除堆栈中的参数)。由于 Sourcer 反汇编的代码长达 13 页,所以要自其中找出线索并不是那么容易,于是我们回头采用上面尝试参数类型的方法。或许这种方式不是很正确,但至少它可以让我们先了解到要如何正确使用 SysErrorBox。接下来利用 CALLFUNC(一个可以直接键入函数名称并加以调用的工具程序)来调用 SysErrorBox(虽然知道引数总线共为 14 bytes,但我们仍不知道其型别,所以传给它 7 个大小写 WORD 的 0):

```
> user syserrorbox 0 0 0 0 0 0 0
```

在按下 Enter 键之前,我们必须在 SysErrorBox 上设定一个中断点,这样我们才能够进入函数中分析其运行。很简单,只要按下切换至 WINICE 的热键并进入:

```
:bpx syserrorbox
:x
```

现在回到 CALLFUNC 并按 Enter 键。几乎是同时,我们又回到 WINICE 中,画面上同时显示着 SysErrorBox 的程序码。按下 F10(WINICE 的按键和 Code View 的很相似)我们会看到这指令:

```
MOV AX,[BP + 10]
MOV DX,[BP + 12]
```

对一个期待有 14 bytes 引数在堆栈中的 FAR PASCAL 函数而言,这两个指令意谓着位于 BP + 12h 的第一个参数不是一个 DWORD 值就是一个远指针(pascal 调用规定是将参数压入堆栈时,按照参数顺序由左至右进行的)。接下来我们利用命令 R(设定寄存器的值)将 IP 移至接近函数返回的地方再继续跟踪;如此一来,当 SysErrorBox 返回至 CALLFUNC 时就不会造成 GP Fault,同时我们也不会因为传递给 SysErrorBox 不正确的参数类型而造成系统不稳定或甚至死机。接着我们再透过 CALLFUNC 重新调用 SysErrorBox 一次,但这次我们将第一个参数改为字符串(大小为 4 bytes 的远程指针):

```
>user syserrorbox "Hello word" 0 0 0 0 0
```

同时,这次我们也可经以更深入一点跟踪函数的运行。经过几次同样步骤的测试后,我们会进展到在 CALLFUNC 中以下面的方式调用 SysErrorBox:

```
>user syserrorbox "Caption?" "Some text" 1 2 3
```

到这里就表示我们离成功已经不远了。

的确,上面将实际测试的过程简化了许多,但最主要的目的是要让各位了解如何将几个工具搭配在一起使用。我们相信还有其它的方法可以帮助你了解每个 Windows 已公开和未公开函数的运行过程,但是上面我们介绍的方法:结合反汇编器所产生的结果、一个类似 WINICE 这样有弹性而且可靠的调试器、和类似 CALLFUNC 这种可做为测试温床用的函数调用直译器,相信会为这个工作带来更多的乐趣,并且大幅减少重新开机的次数。

接下来,我们要介绍如何使用 BPR 来设定记忆体存取中断点(白话黑说,就是将中断点

设定在某个内存范围内，而后如果有任何存取内存范围的动作发生，那么中断点就会被引发)。举例来说，假设你想要知道除了 KERNEL 模块外，是否利用命令 TASK(后面会介绍到)来得到所有执行 task 的 task handle，然后在某些个 Task Database 上设定记忆体存取中断点(透过 BPR)；一个 Task Database 的大小为 200h bytes，并假设 807h 是一个合法的 task handle(指向 Task Database 的选择器)。

```
:bpr 807:0 807:1ff rw
:csip not kernel
:x
```

一旦有任何不属于 KERNEL 模块和程序码对这块大小为 200H bytes 的内存块进行存取动作，我们就会进入 WINICE 中；但在这个情况发生之前，Windows 是以全速在运行，不会受到任何影响。

最后，我们要介绍如何透过 BPINT 所设定的中断点来观察 Windows 内部所产生的中断。例如，如果我们想要知道 Windows 是否会调用被保留的 DPMI　INT　31H　AX = 0701(抛弃某页内存)函数调用，那么我们可以透过 WINICE 来得到答案：

```
:csip off
:bpint 31 ax = 0701
:x
Break Due to BPINT 31 AX = 0701 C = 01
011F:00003BB6   INT   31
```

嗯！看来此函数真的是有被使用到。但是谁调用此函数呢？也就是说，011FH 是在那里？

```
:heap 11f
Han./Sel. Address   Length    Owner    Type    Seg/Rsrc
011F      00018440  0000CC80  KERNEL   Code    01
```

原来是 KERNEL 模块调用被保留的 DPMI 函数。

WINICE 提供的系统信息命令

WINICE 提供许多有关系统信息取得的命令，前面所使用的 HEAP 就是其中一个：

HWND	显示所有窗口的 Handle
CLASS	显示窗口类别的信息
TASK	显示 Windows 的任务列表
MOD	显示 Windows 的模块列表
HEAP	显示 Windows Global Heap 的信息
LHEAP	显示 Windows Local Heap 的信息
VM	显示虚拟机器(Virtual Machine)的信息
VXD	显示虚拟设备驱动程序(VxD)的信息
GDT	显示全局段描述表(GDT)的信息
LDT	显示局部段描述表(LDT)的信息
IDT	显示中断描述表(IDT)的信息

TSS 显示 task 状态节区(TSS)的信息
包括 I/O 映像(对应至实际 I/O 端口,以完成虚拟 I/O)
PAGE 显示分页目录(page directory)和分页表格信息
MAP 显示虚拟机内存对应的信息

例如:

```
:task
TaskName SS:SP     StackTop StackBot Stacklow TaskDB hQueue events
WINOLDAP 146F :105A 1E82    0B18     16D4     14AF   1497   0000
WINFILE  17AF :3716 377C    1012     2546     0807   06BF   0000
DRWATSON 12CF :2A92 3834    1BCA     25FA     12F7   12DF   0000
TASJMAN  1207 :13A4 144E    00E4     0E48     1217   122F   0000
SH       * 1347 3BCC 3C54   28EA     378A     136F   1357   0000
```

利用命令 HEAP,我们可以得知拥有 Task Database 的就是程序本身:

```
:heap 14af
Han./Sel. Address  Length    Owner      Type    Seg/Rsrc
14AF    00025940  00000200  WINOLDAP  TaskDB
```

但是,和 HEAPWALK 所显示的结果一样,所有 Task Queue 结构的拥有者是 USER 模块,尽管使用到此结构的代码大部分都在 KERNEL 模块中。这个结果同时反映出 Windows 对于 KERNEL 和 USER 模块的区分不是很明确:

```
:heap 1497
Han./Sel. Address  Length    Owner     Type    Seg/Rsrc
1497    00025B40  00000120 USER      Alloc
```

透过命令 GDT,我们可以 Windows 内存块地址的对应建立一个清楚概念:

```
:gdt
GDTBASE = 800715BC  Limit = 010F
0008  Code16   Base = 000157F0  Lim = 0000FFFF  DPL = 0  P RE
0010  Data16   Base = 000157F0  Lim = 0000FFFF  DPL = 0  P RW
0018  TSS32    Base = 8000DD74  Lim = 00002069  DPL = 0  P B
0020  Data16   Base = 800715BC  Lim = 0000FFFF  DPL = 0  P RW
0028  Code32   Base = 00000000  Lim = FFFFFFFF  DPL = 0  P RE
0030  Data32   Base = 00000000  Lim = FFFFFFFF  DPL = 0  P RW
003B  Code16   Base = 804A5B20  Lim = 000004CF  DPL = 0  P RE
0043  Data16   Base = 00000400  Lim = 000002FF  DPL = 3  P RW
0048  Code16   Base = 00013290  Lim = 0000FFFF  DPL = 3  P RE
0053  Data16   Base = 00000000  Lim = FFFFFFFF  DPL = 0  P RO
005B  Data32   Base = 804A6000  Lim = 00000FFF  DPL = 3  P RW
0060  Code32   Base = 80059460  Lim = 00001000  DPL = 3  P RE
0068  Code32   Base = 000157f0  Lim = 00001000  DPL = 0  P RE
0073  Data16   Base = 000157f0  Lim = 00000100  DPL = 3  P RW
0078  Code16   Base = 000157f0  Lim = 00033FFF  DPL = 0  P RE
```

```
0080   Data32    Base = 000157f0   Lim = 00033FFF   DPL = 0   P RW
0088   LDT       Base = 000157f0   Lim = 00001FFF   DPL = 0   P
0093   Data16    Base = 00000000   Lim = FFFFFFFF   DPL = 3   P RW
009B   Data32    Base = 805D0000   Lim = 00000FFF   DPL = 3   P RW
00A0   Data32    Base = 00000000   Lim = 00000000   DPL = 0   NP
       …大约省略掉十几个未用区的描述(descriptor)…
```

从上面的结果，我们可以得知：在 Windows 增强模式中，选择器 28H 是一对应至整个线性空间(0 至 4GB)的程节区，而选择器 30H 则是对应至整个线性地址空间的数据区。选择器 40H(上面显示的 43H 表示 aDPL = 3)对应至线性基地址为 400H 且长度为 2FFH bytes 的段；换句话说，40H 是一个对应至 BIOS 数据区的透明选择器，因为『线性基地址 = = 选择器＜4』。选择器 20H 对应至 GDT 本身，而选择器 88H 则指向一个 LDT。

正如我们在前面提到的，WINICE 可以取代 Microsoft 的 WDEB386，当它和附在 DDK 中的调试版 WIN386.EXE 一起使用时，使用者可以直接使用 WIN386 所提供的调试命令：

```
:.?
.VM[#]          显示目前虚拟机的状态
.VC[#]          显示目前虚拟机的控制块(control block)数据
.VH             显示目前虚拟机的 handle
.VR[#]          显示目前虚拟机的寄存器(只有在保护模式下才用)
.VS[#]          显示目前虚拟机的虚拟堆栈(只有在保护模式下才用)
.VL             显示所有有效虚拟机的 handle
.T              切换跟踪(trace)状态
.S [#]          显示有关发生例外的数据，可以用#来指定从那个项目开始显示
.SL[#]          功能同上，但显示数据较详细，同时#是指定只显示那个项目
.LQ             显示最近的 queue-outs 数据
.DS             列出保护模式堆栈中的原始数据
.MH[handle]     显示 handle 的 heap 数据
.MM[handle]     显示 handle 的内存数据(如：大小、线性地址等)
.MV             显示目前虚拟机所配置的内存数据
.MS PFTaddr     显示 PFT(page memory，分页内存)的数据
.MF             显示目前可用内存块的列表
.MI             显示 instance data 的数据
.ML LinAddr     显示线性地址 LinAddr 的分页表格(page table)数据
.MP PhysAddr    显示所有对应至实际内存地址 PhysAddr 的线性地址
.MD             改变调试屏幕显示页
.MO             为所有未被 lock 的 present page(实际存在的内存)安排一个
                page out 事件
.VMM            显示 VMM(虚拟机管理器)的功能选择
.＜dev_name＞   显示虚拟设备 dev_name 的信息
```

例如，命令.VMM 会发生下面的画面：

```
:.vmm
```

```
VMM  DEBUG  INFORMATIONAL  SERVICES
[A] System time
[B] Time-silice information/profile
[C] Dyna-link service profile information
[D] reset dyna-kink prifile counus
[E] I/O port trap information
[F] Reset I/O profile counts
[G] Trun procddure call trace ligging on
[H] V86 interrupt hook information
[I] PM interrupt hook information
[J] Reset PM and V386 interruot ptofile counts
[K] Display event lists
[L] Displu device list
[M] Disply V86 break points
[N] Disply PM break points
[O] Disply interrupt trofile counts
[P] Reset interrupt profile counts
[Q] Display GP fauot profile
[R] Reset GP fault profile counts
[S] Toggle Adjust - Exec - Priority Log and disply
[T] Reset Adjust - Exec - Priority Log info
[U] Toggle verbose device call trace
[V] Fault Hook information

:v86mmgr
Select desired V86MMGR component:
[0]  -General info
[1]  -Memory scan info
[2]  -EMM driver info
[3]  -XMS driver info
[Esc]-Exit V86MMGR debug query

:.dosmgr
Select desired DOSMGR funcition:
[0]   Display DOS trace info
[1]   Set DOSMGR queue - outs
[Esc] Exit DOSMGR debug query
```

部份的.VMM 选项可被用来观察那些 Windows 视为正常运行所产生的中断和例外的实际数目，而这个结果有时会令人感到怀疑。在这么多刻意产生的内部中断和例外中，到底 Windows 是如何来完成任务的呢？例如，Windows 为了要将 ringlevel(可能值为 0～3，表示某个程序使用系统资源的权利，号码越小权利越高)自 1(3.0)或 3(3.1)切换成 0，或是反过来，会刻意产生 Fault(有关这方面的资料，在 WINICE 的使用手册中有很详细的讨论)。

再次的强调，WINICE 可完全取代 WDEB386 和 CVW 的作用；例如，在 Windows 调试版中，WINICE 会拦截所有的 RIP(rest in peace)程序码，提供给使用者选择处理方式的机会。为了达成这项功能，WINICE 会对 Windows 的 INT 41H 底层调试函数调用加以处理。

尽管 WINICE 所提供的功能有一部份会令人觉得很玄奥，但是透过它，你可以接触到 Windows 的全部。你可以用 WINICE 来对 VMM 做更深入的探索，或者只是将它当做是 CVW 或 TDW 的代替品；总之，在 WINICE 这个程序中，你可以找到所有 Windows 调试工具所具备的全部功能。

第 5 章

用汇编语言编写 Windows 应用程序

在 Windows 编程中，至少有以下两个理由说明用汇编语言编写 Windows 程序是必须的：一是涉及到一些低级操作，如针对 BIOS 的读写；另一种情况是在一些必须追求效率的场合，如编写压缩软件。

Windows 汇编与 DOS 汇编实质上没有什么区别，只是 Windows 汇编引入了一些新的宏定义；再者，Windows 汇编经常要调用系统函数，其中自然有一套传递变量的方法。这些变化正是本章的知识要点。

在 Microsoft 的相关文档中，我们看到了一些零散的有关“用汇编语言编写 Windows 程序”的说明，但这些内容只是教导读者在编程时应遵循的原则，并没有一个完整的示例。其实，用汇编语言编写 Windows 程序时要涉及到许多新概念，并非易如反掌的小事，相反，如果没有示例程序，原则描述得再清楚，具体编写起来仍然会觉得无从下手。本章提供的 HelloWin.asm，总共只有 200 多行，竟让笔者花了一个礼拜的时间！

在本章中，我们先分类介绍汇编语言宏指令集 CMARCOS.INC，然后详细讲解其中每一个宏指令的用法，接着给出用汇编语言编写 Windows 程序应遵循的原则，最后列出了一个完整的用汇编语言编写的 HelloWin。如果读者有较多的汇编语言编程经验，根据本书对汇编语言宏指令的讲述，并参照 HelloWin.asm 源程序，应该能比较顺利地用汇编语言编写出新的 Windows 程序。

因为汇编语言与机器语言是直接等价的，所以读者掌握用汇编语言编写 Windows 程序的方法后，就能读懂实际运行的代码，便于跟踪、调试，也是了解 Windows 核心的重要一步。另外，在本书第 8 章中，有一些程序是用汇编语言编写的。

5.1 汇编语言宏指令 CMACROS.INC

用汇编语言编写的 Microsoft Windows 应用程序必须高度结构化的，这样的程序要采用高级语言调用约定和 Windows 的函数、数据类型及编程约定。用宏汇编语言创建的 Windows 程序生成的目标文件要求与 C 编译器生成的目标文件相似。

为了能在汇编语言程序采用高级语言调用约定，Microsoft Windows3.1 的 SDK 及 Microsoft Visual C++ 1.5 等工具中提供了 CMACROS.INC 文件，该文件由各种宏指令组成，这些宏指令定义了创建 Windows 应用程序所需的段、编程模式、函数接口和数据类型。具体汇编程序中将要用到的内存模式和调用约定通过汇编选择项来定义。

在本节中我们介绍 Cmacros 宏指令组。

Cmacros 是一组汇编语言宏指令，Microsoft 宏指令汇编程序(ML)用它来创建汇编语言 Windows 应用程序。Cmacros 提供了高级语言(如 C)的函数和段约定的简化接口。

Cmacros 宏指令分为下列几组：

- 段宏指令
- 内存分配宏指令
- 函数宏指令
- 调用宏指令
- 特殊宏指令
- 错误处理宏指令

下面分别介绍上述宏指令组。

5.1.1　段宏指令

段宏指令用来对应用程序将要用到的代码和数据段进行访问。这些段具有 Windows 所需的名字、属性、类和组：

宏指令名	说　明
createSeg	创建一个具有指定名和段属性的新段
sBegin	打开一个段，这条宏指令与汇编指令 SEGMENT 相似
sEnd	关闭一个段，这条宏指令与汇编指令 ENDS 相似
assumes	相对于 segReg 所给定的段寄存器，对 segName 段中断数据和代码进行访问，这条宏指令与汇编指令 ASSUME 相似
dataOFFSET	生成一个相对于 DATA 段所属组开始的偏移量。这条宏指令与汇编操作符 OFFSET 相似，但能自动提供组名
codeOFFSET	生成一个相对于 CODE 段所属组开始的偏移量，这条宏指令与汇编操作符 OFFSET 相似，但能自动提供组名
segNameOFFSET	生成一个相对于用户定义的 segName 段所属组开始的偏移量。这条宏指令与汇编操作符 OFFSET 相似，但能自动提供组名

Cmacros 宏指令有两个预定义段 CODE 和 DATA，应用程序不作特别定义就可使用它们。

5.1.2　存储分配宏指令

存储分配宏指令分配静态内存(专用的或公共的)，申明外部定义的内存和过程，并定义公共标记：

宏指令名	说　明
staticX	分配专用静态内存
globalX	分配公共静态内存
externX	定义一个或多个外部变量名和函数名
labelX	定义一个或多个公共(全局)的变量和函数

5.1.3 函数宏指令

函数宏指令定义函数的名字、属性、参数和局部变量。

宏指令名	说　明
cProc	定义一个函数的名字和属性
parmX	定义一个或多个函数的参数，这些参数提供了对传递给函数参量的访问
localX	定义指定函数的一个或多个框架变量
cBegin	定义指定函数的入口点
cEnd	定义指定函数的出口点

5.1.4 调用宏指令

调用宏指令可以用来调用 cProc 函数和高级语言函数。这些宏指令按照选择项？PLM 所定义的调用约定传递参量。

宏指令名	说　明
cCall	将指定参量压入栈，保存寄存器内容(如果有的话)，并调用指定函数
Save	指示下一个 cCall 宏指令在调用函数前将指定寄存器内容保存在栈内，当函数返回时，再恢复寄存器内容
Arg	定义了由下一个 cCall 宏指令传递给函数的参量

5.1.5 特殊定义宏指令

特殊定义宏指令将用户定义变量，函数寄存器的使用和寄存器指针传递给 Cmacros。

宏指令名	说　明
Def	用 Cmacros 注册用户变量的名字
FarPtr	定义一个 32 位的指针值，该值在 cCall 指令中可以当作单个变量来传递

5.1.6 错误处理宏指令

错误处理宏指令向控制台发送错误消息并产生消息表，在所产生的错误消息中，显示出错的原因和估计结果。

宏指令名	说　明
errnz	计算一个给定表达的值。如果其结果不为零，则会显示一个错误
errn $	用位置计数器的偏移量减去 label 参数的偏移量，再加 bias 参数，如果其结果不为零，则会显示一个错误消息

错误处理宏指令允许将断言编入汇编语言源程序中。这样就可以根据变量分配或字中标志位位置及假定为真的断言来设计最优的指令序列。

5.2　Cmacros 宏指令的用法详解

本节按字母顺序列出所有 Cmacros 宏指令并骤一作详细介绍。

宏指令名	说明、参数、注释及示例

Arg	
说明	Arg namelist Arg 宏指令定义了由下一个 cCall 宏指令传递给一个函数的参量。这些参量按规定顺序压入栈，该顺序必须与函数参数的顺序相对应。 在每个 cCall 宏指令之前可以有多个 Arg 宏指令，但多个 Arg 宏指令可以并入一行来说明，两者效果相同。
参数	namelist 指定将要传递给函数的参量名，所有参量名必须是预先定义好的。
注释	字节类型参数当做字来传递。高字节没有符号位或零位。立即参量不被支持。

示例	`Arg  var1`
	`Arg  var2`
	`Arg  var3`
	和 `Arg <var1, var2, var3>`
	两者效果相同

Assumes	
说明	Assumes segReg, segName assume 宏指令相对于 segReg 段寄存器，对名为 segName 的预定义中的数据段和代码进行访问，这条宏指令与 ASSUME 汇编指令相似。
参数	segReg 指定段寄存器的名字。 segName 指定一个预定义段即 Code 段或 Data 段，或一个用户自定义的名字。
示例	`assumes cs, Code` `assumes ds, Data`

cBegin	
说明	cBegin [procName] cBegin 宏指令定义了根据参数 procName 给定的函数的实际入口点。这个宏指令还产生了建立框架和保存寄存器的代码。
参数	procName 指定一个函数名。该参数是可选的，如果已给定，那么它必须与 cBegin 宏指令之前的 cProc 宏指令中给出的名字相同。
示例	cProc WndProc, <FAR, PUBLIC>

cCall	
说明	cCall procName, [<argList>], [<underscores>] cCall 宏指令将参数 argList 中的参量压入栈，保存寄存器内容(如果有的话)，并且调用函数 procName.
参数	procName 指定被调用的函数名。 argList 指定传递给函数的参量名表。该参数是可选的。如果在 cCall 宏指令之前使用 Arg 宏指令，就不需要这个参数了。 underscores 指定下划线是否应加到参数 procName 的开头。该参数是可选的，如果没有参量并且调用约定为 C 调用约定，则加一条下划线。
注释	Arg 宏指令中的参量先压入栈，cCall 宏指令的参数 arglist 后压入栈。 字节类型的参数作为字传递。高位字节没有符号位和零位。 立即参量不被支持。
示例	以下三种调用方式 cCall ShowWindow, <aHWnd, nCmdShow> ;- - - - - - - - - - - - - - - - - - - Arg aHWnd cCall ShowWindow, <nCmdShow> ;- Arg aHWnd Arg nCmdShow cCall ShowWindow 效果相同。

cEnd	
说明	cEnd [ProcName] cEnd 宏指令定义了 procName 函数的出口点，产生放弃框架的代码并恢复寄存其的内容，返回到调用程序。
参数	ProcName 指定一个函数名，该参数是可选的。如果已给定，则必须与 cEnd 宏指令之前的 cBegin 宏指令中给定的名字相同。
注释	一旦用 cProc 宏指令定义了一个函数，则任何正式参数都必须用 parmX 宏指令申明，任何局部变量都必须使用 localX 宏指令申明。描述函数的代码必须用 cBegin 宏指令和 cEnd 宏指令。
示例	cProc Wnd <FAR, PUBLIC> parmW aHwnd parmW aMessage parmW WParam parmD lParam localW aHDC localV PS, %(size PaintStruct) localV aRect, %(size Rect) cBegin WndProc ... ... cEnd WndProc

codeOFFSET	
说明	codeOFFSET arg codeOFFSET 宏指令相对于 CODE 段所属组的起始产生一个偏移量。它与 OFFSET 汇编操作符相似，但能自动提供组名。因此可用它代替 OFFEST。
参数	arg 指定一个标记名或偏移量值。
示例	mov word ptr aWndClass.clsLpfnWndProc, codeOffset WndProc

cProc	
说明	cProc procName, <attributes>, <autoSave> cProc 宏指令定义一个函数的名字和属性。
参数	procName 指定函数名。 attributes 指定函数类型。该参数可以是下列类型的组合:

类型	说　明
NEAR	近函数。它只能在它的所在段中被调用。
FAR	远函数。它可从任何段被调用。
PUBLIC	公共函数。在其它源文件中可将此函数申明为外部函数。

	缺省属性是 Near 而且是专用的(即在其它源文件中不可以将此函数申明为外部函数)。 NEAR 和 FAR 属性不能一起使用。如果选择多个属性,则必须用尖括号"<>"。 autoSave 指定一个要保存的寄存器表,这些寄存器的内容在函数被调用时先被保存,在函数退出时恢复。任何一个 8086 寄存器都可被指定。
注释	如果该函数被 C 语言函数调用,则必须保存和恢复 SI 和 DI 寄存器内容。 BP 寄存器的内容一直被保存,不论它在 autoSave 参数所出的表中是否存在。
示例	cProc WinMain, <PUBLIC, si, di> cProc WndProc, <FAR, PUBLIC>

createSeg	
说明	createSeg segName, logName, align, combine, class createSeg 宏指令创建一个具有指定的名字和段属性的新段,并为新段创建一个 assumes 宏指令和一个 OFFSET 宏指令。这条宏指令在中模式 Windows 应用程序中用来定义非驻留段。
参数	segName 指定段的实际名,这个名字被送到连接器。 logName 指定段的逻辑名,这个名字用于后面涉及该段的所有 sBegin, sEnd 和 assumes 宏指令。 align 指定排列类型。该参数可以是下列类型之一:BYTE, WORD, PARA 和 PAGE。 combine 指定段的组合类型。该参数可以是下列类型之一:COMMON, MEMORY, PUBLIC 和 STACK。如果没有给出组合类型,则假定为专用段。 class 指定段的类名,该类名定义那些必须装载到相继内存中的段。

注释	Cmarcos 宏指令具有两个预定义段即:CODE 和 DATA,应用程序不需要特殊定义就可以使用这个段,中模式、大模式和巨模式的应用程序可以用 createSeg 宏指令定义附加段。
示例	createSeg　-AppendCode, AppendCode, PARA, PUBLIC sBegin　AppendCode 　　assumes cs, AppendCode sEnd

dataOFFSET

说明	dataOFFSET arg dataOFFSET 宏指令相对于 DATA 段所属组的开始产生一个偏移量,该偏移量与 OFFSET 汇编操作符相似,但能自动提供组名,因此,可用它来代替 OFFSET。
参数	arg 指定一个标记名或偏移量值。
示例	mov　bx, dataOFFSET szAppName mov　si, dataOFFSET WndCaption

DefX

说明	DefX ＜namelist＞ DefX 宏指令注册了用户用 Comcros 定义的变量。未用 staticX, globalX, externX, parmX 或 localX 宏指令定义的变量在其它宏指令中不能被引用,除非变量名已注册或用 DW 汇编指令定义了变量。
参数	X 指定变量所占的内存大小,该参数可以是下列类型之一:

类型	说　明
B	字节
W	字
D	双字
Q	四字
T	十字节字
CP	代码指针(对于小模式和压缩模式,其大小为一个字)
DP	数据指针(对于小模式和中模式,其大小为一个字)

	namelist 指定要定义的变量名表。
示例	

errn $	
说明	errn $ label, [bias] errn $ 宏指令将位置计数器的偏移量减去 label 参数的偏移量，然后再将 bias 参数加到以上结果中，如果所得的结果不是零，则显示错误消息。
参数	label 指定一个与内存位置相对应的标记。 bias 指定一个带符号的偏移值，需要正号和负号。该参数是可选的。
示例	原先紧跟在另一个代码片段后被定位的一个函数如果改变位置，则 errn $ 宏指令就显示一个错误的消息。

errnz	
说明	errnz <expression> errnz 宏指令计算一个给定表达式的值，如果其结果不为零，则显示一个错误。
参数	expression 指定要计算的表达式，如果表达式中有空格，则需要尖括号把表达式括上。
示例	下面的示例说明 errnz 宏指令的用法： x db ? y db ? mov ax, word ptrx errnz <(OFFSET y) - (OFFEST x) - 1> 如果在汇编过程中，出了顺序存储位置，x 和 y 接收到任何数值，errnz 宏指令都会显示错误消息。 teble1 struc : : table1len equ $ - table1 table1 ends table2 struc : : table2len equ $ - table2 table2 ends errnz table1 Len - table2Len 如果在汇编过程中有两个表的长度不相等，则 errnz 宏指令显示错误消息。

externX	
说明	externX ＜namelist＞ externX 宏指令定义一个或多个外部变量名或函数名。
参数	X 指定内存大小或函数类型。该参数可以是下列类型之一：

类型	说　明
A	在其它文件中用 EQU 和 = 指令申明的常数值
B	字节
W	字
D	双字
Q	四字
T	十字节字
CP	代码指针(对于小模式和压缩模式,其大小为一个字)
DP	数据指针(对于小模式和中模式,其大小为一个字)
NP	近函数指针
FP	远函数指针
P	对于小模式和压缩模式,是近函数值;对于其他模式,是远函数指针

	namelist 指定变量或函数名表。
示例	externP _acrtused externFP ＜LoadIcon＞ externFP ＜GetStockObject＞

FarPtr	
说明	FarPtr name, segment, offset FarPtr 宏指令定义了一个 32 位的指针值,该值在 cCall 宏指令中被当作单个参量来传递。在 FarPtr 宏指令中,segment 和 offset 的值不一定要在寄存器中。
参数	name 指定将要产生的指针名。 segment 指定定义指针的段部分的文字。 offset 指定定义指针的偏移量部分的文字。
示例	FarPtr destPtr,es,＜wordptr3[si]＞ cCall proc,＜destPtr,ax＞

globalX	
说明	globalX name, [initialValue] [replication] globalX 宏指令分配公共静态内存。
参数	X 指定将要分配的内存的大小，该参数可以是下列类型之一：

类型	说 明
B	字节
W	字
D	双字
Q	四字
T	十字节字
CP	代码指针(对于小模式和压缩模式，其大小为一个字)
DP	数据指针(对于小模式和中模式，其大小为一个字)

	name 指定被分配内存的引用名。 initialValue 指定存储器的初始值，该参数是可选的，如果没有被指定值，则缺省为 0 replication 指定重复进行存储器分配的次数，该参数是可选择的，并能产生 DUP 汇编操作符。
示例	`globalW flag, 1` `globalB string, 0, 30`

labelX	
说明	labelX <namelist> labelX 宏指令定义了一个或多个公共(全局)的变量或函数名。
参数	X 指定存储的大小或函数类型。该参数可以是下列类型之一：

类型	说 明
B	字节
W	字
D	双字
Q	四字
T	十字节字
CP	代码指针(对于小模式和压缩模式，其大小为一个字)
DP	数据指针(对于小模式和中模式，其大小为一个字)
NP	近函数指针
FP	远函数指针
P	对于小模式和压缩模式，是近函数值；对于其他模式，是远函数指针

	namelist 指定外部变量或函数的名表。
示例	`labelB <DataBase>` `labelFP <SampleRend>`

localX	
说明	localX ＜namelist＞, size localX 宏指令定义了函数的一个或多个框架变量。为了使这些字保留在已排列好的栈中,这个宏指令保证被分配的总空间为偶数个字节。
参数	X 指定存储大小,该参数可以是下列类型之一:

类型	说　明
B	字节(在栈中分配一个单字的存储空间)
W	字(在字的边界上分配)
D	双字(在字的边界上分配)
V	变量的大小(在字的边界上分配)
Q	四字(在字的边界上排列)
T	十字节(在字的边界上排列)
CP	代码指针(对于小模式和压缩模式,其大小为一个字)
DP	数据指针(对于小模式和中模式,其大小为一个字)

	namelist 指定函数的框架变量名表。 size 指定变量的大小,仅用于 localV 宏指令。

注释	B 类型的变量不必要在字的边界上排列。localD 宏指令产生两个附加符号:即 OFF_name 和 SEG_name, OFF_name 是参数的偏移量部分, SEG_name 是参数的段部分。
	引用变量时只需要变量名。写法如下:
	`lea bx, aWndClass`
	而不能写成:
	`lea bx, word ptr aWndClass[bp]`

示例	`localB ＜L1,L2.L3,＞`
	`localW ahWnd`
	`localV aWndClass, %(size Wndclass)`

parmX	
说明	parmX <namelist> parmX 宏指令定义一个或多个函数参量。这些参数提供对传递给函数的参量的访问,其中的参量必须按函数调用中的参量次序出现。
参数	X 指定存储大小,该参数可以是下列类型之一:

类型	说 明
B	字节(在栈中分配一个单字的存储空间)
W	字(在字的边界上分配)
D	双字(在字的边界上分配)
Q	四字(在字的边界上排列)
T	十字节(在字的边界上排列)
CP	代码指针(对于小模式和压缩模式,其大小为一个字)
DP	数据指针(对于小模式和中模式,其大小为一个字)

	namelist 指定参数名表。

注释	parmD 宏指令产生两个附加符号:OFF_name 和 SEG_name,OFF_name 是参数的偏移量部分,SEG_name 是参数的段部分。
	在引用相应的参量时,只需要参量名。写法如下:
	`cmp aMessage, WM_paint`
	而不能写成:
	`cmp word ptr aMessage[bp], WM_Paint`

示例	`ParmW <aHWnd, aMessage, WParam>` `ParmD lParam`

特别注意:符号重定义和超越类型

在一个函数中用 parmX 宏指令定义的任何符号都可以在任何其他函数中作为一个参数进行重定义。这样,不同的函数就可以用相同的名字引用相同的参数,而不必考虑该参数在栈中的位置。

用 parmX 和 localX 宏指令定义的参数和局部变量实际与下列形式的表达式相对应:

```
localW ahWnd = = > ahWnd equ word ptr[bp + nn]
parmD nCmdShow = = > nCmdShow equ dword ptr[bp + nn]
```

在这个例子中,参数 nn 指定一个从当前 BP 寄存器值开始的偏移量。这些表达式使你不必显示地指定类型就可以使用操作符中的名字。这意味着 ahWnd 和 nCmdShow 可以按下列形式被引用:

```
mov ax, word ptr (ahWnd)
mov (e)ax, dword ptr (nCmdShow)
```

应该注意的是,超越类型必须用括弧"()",如果没有括弧,比如:

```
mov ax, word ptr ahWnd
```

汇编程序就会产生一个错误消息。这种方式的一个例外是 localV 宏指令。用这个宏指令产生的表达式没有与其相关的类型,因此不用括号就可以超越类型:

```
localV aWndClass, % (size WndClass) = = > aWndClass equ[bp + nn]
```

Save	
说明	Save <regList> Save 宏指令使下一个 cCall 宏指令在调用函数前将指定寄存器的内容保存在栈内，并在函数返回后恢复寄存器的内容。这个宏指令可用于保存由被调用函数所破坏的寄存器内容。 Save 宏指令只能用于一个 cCall 宏指令，每个新的 cCall 宏指令必须有一个相应的 Save 宏指令。如果在一个 cCall 宏指令前出现两个 Save 宏指令，则承认第二个。
参数	regList 指定要保存的寄存器表。
示例	Save <cl, bh, si> Save <ax>

sBegin	
说明	sBegin segName sBegin 宏指令可打开一个段，与 SEGMENT 汇编指令相似。
参数	segName 指定要打开的段的名字。它可以是预订的 CODE 段或 DATA 段，或用户自定义段的名字。
示例	sBegin Data sBegin Code

segNameOFFSET	
说明	segNameOFFSET arg segNameOFFSET 宏指令相对于用户自定义段 segName 所属组的开始，产生一个偏移量。这个宏指令与 OFFSET 汇编操作符相似，但能自动提供组名，因此可用它来代替 OFFSET。
参数	arg 指定一个标记名或偏移量值。
示例	mov word ptr aWndClass.clsLpfnProc, codeOffset WndProc mov bx, dataOffset szAppName

sEnd	
说明	sEnd [segName] sEnd 宏指令关闭一个段,它与 ENDS 汇编指令相似。
参数	segName 指定一个用来增加可读性的名字。该参数是可选的,如果该参数已给定,则必须和与其相匹配的 sBegin 宏指令中给出的名字相同。
示例	sEnd sEnd Code

static X		
说明	staticX name[initialValue], [replication] staticX 宏指令分配专用静态内存。	
参数	X 指定所要分配的存储大小。该参数可以是下列类型之一:	
	类型	说 明
	B	字节
	W	字
	D	双字
	Q	四字
	T	十字节字
	CP	代码指针(对于小模式和压缩模式,其大小为一个字)
	DP	数据指针(对于小模式和中模式,其大小为一个字)
	name 指定被分配内存的引用名。 initialValue 指定存储器的初始值。该参数是可选的,如果没有指定,则缺省为零。 replication 指定重复分配的次数。这个可选择项能产生 DUP 汇编操作符。	
示例	staticB szAppName, 'HelloWin' db 0 staticD CursorNo, IDC_ARROW	

5.3 用汇编语言编写 Windows 程序应遵循的规则

在使用 Cmacros 创建汇编语言 Windows 应用程序时,应用程序的汇编语言源文件必须具有下列内容:

(1)指定内存模式 — 把四个选择项 memS memM memC memL 中的一个设置为

1Cmacros 中有四种内存模式可供选择，memS、memM、memC、memL 分别代表小模式、中模式、压缩模式和大模式。这些内存模式其实是指应用程序中可能有的代码段和数据段的多少。

模式	说　明
小模式	一个代码段和一个数据段
中模式	多个代码段和一个数据段
压缩模式	一个代码段和多个数据段
大模式	多个代码段和多个数据段
巨模式	多个代码段和多个数据段，其中一个或多个数据段大于 64K

用汇编语言编写 Windows 程序时，在源文件的开始应选择一种存储模式。可用的选择项名如下：

选择项名	内存模式	代码规模	数据规模
memS	小模式	小	小
memM	中模式	大	小
memC	压缩模式	小	大
memL	大模式	小	大
memH	巨模式	大	大

使用 EQU 可以定义一个选择项名，比如：

```
memM EQU 1
```

如果没有选定内存模式，则缺省模式是小模式。

在选定了一个内存模式选择项之后，便有两个符号 SizeC 和 SizeD 被定义。这两个符号可以用作与内存有关的代码，它们可以具有下列值：

符号	值	意义
SizeC	0	小模式代码
	1	大模式代码
SizeD	0	小模式数据
	1	大模式数据
	2	巨模式数据

(2)选择调用约定 — 置？PLM 为 1 或 0

Cmacros 调用约定选择项指定了应用程序将要到的高级语言调用约定。通过定义选择项？PLM 的值就可以选择调用的约定。调用约定的值可说明如下：

值	约定	说 明
0	标准 c	调用程序将右边的参量最先压入栈,最左边的参量最后压入栈。当控返回后,调用程序将栈内的参量弹出栈。
1	Pascal	调用程序将最左边的参量最先压入栈,最右边的参量最后压入栈。被调用的函数将参量弹出栈。

? PLM 的值可以用赋值指令(=)来设置,其语句如下:

```
? PLM = 1
```

缺省时采用 Pascal 调用约定,被 Windows 调用的函数必须采用 Pascal 调用约定。

(3)指定 Windows 函数特有的前缀及后缀代码是否有效 — 置? WIN 为 0 或 1

Windows 应用程序必须使用 Windows 前缀及后缀选择项。这个选择项指定了每个函数是否使用特定的前缀和后缀代码,该代码定义给定函数的当前数据段。通过定义选择项? WIN 的值可以设置上述的前缀及后缀选择项。

值	意 义
0	使特定的前缀和后缀代码无效
1	使特定的前缀和后缀代码有效

设置? WIN 的值的语句如下:

```
? WIN = 1
```

缺省时,前缀及后缀代码有效。

(4)包含 CMACROS.INC 文件

CMACROS.INC 文件包含所有 Cmacros 宏指令的汇编语言定义。在汇编语言的源文件的开头必须包含该文件,这可通过 INCLUDE 指令来实现。其命令行具有如下形式:

```
INCLUDE CMACROS.INC
```

如果宏指令文件既不在当前目录中,也不在命令行指定的目录中,则必须给出完整的路径名。

Cmacros 宏指令已在前面说明。

(5)创建应用程序入口点 WinMain

创建应用程序入口点 WinMain,且必须申明它为一个公共函数。该函数应该具有如下形式:

```
cProc WinMain, <PUBLIC>, <si,di>
    parmW  hInstance
    parmW  hPrevInstance
    parmD  lpCmdLine
    parmW  nCmdShow
cBegin WinMain
    ...
    ...
```

```
    ...
cEnd WinMain
sEnd
```

WinMain 函数必须定义在标准代码段内。

(6)申明回调函数

回调函数必须以下形式申明:

```
cProc WndProc, <FAR, PUBLIC>
parmW aHWnd
parmW aMessage
parmW wParam
parmD lParam

localW aHDC
localV PS, %(size PaintStruct)
localV aRect, %(size Rect)

cBegin WndProc
...
...
...
cEnd   WndProc
```

回调函数必须在一个代码段内被定义。

(7)库连接

汇编完应用程序源文件以后,就应该将已汇编的目标文件与适当的 C 语言库进行连接。

如果应用程序全部是用汇编语言编写的,为了连接无误,必须在应用程序源文件中上绝对符号_acrtused 的外部定义。

(8)使栈检查有效—置? CHKSTK 为 1

通过定义选择项? CHKSTK 可以使栈检查有效。当检查有效时,前缀代码就调用外部定义程序 CHKSTK 以分配局部变量。

用赋值指令(=)可以定义选择项? CHKSTK,其语句的形式如下:

```
? CHKSTK = 1
```

一旦? CHKSTK 被定义,栈检查对整个文件都有效。

缺省时? CHKSTK 未被定义,表示没有栈检查。

5.4　用汇编语言编写 HELLOWIN

汇编语言的 HELLOWIN 程序需要的四个文件在下面给出。这四个文件为:

HELLOWIN.DEF 模块定义文件

HELLOWIN.LNK 链接文件

HELLOWIN.MAK "制作"文件

为了方便理解,我们也提供了广为流传的用 C 编写的 HELLOWIN,在阅读汇编程序时,可以与 HELLOWIN.C 比照。

```
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;Macro assembler sample program for Windows
; Written by Leijun Oct 2, 1993
;
; HELLOWIN.ASM - Displays "Hello, Windows!" in client area
;                (c) Yellow Rose Software Workgroup, 1993
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

          .286

;选定内存模式、调用约定,使 Windows 函数特有的前缀和后缀代码有效

memS      EQU 1                    ; Small memory model
? PLM     EQU 1                    ; Pascal call
? WIN     EQU 1                    ; Windows program

;包含两个宏命令文件 WINDOWS.INC 和 CMACRO.INC
        include windows.inc
        include cmacros.inc

;数据段。sBegin 与后面的 sEnd 对应
sBegin  Data

;staticB, staticD, staticW 宏指令分配专用的静态内存
;在 staticB 之后又定义一个字节 0,作为字符串的结束标志
staticB szAppName, 'HelloWin'
        db 0
staticB wndCaption, 'The Hello Program'
        db 0
staticB showStr, 'Hello, Mr. Leijun!'
        db 0
staticD  WndStyle, 0
staticW  CW_Use, CW_USEDEFAULT
staticW  wNULL, NULL
staticD  dNULL, NULL
staticD  IconNo, IDI_APPLICATION
staticD  CursorNo, IDC_ARROW
sEnd

;代码段开始
sBegin  Code
        assumes cs,Code
        assumes ds,Data

        externP _acrtused
;说明所有在 HELLOWIN 中调用的 Windows 函数
;这些函数为远函数
```

```
        externFP <LoadIcon>
        externFP <LoadCursor>
        externFP <GetStockObject>
        externFP <RegisterClass>
        externFP <CreateWindow>
        externFP <ShowWindow>
        externFP <UpdateWindow>
        externFP <GetMessage>
        externFP <TranslateMessage>
        externFP <DispatchMessage>
        externFP <BeginPaint>
        externFP <GetClientRect>
        externFP <DrawText>
        externFP <EndPaint>
        externFP <PostQuitMessage>
        externFP <DefWindowProc>

;应用程序入口点 WinMain,它是一个公用函数。整个函数定义在标准代码段 Code 内
cProc WinMain, <PUBLIC,si,di>
;定义函数参数
parmW  hInstance
parmW  hPrevInstance
parmD  lpCmdLine
parmW  nCmdShow

;应定义函数的一个或多个框架变量,size 指定变量的大小
localW  ahWnd
localV  aMsg, %(size MsgStruct)
localV  aWndClass, %(size WndClass)

;函数的实际入口点,该函数还产生了建立框架和保存寄存器的代码
cBegin  WinMain

        cmp     hPrevInstance, 0
        jnz     @@10

;一连串的赋值语句
        mov     word ptr aWndClass.clsstyle, CS_HREDRAW or CS_VREDRAW
        mov     word ptr aWndClass.clsLpfnWndProc, codeOffset WndProc
;clsLpfnWndProc 是一个长字
        mov     word ptr aWndClass.clsLpfnWndProc[2], cs
        xor     ax, ax
        mov     aWndClass.clsCbClsExtra, ax
        mov     aWndClass.clsCbWndExtra, ax
        mov     cx, hInstance
        mov     aWndClass.clsHInstance, cx

;两个 push 0 构成一个 NULL 参数
;函数 LoadIcon 的返回值放在寄存器 ax 中
;该函数调用也可写成 cCall LoadIcon <NULL, IDI_APPLICATION>
        push    0
        push    0
```

```
        push    IDI_APPLICATION
        cCall   LoadIcon
        mov     aWndClass.clsHIcon, ax

        push    0
        push    0
        push    IDC_ARROW
        cCall   LoadCursor
        mov     aWndClass.clsHCursor, ax

        push    WHITE_BRUSH
        cCall   GetStockObject
        mov     aWndClass.clsHbrBackground, ax

        mov     word ptr aWndClass.clsLpszMenuName, 0
        mov     word ptr aWndClass.clsLpszMenuName[2], 0
        mov     word ptr aWndClass.clsLpszClassName, dataOffset szAppName
        mov     word ptr aWndClass.clsLpszClassName[2], ds

        lea     bx, aWndClass
        regptr  ssbx, ss, bx
        cCall   RegisterClass, <ssbx>     ; 登记窗口

@@10:

        mov     bx, dataOffset szAppName
        regptr rAppName, ds, bx
        mov     si, dataOffset WndCaption
        regptr rWndCap, ds, si
        xor     ax, ax
        mov     word ptr WndStyle[2], WS_OVERLAPPEDWINDOW
        cCall   CreateWindow, <rAppName, rWndCap, WndStyle,   ; 创建窗口
                cw_use, cw_use, cw_use, cw_use,wNull, wNull,
                hInstance, dNull>
        mov     aHWnd, ax

        cCall   ShowWindow, <aHWnd, nCmdShow>              ; 显示窗口
        cCall   UpdateWindow, <aHWnd>

@@20:
        lea     ax, aMsg                                   ; 使用消息
        push    ss
        push    ax
        push    0
        push    0
        push    0
        cCall   GetMessage
        or      ax, ax
        jz      @@30
        lea     ax, aMsg
        push    ss
        push    ax
```

```
        cCall   TranslateMessage
        lea     ax, aMsg
        push    ss
        push    ax
        cCall   DispatchMessage
        jmp     @@20

  @@30:
        mov     ax, aMsg.msWParam

cEnd    WinMain


cProc WndProc, <FAR, PUBLIC>
parmW    aHWnd
parmW    aMessage
parmW    wParam
parmD    lParam

localW  aHDC
localV  PS, %(size PaintStruct)
localV  aRect, %(size Rect)

cBegin  WndProc

        cmp     aMessage, WM_Paint
        jnz     @@100

        lea     bx, PS
        regptr dsbx, ss, bx
        cCall   BeginPaint, <aHWnd, ssbx>
        mov     aHDC, ax

        lea     bx, aRect               ;截获消息,用来在屏幕上显示 HelloWin
        regptr dsbx, ss, bx
        cCall   GetClientRect, <aHWnd, ssbx>

        push    aHDC
        push    ds
        push    dataOffset showStr
        push    -1
        push    ss
        lea     bx, aRect
        push    bx
        push    DT_SINGLELINE or DT_CENTER or DT_VCENTER
        cCall   DrawText

        lea     bx, PS
        regptr ssbx, ss, bx
        cCall   EndPaint, <aHWnd, ssbx>

        xor     ax, ax
```

```
        cwd
        jmp     @@900
@@100:
        cmp     aMessage, WM_Destroy
        jnz     @@200
        push    0
        cCall   PostQuitMessage
        xor     bx, bx
        cwd
        jmp     @@900
@@200:
        cCall   DefWindowProc, <aHWnd, aMessage, wParam, lParam>
@@900:
cEnd    WndProc


sEnd
        end

;整个 HELLOWIN.ASM 是高度结构化的,程序流图与 HELLOWIN.C 完全一样,读者可以对照理解

##----------------
##  HELLOWIN.MAK
##----------------

hellowin.exe: hellowin.obj hellowin.lnk
  link @hellowin.lnk

hellowin.obj: hellowin.asm hellowin.lnk
  masm -Mx -Zi -DDEBUG hellowin.asm;

##----------------
## HELLOWIN.LNK
##----------------
hellowin
hellowin.exe/align:16
hellowin/map
snocrtw libw /NOD
hellowin.def


;------------------------------------------------------------
; HELLOWIN.DEF module definition file
;------------------------------------------------------------

NAME            HELLOWIN

DESCRIPTION     'Hello Windows Program'

EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE DISCARDABLE
```

```
DATA          PRELOAD MOVEABLE MULTIPLE
HEAPSIZE      1024
STACKSIZE     8192
```

作为对照，我们给出用 C 语言编写的 HELLOWIN 的源程序：HELLOWIN.C、HELLOWIN.MAK、HELLOWIN.DEF。

```
- - - HELLOWIN.C - - -

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   HELLOWIN.C - - Displays "Hello, Windows" in client area
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

#include <windows.h>

long FAR PASCAL _export WndProc (HWND, UINT, UINT, LONG) ;

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
     {
     static char szAppName[] = "HelloWin" ;
     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  = szAppName ;

          RegisterClass (&wndclass) ;
          }

     hwnd = CreateWindow (szAppName,              // window class name
                          "The Hello Program",    // window caption
                          WS_OVERLAPPEDWINDOW,    // window style
                          CW_USEDEFAULT,          // initial x position
                          CW_USEDEFAULT,          // initial y position
                          CW_USEDEFAULT,          // initial x size
                          CW_USEDEFAULT,          // initial y size
                          NULL,                   // parent window handle
                          NULL,                   // window menu handle
                          hInstance,              // program instance handle
                          NULL) ;                 // creation parameters
```

```
    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 ;
    PAINTSTRUCT ps ;
    RECT   rect ;

    switch (message)
         {
         case WM_PAINT:
              hdc = BeginPaint (hwnd, &ps) ;

              GetClientRect (hwnd, &rect) ;

              DrawText (hdc, "Hello, Windows!", -1, &rect,
                        DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

              EndPaint (hwnd, &ps) ;
              return 0 ;

         case WM_DESTROY:
              PostQuitMessage (0) ;
              return 0 ;
         }

    return DefWindowProc (hwnd, message, wParam, lParam) ;
    }


- - - HELLOWIN.MAK - - -

WINCC     = cl -c -G2sw -Ow -W3 -Zp -Tp
WINLINK   = link /nod
WINLIB    = slibcew oldnames libw commdlg ddeml test
WINRC     = rc -r

hellowin.exe : hellowin.obj hellowin.def
     $(WINLINK) hellowin, hellowin, NUL, $(WINLIB), hellowin
     rc -t hellowin.exe
```

```
hellowin.obj : hellowin.c
     $ (WINCC) hellowin.c

- - - HELLOWIN.DEF - - -

;- - - - - - - - - - - - - - - - - -
; HELLOWIN.DEF module definition file
;- - - - - - - - - - - - - - - - - -

NAME           HELLOWIN
DESCRIPTION    'Hello Windows Program'
EXETYPE        WINDOWS
STUB           'WINSTUB.EXE'
CODE           PRELOAD MOVEABLE DISCARDABLE
DATA           PRELOAD MOVEABLE MULTIPLE
HEAPSIZE       1024
STACKSIZE      8192
```

第 6 章

自装载 Windows 执行文件

在第 4 章中，我们描述了启动应用程序或 DLL 时，Windows 系统为程序分配的内存、建立的相关数据记录及执行时的动态链接机制。

在可执行文件启动和执行以前，还有一个装载过程。对于一般的可执行程序，装载过程是由 Windows KERNEL 自动完成的。在第 1 章中我们就已经知道，有些 Windows 程序是自行装载的，需要自己装载自己的程序应该在 NE 文件头中申明。

Windows 系统已经提供了装载过程，为什么有的程序还需要自行装载？自装载做了一些什么事情？独立的自装载有什么意义？如何编写自装载程序？这些问题都将在本章中回答。

本章是本书的重点章节，也是作者分析 Windows 的最大收获。掌握该项技术，首先可以用来编写 Windows 操作环境中的加密软件，其次可以编写 Windows 上的压缩程序，而在装入的时候解密或展开，达到保护知识产权和节约磁盘空间的问题。还有，在 Windows 中，根据约定，任何一个段的大小都应该小于 64K，但在实际中，有时必须突破 64K 的限制，很多资料介绍了突破这种限制的办法。由于大于 64K 的段不为 Windows 接受，所以此时必须编写自装载程序来装载段大小大于 64K 的程序。

6.1 自装载过程的函数接口

Windows KERNEL 提供了一个标准的装载函数，负责将应用程序装入内存，并从一个特定入口点(即由 CS:IP 指定的启动过程地址，见第 4 章)开始执行。标准的装载函数能够装载符合 NE 文件格式的程序。但是，如果标准的 NE 文件格式被破坏，如程序中的代码段被加密变形或资源段被压缩，那么重定位数据就无法正确地重定位、资源不再符合标准的资源格式，在这种情况下，它必须有自己的装载程序，在装载过程中为被压缩或加密过的文件提供特别的格式处理方法，把被变形的程序还原成标准的格式。

要表明一个程序(Windows AP 或 DLL)是自行装载的，它必须在 NE 文件头的“特征标记值”中含有 0x0800，也就是位 11 必须是 1(见第 1 章)。否则，Windows 就忽略特有装载代码，使用 Windows 内核中的标准装载函数。

Microsoft Windows 提供了编写自装载过程的接口，它总共有 6 个函数：3 个由应用程序开发人员提供；3 个由 Windows 内核提供，它们负责装入、重装入段和复位硬件，共同构成一

个完整的自装载过程。按照 Windows 的约定，自装载程序的自装载代码必须放在可执行文件的 Seg #1 中。在 Seg #1 中除了上述的 6 个函数，还包含一个指向这些函数和装载代码的指针表——装载数据表。

在这一节中，我们要讲解装载数据表和编程人员应该编写的 3 个自装载函数。

6.1.1 装载数据表

自装载 Windows 应用程序的第一个段中含有一个装载数据表，包括指向每一个装载函数的远指针。下表列出了该数据表的格式：

位　置	描　述
0x00	标识版本号(必须是'A0')[注]
0x02	保留
0x04	指向启动过程，由应用程序开发人员提供
0x08	指向重装载过程，由应用程序开发人员提供
0x0c	保留
0x10	指向内存分配过程，由内核提供
0x14	指向入口编号过程，由内核提供
0x18	指向退出过程，由应用程序开发人员提供
0x1c	保留
0x1e	保留
0x20	保留
0x22	保留
0x24	指向属主设置过程，由内核提供

注 B: Microsoft 的文档错误地将该处的内容说成是 0xA0，实际上应该是字符串'A0'。

该表中的所有指针必须指向第一段之内的某一位置，在第一段之外没有意义。可执行文件的段表装入内存之后，每一入口都有一个附加的 16 位值，这就是装载函数创建的段选择器(或句柄)，段选择器与函数在第一段中的地址联合构成一个远指针。装载数据表放在 Seg #1 的最前面。

6.1.2 装入段——BootApp

如果 Windows 头信息块的 16 位值含有 0x0800(也就是位 11 为 1)，那么内核就调用应用程序开发人员提供的函数 BootApp，而不是一般情况下的装入应用程序。BootApp 调用内核提供的 MyAlloc 函数，为将要装入内存的一个段分配内存，如果该段是 PRELOED 或 FIXED，BootApp 还要调用函数 LoadAppSeg(开发人员提供的另一个函数)。装入内存的段的句柄应该与信息块联系起来，否则 Windows 就无法判断刚才装入的段属于那一个信息块，所以函数 BootApp 还调用内核提供的函数 SetOwner。

函数 BootApp 应该分配的第一个段是应用程序的自动数据段，该段中含有应用程序的堆栈。自动数据段必须在 BootApp 调用 Windows 函数 PatchCodeHandle 之前分配。函数 PatchCodeHandle 为自装载应用程序修改 Prolog code。

6.1.3 重装入段——LoadAppSeg

除了装入段的功能以外，函数 LoadAppSeg 还负责重装曾经被 Windows KERNEL 丢弃的段。正因为 LoadAppSeg 要重装段，它必须更新段表中 16 位标志值的第 1 和第 2 位(只有自装载应用程序需要改变 Windows 头或跟在后面的数据表)。位 1 表示是否要为该段分配内存，位 2 表示该段是否现在装入。段表的完整描述见第 1 章。

如果装载程序要为一个段分配内存，但是该段现在不装入(也就是，位 1 是 1，位 2 是 0)，那么函数 LoadAppSeg 应该调用 Windows 函数 GlobalHandle 决定是否为它分配内存，如果没有分配，那么 LoadAppSeg 就调用 Windows 函数 GloadlReAlloc 函数为该段重分内存。

一旦分配了内存，函数 LoadAppSeg 就从执行文件中装入该段，并调用函数 PatchCodeHandle 修正段中每一函数的前导代码，修正后，LoadAppSeg 就可以还原段中的所有远指针；如果指针是由序号标识的，那么就调用内核提供的函数 EntryAddrProc 来还原地址。

6.1.4 复位硬件——ExitProc

关闭一个自装载应用程序时，KERNEL 调用开发者提供的函数 ExitProc 复位可能由动态链接库访问过的硬件。但是，函数 ExitProc 不需要释放或关闭文件。

6.2 自装载函数参考

这里给出 Microsoft Windows 提供的自装载函数接口的信息。它包括 6 个函数，这些函数在自装载 Windows 程序中由应用程序开发人员或 KERNEL 提供。

BootApp

用　法	void BootApp(hblock, hFile) HANDLE hBlock;　　/* 信息块句柄 */ HANDLE hFile;　　/* 可执行文件句柄 */ BootApp 装载指定的应用程序。
参　数	hBlock Windows 头(新格式)中信息块所在的段的选择器。 hFile 可执行文件句柄。参数 hFile 必须是一个有效的 MS-DOS 句柄。
注　释	hBlock 标识 NE 文件头信息块的内容，详细请见本书第 1 章第 3 节中说明。 函数 BootApp 是自装载 Windows 程序所要求的三个函数之一，应用程序开发人员必须提供它的代码，并将指向该函数的指针存入应用程序装载器代码和数据表中的偏移值 0x0004 处。Windows KERNEL 在装入程序文件头和各种数据表后，调用该函数。

EntryAddrProc

用　法	DWORD EntryAddrProc(hBlock, wEntryNo) HANDLE hBlock;　　/* 信息块选择器 */ WORD wEntryNo;　　/* 入口表过程索引 */ EntryAddrProc 取回指定过程的地址。
参　数	hBlock 含有 Windows 头(新格式)中信息块的段选择器。 wEntryNo 入口表中入口的索引,该索引标识一个过程。
返回值	如果调用成功,则返回指定过程的地址;否则,返回 0。
注　释	参数 wEntryNo 也称为过程顺序号。 EntryAddrProc 是 windows 内核提供的三个函数之一。内核将指向该函数的指针存入装载器代码和数据表的偏移值 0x0014 处。内核在调用私有启动过程(函数 BootApp)之前装入该指针。 EntryAddrProc 由函数 LoadAppSeg 调用。LoadAppSeg 由应用程序开发人员提供。

ExitProc

用　法	void ExitProc(hBlock) HANDLE hBlock;　　/* 信息块选择器 */ ExitProc 关闭自装载应用程序。
参　数	hBlock 含有 Windows 头(新格式)中信息块的段选择器。
注　释	由 hBlock 标识的信息块的内容已在本书第 1 章 1.3 节中说明。 函数 ExitProc 是自装载 Windows 应用程序要求的三个函数之一,应用程序开发人员必须提供它的代码,并将指向该函数的指针存入应用程序装载器代码和数据表中的偏移值 0x0018 处。ExitProc 不需要释放应用程序所占的内存,也不需要关闭任何打开的文件。

LoadAppSeg

用　法	WORD　　LoadAppSeg (hBlock, hFile, wSegID) HANDLE　　hBlock;　　/* 模块信息块句柄 */ HANDLE　　hFile;　　/* 可执行文件句柄 */ WORD　　wSegID;　　/* 段标识 */ LoadAppSeg 负责首次装载一个段或重装载一个曾经被丢弃段。该段在由 wSegID 标识。
参　数	hBlock 含有 Windows 头(新格式)中信息块的段选择器。 hFile 可执行文件句柄。hFile 是一个 MS - DOS 文件句柄(如果文件未打开则是 -1)。 wSegID 函数应该重装的段。
返回值	如果成功,返回该段的选择器;否则返回 0。

注 释	由 hBlock 标识的信息块的内容已在本书第 1 章 1.3 节中说明。 第三个参数 wSegID 由链接程序在链接时决定。 函数 LoadAppSeg 是自装载 Windows 应用程序要求的三个函数之一，应用程序开发人员必须提供它的代码，并将其指针存入应用程序装载器代码和数据表的偏移值 0x0008 处。

MyAlloc

用 法	DWORD MyAlloc (wFlags, wSize, wElem) WORD wFlags; /* 段标志 */ WORD wSize; /* 段大小 */ WORD WElem; /* 段大小的移位计数值 */ 在自装载 Windows 程序中，MyAlloc 为某个段分配内存。
参 数	wFlags 段标志。 wSize 表示段的大小。实际的段大小由 wSize 向左移动 wElem 位即可得到。 wElem 段大小的移位计数值。与 NE 文件头信息块中的逻辑扇区移位计数值的类似。
返回值	如果成功，则返回值的低字是段句柄，高字是选择器。 如果失败，高、低字都是 0。
注 释	参数 wFlags 标识的标志在 Windows 头的信息块之后、段表之前，KERNEL 在调用 GlobalAlloc 函数之前将 wFlags 转换成合适的值。 段大小(以字节计)可以这样得到：将 wSize 的值左移 wElem 位。MyAlloc 是 Windows KERNEL 提供的三个函数之一。内核将指向它的指针放在装载代码和数据表的偏移值 0x0010 处。KERNEL 在调用启动过程(函数 BootApp)之前装入该指针。

SetOwner

用 法	void SetOwner(hSel, hOwner) WORD hSel; /* 段选择器 */ HANDLE hOwner /* 信息块句柄 */ SetOwner 将指定段与可执行文件或应用程序联系起来。
参 数	hSel 标识选择器句柄，用以将要与可执行文件或应用程序联系的段。 hOwner 为含有该段的应用程序指定 Windows(新格式)可执行文件头中的信息块。
注 释	参数 hOwner 标识的 Windows 头信息块指定链接版本号，各种数据表长度，这些表的偏移值，堆和堆栈的大小等等。 函数 SetOwner 是自装载 Windows 应用程序要求的三个函数之一，开发人员必须提供它的代码，并将其指针放在应用程序装载代码和数据表的偏移 0x0004 处。 内核调用函数 MyAlloc 为段分配之后，调用 SetOwner。

6.3　一个完整的自装载程序实例

根据前面两节的描述，我们编写了以下的自装载过程。编写 Windows 环境下的自装载过程有相当的难度，它要求编程人员对 Windows 核心，特别是执行机制有全面的理解。下面我们粗略讲解自装载过程，具体细节请详细阅读这里给出的程序。请注意，如果你想将本书前面各章的内容融汇贯通，务必反复阅读这个示范程序，直到觉得“心情舒畅”为止。

Windows 执行一个自装载程序时，KERNEL 将可执行程序的文件头，包括信息块和各种表装入内存，hBlock 就代表这一块内存。

首先看一下函数 BootApp 的实现。BootApp 首先判断 Windows Debugger 是否存在，设置 WinDebugExist 标记，以便 WinDebug 跟踪被装入的段。接下来，装入自动数据段。自动数据段必须首先装入。然后将所有固定的或预装载的段装入内存。

```
BootApp
{
    WinDebug 是否存在?
        设置 WinDebugExist 标志为相应的值;
    LoadAppSeg(AutoDataSeg);
    for(SegNo = 1; SegNo < = NUM-SEGS; SegNo + + ) {
        if(SegAttribute(SegNo) = (FIXED || PRELOAD))
            LoadAppSeg(SegNo);
    }
}
```

函数 loadAppSeg 将一个装入内存。首先调用 GlobalAlloc 从 Global Heap 中分配内存；接下来将要被装入的段与信息块关联起来，以免 Windows 不知道这个段是属于谁的；然后将该段拷贝到内存中；再接下来应该做的就是根据重定位信息进行重定位，重定位函数是 ApplyRelocs，该函数在第二个程序 RELOC. ASM 中实现；调用 PatchCodeHandle 修改前导代码(Prolog Code)后，通知 Windows/Debugger，该段以被成功装入。

提请读者注意的是：Windows 是不能装载大于 64K 的段的，但在这里的示范程序中，我们成功地实现了对大于 64K 的段的装载。

```
LoadAppSeg
{
    GlobalAlloc;
    SetOwner;
    Copy the Segment to Memory;
    ApplyRelocs;
    PatchCodeHandle;
    Notify Windows/Debug;
}
```

至于函数 ExitProc 则相当简单，因为这里没有调用动态链接库，所以只需简单地退出即可。

再来看一看重定位程序 RELOC. ASM，该程序提供函数 ApplyRelocs。重定位程序根据当前被载入段的重定位表中的数据来完成重定位工作。如果读者对本书前面章节讲述的重

定位还有一些不清楚的地方，请看该程序中对不同的重定位类型和不同的重定位地址类型所采用的方法。

下面 APPLOAD.ASM 的源程序。

```
;*******************************************************
;  APPLOAD.ASM   Self-Loading Header
;
;  Copyright (c) 1993.10 by KingSoft Ltd.
;  Written by Mr. Leijun
;*******************************************************

        .286

memL    EQU 1
?PLM    EQU 1                   ; Pascal call
?WIN    EQU 1                   ; Windows program

        include windows.inc
        include cmacros.inc

Beep    Macro
        externFP <MessageBeep>
        push    -1
        call    MessageBeep
        endM

createSeg _TEXT, _TEXT, WORD, PUBLIC, CODE
sBegin  _TEXT
sEnd

createSeg _DATA, _DATA, WORD, PUBLIC, DATA, DGROUP
DGROUP  group   _DATA
sBegin  _DATA
fHandle dw      -1
sEnd

createSeg LOADER__MSLANGLOAD, _LOADER__MSLANGLOAD, PARA, PUBLIC, CODE
sBegin _LOADER__MSLANGLOAD
assumes CS, LOADER__MSLANGLOAD
assumes DS, LOADER__MSLANGLOAD

        codeOffset      equu    offset LOADER__MSLANGLOAD:
        MaxRelocItems   equ     100

        externA  <_AHINCR>
        externFP <OPENFILE>
        externFP <GLOBALALLOC>
        externFP <GLOBALREALLOC>
        externFP <GLOBALHANDLE>
        externFP <GETMODULEFILENAME>
        externFP <GETMODULEHANDLE>
        externFP <PATCHCODEHANDLE>
        externFP <GETPROCADDRESS>
```

```
        externFP <ALLOCCSTODSALIAS>
        externFP <FREESELECTOR>
        externFP <GETWINFLAGS>
        externFP <MESSAGEBOX>

        externNP <ApplyRelocs>

        public  FNCSEEKSECTOR,PSRELOADAPPLICATIONSEGMENT,FBOOTAPPLICATION
        public  FALLOCCODEHANDLES, _MSLANGLOAD, FX87PRESENT, APPLCUR
        public  FHOPENSWAPFILE, GETREALFH, FBOOTAPPLICATIONPWIN,
                EXITAPPLICATION

_MSLANGLOAD:
ApplCur         equ  $
ApplCurTab      db 'A0'                         ; ID
                dw 0                            ; Reserved
                dw codeOffset BootApp           ; 04h   BootApp
                dw LOADER__MSLANGLOAD
                dw codeOffset LoadAppSeg        ; 08h   LoadAppSeg
                dw LOADER__MSLANGLOAD
                dd 0                            ; Reserved

aMyAlloc        dd 0                            ; 10h
aEntryAddrProc  dd 0                            ; 14h
                dw codeOffset ExitProc          ; 18h   ExitProc
                dw LOADER__MSLANGLOAD
                dw 0                            ; 1CH   Reserved
                dw 0                            ; Reserved

                dw 0                            ; Reserved
                dw 0                            ; Reserved
aSetOwner       dd 0                            ; 24h

CurrSel         dw 0                            ; 28h
TempVar         dw 0                            ; 2Ah       ????
CurrBlk         dw 0                            ; 2Ch
PageSize        dw 0                            ; 2Eh

ErrPrompt       db 'MSLANGLOAD——Error in loading module', 0
WinDebug        db 'WinDebug', 0
WinNotify       db 'WINNOTIFY', 0

WinDebugExist   db 0                            ; 69h
WinNotifyAddr   dd 0                            ; 6Ah
FX87Present     db 0                            ; 6Eh

; void ExitProc (hBlock)
; HANDLE hBlock
ExitProc:
ExitApplication:
                PUSH    BP
                MOV     BP,SP
                LEAVE
                RETF    2
```

```
Closefile:
                MOV     BX, -1
                CALL    GetRealFH
                CMP     BX, -1
                JZ      @@0085
                MOV     AH, 3EH
                INT     21H
        @@0085: RETF

GetRealFH:
                MOV     AX, -1
                CMP     BX, AX
                JNZ     @@0092
                ASSUMES SS, __DATA
                MOV     BX, fHandle
        @@0092: RET

FncSeekSector:
;
; Seek segment position
; DI = 0
; SI = Segment table
; BX = Handle
;
                CMP     DI, 0
                JZ      @@009A
                JMP     short @@00B0
        @@009A: MOV     AX, [SI]            ; DI = 0
                XOR     DX, DX
                MOV     CX, [DI + 32h]
        @@00A1: SHL     AX, 1
                RCL     DX, 1
                LOOP    @@00A1              ; Position
                MOV     CX, DX
                MOV     DX, AX
                MOV     AX, 4200h
                INT     21H
        @@00B0: RET

FHOpenSwapFile:
;
; Open file with hInst
;
  hInst             equ     [bp + 4]
  OpenBuffer        equ     [bp - 80h]
  ModuleFileName    equ     [bp - 180h]

                PUSH    BP
                MOV     BP, SP
                ADD     SP, -180h
                PUSH    DS
                ASSUMES SS, __DATA
                CMP     Word Ptr fHandle, -01
```

```
                JZ      @@00C3
                JMP     short @@00F2
@@00C3: LEA     BX,ModuleFileName       ; Handle = -1
        MOV     AX,100h
        PUSH    BX

        PUSH    [BP+04]                 ; hInst
        PUSH    SS                      ; lpszFileName
        PUSH    BX
        PUSH    AX                      ; cbFileName
        CALL    GetModuleFileName

        POP     BX
        OR      AX,AX
        MOV     AX,-1
        JZ      @@00EE

        LEA     DX,OpenBuffer
        MOV     AX,0                    ; fuMode
        PUSH    SS                      ; lpszFileName
        PUSH    BX
        PUSH    SS                      ; lpOpenBuff
        PUSH    DX
        PUSH    AX                      ; fuMode
        CALL    OpenFile
@@00EE: ASSUMES SS,__DATA
        MOV     fHandle,AX              ; Handle
@@00F2: POP     DS
        LEAVE
        RET     2

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; @@00F7:
; WORD LoadAppSeg (hBlock, hFile, wSegID)
; HANDLE hBlock;
; HANDLE hFile;
; WORD wSegID
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LoadAppSeg:
psReloadApplicationSegment:

  lawSegID      equ     [bp+6]
  lahFile       equ     [bp+8]
  lahBlock      equ     [bp+0ah]

  Sel1          equ     [bp-2]
  Sel2          equ     [bp-4]
  csSel         equ     [bp-6]
  Flag          equ     [bp-8]
  dsSel         equ     [bp-0Ah]

                PUSH    BP
                MOV     BP,SP
                ADD     SP,-0Ah
```

```
                PUSH    DS
                PUSH    SI
                PUSH    DI
                MOV     DS, lahBlock
                XOR     DI, DI
                MOV     SI, lawSegID
                DEC     SI
                MOV     AX, DS
                CMP     AX, CS:CurrBlk          ; aBlock OK ?
                JZ      @@0115
                JMP     DisplayError2
@@0115:         CMP     SI, WORD PTR [1Ch]      ; < = max segment count ?
                JB      @@011E
                JMP     DisplayError2
@@011E:         SHL     SI, 1
                MOV     BX, SI
                SHL     SI, 1
                SHL     SI, 1
                ADD     SI, BX
                ADD     SI, [DI + 22h]
                                                ; SI = SegID * 0ah + ofsSegTab
                CMP     Word Ptr lahFile, - 01  ; Open ?
                JNZ     @@015F
                CMP     SI, [DI + 8]            ; SegTab OK?
                JZ      @@019E
                MOV     AX, lawSegID
                CMP     AX, [DI + 16h]          ; CS Segment
                JZ      @@019E                  ; Is entry segment
                ASSUMES SS, __DATA
                CMP     Word Ptr fHandle, - 01
                JNZ     @@014f
                PUSH    lahBlock
                CALL    FHOpenSwapFile
                MOV     lahFile, AX
@@014F:         CMP     Word Ptr lahFile, - 01  ; Open
                JNZ     @@015F
                ASSUMES SS, __DATA
                CMP     Word Ptr fHandle, - 01
                JNZ     @@015F
                JMP     short @@019E
                                                ; Open
@@015F:         MOV     AX, [SI + 06]           ; size
                MOV     BX, [SI + 04]           ; type
                MOV     DX, [SI + 08]           ; selector
                TEST    BX, 4                   ; loaded
                JZ      @@0171
                JMP     DisplayError2
@@0171:         CMP     SI, [DI + 8]            ;
                JZ      @@01A5
                TEST    BX, 2                   ; allocated
                JNZ     @@017F
                JMP     DisplayError2
@@017F:         TEST    BX, 1                   ; is code
                JNZ     @@019C
```

```
                XOR     CX,CX                   ; data segment
                OR      AX,AX
                JNZ     @@018C
                INC     CX
@@018C:         PUSH    DX

                PUSH    DX                      ; hglb
                PUSH    CX                      ; cbNewSize
                PUSH    AX                      ;
                PUSH    00                      ; fuAlloc
                CALL    GlobalRealloc

                POP     DX
                CMP     AX,DX
                JNZ     @@019E
@@019C:         JMP     short @@01F1            ; code

@@019E:         XOR     AX,AX                   ; entry segment
                XOR     DX,DX
                JMP     Exit
@@01A5:         TEST    BX,2                    ; allocated
                JNZ     @@01F1
                TEST    BX,0010h                ; Movable / fixed
                JNZ     @@01B4
                JMP     DisplayError2
@@01B4:         XOR     CX,CX
                ADD     AX,[DI+12h]             ; stack size
                JNB     @@01c2
                OR      AX,AX
                JZ      @@01c1
                JMP     short @@019E
@@01c1:         INC     CX
@@01c2:         ADD     AX,[DI+10h]             ; heap size
                JNB     @@01ce
                OR      AX,AX
                JZ      @@01CD
                JMP     short @@019E
@@01CD:         INC     CX
@@01CE:         OR      BX,4000h
                MOV     BX,2
                PUSH    BX                      ; fuAlloc
                PUSH    CX                      ; size
                PUSH    AX
                CALL    GlobalAlloc
                OR      AX,AX
                JZ      @@019E
                MOV     [SI+08],AX              ; AllocSel
                OR      Byte Ptr [SI+04],02
                PUSH    AX
                PUSH    AX                      ; sel
                PUSH    DS                      ; hOwner
                CALL    DWORD PTR CS:[aSetOwner]
                POP     DX
@@01F1:         MOV     csSel,DX                ; code segment
```

```
                MOV     sel1,DX
                MOV     sel2,DX
                MOV     Word Ptr dsSel, 0
                TEST    Word Ptr [SI + 04],8000h
                JZ      @@0214
                MOV     CX,CS
                AND     CX, + 03
                MOV     AX,DX
                AND     DX, - 04
                OR      DX,CX
                JMP     short @@0222
@@0214:         PUSH    DX                      ;
                CALL    GlobalHandle            ;
                CMP     AX,sel1
                JZ      @@0222
                JMP     DisplayError2
@@0222:         TEST    Word Ptr [SI + 04],0001
                JZ      @@0231
                MOV     sel1,DX
                MOV     sel2,DX
                JMP     short @@0244
@@0231:         PUSH    DX                      ; code
                CALL    AllocCStoDSAlias
                OR      AX,AX
                JNZ     @@023E
                JMP     @@019E
@@023E:         MOV     dsSel,AX
                MOV     sel2,AX
@@0244:         MOV     Word Ptr Flag,1
                JMP     short @@0260
@@024B:         XOR     DX,DX
                JMP     Exit
@@0250:         POP     DS
                POP     SI
                POP     DI
@@0253:
                POP     SI
                POP     DS
                XOR     AX,AX
                DEC     Word Ptr Flag
                JNZ     @@024B

                MOV     AH,0Dh
                INT     21h                     ; Disk reset
@@0260:         CMP     Word Ptr [SI], + 00
                JZ      @@02A4
                PUSH    DS
                PUSH    SI
                MOV     BX,lahFile
                CALL    GetRealFH
                MOV     lahFile,BX
                CMP     BX, - 01
                JNZ     @@0278
                JMP     DisplayError2
```

```
@@0278:  CALL    FncSeekSector
         JB      @@0253
         MOV     CX,[SI+02]              ; size
         MOV     DS,sel2                 ;
         XOR     DX,DX                   ;
         MOV     AH,3Fh
         OR      CX,CX
         JNZ     @@029A
         MOV     CX,8000h                ; = 0 first 32K
         INT     21h
         JB      @@0253
         CMP     AX,CX
         JNZ     @@0253
         MOV     DX,CX
         MOV     AH,3Fh                  ; = 0 second 32K
@@029A:  INT     21h                     ; 32K
         JB      @@0253
         CMP     AX,CX
         JNZ     @@0324
         db      075h, 0b1h
         POP     SI
         POP     DS
@@02A4:  TEST    Word Ptr [SI+04],100h
                                         ; contains relocation data ?
         JZ      @@0324
         PUSH    DI
         PUSH    SI
         PUSH    DS
         PUSH    CS
         CALL    AllocCStoDSAlias
         MOV     DS,AX                   ;
         MOV     CurrSel,AX              ;
         MOV     DX,codeOffset TempVar   ;
         MOV     CX,0002
         MOV     AH,3Fh
         INT     21h                     ; Get reloc items
         JB      @@0250
         CMP     AX,CX
         JNZ     @@0250
@@02C9:  MOV     CX,CS:TempVar           ; < = 64h
         CMP     CX,MaxRelocItems
         JBE     @@02d6
         MOV     CX,MaxRelocItems
@@02d6:  MOV     SI,CX
         SHL     CX,03                   ;
         MOV     DX,codeOffset Buffer    ;
         MOV     BX,lahFile
         MOV     AH,3Fh
         INT     21h                     ; Read Relocation Table
         JNB     @@02EA
         JMP     @@0253
@@02EA:  CMP     AX,CX
         JZ      @@02F1
         JMP     @@0253
```

```
@@02F1:  MOV     DI,DX
         ADD     DX,CX
         XOR     AX,AX
         MOV     ES,lahBlock             ;[BP+0Ah]

         PUSH    lahBlock                ;[BP+0Ah]
         PUSH    Sel2                    ;[BP-04h]

         PUSH    DS                      ;Relocation Table
         PUSH    DI

         PUSH    DS                      ;End of Relocation Table
         PUSH    DX

         PUSH    ES                      ;segment beginning
         PUSH    AX
         CALL    ApplyRelocs

         SUB     TempVar,SI
         JNZ     @@02c9
         MOV     AX,CS:CurrSel
         MOV     Word Ptr CurrSel,0
         PUSH    CS
         POP     DS
         PUSH    AX
         CALL    FreeSelector
         POP     DS
         POP     SI
         POP     DI
@@0324:  MOV     CX,[SI+06]              ;alloc size
         MOV     DI,[SI+02]              ;size
         CMP     CX,DI
         JNZ     @@0332
         OR      CX,CX
         JNZ     @@034E
@@0332:  XOR     DX,DX                   ;
         OR      CX,CX
         JNZ     @@0339
         INC     DX
@@0339:  SUB     CX,DI                   ;Clear zero
         SBB     DX,+00
         XOR     AX,AX
         RCR     DX,1
         RCR     CX,1
         MOV     ES,sel2
         CLD
         REPZ    STOSW
         RCL     CX,1
         REPZ    STOSB
@@034E:  OR      Byte Ptr [SI+04],4      ;allocated
         TEST    Word Ptr [SI+04],1      ;code ?
         JNZ     @@0361
         PUSH    csSel
         CALL    PatchCodehandle
```

```
@@0361:  MOV     CX, dsSel
         JCXZ    @@036C
         PUSH    CX
         CALL    FreeSelector
@@036C:  CMP     Byte Ptr CS:WinDebugExist, 00
         JZ      @@039A
         TEST    Word Ptr [SI + 04], 8000h
         JNZ     @@039A

         MOV     AX, 0050h
         MOV     BX, lawSegID
         DEC     BX
         MOV     CX, sel1
         MOV     DX, 0000
         MOV     SI, [SI + 04]          ; type
         MOV     ES, lahBlock           ; hBlock
         MOV     DI, 0026h              ; resident name table
         MOV     DI, ES:[DI]            ; ???
         INC     DI
         CALL    DWORD PTR CS:WinNotifyAddr

@@039A:  MOV     AX, sel1
         MOV     DX, csSel

Exit:
         POP     DI
         POP     SI
         POP     DS
         LEAVE
         RETF    6

DisplayError2:
         PUSH    00                     ; hwndParent
         PUSH    CS                     ; lpszText
         PUSH    codeOffset ErrPrompt
         PUSH    NULL                   ; lpszTitle
         PUSH    NULL
         PUSH    MB_ICONHAND            ; fuStyle
         CALL    MessageBox
         XOR     AX, AX
         JMP     short Exit


; void BootApp (hBlock, hFile)
hBlock   equ     [bp + 8]
hFile    equ     [bp + 6]
BootApp:
FBootApplication:
         PUSH    BP
         MOV     BP, SP
         PUSH    DS
         PUSH    SI
         PUSH    DI
         MOV     AX, _ _AHINCR
```

```
                CMP     AX,1000h
                JZ      @@03D5
                PUSH    hBlock
                PUSH    hFile
                CALL    FBootApplicationPWin
                JMP     short @@03D7
@@03D5:         XOR     AX,AX
@@03D7:         MOV     ES,hBlock
                POP     DI
                POP     SI
                POP     DS
                LEAVE
                RETF    4

FAllocCodeHandles:
; DI - - NE Header
; SI - - Segment table
                MOV     SI,[DI+22h]             ; segment table
                MOV     CX,[DI+1Ch]             ; segment count
                SUB     SI,+0Ah
@@03EA:         DEC     CX
                ADD     SI,0Ah
                CMP     Word Ptr [SI+08],0      ; Selector
                JZ      @@03F7
                JMP     @@047A
@@03f7:         CMP     [DI+8],SI               ; Alloc
                JZ      @@047A                  ; Boot segment ?
                MOV     BX,[SI+04]              ; Type
                OR      BX,4000h                ; set 14 bit
                TEST    BX,1                    ; Code segment
                JNZ     @@0417                  ;
                PUSH    CX
                PUSH    BX                      ; wFlags
                PUSH    AX                      ; wSize         ???
                PUSH    AX                      ; wElem         ???
                CALL    DWORD PTR CS:aMyAlloc
                POP     CX
                MOV     AX,DX                   ; AX:Handle  DX:Selector
                JMP     short @@0466
@@0417:         CMP     Word Ptr [SI+06],0      ; Data segment
                JZ      @@042D                  ; 64K
                PUSH    CX
                PUSH    2; GMEM_MOVABLE         ; fuAlloc = 2
                PUSH    00                      ; DWORD
                PUSH    [SI+06]                 ;  object size
                CALL    GlobalAlloc
                POP     CX
                JMP     short @@0466

@@042D:         PUSH    SI                      ; Data segment
                PUSH    CX
                XOR     DX,DX
@@0431:         INC     DX
                ADD     SI,0Ah
```

```
                CMP     Word Ptr [SI + 06], + 00        ; 64K ?
                LOOPZ   @@0431
                PUSH    DX
                MOV     CX, [SI + 06]                   ;
                PUSH    2; GMEM_MOVABLE                 ; 02
                PUSH    DX                              ; > 64K
                PUSH    CX                              ;
                CALL    GlobalAlloc
                POP     DX
                POP     CX
                POP     SI
                OR      AX, AX
                JZ      @@047A                          ; if failed go to @@047A
                PUSH    SI
                PUSH    AX
        @@0451: DEC     DX                              ;
                ADD     SI, 0Ah                         ;
                ADD     AX, __AHINCR                    ; DS Selector !
                OR      Word Ptr [SI + 04], 8002h       ; Set allocated ID
                MOV     [SI + 08], AX                   ; selector
                OR      DX, DX
                JNZ     @@0451
                POP     AX
                POP     SI

        @@0466: OR      AX, AX                          ; Alloc success ?
                JZ      @@047A
                OR      Byte Ptr [SI + 04], 02          ; Has allocated
                MOV     [SI + 08], AX                   ; Selector
                PUSH    CX
                PUSH    AX                              ; hSel
                PUSH    DS                              ; hOwner
                CALL    DWORD PTR CS:aSetOwner
                POP     CX
        @@047A: OR      CX, CX                          ; End ?
                JZ      @@0481
                JMP     @@03EA
        @@0481: MOV     AX, SP
                RET

KERNEL          DB 'KERNEL', 0
FBootApplicationPWin:
        hBlock_2 equ    [bp + 6]
        hFile_2  equ    [bp + 4]
                PUSH    BP
                MOV     BP, SP
                XOR     DI, DI
                MOV     AX, codeOffset KERNEL
                PUSH    CS
                PUSH    AX
                CALL    GetModuleHandle                 ; Is KERNEL loaded ?
                OR      AX, AX
                JZ      @@04AA
                MOV     SI, AX
```

```
                PUSH    CS
                CALL    AllocCStoDSAlias
                OR      AX, AX
                JNZ     @@04AD
@@04AA:         JMP     @@058B
@@04AD:         MOV     DS, AX
                MOV     CurrSel, AX
                MOV     ES, hBlock_2
                MOV     CurrBlk, ES
                MOV     AX, 1
                MOV     CX, ES:[0032h]          ; NE shift factor
                SHL     AX, CL
                MOV     PageSize, AX
                PUSH    DS
                PUSH    codeOffset WinDebug
                CALL    GetModuleHandle         ; WinDebug
                OR      AX, AX
                JZ      @@04F2
                MOV     Byte Ptr WinDebugExist, 01
                PUSH    AX
                PUSH    DS
                PUSH    codeOffset WinNotify
                CALL    GetProcAddress
                MOV     word ptr WinNotifyAddr, AX
                MOV     word ptr winNotifyAddr[2], DX
                OR      AX, DX
                JNZ     @@04F2
                MOV     Byte Ptr WinDebugExist, 00
@@04f2:         CALL    GetWinFlags
                TEST    AX, WF_80x87            ; 0400h
                JZ      @@0500
                INC     Byte Ptr FX87Present
@@0500:         MOV     AX, CurrSel
                MOV     Word Ptr CurrSel, 0
                PUSH    CS
                POP     DS
                PUSH    AX
                CALL    FreeSelector

                MOV     DS, hBlock_2
                MOV     SI, [DI + 8]            ; Reserved/New seg table
                MOV     [SI + 08], CS           ; set segment 1 selector
                MOV     SI, [DI + 8]
                CMP     SI, + 00
                JNZ     @@0524
                JMP     short DisplayError
@@0524:         TEST    word Ptr [SI + 04], 0010h   ; Movable
                JNZ     @@052D
                JMP     short DisplayError
@@052d:         CALL    FAllocCodeHandles       ;
                OR      AX, AX
                JZ      @@058B
                                                ; Load auto data segment
                PUSH    DS                      ; hBlock
```

```
                PUSH    hFile_2                 ; hFile
                PUSH    [DI + 1Ch]              ; wSegID
                PUSH    CS                      ; FAR
                CALL    LoadAppseg
                MOV     CX, 1
                MOV     SI, [DI + 22h]          ; ofsSegTable
                JMP     short @@056f
        @@0547: TEST    Word Ptr [SI + 04], 4   ; Loaded ?
                JNZ     @@056B
                TEST    Word Ptr [SI + 04], 1   ; Code segment ?
                JNZ     @@055C
                TEST    Word Ptr [SI + 04], 40h ; PreLoad ?
                JZ      @@056b
        @@055c: PUSH    CX                      ;
                PUSH    DS                      ; hBlock
                PUSH    hFile_2                 ; hFile
                PUSH    CX                      ; wSegID
                PUSH    CS                      ; FAR
                CALL    LoadAppseg
                POP     CX
        @@0567: OR      AX, AX
                JZ      @@058b
        @@056B: ADD     SI, 0Ah                 ; ?
                INC     CX
        @@056f: CMP     CX, [DI + 1Ch]
                JBE     @@0547
                MOV     AX, SP
        @@0576: LEAVE
                RET     4

DisplayError:
                PUSH    00
                PUSH    CS
        ;       PUSH    0030h
                db      068h, 30h, 0
                PUSH    00
                PUSH    00
                PUSH    10h
                CALL    MessageBox
        @@058b: XOR     AX, AX
                JMP     short @@0576

  Buffer        DB      MaxRelocItems * 8 dup(0)
                sEnd

                end
```

下面 RELOC.ASM 的源程序。

```
;
;  Reloc.ASM
;
;
```

```
        .286

memL    EQU 1
? PLM   EQU 1                   ; Pascal call
? WIN   EQU 1                   ; Windows program

        include windows.inc
        include cmacros.inc

createSeg LOADER__MSLANGLOAD, _LOADER__MSLANGLOAD, WORD, PUBLIC, CODE
sBegin _LOADER__MSLANGLOAD
assumes CS, _LOADER__MSLANGLOAD
assumes DS, _LOADER__MSLANGLOAD

        codeOffset      equ     offset LOADER__MSLANGLOAD:

        externFP  <MESSAGEBOX>
        externFP  <GETPROCADDRESS>

        extrn   APPLCUR: ABS
        extrn   FX87PRESENT: Byte

        public     APPLYRELOCS

  RelocMsg DB 'Unknown relocation type found.',0

; @@08cf:
ApplyRelocs     proc        near
;
; hBlock        equ         [bp+12h]
; RelocLen      equ         [bp+8]
;
; RelocAddr     equ         [bp-6]
;

                PUSH        BP
                MOV         BP,SP
                ADD         SP,0FEF4h
                PUSH        SI
                PUSH        DI
                PUSH        DS
                MOV         SI,[BP+0Ch]
                MOV         ES,[BP+0Eh]
                MOV         DS,[BP+10h]
                JMP         @@0ACE

      @@08E5:
                MOV         BL,ES:[SI+01]
                AND         BX,3
                SHL         BX,1
                JMP         CS:RelocTypeTable[BX]

  RelocTypeTable DW Offset @@08FB
                 DW Offset @@09DA
```

```
                DW Offset @@0A05
                DW Offset @@0A4B

@@08FB:                                         ; Type = 0
                CMP     Byte Ptr ES:[SI + 04], 0FFh
                JNZ     @@0918
                PUSH    ES                      ; Movable segment
                PUSH    [BP + 12h]              ; hBlock
                PUSH    ES:[SI + 06]            ; wEntryNo
                CALL    DWORD PTR CS:APPLCUR[14h] ; aEntryAddrProc
                POP     ES
                MOV     [BP - 8], AX
                MOV     [BP - 6], DX
                JMP     short @@0943

@@0918:         MOV     AX, ES:[SI + 6]         ; Type = 0  Fixed segment
                MOV     [BP - 08], AX
                XOR     AX, AX
                MOV     AL, ES:[SI + 04]
                DEC     AL
                SHL     AX, 1
                MOV     BX, AX
                SHL     AX, 1
                SHL     AX, 1
                ADD     AX, BX
                LDS     BX, [BP + 04]
                MOV     BX, [BX + 22h]
                ADD     BX, AX
                MOV     AX, [BX + 08]
                DEC     AX
                MOV     [BP - 06], AX
                MOV     DS, [BP + 10h]

@@0943:                                         ; Relocation procedure
                MOV     DI, ES:[SI + 02]        ; Position
                MOV     BL, ES:[SI]             ; RelocAddrType
                AND     BX, + 0Fh
                SHL     BX, 1
                JMP     CS:RelocAddrTypeTable[BX]

RelocAddrTypeTableDW Offset @@0974  ; 0 Low byte at the specified offset
                DW Offset @@09C3        ; 1
                DW Offset @@0977        ; 2 16b selector
                DW Offset @@0996        ; 3 32b pointer
                DW Offset @@09C3        ; 4
                DW Offset @@09BE        ; 5 16b offset
                DW 10 DUP (Offset @@09C3)
                                        ; 11 48b pointer
                                        ; 13 32b offset

@@0974:
                JMP     @@0ACB

@@0977:                                         ; 16b selector
```

```
                MOV         AX, [BP - 06]
        @@097A:                                         ;
                TEST        Byte Ptr ES:[SI + 01], 04
                JZ          @@0985
                ADD         [DI], AX
                JMP         short @@0993
        @@0985: JMP         short @@098D
        @@0987: MOV         CX, AX
                XCHG        CX, [DI]
                MOV         DI, CX
        @@098D:
;               CMP         DI, 0ffffh
                db          081h, 0ffh, 0ffh, 0ffh
                JNZ         @@0987
        @@0993: JMP         @@0ACB

        @@0996:                                         ; 32b pointer
                MOV         AX, [BP - 08]
                MOV         DX, [BP - 06]
                TEST        Byte Ptr ES:[SI + 01], 04
                JZ          @@09AA
                ADD         [DI], AX
                ADD         [DI + 02], DX
                JMP         short @@09BB
        @@09AA: JMP         short @@09B5
        @@09AC: MOV         CX, AX
                XCHG        CX, [DI]
                MOV         [DI + 02], DX
                MOV         DI, CX
        @@09B5:
;               CMP         DI, 0ffffh
                db          081h, 0ffh, 0ffh, 0ffh
                JNZ         @@09AC
        @@09BB: JMP         @@0ACB

        @@09BE:                                         ; 16b offset
                MOV         AX, [BP - 08]
                JMP         short @@097A

        @@09C3:                                         ; Error process
                XOR         AX, AX
                PUSH        AX
                PUSH        CS
                MOV         AX, Offset RelocMsg
                PUSH        AX
                XOR         AX, AX
                PUSH        AX
                PUSH        AX
                XOR         AX, AX
                PUSH        AX
                CALL        MessageBox
                JMP         @@0ACB
```

```
@@09DA:                                          ; RelocType = 1 (Imported ordinal)
        LDS     BX,[BP+04h]                      ; Module reference table
        MOV     BX,[BX+28h]
        MOV     CX,ES:[SI+04]                    ; module index
        DEC     CX
        SHL     CX,1
        ADD     BX,CX
        MOV     AX,[BX]
        PUSH    ES

        PUSH    AX
        PUSH    ES:[SI+06]
        CALL    DWORD PTR CS:APPLCUR[14h] ; aEntryAddrProc

        POP     ES
        JCXZ    @@09C3
        MOV     DS,[BP+10h]
        MOV     [BP-08],AX
        MOV     [BP-06],DX
        JMP     @@0943

@@0A05:                                          ; RelocType = 2 (Imported name)
        LDS     BX,[BP+04]
        MOV     DI,[BX+28h]
        MOV     CX,ES:[SI+04]
        DEC     CX
        SHL     CX,1
        ADD     DI,CX
        MOV     AX,[DI]
        MOV     BX,[BX+2Ah]
        ADD     BX,ES:[SI+06]
        XOR     CX,CX
        MOV     CL,[BX]
        INC     BX
        MOV     DI,BX
        ADD     DI,CX
        MOV     CL,[DI]
        MOV     [DI],CH
        PUSH    CX
        PUSH    ES
        PUSH    AX
        PUSH    DS
        PUSH    BX
        CALL    GetProcAddress
        POP     ES
        POP     CX
        MOV     [DI],CL
        MOV     DS,[BP+10h]
        MOV     [BP-08],AX
        MOV     [BP-06],DX
        OR      DX,DX
        JNZ     @@0A48
        JMP     @@09C3
```

```
@@0A48:  JMP     @@0943


@@0A4B:                                  ; RelocType = 3 (OSFixup)
         MOV     DI,ES:[SI+02]
         MOV     BX,ES:[SI+04]
         CMP     BX,+06
         JBE     @@0A5B
         JMP     @@09C3
@@0A5B:  SHL     BX,1
         JMP     CS:FixupTable[BX]

FixupTable DW    Offset @@09C3           ;0
         DW      Offset @@0A70           ;1
         DW      Offset @@0A83           ;2
         DW      Offset @@0A96           ;3
         DW      Offset @@0AA9           ;4
         DW      Offset @@0AB7           ;5
         DW      Offset @@0AC5           ;6

@@0a70:
         CMP     BYTE PTR CS:FX87Present,0
         JNZ     @@0A81
         ADD     Word Ptr [DI],0FE32H    ;1
         ADD     Word Ptr [DI+01],4000H
@@0A81:  JMP     short @@0ACB

@@0A83:
         CMP     Byte Ptr CS:FX87Present,0
         JNZ     @@0A94
         ADD     Word Ptr [DI],0632H     ;2
         ADD     Word Ptr [DI+01],8000H
@@0a94:  JMP     short @@0ACB

@@0A96:
         CMP     Byte Ptr CS:FX87Present,0
         JNZ     @@0AA7
         ADD     Word Ptr [DI],0E32H     ;3
         ADD     Word Ptr [DI+01],0C000H
@@0AA7:  JMP     short @@0ACB

@@0AA9:
         CMP     Byte Ptr CS:FX87Present,0
         JNZ     @@0AB5
         ADD     Word Ptr [DI],1632H     ;4
@@0AB5:  JMP     short @@0ACB

@@0AB7:
         CMP     Byte Ptr CS:FX87Present,0
         JNZ     @@0AC3
         ADD     Word Ptr [DI],5C32H     ;5
@@0AC3:  JMP     short @@0ACB

@@0AC5:
```

```
                ADD         Word Ptr [DI],0A23DH      ;6
                JMP         short @@0ACB

        @@0ACB: ADD         SI,8

        @@0ACE: CMP         SI,[BP+08]
                JNB         @@0AD6
                JMP         @@08E5
        @@0AD6: POP         DS
                POP         DI
                POP         SI
                MOV         SP,BP
                POP         BP
                RET         10h
ApplyRelocs     endp

                sEnd
                End
```

6.4　自装载的 HELLOWIN

在本书所附的磁盘中，目录 CHAP6 下面是一个完整的自装载程序 HelloWin。读者应该很熟悉该程序，所以我们在这里略去它的源程序，只是讲解使用自装载库的方法。

首先编译上一节中提供的两个汇编程序，把它作为一个库放在 test.lib 中。然后通过命令 nmake 执行 makefile。

文件 C.BAT

```
masm appload;
masm reloc;
lib test.lib +appload +reloc;
nmake
```

在该文件中指定链入 test.lib 库。

文件 MAKEFILE

```
WINCC     = cl -c -G2sw -Ow -W3 -Zp -Tp
WINLINK   = link /nod
WINLIB    = slibcew oldnames libw commdlg ddeml test
WINRC     = rc -r

hellowin.exe : hellowin.obj hellowin.def
     $(WINLINK) hellowin, hellowin, NUL, $(WINLIB), hellowin
    rc -t hellowin.exe
hellowin.obj : hellowin.c
     $(WINCC) hellowin.c
```

在定义文件中指定装载过程为“__MSLANGLOAD”，该过程存放在 test.lib 库中。

文件 HELLOWIN.DEF

```
;- - - - - - - - - - - - - - - - - - -
; HELLOWIN.DEF module definition file
;- - - - - - - - - - - - - - - - - - -
NAME            HELLOWIN
DESCRIPTION     'Hello Windows Program'
APPLOADER       '__MSLANGLOAD'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       8192
```

有兴趣的读者可以先运行 C.BAT,再用第 2 章介绍的文件分析工具 PDUMP 来分析 hellowin.exe。

第 7 章

LZ 压缩算法原理与具体实现

在本书第 1 章，我们全面讲解了 Windows 执行文件的内容，其中包括未写入文档的部分和 Microsoft 出版物中不详细甚至错误的地方；在第 2 章，介绍了如何利用 Power 系列分析工具，从不同的角度全方位剖析 Windows 执行文件；接下来的第 3 章，我们看到的是一个功能完整的 File Object 的定义，以及定义这种对象的理由，并基于这一个 File Object，演示了开发一个类似 Microsoft EXEHDR 的执行文件分析工具的整个过程，其中一个个短小简洁的子程序肯定给你留下了深刻的印象。前 3 章构成了第一部分，是本书的基础。

进入第 4 章后，我们着意一步一步地引导读者跟随我们探索缤纷灿烂的 Windows 可执行文件内部世界：在第 4 章，我们神奇地“篡改”了 Windows 可执行文件，而 Windows 对此竟然全无觉察！ 在第 5 章里，有一个用宏汇编语言编写的 HELLOWIN 的例子，如果您确实读懂了这个例子，再和您用高级语言编写 Windows 应用程序的经验比较一下，就会感觉到：用汇编语言编写 Windows 应用程序更具有挑战性，需要付出更多的钻研时间。当然，报酬也是丰厚的：您能够实现那些用高级语言很难达到的性能。在第 6 章，我们搞清楚了 Windows 可执行文件的自装载技术。自装载技术是有很高的实用价值，比如在软件加密和 EXE 压缩上；同时又极为复杂，它是衡量您的 Windows 水平的标尺。知道如何编写自装载代码，您就可以在 Windows 世界里天马行空，无拘无束。有了第 4 章到第 6 章的这些高级知识，Windows 执行机制不再神秘莫测。

在本书中的最后一个部分，我们要利用前面的这些知识来探讨两个具体应用问题：Windows 可执行文件的压缩和 Windows 可执行文件的加密。Windows 可执行文件通常占用很大的存储空间，比如一套完整的 Visual C++ 1.5 需要 150M 以上的磁盘空间，常用的 Microsoft EXCEL、Word 也需要占用几十兆的空间。如何有效利用有限的存储介质？数据压缩是一条途径。当然，这种压缩要求是能够自动实时解压缩的。另外，Windows 取代 DOS 成为微机上新的软件开发平台已是事实，很多的软件开发厂商都在开发基于 Windows 的应用软件。在软件版权的法律保护并不完善的今天，过去常用的通过各种加密手段来保护软件版权的办法，同样也遇到了一个从 DOS 到 Windows 的转变的问题。

这两种需要至少有以下三个共同点：

1. 它们的操作对象都是 Windows 可执行文件，属于本书第 3 章中 File Object 的范畴。

2. 不管是压缩还是加密，它们都是在修改 Windows 可执行文件，肯定要涉及本书第 4 章的内容。

3. 在被压缩过或被加密过的程序中，Windows 所需的装载和运行信息已被隐藏，Windows 无法正常执行该程序。被压缩或加密的程序必须有自己的自装载代码。这就是前一

章讲解的技术。

作为第 8 章中 PACKWIN 软件的基础，这一章我们先来讨论“数据压缩”这一个题目。关于数据压缩的知识极为丰富，有兴趣的读者请参阅香港金山公司编写的《压缩技术经典》(学苑出版社 1994 年 8 月出版)。

7.1 数据压缩概论

数据压缩的目的是减少用于存储或传递的比特数。一般来说，数据压缩由“读符号串”和把它“转换为码字”这两者组成。如果作为结果码的代码串长度小于原始符号串，我们则说这个压缩是有效的。

对于一给定的符号或一组符号，所输出的结果取决于模型。简单地说，模型是一个数据和规则的集合。这些数据和规则用于处理输入符号并决定输出哪些码字。程序使用模型为每一符号精确地定义一个概率，而编码器再根据这些概率产生一个适当的代码。建模和编码是不同的两件事：对同一原始符号串，使用相同的编码程序，如果模型数据不同，则输出结果亦不相同。

为了很好地压缩数据，我们需要选取能以高概率预测符号的模型。具有高概率的符号信息含量低，编码上需要的比特数较少。一旦模型产生了高概率，则接下来的工作就是用一个适当数量的比特位为符号编码。

压缩数据时，需要用符号所包含的精确的信息比特数对符号进行编码。假如字符“e”只给我们 4 比特信息，那么就应该用 4 比特给它编码，假如“x”含有 12 比特信息，就应该用 12 比特给它编码。

7.2 LZ 压缩算法原理

7.2.1 原理

LZ 压缩算法是一种基于字典的压缩方法。它由 Jacob Ziv 和 Abraham Lempel 于 1977 年和 1978 年提出。字典方案是：它读入输入数据，在字典中查找出现的符号组，若找到一个串匹配，就把一个指向字点的指针或索引输出出去，而不是为每一个符号输出一个代码。匹配越长，压缩比就越高。

Ziv 和 Lempel 提出的第一个压缩算法通常被称为 LZ77，这个算法较为简单。字典由在一个“窗口”(注意：这里的“窗口”与 Microsoft Windows 中的“window”是不同的两个概念)中的所有数据串所组成，这些数据串是先前被读入窗口的输入数据。举例来说，一个文件压缩程序可以使用 4K 字节作为字典。尽管新的符号组被读入，但算法只对先前已经读入的

4K 字节数据中的串查找匹配。任何被编码的匹配以指针形式发送到输出流中。

LZ77 和它的变体产生了极具吸引力的压缩算法。其特点是:模型维护简单,编码输出简单,能编写出高效程序。市面上已经被广泛使用的压缩软件,像 PKZIP 和 LHARC,使用的就是 LZ77 程序的变体。

LZ78 程序采用另一种方法建立和维护字典。LZ78 不是在先期文本上开一个有限大小的窗口,而是使它的字典包括所有在输入文本中出现的符号。字典中的串每次加长一个字符,而不是使字典包含所有的先期文本中出现的字符串。比如,第一次"Mark"出现时串"Ma"被加入字典中,下一次"Mar "被加入,如果"Mark"再出现一次,它也被加到字典中。这个递增过程特别有利于分解使用频率高的串,并把它们加进字典中去。与 LZ77 不同,在 LZ78 中,串可以很长,这便于获得较高的压缩比。基于字典的压缩算法是:把不同长度的符号编码为一些独立的标记(token),这些标记就形成一个字典的索引。假如这些标记小于它们所代替的词组,那么就实现了压缩。

7.2.2　基于字典的压缩是如何工作的?

作为一个例子,我们先用一个流行的字典(Longman Dictionary)作为我们的"字典"。用这个字典的体制来作为编码消息的密钥,就能实现很好的压缩。使用这个方案时,我们将"the mechanics of a dictionary based compression method"8 个单词编码为:

1099/5 649/11 717/9 1/1 282/34 73/10 206/24 655/9

这个基于字典的编码方案由一个简单的查寻表构成。第一个数给出字典的页数,第二个数告诉我们该页上的单词的序号。这本字典有 1230 页,每页上单词个数小于 128,这样,1099/5 就是第 1099 页上第 5 个单词"the"的码字,1/1 就是第 1 页上第 1 个单词"a"的码字。

为了看清这个方案到底能节约多少空间,我们来考查了编码一个单词实际要用的比特数。由于一个单词可能出现在 1230 个页码中的每一页,因此我们需用 11 比特来编码页数。由于每一页不多于 128 个单词,因此每页上单词的序号仅需用 7 比特来编码,这就给出了用这个字典来编码任何一个单词所需的比特数:18 比特,也就是说每个单词需 2.25 个字节。那么,我们所编码的消息中的 8 个单词的 ASCII 码表示共需 44 个字节,而用我们现在的方案来编码仅需 18 个字节。这样,我们用字典编码将我们的报文压缩了近 60%。

现在,众多已知的基于字典的算法都是自适应的。在压缩开始时,自适应方案要么开始时不用字典,要么用一本缺省的基本字典,此时它并没有一本已完全定义好的字典。随着压缩的不断进行,算法不断添加新词组来作为已编码标记供以后使用。

下面一段 C 语言代码说明了自适应字典程序应遵循的原则。

```
for ( ; ; ) {
  word = read_word( input_file );
  dictionary_index = look_up( word, dictionary );
  if ( dictionary_index < 0 ) {
    output( word, output_file );
    add_to_dictionary( word , dictionary );
    }
  else
    output( dictionary_index, output_file );
  }
```

7.3 压缩与还原算法的实现

7.3.1 压缩

Windows 执行文件中包含代码、数据和各种资源。这些不同部分的数据有不同的统计特性，在建立模型时，应分别适应才能得到高效率的压缩比。在第 8 章讲述的 PACKWIN 中存在好几种压缩方法，它们之间互有差别，但大致途径是相同的。PACKWIN 中的压缩算法是用汇编语言编写的，而且涉及的细节处理很多，我们以 C 语言作简明的讲解。

首先将字母表中的所有符号作为单字符短语预加载到短语字典中，这样，所有的符号，既使它未曾出现在输入流中，也都能够被立即编码。

以下是压缩算法的最简单的形式。该算法总是试图为每一个已知的串输出代码。每输出一个新的代码，就向短语字典中加入一个新的串。请仔细阅读下面压缩算法的描述。

```
old_string[ 0 ] = getc(input);
old_string[ 1 ] = '\0';
while ( ! feof( input ) ) {
  character = getc( input );
  strcpy( new_string, old_string );
  strncat( new_string, &character, 1 );
  if ( in_dictionary( new_string ) );
    strcpy( old_string, new_string );
  else {
    code = look_up_dictionary( old_string );
    output_code( code );
    add_to_dictionary( new_string );
    old_string[ 0 ] = character;
    old_string[ 1 ] = '\0';
    }
  }
code = look_up_dictionary( old_string );
output_code( code );
```

接下来给出了一个用于演示算法的样本串。输入串是一组英语单词，它取自一个拼字字典，由空格字符“ ”分离开。首次通过循环时，执行一个检查，用以查看串“ W”是否在表中。由于它不在表中，故为“ ”输出一代码，而且串“ W”被加进表中，由于字典已有代码 0 至 255——它们被定义作为 256 个可能字符的值，所以第一个串被定义为代码 256。第三个字符后，“E”已被读入，第二个串代码“WE”被加入表中，而字母“W”的代码被输出。在第二个词中，读入字符“ ”和“W”，它与串数值 256 相匹配。然后输出代码 256，并把一个三字符串加进串表中。这个过程连续地进行下去，直到串被用完并且所有的代码已经都输出。

输入串:" WED WE WEE WEB WET"		
输入字符	输出代码	新码值及对应的串
" W"	' '	256 = " W"
"E"	'W'	257 = "WE"
"D"	'E'	258 = "ED"
" "	'D'	259 = "D "
"WE"	256	260 = " WE"
" "	'E'	261 = "E "
"WEE"	260	262 = " WEE"
" W"	261	263 = "EW"
"EB"	257	264 = "WEB"
" "	'B'	265 = "B "
"WET"	260	266 = " WET"
<EOF>	'T'	

串的样本输出是用结果串表给出的。由于每输出一代码就加进新串,因此串表迅速装满。对这一高冗余的输入,输出了 5 个置换代码和 7 个字符。若我们输出 9 比特码,则 19 个字符的输入串将减至 13.5 字节的输出串。当然,本例是特别选来演示代码置换的。在实际的例子中,通常是建起相当长的表之后才开始压缩,一般是在读入了至少上百个字节之后。

7.3.2　还原

还原算法是与压缩算法相伴而来的,它取得压缩算法产生的输出代码流,再生成确切的输入流,该算法高效率的表现之一是无需传递字典给还原器。利用数据输入流可以在压缩期间精确地建立表,这是因为压缩算法中,总是在输出流使用之前输出代码的短语和字符项。这样,被压缩的数据就无需携带一个巨大的字典。

```
old_string[ 0 ] = input_bits();
old_string[ 1 ] = '\0';
putc( old_string[ 0 ], output )
while ( ( new_code = input_bits() ! = EOF ) {
  new_string = dictionary_lookup( new_code );
  fputs( new_string, output );
  append_char_to_string( old_string, new_string[ 0 ] );
  add_to_dictionary( old_string );
  strcpy( old_string, new_string )
}
```

上面程序是一个粗略的 C 语言片段。与压缩算法一样,它每次读入一新代码,并把新串加进串表中。另外,把每个到来的代码译为串并发给输出。下面是对前面压缩所得结果

算法的输出。结果串表看起来与压缩期间建立的表完全相同。输出串与压缩算法的输入串是完全相同的。与压缩程序一样，前 256 个代码用于定义单字符串。

输入代码:" WED<256>E<260><261><257>B<260>T"				
新输入代码	旧代码	输出串	字符	新表
' '	' '	""	'W'	256 = " W"
'W'	' '	"W"	'E'	257 = "WE"
'E'	'W'	"E"	'D'	258 = "ED"
'D'	'E'	"D"	"	259 = "0"
256	'D'	"W"	'E'	260 = " WE"
'E'	256	"E"	' '	261 = "E "
260	'E'	"WE"	'E'	262 = " WEE"
261	260	"E"	'W'	263 = "E W"
257	261	"WE"	'B'	264 = "WEB"
'B'	257	"B"	' '	265 = "B "
260	'B'	" WE"	'T'	266 = " WET"
'T'	260	"T"		

7.4 PACKWIN 中用到的压缩函数

本节介绍 PACKWIN 中用到的一组压缩函数，这些程序放在本书所附磁盘的 CHAP7 目录下。

```
{ PACKWIN 中用到的一组压缩函数。}

unit LzPack;

interface

{ LZ 数据流压缩。需要的参数有:
SBuf:       Pointer to Source Buffer
SBufLen:    Length of Source Buffer
ReadFunc:   Read access function pointer
TBuf:       Pointer to Target Buffer
TBufLen:    Length of Target Buffer
WriteFunc:  Write access function pointer}
procedure LzStreamPack(
  SBuf: Pointer;
  SBufLen: Word;
  ReadFunc: Pointer;
  TBuf: Pointer;
```

```
  TBufLen: Word;
  WriteFunc: Pointer); far;

{ LZ 数据流解压缩。需要的参数有:
SBuf:        Pointer to Source Buffer
SBufLen:     Length of Source Buffer
ReadFunc:    Read access function pointer
TBuf:        Pointer to Target Buffer
TBufLen:     Length of Target Buffer
WriteFunc:   Write access function pointer }
procedure LzStreamUnPack(
  SBuf: Pointer;
  SBufLen: Word;
  ReadFunc: Pointer;
  TBuf: Pointer;
  TBufLen: Word;
  WriteFunc: Pointer); far;

{ LZ 数据块压缩。需要的参数有:
SBuf:Pointer to Source Buffer
SLen:Length of Source Buffer
TBuf:Pointer to Target Buffer
TLen:Length of Target Buffer
返回值为一个 Word,即压缩后数据的长度。}
function LzBlockPack(
  SBuf: Pointer;
  SLen: Word;
  TBuf: Pointer;
  TLen: Word        ): Word; far;

{ LZ 数据块解压缩。需要的参数有:
Buf:Pointer to Buffer
LzedLen:Length of Packed data in Source Buffer
BufLen:Total Length of the Buffer
解压缩出的数据仍然在 Buffer 中,其长度用返回值(Word)表示。}
function LzBlockUnPack(
  Buf: Pointer;
  LzedLen: Word;
  BufLen: Word): Word; far;

{ LZ 数据块解压缩。需要的参数有:
SBuf:Pointer to Source Buffer
TBuf:Pointer to Target Buffer
压缩过的数据放在 SBuf 中,其长度根据 SBuf 中的字符串结束标志判断。
解压缩出的数据的长度用返回值(Word)表示。}
function LzBlockUnPackEx(
  SBuf: Pointer;
  TBuf: Pointer): Word; far;

{ LZ 文件压缩。需要的参数有:
SF:Source File Handler
```

```
TF:Target File Handler
返回值为文件压缩是否成功的标志。}
function LzFilePack(
  SF: Word;
  TF: Word): Boolean; far;

{ LZ 文件解压缩。需要的参数有:
SF:Source File Handler
TF:Target File Handler
返回值为文件解压缩是否成功的标志。}
function LzFileUnPack(
  SF: Word;
  TF: Word          ): Boolean; far;

{ 压缩 EXE 文件。需要的参数有:
InFile:      Input File Name
OutFile:     Output File Name
PrintMsg:
ForcePack:   是否强行压缩
LinkOverlay:
CutOverlay:
RemoveDoubleReloc:    }
function LiteExe(
  InFile: String;
  OutFile: String;
  PrintMsg: Boolean;
  ForcePack: Boolean;
  LinkOverlay: Boolean;
  CutOverlay: Boolean;
  RemoveDoubleReloc: Boolean): Byte; far;

implementation

uses ExtTools;

{ $L LZPACKA.OBJ}
{ 以下这些过程的实现放在目标文件 LZPACKA.OBJ 中。}
procedure LzStreamPack; external;
procedure LzStreamUnPack; external;
function LzBlockPack; external;
function LzBlockUnPack; external;
function LzBlockUnPackEx; external;
function LzFilePack; external;
function LzFileUnPack; external;

procedure Booter; external;
{ $L Booter.obj}
type
  PBooter = ^TBooter;
  TBooter = record
    _IP,
```

```
    _CS,
    _SP,
    _SS,
    _LzedEXEParaLen,
    _BooterPlaceSeg,
    _BooterLen,
    _MarkLen: Word
  end;

{ 压缩 EXE 文件。}
function LiteExe(
  InFile: String;
  OutFile: String;
  PrintMsg: Boolean;
  ForcePack: Boolean;
  LinkOverlay: Boolean;
  CutOverlay: Boolean;
  RemoveDoubleReloc: Boolean): Byte;

type
  ExeHeader = Record
    Sign,                    { 00h  'MZ' or 'ZM' }
    lenImage,                { 02h  Length of image }
    numPages,                { 04h  Size of file in 512 - byte pages }
    numRelocEntry,           { 06h  Number of relocation - table items }
    sizeHeader,              { 08h  Size of header in paragraphs }
    numMinAlloc,             { 0Ah  Minimum number of paragraphs above }
    numMaxAlloc,             { 0Ch  Maximum number of paragraphs above }
    _SS,                     { 0Eh  Displacement of stk seg in paragraphs }
    _SP,                     { 10h  offset in SP register }
    CheckSum,                { 12h  word Checksum }
    _IP,                     { 14h  IP register offset }
    _CS,                     { 16h  code segment displacement }
    ofsRelocList,            { 18h  Displacement of first relocation item }
    noOverlay: word;         { 1Ah  Overlay number (Resident code = 0) }
    Reserved: array [ $1C.. $3B] of char;
    ofsWinHeader: LongInt;   { 3Ch  New exe file header offset }
  end;
  TReloc = record
    Ofs,
    Seg: word;
  end;
  TRelocTab = array[0..0] of TReloc;
  PRelocTab = ^TRelocTab;
  TLongints = array[0..0] of Longint;
  PLongints = ^TLongints;
const
  Len = $100;
  CopyRight: String = 'ZLITE (C) 1993 WYellow Rose Software.';
  TempFile : String = 'ZLITE.$ $ $';
var
```

```
  InF, OutF: File;
  Buf: Array[0..Len-1] of byte;
  rLen, wLen: Word;
  OldHead, NewHead: ExeHeader;
  NewBooter: TBooter;
  ofsBooter: Longint;
  lenBooter: Word;
  Relocs: PRelocTab;
  I, J: Word;
  tmpB: Byte;
  tmpW: Word;
  ri1, ri2: Longint;
  DoubleRelocNum: Word;
  HasOverlay: Boolean;
  IsWinProg: Boolean;
  NeHeaderLen: Word;
  function MinWord(W1, W2: Word): Word;
  begin
    if W1 < W2 then
      MinWord := W1
    else
      MinWord := W2;
  end;
  procedure QuickSort(var A: TLongints; Lo, Hi: Integer);
    procedure Sort(l, r: Integer);
    var
      i, j: integer;
      x, y: longint;
    begin
      i := l; j := r; x := a[(l+r) DIV 2];
      repeat
        while a[i] < x do i := i + 1;
        while x < a[j] do j := j - 1;
        if i <= j then
        begin
          y := a[i]; a[i] := a[j]; a[j] := y;
          i := i + 1; j := j - 1;
        end;
      until i > j;
      if l < j then Sort(l, j);
      if i < r then Sort(i, r);
    end;
  begin {QuickSort};
    Sort(Lo,Hi);
  end;

  function CopyFileBlock(var sf, tf: File; Len: LongInt): boolean;
  const
    DefMem = $8000;
  var
    P: PBytes;
```

```
    Error: Boolean;
    MemRead, NumRead, NumWritten: Word;
  begin
    GetMem(P, DefMem);
    Error := False;
    while (Len <> 0) and (not Error)do begin
      if Len < DefMem then
        MemRead := Len
      else MemRead := DefMem;
      BlockRead(sF, P^, MemRead, NumRead);
      BlockWrite(tF, P^, NumRead, NumWritten);
      if NumRead <> NumWritten then Error := True;
      Dec(Len, MemRead);
    end;
    FreeMem(P, DefMem);
    CopyFileBlock := not Error;
  end;

begin
  LiteExe := 0;
  HasOverlay := False;
  IsWinProg := False;
  Assign(InF, InFile);
  Reset(InF, 1);

{- - - - - - - - - Separate EXE head - - - - - - - - - -}

  BlockRead(InF, OldHead, SizeOf(OldHead), rLen);
  if rLen < $20 then
  begin
    if PrintMsg then
      Writeln('File too small');
    LiteExe := 1;
    Exit;
  end;
  if (OldHead.Sign <> $4D5A) and (OldHead.Sign <> $5A4D) then
  begin
    if PrintMsg then
      Writeln('Not .EXE file, use COM2EXE convert it.');
    LiteExe := 2;
    Exit;
  end;
  ri1 := Longint(OldHead.numPages - Byte(OldHead.lenImage<>0)) * $200
+ OldHead.lenImage;
  if (OldHead.noOverlay <> 0) or (ri1 <> FileSize(InF)) then
  begin
    if not CutOverlay then
    if not LinkOverlay then
    begin
      LiteExe := 3;
      Write('File: ', UpcaseStr(InFile),
```

```
        ' may contain overlays. Compress (y/n)? ');
      if Upcase(ReadKey) <> 'Y' then
        Exit;
      Writeln('yes');
    end;
    if PrintMsg then
      Write('Creatting...');
    if (OldHead.ofsWinHeader > $70) and (OldHead.ofsWinHeader < ri1) then
    begin
      Seek(InF, OldHead.ofsWinHeader);
      Blockread(InF, tmpW, 2);
      if tmpW = $454E then { 'NE' Window program, Cut NE_Header }
      begin
        NeHeaderLen := ri1 - OldHead.ofsWinHeader;
        ri1 := OldHead.ofsWinHeader;
        OldHead.lenImage := ri1 mod $200;
        OldHead.numPages := ri1 div $200 + Byte(OldHead.lenImage<>0);
        IsWinProg := True;
      end;
    end;
    Close(InF);
    Assign(InF, InFile);
    Assign(OutF, TempFile);
    Reset(InF, 1);
    Rewrite(OutF, 1);
    if not CopyFileBlock(InF, OutF, ri1) then
    begin
      LiteExe := 4;
      if PrintMsg then
        Writeln('Error');
      Close(InF);
      Close(OutF);
      Exit;
    end;
    Close(InF);
    Close(OutF);
    Assign(InF, TempFile);
    Reset(InF, 1);
    HasOverlay := not CutOverlay;
    if PrintMsg then
      Writeln('ok');
  end;
  if (OldHead.Reserved[$1E] = 'Y') and (OldHead.Reserved[$1F] = 'R') then
  begin
    LiteExe := 10;
    if PrintMsg then
      Writeln('File already packed');
    if not ForcePack then
      Exit;
  end else
  if (OldHead.Reserved[$1E] = 'P') and (OldHead.Reserved[$1F] = 'K') then
```

```
begin
  LiteExe := 11;
  if PrintMsg then
    Writeln('File already packed by PKLITE');
  if not ForcePack then
    Exit;
end else
if (OldHead.Reserved[$1C] = 'L') and (OldHead.Reserved[$1D] = 'Z') then
begin
  LiteExe := 12;
  if PrintMsg then
    Writeln('File already packed by LZEXE');
  if not ForcePack then
    Exit;
end;  { else

  Borland Link - head $20 'rj' or 'jr'
  Borland Overlays - follow the load image word is 46 F4
  Borland Symbal - follow the load image word is FB 52
}
Assign(OutF, OutFile);
if FileExists(OutFile) then
  Erase(OutF);
ReWrite(OutF, 1);
FillChar(NewHead, SizeOf(NewHead), 0);
NewHead.Sign := $5A4D;
NewHead.lenImage := 0;
Newhead.numPages := 0;
NewHead.numRelocEntry := 0;
NewHead.sizeHeader := SizeOf(NewHead) shr 4;
NewHead.numMinAlloc := 0;
NewHead.numMaxAlloc := $FFFF;
NewHead._SS := 0;
NewHead._SP := 0;
NewHead.CheckSum := $6511;
NewHead._IP := 0;
NewHead._CS := 0;
NewHead.ofsRelocList := $1C;
NewHead.noOverlay := 0;
NewHead.Reserved[$1E] := 'Y'; NewHead.Reserved[$1F] := 'R';
NewHead.ofsWinHeader := $40;
Move(CopyRight[1], NewHead.Reserved[$20], $3B - $20);
BlockWrite(OutF, NewHead, SizeOf(NewHead), wLen);
if wLen <> SizeOf(NewHead) then
begin
  LiteExe := 20;
  if PrintMsg then
    Write('File write error!');
  Close(InF);
  Close(OutF);
```

```
    Exit;
  end;

{- - - - - - - - - - - - - Packing - - - - - - - - - - - - - - - - -}

  if PrintMsg then
    Write('Packing...');
  Seek(InF, Longint(OldHead.sizeHeader) shl 4);
  if LzFilePack(word((@InF)^), word((@OutF)^)) then
  begin
    if PrintMsg then
      Writeln('ok');
  end
  else
  begin
    LiteExe := 30;
    if PrintMsg then
      Writeln('error found');
    Close(InF);
    Close(OutF);
    Exit;
  end;
  FillChar(NewBooter, SizeOf(NewBooter), $65);
  ofsBooter := FilePos(OutF);
  if ofsBooter mod $10 <> 0 then      { para align }
  begin
    lenBooter := ofsBooter shr 4 shl 4 + $10 - ofsBooter;
    BlockWrite(OutF, NewBooter, lenBooter and $F);
  end;

{- - - - - - - - - - - - - - - Mount Booter - - - - - - - - - - - - - - - - - - -}

  Move((@Booter)^, NewBooter, SizeOf(NewBooter));
  lenBooter := NewBooter._IP;
  ofsBooter := FilePos(OutF);
  if not PrintMsg then    { Hide Long Mark String }
    Dec(lenBooter, $10);
  BlockWrite(OutF, (@Booter)^, lenBooter, wLen);
  if wLen <> lenBooter then
  begin
    LiteExe := $40;
    if PrintMsg then
      Writeln('Mount BOOTER error!');
    Close(InF);
    Close(OutF);
    Exit;
  end;
  if not PrintMsg then    { Mount Short Mark String }
  begin
    tmpW := $4751; {'QG'}
    BlockWrite(OutF, tmpW, 2, wLen);
```

```
  if wLen <> 2 then
  begin
    LiteExe := $41;
    if PrintMsg then
      Writeln('Mount BOOTER error!');
    Close(InF);
    Close(OutF);
    Exit;
  end;
end;

{---------- Optimize and Mount Relocation Tabel ----------}

if OldHead.numRelocEntry <> 0 then
begin
  if PrintMsg then
    Write('Optimizing...');
  if PrintMsg then
    Write('read...');
  GetMem(Relocs, OldHead.numRelocEntry * 4);
  if Relocs = nil then
  begin
    LiteExe := $48;
    Close(InF);
    Close(OutF);
    if PrintMsg then
      Writeln(OldHead.numRelocEntry,
          ' Reloc Entrys, Out of memory');
    Exit;
  end;
  Seek(InF, Longint(OldHead.ofsRelocList));
  BlockRead(InF, Relocs^, OldHead.numRelocEntry * 4, rLen);
  if rLen <> OldHead.numRelocEntry * 4 then
  begin
    LiteExe := 50;
    if PrintMsg then
      Writeln('Relocation Table error!');
    Close(InF);
    Close(OutF);
    Exit;
  end;

  {..... convert Seg.Ofs to longint .....}
  for I := 0 to OldHead.numRelocEntry do
    Longint(Relocs^[I]) := Longint(Relocs^[I].Seg) shl 4 +
                                    Relocs^[I].Ofs;
{..... sort reloc table .....}
  if PrintMsg then
    Write('sort...');
  case OldHead.numRelocEntry of
    1: ;
```

```
    2: begin
         I := 1;
         if Longint(Relocs^[0]) > Longint(Relocs^[I]) then
         begin
           Move(Relocs^[0], ri1, 4);
           Move(Relocs^[I], Relocs^[0], 4);
           Move(ri1, Relocs^[I], 4);
         end;
       end;
    else
     (* Insert sort
       for I := 0 to OldHead.numRelocEntry - 2 do
       begin
         for J := I + 1 to OldHead.numRelocEntry - 1 do
           if Longint(Relocs^[I]) > Longint(Relocs^[J]) then
           begin
             Move(Relocs^[I], ri1, 4);
             Move(Relocs^[J], Relocs^[I], 4);
             Move(ri1, Relocs^[J], 4);
           end;
         if PrintMsg then
           case I mod 4 of
             0: Write('-'#8);
             1: Write('\'#8);
             2: Write('|'#8);
             3: Write('/'#8);
           end;
       end;
     *)
     { Quik Sort }
       QuickSort(TLongints(Relocs^), 0, OldHead.numRelocEntry - 1);

end; {case}

{.............. Pack reloc table .....................}
if PrintMsg then
  Write('pack...');
ri1 := 0;
DoubleRelocNum := 0;
for I := 0 to OldHead.numRelocEntry - 1 do
begin
  ri2 := Longint(Relocs^[I]);
  tmpW := ri2 - ri1;
  tmpB := tmpW and $FF;
  if ri2 - ri1 = 0 then
  begin
    Inc(DoubleRelocNum);
    if not RemoveDoubleReloc then
    begin
      tmpB := 0;
      tmpW := $1;
```

```
        BlockWrite(OutF, tmpB, 1);
        BlockWrite(OutF, tmpW, 2);
      end;
    end else
    if ri2 - ri1 <= $FF then
    begin
      BlockWrite(OutF, tmpB, 1);
    end else
    if ri2 - ri1 <= $FFFF then
    begin
      tmpB := 0;
      BlockWrite(OutF, tmpB, 1);
      BlockWrite(OutF, tmpW, 2);
    end else
    if ri2 - ri1 <= $FFFFFFFF then
    begin
      tmpB := 0;
      tmpW := 0;
      ri1 := ri2 - ri1;
      repeat
        BlockWrite(OutF, tmpB, 1);
        BlockWrite(OutF, tmpW, 2);
      until ri1 < $FFF;
      tmpW := ri1;
      BlockWrite(OutF, tmpB, 1);
      BlockWrite(OutF, tmpW, 2);
    end;
    ri1 := ri2;
  end;
  FreeMem(Relocs, OldHead.numRelocEntry * 4);
  if PrintMsg then
  begin
    case DoubleRelocNum of
      0: ;
      1: if RemoveDoubleReloc then
           Write('Remove ', DoubleRelocNum, ' double reloc...')
         else
           Write('found ', DoubleRelocNum, ' double reloc...');
      else
         if RemoveDoubleReloc then
           Write('Remove ', DoubleRelocNum, ' double relocs...')
         else
           Write( 'found ', DoubleRelocNum, ' double relocs...');
    end;
    Writeln('ok');
  end;
end;
tmpB := 0; tmpW := $65;  { Optimize Reloc Table end mark }
BlockWrite(OutF, tmpB, 1);
BlockWrite(OutF, tmpW, 2);
```

```
{- - - - - - - - - - - - - Modify Booter Parameter - - - - - - - - - - - - - - -}

  NewBooter._IP := OldHead._IP;
  NewBooter._CS := OldHead._CS;
  NewBooter._SP := OldHead._SP;
  NewBooter._SS := OldHead._SS;
  NewBooter._LzedEXEParaLen := (ofsBooter - SizeOf(NewHead)) shr 4;
  NewBooter._BooterLen := FilePos(OutF) - ofsBooter;
  NewBooter._BooterPlaceSeg :=
            (FileSize(InF) - Longint(OldHead.sizeHeader shl 4) + $F) shr 4
          - NewBooter._LzedEXEParaLen
          + $10
          + $2
          + MinWord(OldHead.numMinAlloc, $400);
  if PrintMsg then
    NewBooter._MarkLen := $10   { Long Mark }
  else
    NewBooter._MarkLen := $02; { Short Mark }
  Seek(OutF, ofsBooter);
  BlockWrite(OutF, NewBooter, SizeOf(NewBooter));

{- - - - - - - - - - - - - - - Modify Exe Head - - - - - - - - - - - - - - - - -}

  NewHead.lenImage := FileSize(OutF) mod $200;
  NewHead.numPages := FileSize(OutF) div $200 + Byte(NewHead.lenImage<>0);
  if IsWinProg then
  begin
    NewHead.lenImage := (FileSize(OutF) + NeHeaderLen) mod $200;
    NewHead.numPages := (FileSize(OutF) + NeHeaderLen) div $200 +
Byte(NewHead.lenImage<>0);
  end;
  NewHead._SS := NewBooter._BooterPlaceSeg
          + NewBooter._LzedEXEParaLen
          + (NewBooter._BooterLen + $F) shr 4;
  NewHead._SP := $80;
  NewHead._IP := SizeOf(NewBooter);
  NewHead._CS := NewBooter._LzedEXEParaLen;
  NewHead.numMinAlloc := OldHead.numMinAlloc
          + (NewBooter._BooterLen + NewHead._SP + $F) shr 4
          + $2;
  NewHead.numMaxAlloc := wMax(OldHead.numMaxAlloc, NewHead.numMinAlloc);
  NewHead.ofsWinHeader := FileSize(OutF);
  Seek(OutF, 0);
  BlockWrite(OutF, NewHead, SizeOf(NewHead), wLen);
  if wLen <> SizeOf(NewHead) then
  begin
    LiteExe := 60;
    if PrintMsg then
      Writeln('Update EXE Header error!');
    Close(InF);
    Close(OutF);
```

```
    Exit;
  end;

{- - - - - - - - - - - - - - - - Link Overlay - - - - - - - - - - - - - - - - - -}

  if HasOverlay and LinkOverlay then
  begin
    if PrintMsg then
      Write('Linking...');
    Close(InF);
    Assign(InF, InFile);
    Reset(InF, 1);
    if IsWinProg then
      Seek(InF, Longint(OldHead.numPages -
           Byte(OldHead.lenImage<>0)) * $200
               + OldHead.lenImage - NeHeaderLen)
    else;
      Seek(InF, Longint(OldHead.numPages -
           Byte(OldHead.lenImage<>0)) * $200
               + OldHead.lenImage);
    Seek(OutF, FileSize(OutF));
    CopyFileBlock(InF, OutF, FileSize(InF) - FilePos(InF) + 1);
    if PrintMsg then
      Writeln('ok');
  end;

{- - - - - - - - - - - - - - - - Show Result - - - - - - - - - - - - - - - - - -}

  if ((FileSize(OutF) + FileSize(OutF) shr 4) > FileSize(InF)) and
     not ForcePack then
  begin
    LiteExe := $70;
    if PrintMsg then
      Writeln('File: ', UpcaseStr(InFile), ' pack needless');
    Assign(OutF, OutFile);
    Erase(OutF);
    Close(InF);
  end
  else
  begin
    if PrintMsg then
    begin
      ReSet(InF, 1);
      ReSet(OutF, 1);
      Writeln;
      Writeln('   Orignal Size: ', Longint(FileSize(InF)),
              '   Packed Size: ', FileSize(OutF),
              '   Rate: ', (FileSize(OutF) * 100) div FileSize(InF), '%');
    end;
    Close(InF);
    Close(OutF);
```

```
    end;
    if FileExists(TempFile) then
    begin
      Assign(InF, TempFile);
      Erase(InF);
    end;
  end; { LiteExe }

  end.
```

第 8 章

开发 Windows 执行文件压缩软件

执行文件压缩工具能减少磁盘空间，节约传输时间和磁盘 CACHE 资源，有效地防止运用逆向工程的方法破译程序。目前，Windows 的装机量已突破 5000 万套，Windows 操作系统正逐步取代 DOS，成为 PC 上的主流操作系统。DOS 上的压缩归档工具同样适用于 Windows，但 Windows 上的执行文件压缩工具很少见。

Windows 执行文件的尺寸比 DOS 上的要大得多，如 WINWORD.EXE 长达 2M 多。开发一个 Windows 上的执行文件压缩工具非常必要。Windows 执行文件格式较 DOS 的文件格式复杂得多，一般程序员对于 Windows 系统的了解还不够深入，开发压缩工具存在一定的难度。

经过一段时间的研究，我们开发了一个 Windows 执行文件压缩工具 PACKWIN。目前的 PACKWIN 能够压缩绝大部分 Windows 执行文件，但还需要进一步完善。考虑到目前还存在许多 DOS 上的用户，PACKWIN 也能够压缩 DOS 上的执行程序。PACKWIN 的 shareware 版本在 Internet 的 ftp.cica.indiana.edu 节点上可以找到，所在的目录是/pub/pc/win/util，读者可以用 FTP 取得该软件。

本章讲述 Windows 执行文件压缩工具 PACKWIN 具体实现过程。在前面的章节中，我们已经清楚地讲述了如何直接修改 Windows 执行文件的信息，并且介绍了常用的压缩算法和文件对象 File Object。要编写 Windows 执行文件的压缩软件，须把前面的这些知识组合起来。

8.1 DOS 下压缩软件简述

运行在 DOS 上的压缩工具分为两类：压缩归档工具和执行文件压缩工具。

压缩归档工具的功能是把多个文件压缩在一个文件中，以节约磁盘空间，便于保存和传递，需要时用相应的解压缩工具展开。著名的压缩归档软件有 PKZIP，LHA，ARJ 等，这些软件已经相当成熟，功能完备适用，是广大用户非常喜爱的必备工具。由于与本章的主题无关，这里不介绍这些压缩归档软件。

另一类工具是执行文件压缩工具。与压缩归档工具只做单纯的压缩不同，执行文件压缩工具还必须具备另外的特性：压缩过的可执行文件应该能直接执行，自动展开，文件的可执行性不能被破坏。即：压缩过的文件中存在展包代码，随着程序的执行，展包代码自动展开压缩过的内容。这一点与归档软件明显不同。流行的执行文件压缩工具有 PKLITE、

LZEXE 等。MS－DOS 6.0 的外部命令中有不少程序是用 PKLITE 压缩过的。

使用执行文件压缩工具还有一个好处:它能增大程序的跟踪和破译难度,在一定程度上保护程序。如:PKLITE 有一个参数(－E),该参数的功能是把一个程序压缩后不允许展开,可以防止程序被 Sourcer 这样的工具反汇编。

下面介绍 PKLITE 的使用方法,以加深读者对执行文件压缩工具的理解。

不带任何参数,直接执行 PKLITE,PKLITE 会显示使用方法。

```
C:\ >PKLITE
PKLITE (tm) Executable File Compressor Version 1.15 7-30-92
Copyright 1990-1992 PKWARE Inc. All Rights Reserved. Patent No. 5,051,745

Usage: PKLITE [options] [d:][/path]Infile [[d:][/path]Outfile]
Options are:
  -a = always compress files with overlays and optimize relocations
  -b = make backup .BAK file of original
  -e = make compressed file unextractable ( * commercial version only * )
  -l = display software license screen
  -n = never compress files with overlays or optimize relocations
  -o = overwrite output file if it exists
  -r = remove overlay data
  -u = update file time/date to current time/date
  -x = expand a compressed file

( * ) See documentation and license screen for more information

If you find PKLITE easy and convenient to use, a registration of $ 46.00
would be appreciated. Registration includes one free upgrade to the
software and a printed manual. Please state the version of the software
that you currently have. Send check or money order to:
                        PKWARE, Inc.
                        9025 N. Deerwood Drive
                        Brown Deer, WI 53223
```

PKLITE 的命令行参数含义如下:

-a 压缩带覆盖的文件,并优化重定位项。这样能够获得更大的压缩比。

-b 生成备份文件。

-e 生成不能展开的文件。

-l 显示软件的版权信息。

-n 不压缩带覆盖的文件,不优化重定位项。这样兼容性比较好。

-o 假如输出文件存在,覆盖输出文件。

-r 去掉覆盖数据。

-u 更新文件的时间日期。

-x 展开压缩过的文件。

用 PKLITE 把 PCTOOLS.EXE 压缩成 PCT.EXE。PCT.EXE 可以直接运行。

```
C:\ > PKLITE PCTOOLS.EXE PCT.EXE
PKLITE (tm) Executable File Compressor Version 1.15 7-30-92
Copyright 1990-1992 PKWARE Inc. All Rights Reserved. Patent No. 5,051,745
```

```
File: PCTOOLS.EXE may contain overlays. Compress (y/n)? Y

Compressing: PCTOOLS.EXE into file PCT.EXE
Original Size: 172035  Compressed Size: 103440  Ratio: 39.9
```

带 -X 参数把 PCT.EXE 展开成 PCTOOLS.EXE。

```
C:\ > PKLITE -X PCT.EXE
PKLITE (tm) Executable File Compressor Version 1.15 7-30-92
Copyright 1990-1992 PKWARE Inc. All Rights Reserved. Patent No. 5,051,745

Expanding: PCT.EXE
Original Size: 103440  Expanded Size: 172035
```

8.2　Windows 执行文件压缩工具 PACKWIN

本节介绍 PACKWIN 的技术特点和功能，下一节说明 PACKWIN 的实现过程。

PACKWIN 能快速压缩 MS-DOS 和 Microsoft Windows 上的应用软件，压缩后不影响他们的运行，而且压缩后程序的执行速度和压缩前相当，用户一般觉察不到因压缩展开而引起的延迟。比如说，用 PACKWIN 压缩 Windows 3.1 中 SETUP.EXE，程序大小由 422K 减少为 250K，压缩掉 172K，约占原来大小的 41%。由于磁盘的访问时间比程序展开时间要慢，程序压缩比比较高时，压缩的过程甚至提高了装入速度。

PACKWIN 的用法很简单，只需在 PACKWIN 后面加上需要压缩的文件即可。DOS 执行文件名后缀为“EXE”或“COM”，Windows 上的文件后缀名为“EXE”或“DLL”。还可以使用“*”和“?”匹配一批文件。如：PACKWIN W*.EXE。PACKWIN 能够保留原来的文件，将原来的程序后缀名改为“OLD”。

```
C:\ > PACKWIN
    PACKWIN DOS and Windows Executable File Compressor V1.0 hareware
    Copyright (c) 1993, 1994 Yellow Rose Workgroup. All Rights Reserved.

    Written by Mr. Lei Jun and Mr. Wang Quanguo

    Usage: PACKWIN <Infile> [<Outfile>] [Options]
    Options:
      /r   -- Disable Resource compression
      /b   -- Enable Bitmap compression
      /w   -- Disable Winstub compression
      /m   -- Display more messages
      /win -- Only compress Windows Application
      /dos -- Only compress MS-DOS Application
```

命令行参数的含义如下：

/r 关掉资源压缩的开关
（有些文件的资源格式不标准，如果压缩后有问题，选择这个选择项试一下）

/b 激活 BITMAP 图象的压缩开关(选用该选择项可以进一步提高压缩比)
/w 关掉 WINSTUB 的压缩开关(如果文件压缩后在 DOS 下运行出错,关掉该压缩开关)
/m 显示更多的调试信息
/win 只压缩 Windows 应用程序
/dos 只压缩 DOS 应用程序

程序压缩后应在 Windows 下试运行加以检验。如果在 Windows 上运行有问题,可以改变命令行选择项开关试一试,如不压缩资源,不压缩 WINSTUB 等。如果没有问题,可以进一步压缩 BITMAP。有些程序中的 BITMAP 格式不标准,压缩后的程序在 Windows LoadBitmap 时可能会出错。

需要说明的是有些文件是无法被 PACKWIN 压缩的:

(1) 非标准的 DOS MZ 格式或 Windows 3.1 的 NE 文件格式,PACKWIN 不能正确压缩。如:用 Borland Pascal 编译器编译成的保护模式下应用程序是 BOSS 格式,该格式虽然能被 DOS 承认,但 PACKWIN 无法正确接受。

(2) 有的执行文件为了防病毒或者防止程序被破坏,在执行的过程中会随时检查自身代码的完整性。如果压缩这样的程序,则运行时会提示程序被修改或感染有病毒,不能正常运行。如 Visual Basic 中的 VB.EXE, Windows 系统中的 WINHELP.EXE。

(3) 有的执行程序把配置信息写入自身的代码中,如果程序被压缩或加密的话,数据就有可能被破坏。如一些加密安装软件。

(4) PACKWIN 不能压缩自装载的 Windows 程序,这是因为 PACKWIN 不能识别某个程序特殊的自装载过程。

PACKWIN 用法如下:

```
C:\ > PACKWIN \WINDOWS\WRITE.EXE W.EXE
PACKWIN DOS and Windows Executable File Compressor V1.0 Shareware
Copyright (c) 1993, 1994 Yellow Rose Workgroup. All Rights Reserved.

Compressing: \WINDOWS\WRITE.EXE into file W.EXE ...
Original: 244976 Compressed: 169104 Ratio: 30.97 %
```

PACKWIN 目前还没有展开(回收)功能,比如说将上面的 w.exe 还原成 write.exe,因为这样做实现起来比较麻烦。在将来的正式发行版本中可能会设计展开功能,我们会随时将最新版本放在 Internet 国际网中,读者可以用 FTP 获取。同时我们希望拥有 PACKWIN 的读者把 PACKWIN 介绍给其他的朋友。

8.3 PACKWIN 的实现过程

PACKWIN 不仅能压缩 Windows 执行文件,还能压缩 DOS 执行文件。PACKWIN 压缩 DOS 执行文件的过程与 PKLITE 大致相同,压缩 Windows 执行文件的方法却相当复杂。本节先简述压缩 DOS 执行文件的一般过程以及被压缩程序的执行过程,然后叙述 PACKWIN 压缩 Windows 执行文件的基本原理,接着讨论压缩中技术难点的实现,最后给出

PACKWIN 的编程实现。PACKWIN 中涉及的技术问题太多，我们只讲解了实现 PACKWIN 的关键技术。

8.3.1　DOS 执行文件的压缩和执行

DOS 执行文件的压缩过程是先压缩代码和数据，然后根据压缩算法再加上相应的展包模块。在 DOS 下启动被压缩程序时，先进入展包模块，展包模块在内存中展开被压缩的代码和数据段，并进行重定位，最后把控制权转入原来的程序进入点。下面简述压缩过程和被压缩程序的执行过程，有兴趣的读者可以反解 PKLITE 来仔细研究。

压缩过程：

Step 1.　把代码段数据段等压缩；

Step 2.　在压缩的数据后面附上原来程序的全部重定位表；

Step 3.　加上展包模块，把原来的程序入口存放在展包程序中；

Step 4.　设计新的文件头，并把入口指针指向展包模块。

被压缩程序的执行过程：

Step 1.　系统将程序全部装入，并把控制权交给展包模块；

Step 2.　展包模块把自身的代码搬到内存高端，为下一步要展开的代码挪出空间；

Step 3.　将被压缩数据全部展开；

Step 4.　根据重定位表，把展开后的代码重新定位；

Step 5.　释放多余的空间，并把控制权转入原来的程序进入点。

PKLITE 不能完美地处理各种特殊格式的执行文件，也不能完美地处理各种带覆盖的文件。

8.3.2　Windows 执行文件压缩的实现

PACKWIN 压缩 Windows 执行文件的基本原理

和 DOS 执行文件压缩工具一样，在 Windows 执行文件压缩软件中也必然存在展包模块。DOS 下的展包模块最先开始执行，在应用程序已经装入内存、但应用程序实体还没有运行前，它把压缩的代码段和数据段全部展开，展开完毕后，控制权转入原来的程序进入点，展包模块不再起作用。但这种特性的展包模块不能照搬到 Windows 上，这是因为 Windows 有特殊的分段执行机制。我们下面来仔细讨论一下这个问题。

NE 结构文件是分段的，Windows 的分段执行机制正是缘于这种结构。Windows 执行文件中的段有不同的属性，有的段需预装入(Preload)，有的段是在需要时装入(Load on Call)；有的段是固定的(Fixed)，有的段是可移动的(Movable)，有的段是可抛弃的(Discardable)。可抛弃的段在内存不够时，可能会被 Windows 暂时释放，需要时再重新装入。为了提高 Windows 系统的性能，一个 Windows 可执行文件中的可抛弃段往往占很大比例。

Windows 执行程序中都含有启动过程，程序就从这里开始执行，这一点在本书第 4 章已经讲明。如果象 DOS 执行文件压缩工具一样，把展包模块放在启动过程中，则只能展开预装入内存中的段。Windows 程序中的启动过程只被执行一次，这样，当启动过程完毕后，

展包模块再也无法获得控制权，没有预先装入的段再也没有被展开的机会；同样，当某个被抛弃的段再次被装入时，展包模块也是无能为力。当然，“把展包模块放在启动过程中”这种方法还是可以勉强实现的，条件是：把 Windows 执行文件中所有的段强行设置为预装入并且不能抛弃的。这自然会影响 Windows 的系统性能，破坏执行文件本身的特点，这是不现实的。

（对于其它格式的文件，比如，Win32 的 PE 格式文件，这一种展包模式是可行的。我们可以用这种方法开发一个 Windows NT 应用加密软件。）

如何解决这个难题呢？到底该把展包模块放在哪里？

我们的答案是把展包模块放在装载过程中。回忆第 6 章中的有关知识，我们知道装载函数负责将应用程序中的各个段装入内存，然后从一个特定的入口点开始执行。装载函数在应用程序运行期间一直驻留在 Windows 中，不会被抛弃，它随时处理当前要被装入的段。装载函数的这种功能正好满足压缩工具的要求：编写一个被压缩文件的自装载过程，其中的装载函数中含有展包模块，如果被装入段是压缩过的，它就先进行展包处理，这样就绕过了前面提到的所有难题，也是实现展包功能的最佳方案。

在第 6 章中已经指出，要实现自装载过程，程序员必须自己编写三个装载函数，即 BootApp()，LoadAppSeg()和 ExitApp()。

当程序开始运行时，系统先装入程序的自装载模块，然后调用函数 BootApp()。这个函数可以说是装载模块的初始化过程。BootApp()调用 LoadAppSeg()，装入当前需要装载的段。自装载过程结束时，系统调用函数 ExitApp()，复位可能由动态连接库访问过的硬件。

与装入每个段直接相关的函数是 LoadAppSeg()。PACKWIN 是根据 Windows 执行文件中原有的段来分段压缩的，这样，只要把展包代码插入到该函数的实现过程后面即可。在后面的示范源程序中可以看到，展包过程相当简单，这就是用户根本觉察不到 PACKWIN 对程序执行效率影响的原因。

上面说的是 PACKWIN 的设计思想。实现 PACKWIN，有以下几个技术要点：

1. 如何改写自装载模块
2. 如何插入自装载段
3. 如何压缩每一个具体的代码段和数据段以及他们的重定位数据
4. 如何压缩资源

还有一点应该注意，就是如何压缩 Windows 执行文件的 DOS 首部，即 STUB。下面分别讲述这几个问题的要领。

压缩 STUB

每个 Windows 执行文件头部都有一个 STUB，WINSTUB 是一个标准的 DOS 程序。一般的 STUB 只是提示该程序需要在 Windows 环境下运行，有的程序的 WINSTUB 实质上是该程序的 DOS 版本。针对不同的情况，需要做不同的处理。

1. 如果用户使用了不压缩 WINSTUB 的选择项，就不压缩 STUB。

2. 如果 STUB 是标准的 STUB，它的代码尺寸小，使用一般压缩方法的压缩效果并不明显。标准的 STUB 功能明确，容易用汇编语言实现。在 PACKWIN 中，我们用第一章讲述过的只有 128 个字节的 MINISTUB 来替换标准的 STUB，实现过程在第 4 章中。

3. 如果不是标准的 STUB,可以先压缩一下试一试。如果压缩比不大,就使用原来的 STUB;如果能压缩到一定的比例,就使用压缩过的版本。

压缩 STUB 时,必须注意压缩后的 MZ Header 必须大于 40H 个字节,第一个重定位项开始偏移最好要大于 40H。其它的具体过程与 DOS 执行文件的压缩基本一致。

改写自装载模块

第 1 章中我们已经指出,自装载模块的标志是英文字符"A0",而不是 Microsoft 开发手册中所说的 0xA0。这真是"失之毫厘,谬以千里"。另外,Microsoft 的手册中只粗略给出了编写自装载模块的过程,但一般的程序员对照 Windows 手册中对装载过程的讲述,很难编写出完整的自装载模块,因为 Microsoft 有太多的保留。在第 6 章中,我们通过反解 Windows 的装载核心和有关的程序,编写了一个完善的自装载示范。欲深入研究 Windows 压缩、加密技术的读者,必须对这个示范程序的方方面面融会贯通。

在 PACKWIN 中,我们采用的办法是在自装载模块中加入展包代码。在第 6 章 APPLOAD.ASM 基础上增加 UNLZ 展包程序和 UNZIP 展包过程,增加部分如下。完整的 APPLOAD.ASM 在本书所附的磁盘中给出。

请注意:在展包过程 UNZIP 中,它首先判断一个段的压缩标记 ZipMark 是否为"*",如果为"*",则执行真正的展包代码,否则,跳过展包代码。

```
;-------------------------------------------
; UNLZ展包程序的入口参数的说明
;
; Entry:
;   DS:SI — Packed data buffer        压缩数据的缓冲区
;   ES:DI — Unpacked data buffer      展开数据的缓冲区
; Return:
;   CX    — Length of unpacked data   展开后数据的长度
;
; Change registers:                   破坏的寄存器
;   AX, BX, CX, DX, SI, DI
; -------------------------------------------
;
LoadBitFlag     Macro
                mov     dx,10h      ; Bit counter
                lodsw               ;Get flag word from buffer
                mov     bp,ax
                endM

CmpBitFlag      Macro
                Local   @@0
                shr     bp,1        ;Get a bit flag
                dec     dx          ; Cmp end of word flag
                jnz     @@0
                lodsw               ; LoadBitFlag
                mov     bp,ax
                mov     dl,10h
        @@0:
                endM
```

```
; ---------------------------------------------------
UNLZ PROC
            cld
            push    bp
            LoadBitFlag
      @@3:
            CmpBitFlag
            jnc     @@5         ; Cmp a bit flag
            movsb               ; 1     Copy a byte to target
            jmp     short @@3

      @@5:
            xor     cx,cx       ; 0
            CmpBitFlag
            jc      @@9
                                ; 00xx Counter save in BitFlag (2 bits)
                                ;       Counter  < = 5
                                ;       Lookback < 100h (1 byte)
            CmpBitFlag
            rcl      cx,1
            CmpBitFlag
            rcl     cx,1
            inc     cx          ; add 2 to counter  < = 5
            inc     cx
            lodsb               ; Lookback distance < 100h
            mov     bh,0FFh
            mov     bl,al
            jmp     @@10

      @@9:                      ; 01 Lookback distance < 2000h (2 bytes)
            lodsw
            mov     bx,ax
            mov     cl,3
            shr     bh,cl
            or      bh,0E0h     ; 1 1 1 b7 b6 b5 b4 b3 b3 (b2 b1 b0)
            and     ah,7
            jz      @@11
                                ; counter = (b2 b1 b0)  < = 9
            mov     cl,ah
            inc     cx
            inc     cx

      @@10:
            mov     al,es:[bx+di] ; Copy repeat block
            stosb
            loop    @@10

            jmp     short @@3

      @@11:                     ; counter = 0
            lodsb
```

```
                or      al,al       ; exit ?
                jz      @@13

                cmp     al,1        ; add segment register
                je      @@12
                                    ; 01  Counter < = 100h (1 byte)
                mov     cl,al
                inc     cx
                jmp     short @@10
        @@12:
                jmp     @@3

        @@13:
                pop     bp
                mov     cx, di
                ret
Unlz            ENDP


Unzip:
; ---------------------------------------
; 展包过程
; ds:0  -- 缓冲区指针
; cx    -- 缓冲区长度
; ---------------------------------------
        cmp     cs:ZipMark, '*'         ; 比较是否是 PACKWIN 调用
        jz      @@exit
        cmp     word ptr ds:[0], Mark   ; 这块数据是否被压缩
        jnz     @@exit
        push    ax
        push    bx
        push    dx
        push    si
        push    di
        push    ds
        push    es

        push    word ptr ds:[2]         ; 压缩过的数据块[02H]处存放的是压缩长度
        push    cx                      ; 申请用来存放压缩过的段的空间
        PUSH    2
        PUSH    0
        PUSH    CX
        CALL    GlobalAlloc
        pop     cx
        mov     es, ax
        mov     si, 4
        xor     di, di
        sub     cx, si
        push    cx
        rep     movsb                   ;把段移过去
        pop     cx
        push    ds
```

```
        push    es
        pop     ds
        pop     es
        xor     si, si
        xor     di, di
        call    UnLz                    ;调用 UNLZ 过程解码
        push    cx                      ;释放多余的空间
        push    ds
        CALL    GLOBALFREE
        pop     cx
        pop     ax
        cmp     ax, cx

        pop     es
        pop     ds
        pop     di
        pop     si
        pop     dx
        pop     bx
        pop     ax
        jnz     @@error_exit            ;比较展开后的长度与原来的长度是否一致
@@exit:
        clc
        ret
@@error_exit:
        stc
        ret
```

改写完自装载过程，我们按照第 6 章中所给出的编译连接方法生成新的 test.lib，该 Library 中新的装载过程"--MSLANGLOAD"能展开被压缩的代码数据段。

```
masm appload;
masm reloc;
lib test.lib +appload +reloc;
nmake
```

接下来通过命令 nmake 执行 makefile，其中的执行文件还是以 HELLOWIN 为例。makefile 和 HELLOWIN.DEF 与第 6 章中给出的完全相同。请参看第 6 章了解 makefile 是如何将新的装载过程加入到 HELLOWIN.EXE 中去的。

执行完 nmake 之后，虽然 HELLOWIN.EXE 有了新的装载过程，但 HELLOWIN.EXE 的代码数据段并没有被压缩，展包过程 UNZIP 在执行时跳过了真正的展包代码。这一点请读者仔细思考。

```
getseg hellowin.exe 1 selfload.bin
binobj selfload.bin selfload.obj apploader
```

自装载程序 HELLOWIN 的 Seg #1 是自装载模块的全部，我们用第 1 章中介绍的工具 GetSeg 把它取出来，放在 selfload.bin 中，然后用 Borland Pascal 提供的 BINOBJ 工具将它转换为 OBJ 文件 selfload.obj，该 obj 的调用名为 apploader。

这样，我们就从一个未被压缩的 HELLOWIN 取出了一个具有展开功能的自装载模块

的 obj。这是一种“金蝉脱壳”的技巧。下面是完整的编译过程。

```
masm appload;
lib test.lib + appload + reloc;
nmake
getseg hellowin.exe 1 selfload.bin
binobj selfload.bin selfload.obj apploader
```

如何使用通过上述手段得到的自装载模块呢？PACKWIN 为了方便调试，如果当前目录下有 selfload.bin 文件，则直接装入这个模块。否则使用已经预编译进 PACKWIN 的 apploader，这样，即使 selfload.bin 有 BUG，只要直接修改 selfload.bin 即可，改动 PACKWIN。这个处理过程请见 TPackWin.LoadBootSeg。

在阅读后面的编程实现时，有一个问题应当引起注意：

selfload.bin 这个自装载模块中需要调用 Windows 系统的两个模块 KERNEL 和 USER，Windows 执行程序一般会调用这两个模块中的函数并以“模块名.序号”的形式给出具体函数，模块名在模块引用表中定义。现在的问题是，比如说，在执行程序 A 中，模块 KERNEL 和 USER 的模块引用表序号为 3 和 4，在执行程序 B 中，模块引用表序号可能是 5 和 8，甚至还有可能这两个模块没有被引用（实际的序号安排由编译器决定）。如果对程序实行压缩，这就极可能与 selfload.bin 中这两个模块的序号不一致，导致同一个序号所指的模块不同。要处理好这个相当麻烦的问题，参见本章“PACKWIN 编程实现”中的 TPackWin.SetNewModuleName，该过程更新了模块名，使两者统一；另外，NewModuleNo 把新的模块引用号更新到重定位表中。这个细节问题一定不能放过。

插入自装载段

实现了自装载模块 selfload.bin，接下来的难题是如何将这个自装载段插入到执行程序中。这个工作十分复杂，虽然我们在第四章中讲解了如何插入代码段，但具体来说还是有些不同。

插入一个自装载的段，相当于在插入第一个段的基础上增加一些操作。具体来说，需要做如下操作：

1. 改变 NE 首部段表的项数，把该段的段表项加在最前面，顺序往后挪动其他的段表；
2. 将自动数据段的段号加 1；
3. 将入口段的段号加 1；
4. 将堆栈段的段号加 1；
5. 新增加的段需要增加引用 KERNEL 和 USER 模块，根据需要修改模块引用表的项数；
6. 计算快速装载区域；
7. 修改资源表的每项的值；
8. 调整入口表、段表、资源表、模块引用表、输入名表、非驻留名表等各种表的入口偏移值；
9. 将信息块中的“特征标记”设置为自装载的程序；
10. 根据给定的位置插入新的段。

这些操作实现参见 TPackWin.Packing。

压缩代码段和数据段以及重定位部分

插入自装载段后的执行文件如同前面编译生成的 HELLOWIN，虽然它的自装载模块可以展开被压缩过的代码数据段，但它的代码数据段并没有被压缩。现在来看如何压缩代码数据段及它们的重定位表。

Windows 的段有 64K 的大小限制，最大不能超出 64K。如果某一段尺寸大于 F000H，PACKWIN 不压缩这个段。这是因为压缩可能出现颠簸，导致压缩后大小反而增加，经验数据表明，压缩颠簸很少超过 10%。PACKWIN 把段大小限制在 F000H，即使出现严重的压缩颠簸，段大小也只会有 FF00H，不会因超过 64K 而造成严重错误；另一方面，Windows 执行文件中的段大小很少超过 F000H，PACKWIN 不压缩这样的段也不至于严重影响 PACKWIN 的压缩效率。

压缩后的段第一个字存放的是标志'PW'，第二个字存放的是压缩前的长度，该段在段表中"需要申请的空间大小"就是压缩前的长度，不要修改，这样系统会自动分配压缩以前需要的空间。随后把压缩后段的长度设置到该段段表中"段的长度"项中。

重定位表是单独压缩的。压缩后的重定位表的第一个字存放的是重定位的项数，第二个字存放的是标志'PW'，第三个字存放的是重定位表压缩后的长度。由于该段重定位表压缩前的长度可由重定位项数计算出来，因此第三个字中放压缩后的长度。

参见 ProcessNode 中的 WriteCodeDataSeg 和 WriteRelocList 过程。

压缩资源

执行文件中的资源类型很多，实际中空间比例最大的资源是 BITMAP。对于 BITMAP，如果它未被压缩过，PACKWIN 按照 Microsoft 手册中介绍的压缩方法把它变成压缩格式的 BITMAP；参见 ProcessNode 中的 WriteResBlock。对于其他类型的资源，由于他们一般不长，PACKWIN 只是选用合适的对齐因子去掉多余的填充数据，因为压缩一小段数据看不到明显的压缩效率。

在压缩一个资源前需要得到它的具体资源的长度，FILEDEF.TPU 的 TNEHeader 对象中的 GetResLength 用于计算某个指定资源的大小。具体的实现可以参见第 3 章。

```
{ 这里 S 参数是调试用的。传资源的类型、位置、长度就可以获得一个具体资源的长度 }
function GetResLength(S: String; ResTypeID: Word;
  ResOfs, ResLen: LongInt): LongInt;
```

资源压缩还可以截获 Windows 函数的 LoadResource，在系统装入资源时再展开。

8.3.3 PACKWIN 的编程实现

根据前面对 Windows 执行文件压缩方法的阐述，这里给出 PACKWIN 的源程序。请读者应前后参照，深入体会 PACKWIN 的设计思想。

在下面的 PACKWIN.PAS 中，经常使用数据结构 InfoTree，有关 InfoTree 的操作有如下四个，在此特作说明。FILEDEF 中有具体的定义和实现过程。

```
procedure BuildInfoTree;
procedure DisposeInfoTree;
procedure SetInfo(myType: byte; myIndex: word;
        myOfs, myLen: LongInt; myOwner: String);
procedure BrowseInfoTree(NRoot: PBlkInfo; DoNode: ProcDo);
```

NE 格式文件结构非常复杂，为了方便计算和处理，我们可以把它二叉树对待。BuildInfoTree 用于构造二叉树结构 InfoTree。NE 结构中的 STUB、各种表、段以及重定位部分和资源等都被看成是这个二叉树的节点。用二叉树方式可以快速搜索出按偏移排序的表。用 PFI 显示来显示一般的 NE 结构文件：

```
C:\ >pfi hellowin.exe

Power FileInfo  Version 1.0   Copyright (c) 1993 Yellow Rose Workgroup

File name: hellowin.exe
Module name: HELLOWIN
Description: Hello Windows Program
Page size: 16     Resource page size: 1

    Position        Length     Owner
    --------        ------     ----------------
    0000h-03FFh     0400h    WINSTUB
    0400h-0494h     0095h    New Executable file header
    0400h-043Fh     0040h      NE header record
    0440h-0457h     0018h      Segment table
    0458h-0458h     0000h      Resource table
    0458h-0463h     000Ch      Resident name table
    0464h-0469h     0006h      Module reference table
    046Ah-047Ah     0011h      Imported names table
    047Bh-047Bh     0001h      Entry table
    047Ch-0494h     0019h      Non-resident names table
    04A0h-06ABh     020Ch    seg1 code movable preload
    06ACh-075Dh     00B2h    seg1 Relocation: 22 entries
    0760h-1249h     0AEAh    seg2 code movable preload
    124Ah-1273h     002Ah    seg2 Relocation: 5 entries
    1280h-12E1h     0062h    seg3 data movable preload

  File size is 4834  [ Stub: 21.1 %   Code: 68.6 %   Data: 2.0 % ]
```

PFI 所显示的每一项都是 InfoTree 的一个节点。

使用二叉树后，在这个表中插入一个段可以理解成插入一个节点，SetInfo 用来插入这样的节点；对一个文件的压缩就转换成对每一个节点的处理。正是基于这种理解，实现 PACKWIN 的过程看起来极其简洁：只需要写针对具体节点的处理子程序，然后调用 BrowseInfoTree。比如，编写 PFI.PAS 的关键是获取每一个具体节点的信息并显示它们，PACKWIN 中也有类似的过程 ProcessNode。

运行 PACKWIN 带/m 参数有助于加深我们对 InfoTree 的理解。

```
C:\ >packwin /m hellowin.exe
```

```
PACKWIN   DOS and Windows Executable File Compressor  Version 1.0
Copyright (c) 1993, 1994  Yellow Rose Workgroup.   All Rights Reserved.

File: HELLOWIN.OLD already exists.  Overwrite (y/n)? Y

Compressing: HELLOWIN.EXE ...

New page size is 16                                    <-- 计算合适的对齐因子
New resource page size is 1

<偏移>    <长度>    <节点说明>
  0000    0080   WINSTUB
  0080    0040     NE header record
  00C0    0020     Segment table
  00E0    0000     Resource table
  00E0    000C     Resident name table
  00EC    0006     Module reference table
  00F2    0011     Imported names table
  0103    0001     Entry table
  0104    0019     Non-resident names table
  0120    09BB   new seg1                            <-- 插入自装载段
  0AE0    0180   seg1 code movable preload
  0C60    008A   seg1 Relocation: 22 entries
  0CF0    02CD   seg2 code movable preload
  0FBD    002A   seg2 Relocation: 5 entries
  0FF0    004B   seg3 data movable preload

Original: 4834  Compressed: 4160  Ratio: 13.94 %
```

PACKWIN.PAS 中需要频繁使用这种遍历技术，因为 Windows NE 格式中有许多表格，如果每执行一次操作就加入一段遍历代码，这将使得代码变得非常冗长。运用 Borland Pascal 7.0 中提供把函数作为参数的方法，我们把一个一个的表的遍历操作事先实现，需要作遍历操作时，只要把具体的操作函数以参数形式给出即可。比如针对重定位表，只要编写对一个象 ADoRelocItem 这样具体的重定位项操作过程。

下面是一个实现的例子。

```
{说明部分}
DoRelocItem = procedure (PSelf: PNeHeader; P: PRelocItem);
{遍历的具体程序}
procedure BrowseRelocNode(MyDoRelocItem: DoRelocItem);
beign
  for I := 1 to RelocLen do begin
    P := @(PList^[(I-1) * Sizeof(RelocItem)]);
    DoRelocItem(@Self, P);
  end;
end;

{具体节点的处理程序}
procedure ADoRelocItem (PSelf: PNeHeader; P: PRelocItem);
begin
  {...}
```

```
end;

begin
  {...}
  BrowseRelocNode(ADoRelocItem);
  {...}
end.
```

FILEDEF.TPU 中定义了 NE 结构中所有表的遍历过程，用 FILEDEF 开发一个修改 Windows 执行程序的软件应该是相当轻松的。如果不想用 FILEDEF，也可以自己实现。

这里是 PACKWIN.PAS 的源程序。

```
{ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * }
{ *   PACKWIN  Version 1.0                                                        * }
{ *   Copyright (c) 1993 by Yellow Rose Workgroup                                 * }
{ *   Written by Mr. Lei Jun, Mr. Wang Quanguo                                    * }
{ * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * }

{ $ DEFINE DEBUG}   { Display some debug informations }

Uses FileDef, ExtTools, PackUnit, Dos, Strings;

Const
  -ver        = '1.0';                { 版本号                     }
  verA        = -ver + #224;          { 内部测试版  Alpha          }
  verB        = -ver + #225;          { 正式测试版  Beta           }
  ver         = verA;                 { 当前版本                   }
  TradeMark  =                        { 软件商标                    }
    'PACKWIN   DOS and Windows Executable File Compressor   ' +
    'V' + ver + '    Shareware '#10#13 +
    'Copyright (c) 1993, 1994   Yellow Rose Workgroup.    ' +
    'All Rights Reserved.' + #13#10;

  fPack       : Boolean = True;       { 软件是否压缩                }
  fPackDos    : Boolean = True;       { 是否压缩 DOS 的 AP          }
  fPackWin    : Boolean = True;       { 是否压缩 WIN 的 AP          }
  fGroupPack  : Boolean = False;      { 批压缩方式 支持一批文件的压缩 }

procedure DisplayHelp;
begin
  Writeln('Written by Mr. Lei Jun and Mr. Wang Quanguo');
  Writeln;
  Writeln('Usage: PACKWIN <Infile> [<Outfile>] [Options]');
  Writeln('Options:');
  Writeln('  /r   -- Disable Resource compression');
  Writeln('  /b   -- Disable Bitmap compression');
  Writeln('  /w   -- Disable Winstub compression');
  Writeln('  /m   -- Display more messages');
  Writeln('  /win -- Only compress Windows Application');
  Writeln('  /dos -- Only compress MS-DOS Application');
  Writeln;
end;
```

```
{ Command line options }

const
  fPackBmp  : Boolean = True;        { 是否压缩 BITMAP 资源          }
  fPackStub : Boolean = True;        { 是否压缩 WINSTUB              }

procedure Apploader; external; { $L SELFLOAD.OBJ}

{ New executable file header object ---------------------- }

const
  NewSeg = 0;
  BLWFlag = $45534f52;               { BITLOK 的标志                 }
  BLW: boolean = false;              { 是否被 BITLOK 加密的标志      }

type
  NewModule = record                 { 自装载部分的输入模块的记录    }
    BootIdx,
    Idx: word;
    Append: boolean;
    Ofs: word;
    Name: String10;
  end;

  PackedAp = record
    ofsNe: LongInt;                  { 新的首部的偏移                }
    NE: tagNeHeader;                 { 首部                          }
    lenHeaderAdd,                    { 首部增加部分的长度            }
    ofsBootSeg: Word;                { 自装载段的位置                }
    PageSize,                        { 新的页大小                    }
    ResAlign,                        { 资源对齐的大小                }
    ResPageSize: Word;               { 资源的对齐因子                }
    GangLoadStart: Boolean;          { 是否到快装区域                }
    NewOfsGangLoad, NewLenGangLoad: LongInt; { 快装区域              }
    PMySegTab, PMyResource: PBuf;    { 段表资源表                    }
  end;

  SelfLoadingSeg = record            { 自装载段的记录                }
    Data: TBuf;                      { 自装载段的存放地址            }
    PASeg: PSegmentEntry;            { 自装载段的段表指针            }
  end;

  PPackWin = ^TPackWin;
  TPackWin = object (TNeHeader)
    FName: PathStr;                  { 原文件                        }
    FT: File;                        { 目标文件的句柄                }
    T: PackedAP;                     { 存放压缩后文件的信息          }
    PBoot: PBuf;                     { 存放自装载段                  }
    BootSeg: SelfLoadingSeg;         { 自装载段的信息                }
    Kernel, User: NewModule;         { 记录 KERNEL USER 的信息       }
    CurrRes: tagBlkInfo;             { 当前的资源块                  }
    CurrOfs, NextOfs, OfsSegBegin: LongInt;
```

```
                                    { 块遍历的变量                    }

    constructor Init(SName, TName: PathStr);
    destructor Done; virtual;
    { 压缩 STUB,并返回压缩的类型 }
    function PackStub (TName: PathStr): LongInt;

    { 设置新的页大小 }
    procedure SetNewPageSize;
    { 设置新的输入模块名 }
    procedure SetNewModuleName;
    { 装入自装载段 }
    procedure LoadBootSeg;
    { 释放自装载段 }
    procedure FreeBootSeg;
    { 初始化目标数据 }
    procedure InitTargetData;
    { 压缩主过程 }
    procedure Packing;
    procedure Run; virtual;
  end;

{ 遍历信息块,求出偏移最小的段的位置 }
procedure GetOfsSegBegin (PSelf: PNeHeader; P: Pointer); far;
begin
  with PPackWin(PSelf)^, PBlkInfo(P)^ do
    if (flags in [cSeg, dSeg, Reloc, Res, theEnd]) and
       (ofs > 0) and                    { valid segment entry }
       (ofs < ofsSegBegin) then
      ofsSegBegin := ofs
end; { GetOfsSegBegin }

constructor TPackWin.Init(SName, TName: PathStr);
var
  PackedLength: LongInt;
begin  { Init }
  inherited Init (SName);
  FName := SName;
  if rtError <> 0 then fail;
  if (neHeader.TargetOS <> 2) then begin
    rtError := $EE10;
    fail;
  end;
  if IsSelfLoadingAP(neHeader) then begin
    rtError := $EE11;
    fail;
  end;
  if (neHeader.-CS = 0) then begin
    rtError := $EE12;
    fail;
  end;
```

```
  if (ModName = 'win386') or (ModName = 'w386dll') then begin
    rtError := $EE13;
    fail;
  end;

  BuildInfoTree;                          { 建立一个信息树                    }
  ofsSegBegin := $7fffffff;
  BrowseInfoTree(Root, GetOfsSegBegin);   { 调用求最小段的偏移                }
  InitTargetData;
  PackedLength := PackStub(TName);        { 先压缩 STUB                       }
  T.ofsNe := AlignSize(PackedLength, $10); { 计算压缩过的 STUB 长度           }
  Assign(FT, TName);
  Reset(FT, 1);
end;

destructor TPackWin.Done;
begin
  Close(FT);
  FreeBootSeg;
  with T do begin
    DisposePBuf(PMyResource);
    DisposePBuf(PMySegTab);
  end;
  DisposeInfoTree;
  inherited Done;
end;

procedure TPackWin.SetNewPageSize;
var
  I, J: word;
begin
  if fPack then
    I := wMin(GetPageSize, $10)    { 最小的对齐因子为 16                       }
  else I := GetPageSize;
  while (I < DWord(FSize).HiWord) do
    I := I*2;                     { 对齐因子必须能够表示整个文件的大小        }
  J := Log2(I);
  if fPack then
    T.ResPageSize := wMin(I, GetResPageSize)
  else T.ResPageSize := GetPageSize;
  T.ResAlign := Log2(T.ResPageSize);
  PWord(T.PMyResource^.P)^ := T.ResAlign;
  T.PageSize := I;
  T.NE.SegmentAlign := J;

  if fPrompt then begin
    Writeln;
    Writeln('New page size is ', T.PageSize);
    Writeln('New resource page size is ', T.ResPageSize);
    Writeln;
  end;
end;
```

```
{ 修改自装载块的 Module 号 }
procedure NewModuleNo (PSelf: PNeHeader;
  P: PRelocItem; Body: PBytes); far;
begin
  With PPackWin(PSelf)^ do
    if (P^.RelocType and 3 = 1) then begin
      if (P^.Idx = User.BootIdx) then
        P^.Idx := User.Idx
      else if (P^.Idx = Kernel.BootIdx) then
        P^.Idx := Kernel.Idx
      else Write('S');
    end; { if }
end;

procedure TPackWin.LoadBootSeg;
const
  BootSegFName = 'SELFLOAD.BIN';
var
  TmpF: File;
  PA: PBuf;
begin
  { 假如存在自装载模块,就装入 }
  if FileExists(BootSegFName) then begin
    assign(TmpF, BootSegFName);
    reset(TmpF, 1);
    NewPBufEx(PBoot, FileSize(TmpF), nil);
    BlockRead(TmpF, PBoot^.P^, PBoot^.Len);
    close(TmpF);
  end
  else begin
  { 使用代码中的自装载模块 }
    NewPBufEx(PBoot,
      PWord(@PBytes(@AppLoader)^[ $ 8])^ + $ 10,
      PBytes(@AppLoader) );
  end;

  BootSeg.Data.Len := PWord(@PBoot^.P^[ $ 8])^;
  if not fPack then Inc(BootSeg.Data.Len, $ 20);
  { Only used by add self-loading Module }

  BootSeg.Data.P    := @(PBoot^.P^[ $ 10]);
  BootSeg.PASeg     := @(PBoot^.P^);
  Kernel.BootIdx    := PWord(@PBoot^.P^[ $ C])^;
  User.BootIdx      := PWord(@PBoot^.P^[ $ E])^;

  if fPack then BootSeg.Data.P^[ $ 31 ] := Ord('o');
  New(PA);
  PA^.P := @(BootSeg.Data.P^[BootSeg.PASeg^.lenSeg + 2]);
  PA^.Len := (BootSeg.Data.P^[BootSeg.PASeg^.lenSeg]) * 8;
  BrowseAsegRelocList(PA, 0, NewModuleNo);
  Dispose(PA);
end;
```

```
procedure TPackWin.FreeBootSeg;
begin
  DisposePBuf(PBoot);
end;

function TPackWin.PackStub (TName: PathStr): LongInt;
var
  D: DirStr;
  N: NameStr;
  E: ExtStr;
  Result: Word;
  TmpStubFile: PathStr;
begin
  if fPackStub and fPack then begin
    FSplit(TName, D, N, E);
    TmpStubFile := D + 'PSTUB.TMP';
    GetWinStub(TmpStubFile);
    Result := (PackWinStub (TmpStubFile, TName, false));
    if (Result > 1) then
      RenameFile (TmpStubFile, TName)
    else DelFile (TmpStubFile);
  end
  else GetWinStub(TName);
  PackStub := GetFileSize(TName);
end;

{ 设置 KERNEL 和 USER 的信息 }
procedure TPackWin.SetNewModuleName;
begin
  With T do begin
    With Kernel do begin
      Idx := GetModuleIdx('KERNEL');
      Append := (Idx = 0);
      if Append then begin
        Name := 'KERNEL';
        Ofs  := NE.lenImportedNameTab;
        Inc(NE.lenModuleReferenceTab, 2);
        Inc(NE.numModuleReferenceTab);
        Inc(NE.lenImportedNameTab, Length(Name) + 1);
        Inc(lenHeaderAdd, Length(Name) + 1 + 2);
        Idx := NE.numModuleReferenceTab;
      end;
    end;  { with Kernel }
    With User do begin
      Idx := GetModuleIdx('USER');
      Append := (Idx = 0);
      if Append then begin
        Name := 'USER';
        Ofs := NE.lenImportedNameTab;
        Inc(NE.lenModuleReferenceTab, 2);
        Inc(NE.numModuleReferenceTab);
        Inc(NE.lenImportedNameTab, Length(Name) + 1);
```

```
        Inc(lenHeaderAdd, Length(Name) + 1 + 2);
        Idx := NE.numModuleReferenceTab;
      end
    end;  { with User }
  end;  { with T }
end;

procedure TPackWin.InitTargetData;
begin
  with T do begin
    NE := neHeader;
    lenHeaderAdd := 8;
    Inc(NE.lenSegmentTab, 8);
    Inc(NE.numSegmentTab);
    SetNewModuleName;
    RecalcTablesOfs(NE);
    NewPBufEx(PMySegTab, neHeader.lenSegmentTab, PSeg);
    NewPBufEx(PMyResource, neHeader.lenResourceTab, PResource);
  end; { with }
  SetNewPageSize;
  LoadBootSeg;
end;

{ 通过遍历的功能,把每个入口表的段号加一 }
procedure IncEntrySegNo (PSelf: PNeHeader; P: Pointer); far;
begin
  with PPackWin(PSelf)^, PAEntry(P)^ do begin
    if (Idx > 0) and (Bundle^.flags = $ff) then begin  { Non bundle }
        Inc(MEntry^.SegNum);
    end                                    { Movable segment entry }
    else if (Idx = 0) and
      (Bundle^.Flags > 0) and (Bundle^.Flags < $ff) then begin
      Inc(Bundle^.Flags)
    end;                                   { Fixed segment Bundle }
  end;
end;  { IncEntrySegNo }

{ 通过遍历的功能,把每个段的重定位项段号加一 }
procedure IncRelocItemSegNo (PSelf: PNeHeader;
  P: PRelocItem; Body: PBytes); far;
begin
  With PSelf^ do begin
    if ((P^.RelocType and 3 = 0) and (P^.Idx <> $ff)) then
      Inc(P^.Idx);
  end;  { with }
end;  { IncRelocItemSegNo }

{ 通过遍历的功能,替换资源表项的偏移 }
procedure NewResOffset (PSelf: PNeHeader; P: Pointer); far;
begin
  with PPackWin(PSelf)^, PEachRes(P)^ do
    if (Idx = CurrRes.Idx) and (T.ResPageSize <> 0) then begin
```

```
      NameInfo^.rnOffset := CurrOfs div T.ResPageSize;
      NameInfo^.rnLength :=
        (CurrRes.Len + T.ResPageSize - 1) div T.ResPageSize;
      BrowseStop := True;
    end;
end;  { NewResOffset }

{ 遍历每个块 }
procedure ProcessNode (PSelf: PNeHeader; P: Pointer); far;

  procedure WriteDataBlock;
  begin
    with PPackWin(PSelf)^, PBlkInfo(P)^ do begin
      if (Len <> 0) then begin
        Seek(F, ofs);
        CopyFileBlock(F, FT, Len);
      end; { if }
    end; { with }
  end; { WriteDataBlock }

  procedure WriteSysMessage;
  var
    PA: PBuf;
  begin  { WriteSysMessage }
    with PPackWin(PSelf)^, PBlkInfo(P)^ do begin
      case Idx of
        sysNe:     { 处理首部的头 }
                   BlockWrite(FT, T.NE, $40);
        sysSeg:    { 段表 }
                   begin
                     BlockWrite(FT, BootSeg.PASeg^, 8);
                     BlockWrite(FT, T.PMySegTab^.P^, T.PMySegTab^.Len);
                     Inc(Len, 8);
                   end;
        sysRes:    { 资源表 }
                   BlockWrite(FT,
                     T.PMyResource^.P^, T.PMyResource^.Len);
        sysModRef: { 模块引用表 }
                   begin
                     WriteDataBlock;
                     if Kernel.Append then begin
                       BlockWrite(FT, Kernel.Ofs, 2);
                       Inc(Len, 2);
                     end;
                     if User.Append then begin
                       BlockWrite(FT, User.Ofs, 2);
                       Inc(Len, 2);
                     end;
                   end;
        sysImpName:{ 输入名表 }
                   begin
                     WriteDataBlock;
```

```
                if Kernel.Append then begin
                  BlockWrite(FT, Kernel.Name[0],
                    Length(Kernel.Name) + 1);
                  Inc(Len, Length(Kernel.Name) + 1);
                end;
                if User.Append then begin
                  BlockWrite(FT, User.Name[0], Length(User.Name) + 1);
                  Inc(Len, Length(User.Name) + 1);
                end;
              end;
      sysEntry: { 入口表 }
              begin
                NewPBufEx(PA, Len, PEntry);
                BrowseEntryTab(PA, IncEntrySegNo);  { Browse }
                BlockWrite(FT, PA^.P^, PA^.Len);
                DisposePBuf(PA);
              end;
      else WriteDataBlock; { NonresidentNameTable  ResidentNameTab }
    end;  { case }
  end;  { with }
end;  { WriteSysMessage }

{ 处理自装载段 }
procedure WriteBootSeg;
begin
  with PPackWin(PSelf)^, PBlkInfo(P)^ do begin
    T.ofsBootSeg := newOfs div T.PageSize;
    BlockWrite(FT, BootSeg.Data.P^, BootSeg.Data.Len);
  end;  { with }
end; { WriteNewSeg }

{ 压缩一个块 }
function Encode (var A: TBuf): word;
var
  PB: PBuf;
begin
  PB := Pack (@A);
  FillChar(A.P^, A.Len, '.');
  Move(PB^.P^, A.P^, PB^.Len);
  Encode := PB^.Len;
  DisposePBuf(PB);
end;  { Encode }

function Encode0 (var A: TBuf): word;
begin
  Encode0 := A.Len;
end;

{ 写代码数据段 }
procedure WriteCodeDataSeg;
var
  PA: PBuf;
```

```
  AFlag, WriteLen: LongInt;
begin   { WriteCodeDataSeg }
  with PPackWin(PSelf)^, PBlkInfo(P)^ do begin
    if (Len = 0) or (Len > $f000) then begin    { No packed }
      if (Len = 0) then WriteLen := $10000;     { 64K }
      Seek(F, ofs);
      CopyFileBlock(F, FT, WriteLen);
      NewPBuf(PA);
    end
    else begin
      PA := ReadASeg(Idx);
      if (Idx = neHeader._cs) then begin        { July 20, 1994 }
        Move(PA^.P^[neHeader._ip + $c], AFlag,  4);
        if (AFlag = blwFlag) then BLW := true;
      end;
      if fPack then
        WriteLen := Encode(PA^)
      else WriteLen := Encode0(PA^);
      BlockWrite(FT, PA^.P^, WriteLen);
    end;
    with GetSegmentEntry(T.PMySegTab^.P, Idx)^ do begin
      lenSeg := WriteLen;
      Ofs := NewOfs div T.PageSize;
    end;
    Len := WriteLen;
    DisposePBuf(PA);
  end;   { with }
end;   { WriteCodeDataSeg }

{ 写重定位表 }
procedure WriteRelocList;
var
  I: Word;
  PA: PBuf;
begin
  with PPackWin(PSelf)^, PBlkInfo(P)^ do begin
    PA := ReadARelocList(Idx);
    BrowseAsegRelocList(PA, Idx, IncRelocItemSegNo);   { Browse }
    I := PA^.Len div SizeOf(tagRelocItem);
    BlockWrite(FT, I, 2);
    if fPack and not ((Idx = neHeader._cs) and Blw) then
      I := Encode(PA^)
    else I := Encode0(PA^);
    if (PWord(PA^.P)^ = PackMark) then PWords(PA^.P)^[1] := I;
    BlockWrite(FT, PA^.P^, I);
    DisposePBuf(PA);
    Len := I + 2;
  end;   { with }
end;   { WriteRelocList }

{ 写资源表 }
procedure WriteResBlock;
```

```
begin
  with PPackWin(PSelf)^, PBlkInfo(P)^ do begin
    CurrRes := PBlkInfo(P)^;
    CurrOfs := NewOfs;
    if fPackBmp and (StrPos(Owner, 'Bitmap') <> nil) then begin
      CurrRes.Len := PackBitmap(F, Ofs, FT, CurrOfs, Len);
      Len := CurrRes.Len;
    end
    else WriteDataBlock;
    BrowseResourceTab (T.PMyResource, NewResOffset);     { Browse }
  end;
end;  { WriteResBlock }

{ 计算快装区域 }
procedure CalcGangLoad;
begin
  with PPackWin(PSelf)^, PBlkInfo(P)^ do begin
    if GangLoad then begin
      if (not T.GangLoadStart) and
        (OtherFlag and GangLoadFlag > 0) then begin
        T.GangLoadStart := True;
        T.NewOfsGangLoad := NewOfs;
      end;
      if T.GangLoadStart and
        ((OtherFlag and GangLoadFlag = 0) or (Flags = TheEnd))
      then begin
        T.GangLoadStart := False;
        T.NewLenGangLoad := AlignSize
          (NewOfs-T.NewOfsGangLoad, T.PageSize);
      end; { if }
    end; { if }
  end; { with }
end; { CalcGangLoad }

{ 显示提示信息 }
procedure ShowMessage;
begin
  with PPackWin(PSelf)^, PBlkInfo(P)^ do begin
    if fPrompt then begin
      if (flags = res) then
        Writeln( lwhStr(newofs):6, '  ',
                 lwhStr(len):6, '  ', Idx, Owner)
      else Writeln( lwhStr(newofs):6, '  ',
                    lwhStr(len):6, '  ', Owner);
    end
    else begin
      Write(#8#8#8#8#8#8#8' %');
      WriteRatio(FSize, newOfs);
    end;
  end;
end;  { ShowMessage }
```

```
begin  { Process Node }

  with PPackWin(PSelf)^, PBlkInfo(P)^ do begin

    if fPack then begin
      CurrOfs := NextOfs;
      { $ IFDEF DEBUG}
      if CurrOfs <> FilePos(FT) then
        Write('[', lhStr(FilePos(FT)),'] ');
      { $ ENDIF}
    end
    else CurrOfs := NewOfs;

    if fPack then begin
      if ( flags in [cseg, dseg, TheEnd] ) then     { 对齐代码和数据 }
        CurrOfs := AlignSize(NextOfs, T.PageSize)
      else if ( flags = Res ) then                  { 对齐资源段     }
        CurrOfs := AlignSize(NextOfs, T.ResPageSize);
    end;

    NewOfs := CurrOfs;
    if (flags in [cseg, dseg, res, TheEnd]) or
       ((flags = sys) and (Idx = sysAll)) then
      WriteSeek (FT, NewOfs);

    if (flags <> TheEnd) then
    begin
      if ((flags = sys) and (idx = sysAll)) then
        exit;
      if (flags = stub) then begin
        Len := T.OfsNe;
        Seek(FT, T.ofsNe);
      end;

      if (flags = cseg) and (Idx = newseg) then
        WriteBootSeg
      else case flags of
        sys:   WriteSysMessage;
        cseg,
        dseg:  WriteCodeDataSeg;
        Reloc: WriteRelocList;
        Res:   WriteResBlock;
      end;
      ShowMessage;
      NextOfs := NewOfs + Len;
    end;
    CalcGangLoad;
  end;  { with }
end;  { ProcessNode }

{ 计算段表项的位置 }
```

```
procedure IncSegOfs(PSelf: PNeHeader; P: Pointer); far;
begin
  with PPackWin(PSelf)^, PBlkInfo(P)^ do
    if ( ofs >= ofsSegBegin) then
      NewOfs := Ofs + AlignSize(BootSeg.Data.Len, T.PageSize);
end;

procedure TPackWin.Packing;

  { 在信息表中插入自装载块 }
  procedure InsertNewSeg;
  begin
    BrowseInfoTree(Root, IncSegOfs);
    SetInfo(cseg, newseg,
      ofsSegBegin, BootSeg.Data.Len, 'new seg1');
    if not fPack then Inc(PCurrInfo^.NewOfs, $20);
    { Only used by Add Self-Loading Module }
  end;

var
  MZ: tagMzHeader;

begin  { Packing }
  if Root <> nil then begin
    with T, T.NE do begin

      { 设置自装载的标志 }
      Flags := Flags or $800;   { Set Self Loading Flag }

      { 设置新的自动数据段 }
      if (AutoDataSeg > 0) then Inc(AutoDataSeg);

      { 启动段的段号加一 }
      if (_CS > 0) then Inc(_CS);

      { 堆栈段的段号加一 }
      if (_SS > 0) then Inc(_SS);

      { 将快装区域置零 }
      GangLoadStart := false;
      NewOfsGangLoad := 0;
      NewLenGangLoad := 0;
    end;

    if not fPrompt then Write('    ');
    InsertNewSeg;
    CurrOfs := 0;
    NextOfs := 0;
    Seek(FT, 0);
    BrowseInfoTree(Root, ProcessNode);

    { 设置 MZ 文件头 }
```

```
    Seek(FT, 0);
    BlockRead(FT, MZ, $40);
    SetImageSize(mz, T.ofsNe + GetOfsEndOfHeader(T.NE)-ofsNe);
    MZ.ofsWinHeader := T.ofsNe;
    MZ.ofsRelocList := $1C;                    { TDump Error }
    MZ.noOverlay    := 1;
    Seek(FT, 0);
    BlockWrite(FT, MZ, $40);

    { 设置快装区域 }
    if GangLoad then begin
      T.NE.ofsGangLoad :=
        wMin(LongInt(T.NE.ofsGangLoad) * GetPageSize div T.PageSize,
             T.NewOfsGangLoad div T.PageSize);
      T.NE.lenGangLoad :=
        T.NewLenGangLoad div T.PageSize;
    end;

    { 设置非驻留名表 }
    T.NE.ofsNonresidentNameTab :=
      T.NE.ofsNonresidentNameTab - ofsNe + T.ofsNe;
    Seek(FT, T.ofsNe);
    BlockWrite(FT, T.NE, $40);

    Seek(FT, T.ofsNe + T.NE.ofsSegmentTab);
    BlockWrite(FT, T.ofsBootSeg, 2);
    Seek(FT, T.ofsNe + T.NE.ofsSegmentTab + 8);
    BlockWrite(FT, T.PMySegTab^.P^, T.NE.lenSegmentTab-8);
    PWord(T.PMyResource^.P)^ := T.ResAlign;
    Seek(FT, T.ofsNe + T.NE.ofsResourceTab);
    BlockWrite(FT, T.PMyResource^.P^, T.NE.lenResourceTab);
    if not fPrompt then  Write(#8#8#8#8#8#8#8#8#8'          ');
  end;
end;

procedure TPackWin.Run;
begin
  Packing;
end;

{ MS-DOS file header object ------------------------------ }

type
  PPackDos = ^TPackDos;
  TPackDos = Object (TMzHeader)
    Name1, Name2: PathStr;
    constructor Init(SName, TName: PathStr);
    procedure Run; virtual;
  end;

constructor TPackDos.Init(SName, TName: PathStr);
begin
```

```
  inherited Init(SName);
  Name1 := SName;
  Name2 := TName;
end;

procedure TPackDos.Run;
var
  PackOvl: Boolean;
begin
  PackOvl := false;
  if ExistOverlay then begin
    Writeln;
    Write('File: ', Name1, ' may contain overlays.  Compress (y/n)?');
    if Upcase(ReadKey) = 'Y' then begin
      PackOvl := True;
      Writeln('Y');
    end
    else begin
      Writeln('N');
      Exit;
    end;
  end;
  PackExe(Name1, Name2, PackOvl);
end;

{ File Struct Object ------------------------------------ }

type
  PFileApp = ^TFileApp;
  TFileApp = Object
    P: PMzHeader;
    constructor Init(FName, FName2: PathStr);
    destructor Done; virtual;
    procedure Run;
  end;

constructor TFileApp.Init;
var
  D: DirStr;
  N: NameStr;
  E: ExtStr;
begin
  if (FileExists(FName)) then begin
    case GetFileSign(FName) of
      WinAppSign: begin
                    if fPackWin then
                      P := New(PPackWin, Init(FName, FName2))
                    else rtError := $EE11;
                  end;
      ExeAppSign: begin
                    if fPackDos then
                      P := New(PPackDos, Init(FName, FName2))
```

```
                    else rtError := $EE11;
                  end;
      else rtError := $EE11;
    end;
    if rtError <> 0 then fail;
  end
  else begin
    rtError := $EE00;
    fail;
  end;
end;

procedure TFileApp.Run;
begin
  P^.Run;
end;

destructor TFileApp.Done;
begin
  Dispose(P, Done)
end;

{ ---------------------------------------------------- }

const
  TmpFName = 'PACKWIN.TMP';

{ 压缩一个 Windows 程序 }
function PackWinAp(FName, FName2: PathStr): LongInt;
var
  D: DirStr;
  N: NameStr;
  E: ExtStr;
  fSetName: Boolean;
  PTest: PFileApp;
  OldSize, PackedSize: LongInt;
begin
  if Pos('.', FName) = 0 then FName := FName + '.EXE';
  if FName2 = '' then begin
    fSetName := True;
    FSplit(FName, D, N, E);
    FName2 := D + N + '.OLD';
  end
  else fSetName := false;
  FName := UpcaseStr(FName);
  FName2 := UpcaseStr(FName2);

  if FileExists(FName2) then begin
    Write('File: ', FName2, ' already exists.  Overwrite (y/n)? ');
    if (Upcase(ReadKey) = 'Y') then
      Writeln('Y')
    else begin
```

```
    Writeln('N');
    Writeln;
    PackWinAp := 0;
    Exit;
  end;
  Writeln;
end;

if fPack then
  Write('Compressing: ', FName)
else Write('Processing: ', FName);
if fSetName then
  Write(' ...')
else Write(' into file ', FName2, ' ...');
if fPrompt then Writeln;

PTest := New(PFileApp, Init(FName, FName2));
if PTest = nil then begin
  if not fPrompt then begin
    Writeln;
    if not fGroupPack then Writeln;
  end;
  case rtError of
    $ ee10:Writeln('PACKWIN only compress DOS and WIN executable file.');
    $ ee11:Writeln('File could not be compressed.');
    $ ee12:Writeln('PACKWIN only compress executable file.');
    $ ee13:Writeln('Cannot compress file compiled by WATCOM-C.');
    { ...  Append New Error Handle ... }
    else DisplayRtError;
  end;
end
else begin
  PTest^.Run;
  Dispose(PTest, Done)
end;

if fPack then begin
  OldSize := GetFileSize(FName);
  PackedSize := GetFileSize(FName2);
  if (PackedSize <> 0) and
     (PackedSize < OldSize) then begin
    if fSetName and not fGroupPack then begin
      RenameFile(FName, TmpFName);
      RenameFile(FName2, FName);
      RenameFile(TmpFName, FName2);
    end;

    Writeln;
    Write('Original: ', OldSize,
          '  Compressed: ', PackedSize,
          '  Ratio: ');
    WriteRatio(OldSize, OldSize - PackedSize);
```

```
      Writeln('%');
    end
    else begin
      DelFile(FName2);
      if (not fGroupPack) and (rtError = 0) then
        Writeln('File could not be compressed.');
    end;
  end; { if }
  PackWinAp := PackedSize;
  Writeln;
end;

{ 压缩一组应用程序 }
procedure GroupPackAp(FName: PathStr);
var
  D: DirStr;
  N: NameStr;
  E, E0: ExtStr;
  DirInfo: SearchRec;
  TotalFileNum: Word;
  Size, Size2, TotalOldSize, TotalPackedSize: LongInt;
begin
  fGroupPack := True;
  TotalFileNum := 0;
  TotalOldSize := 0;
  TotalPackedSize := 0;
  FName := UpcaseStr(FName);
  FSplit(FName, D, N, E0);

  if ((Pos('.EXE', FName) > 0) or
      (Pos('.DLL', FName) > 0)) then begin

    FindFirst(FName, Archive, DirInfo);
    while (DosError = 0) do begin
      Inc(TotalFileNum);
      FName := D + DirInfo.Name;
      Size := GetFileSize(FName);
      Size2 := PackWinAp(FName, '');
      if (Size2 > 0) then begin
        Inc(TotalOldSize, Size);
        Inc(TotalPackedSize, Size2);
      end;
      FindNext(DirInfo);
    end;  { while }

    if TotalFileNum > 0 then begin
      FindFirst(D + '*.OLD', Archive, DirInfo);
      while (DosError = 0) do begin
        FName := D + DirInfo.Name;
        FSplit(FName, D, N, E);
        RenameFile(FName, D + TmpFName);
        RenameFile(D + N + E0, FName);
```

```
        RenameFile(D + TmpFName, D + N + E0);
        FindNext(DirInfo);
      end;  { while }
    end;  { if }

    if (TotalOldSize <> 0) and (TotalPackedSize <> 0) then begin
      Writeln;
      Write('Total Original size: ', TotalOldSize,
        ' Total Compressed size: ', TotalPackedSize);
      if TotalOldSize > TotalPackedSize then begin
        Write('  Ratio: ');
        WriteRatio(TotalOldSize, TotalOldSize-TotalPackedSize);
        Writeln('%');
      end; { if }
    end; { if <> 0 }

  end; { if .EXE or .DLL }
end;

{ ---------------------------------------------------- }

{ 处理命令行参数 }
procedure ProcessCmdLine(var FName, FName2: PathStr);
const
  FirstSetName: boolean = true;
var
  I: Word;
  S: PathStr;
begin
  FName  := '';
  FName2 := '';
  fPrompt := False;
  if ParamCount > 0 then begin
    I := 0;
    repeat
      Inc(I);
      S := ParamStr(I);
      if (not (S[1] in OptionChars)) then
        if FirstSetName then begin
          FName := S;
          FirstSetName := False;
        end
        else FName2 := S;
    until (I = ParamCount);
    for I := 1 to ParamCount do begin
      S := UpcaseStr(ParamStr(I));
      if S[1] in OptionChars then begin
        Delete(S, 1, 1);
        if S = 'B' then
          fPackBmp  := False
        else if S = 'R' then
          fPackRes  := False
```

```
          else if S = 'W' then
            fPackStub := False
          else if S = 'M' then
            fPrompt := True
          else if S = 'S' then
            fPack := False
          else if S = 'WIN' then
            fPackDos := False
          else if S = 'DOS' then
            fPackWin := False;
        end;
      end;
    end;
    if not fPackRes then
      fPackBmp := false;
end;   { ProcessCmdLine }

var
  FName, FName2: PathStr;

begin     { Main Program }

  ProcessCmdLine(FName, FName2);

  Writeln(TradeMark);
  if FName = '' then begin
    DisplayHelp;
    Halt(1);
  end;

  if ((Pos('*', FName) > 0) or (Pos('?', FName) > 0)) and
     (FName2 = '') then
    GroupPackAp(FName)
  else PackWinAp(FName, FName2);

end.

{ ---------- End of PACKWIN.PAS ---------- }
```

第 9 章

开发 Windows 加密软件

Windows 在国内推广还存在一个比较大的困难，就是即使开发了 Windows 的产品也难以销售，因为市场上没有一套实用的 Windows 版加密软件，在中国的市场上，不加密的软件用户不愿意购买，销售商不愿意销售。本章介绍我们自己开发的 Windows 加密软件 BITLOK for Windows(BLW)的主要技术，以协助读者应用加密手段更好地保护自己开发的 Windows 应用软件，是国内用户能够享用 Windows 的好处。

9.1 软件加密基础与典型的加密软件

9.1.1 软件为什么要加密

人们经常会提出这样的问题，“国外的软件都是不加密的，国内软件为什么要加密?”、“加密过的软件没有解不开的，为什么还要加密?”等等。加密的确给用户带来许多不便，也给软件开发者带来了不少麻烦，可谓“弊多利少”。但国内软件厂商为何还要选择加密这一劳民伤财的保护措施呢?

首先，让我们看看国外的软件市场。在 PC 机软件开发初期，象 dBase 这样的通用软件都是加密的。在那个时代，有不少好的加密软件和解密工具，如激光孔加密软件 PROLOK，万能拷贝工具 CopyWrite 和 Copy II PC 等，有关加密技术的文章随处可见。近年来，绝大多数的软件是不加密，这是因为国外的版权法非常严格，对盗版现象有严厉的处罚措施；一般一部机器必须买一套软件。软件开发者不需要通过加密的手段就能获得应有的利益，而且省去了加密给软件的维护带来的麻烦。因此，国外流行的通用软件大都是不加密的。当然也有例外，大型系统如网络版 AutoCAD 12.0 由于销量小、价格高，不得已仍然无法放弃加密手段。看来，加不加密的问题在美国也是具体情况具体分析。

在国内，由于法制观念不强，软件保护法规不完善，盗版现象非常严重。这严重打击了软件开发者的积极性，也影响了软件行业的健康发展。如果软件不加密，大家都可以拷来拷去，花钱购买的用户便会觉得不合算。这样不仅用户不愿意买，而且销售商不愿意卖。国内的软件加密不是开发者愿意劳神，而是为了保护自己的版权。如果大家都使用正版软件的话，估计国内的软件行业会和国外一样，不再选择加密这种方式。目前，国内也有一些软件开发者正在尝试不加密上市自己的产品，这是一个好的开端。社会只有逐步重视软件的知识产权，软件行业才能蓬勃发展起来。

加密是目前保护知识产权的一种有效方式。但任何加密软件都可能被破译，我们不能因噎废食。加密与解密，是矛和盾的关系，要想防止被解密，必须提高加密技术。对于目前

的软件行业来说，加密的目的只是争取尽可能长的时间内不被解密，在这段时间内收回开发投资并开发产品的新版本。

加密软件因为它的作用特殊，在一段时间内还将广泛存在。未来加密软件的重点会由防拷贝转向反跟踪反破译。采用新技术和算法开发的加密软件，为了防止竞争对手知晓这些技术和算法，必须采取一些自我保护方法，如反跟踪、反破译等手段。国内这样的软件还不多，主要的代表作是黄玫瑰的 BITSHELL。

9.1.2 加密软件的原理

一般的用户都知道“加密软件”这个概念，但知道“加密软件具体是如何工作”的人相当少。我们首先介绍一下加密软件的工作方式。

加密软件有如下三种方式：

外壳式：加密软件把一段加密代码附加到执行程序上，并把程序入口指向附加代码中。当被加密的程序装入内存后，附加代码首先执行，检查是否有跟踪程序存在，如果没有再检查密钥是否正确，如果正确，则转入原来的程序中。

这种方式的优点是不需要修改源代码，使用简单。然而，其缺点也很明显：一旦附加代码被击破，就会被解得干干净净，没有一点遗留的问题。

内含式：加密代码以 OBJ 文件形式存在，在应用程序调用这些加密代码，最后与要加密的程序编译连接到一起。

这种方式需要修改源代码，比较可靠，但是代码复杂性不如外壳式，不容易对二进制代码做复杂变形，容易被跟踪。这种方式主要用于使用软件狗和加密卡的加密程序。

结合式：把上述两种方法结合起来。用 OBJ 去检查外壳的可靠性，内外结合。

9.1.3 加密软件最好定制

不管采用哪一种加密方式，通用的加密软件有一个致命的弱点：解密者可以从市场上买一套反复研究，并将解密方法套用到其它软件上。专门定制的软件在市面上不易得到，而且可以根据具体需求作特别改制。相对来说，定制的加密软件的功效最强。软件厂商一般是通过定制专用的加密软件来提高软件的保险系数。

9.1.4 加密软件与密码学的关系

加密软件与密码学有着很深的联系，但不是一回事。加密软件的变形算法都源于密码学的理论。

有的加密软件采用的变形算法比较简单，认为无关紧要，其实这样给解密者留下了很大的后门，比如说用这个加密软件加密“全零”等各种有规律的数据来研究推断，不用分析程序就可以轻松地解开。为了提高加密软件的安全性，必须使用复杂可靠的算法。例如 BITLOK 中采用了十几套随机可选的算法，有效地增加了解密难度。

9.1.5 加密软件的市场现状

国外的加密软件以 PROLOK 等著称。国内的加密系统由于市场的需求，更是百花齐放，不胜枚举。目前市场上销售的有 BITLOK 系统(黄玫瑰开发)，北京微宏的 LOCK93 及

LOCK93 NT，昆明明星电脑公司的中国锁，清华正方的终结者和得力研究所的 KEYMAKER 3.0 等。

加密软件的生存基础：加密软件生存基础是密钥可靠性和软件兼容性。密钥的可靠性包括密钥的不可复制性和密钥判读的可靠性。如果密钥能够被复制，那么软件开发者的利益无法保护；如果密钥判读不可靠，有的机器可以读，有的机器不能读，或着有时能读出来，有时读不出来，那么就不能保证用户购买的软件能始终正常运行。软件兼容性指加密软件应该不影响原来软件的兼容性，即原来的软件能在一部机器、一种软件环境上运行，加密后的软件应该同样能在这部机器、这种软件环境上运行。影响加密软件兼容性的原因是，为了追求好的反跟踪效果，加密软件的代码都是尽可能复杂、特殊，势必会影响软件的兼容性。所以，密钥可靠性和软件兼容性是选购加密软件的最重要的两点。

市场上加密软件的特点：市场上加密软件的功能特点各不相同，下面以 BITLOK 为例介绍通用加密软件的特点。

BITLOK 是一套可安装到硬盘、网络上的功能全面的加密系统，主要用于保护软件开发者的合法权益，防止未经授权的复制、算法解读及目标码反汇编。BITLOK 最初研制始于 1989 年夏天，同年 8 月 17 号推出第一个商品版 0.99。在广大用户和许多朋友的支持下，经过四年的不断改进和增强，终于在 1993 年 8 月由代理商北京微远公司推出了 BITLOK 1.0。

跟踪了半年的用户反馈信息后，在 1994 年 3 月推出了 BITLOK 1.5，1994 年 7 月推出了 BITLOK 1.6，1994 年 9 月将推出 BITLOK 2.0。在 BITLOK 几年的发展过程中，黄玫瑰软件工作组解决了数不尽的软硬件问题，可以这样说，BITLOK 的每一个字节都凝聚着创造者的心血和汗水，每一比特都体现了开发者对软件版权问题的关注。时至今日，BITLOK 功能完善，运行可靠，获得了广泛的好评。超过 30 万台各种各样的 PC 每天都在运行着经 BITLOK 加密的各种系统，足见其兼容性与可靠性非同寻常。

主要功能和特能如下：

使用最新的密钥技术，不生成坏道坏块、密钥可靠、能自动识别插密匙的驱动器。内含随机可选的 20 套加密算法，可构造出千变万化的加密方案，特别适合加密有多个单独执行程序的软件，买一套 BITLOK 至少相当于得到 20 套普通加密软件。这一点是其他系统无法比拟的。同时，它采用加密虚拟机、多层间址多堆栈链接等独创的世界领先技术，结合传统加密方法，具备超强的动态反跟踪能力，防范各种软硬件调试器对其破译。不仅防止了国际上通用的调试器，如 SoftICE，Turbo Debugger，SymDeb 等，还重点防范了国内开发的各种专用调试器。BITLOK 兼容各种厂牌、各种档次的 PC。BITLOK 可加密各种带覆盖模块的 EXE 文件，特别适合加密由 Clipper、FoxPro 等编译的各种数据库应用系统。加密母盘可以使用任意多次，加密过的软件支持硬盘安装和网络安装，安装次数可设定。提供的 OBJ 加密模块可与外壳部分相互关联，有效地提高破译难度。

用 BITLOK 制作加密的商品软件速度快。通常的加密软件的制作工序复杂，生产率低。如果制作一套商品盘需要 10 分钟，制作第二套时，所有的工序还必须走一次。如第一次加密两个文件，第二次还是需要加密这两个文件。而 BITLOK 具有超群的批制作功能：制作出第一张加密盘后，再生成第二套软件除去拷盘时间仅需几秒种。BITLOK 提供这种轻松的制作方式的好处是显而亦见的。

与 BITLOK 配套的系统还有 BITSHELL。BITSHELL 是一套反跟踪反破译系统，用户可以自己开发加密卡、加密盘配合使用。它功能上与 BITLOK 相似，只是不含密钥生成和识别部分。

9.2 加密软件的核心技术

目前，加密软件主要是在 DOS 上开发的。加密软件的主要技术方法都是类似的，我们这里讨论 DOS 上的软件加密技术，在接下来的几节中，我们将这些技术应用到 Windows 上的加密软件中。软件加密技术主要由密钥技术、反跟踪技术和代码插入技术构成，缺一不可。

9.2.1 密钥技术

密钥技术是指与密钥有关的全部技术，如密钥盘的制作、识别密钥盘、安装和回收密钥等。密钥的特点就是不能或不容易复制。密钥主要分软盘、加密卡和并行口加密盒(也称“软件狗”)。这三种密钥的特点如下。

	软 盘	加密卡	加密盒
成本	低	高	较低
可靠性	一般	稳定	稳定
易安装性	不需要安装	复杂	容易
使用方便性	不方便	方便	方便
系统开销	判别时需插盘	占扩展槽和地址空间	并行口
产品盘备份	不能	可以	可以

由上表可见，三种密钥各有千秋，可以根据不同的需求选用。比如，对于一般的小型的软件来说，由于成本问题，选用软盘密钥的加密软件合适一些。我们来看一下软盘密钥的原理和制作的方法。

在软盘上制作密钥的技术有很多种。最早的 PROLOK 是用激光在软盘上打孔，在软盘上制作无法复制的硬错误。还可以在软盘上制作弱位。软盘上的信息是二进制，即 0 与 1。但弱位不是 0 也不是 1 — 有时是 0，有时是 1。这种信息用计算机无法复制。目前，加密软件主要采用道缝加密。将数据写到某一已格式化过的扇区时，因为不同的磁盘机会有转速的差别，因此数据写完后不可能与下一个扇区连接得无缝隙，而且这缝隙中就会出现一些噪音信号。利用这种噪音信号来作为辨别是否是原盘的依据。这种方式可使目前所有拷贝程序(包括拷贝卡)都无法复制。具体地说，由于磁道是圆形的，因此必然有头尾相接的地方，用这种接缝作为“指纹”来模仿 PROLOK 所用的激光孔“指纹”，可使每片母盘都不同，母盘的辨认程序的细节也会不同。制作方法的简要说明如下：

1. 将格式化磁道的参数表中最后一个扇区 N 置为 6；

2. 用 INT 13H 格式化一个磁道(实际上还是用正常大小格式化,第一步的结果只是改动了最后一个扇区的地址场);

3. 把 INT 1EH 指向的参数表中的 N 改为 6;

4. 读最后一个扇区,就可把接缝读出来(此时错误号为 10H);

5. 把 INT 1EH 指向的参数表中 N 恢复为 2;

不同的盘、不同的磁道,制作出来的密钥是不同的。下面是一个用来制作密钥和判读密钥的完整程序。该程序格式化插在 B 驱动器中的 360K 的软盘,密钥道在 36 道 1 面。该程序用 Turbo Assembler 编译成 COM 文件。

```
        dosseg
            .model    tiny
            .code
            org       100h

Main:
            push      cs
            pop       ds
            jmp       FormatTrack

Drive       equ       1          ; Drive B

FormatID    db   24h,1,1,2,   24h,1,2,2
            db   24h,1,3,2,   24h,1,4,2
            db   24h,1,5,2,   24h,1,6,2
            db   24h,1,7,2,   24h,1,8,2
            db   24h,1,9,6      ; 注意:最后 N 为 6
                                                ; STEP 1

FormatTrack:
            mov  ax, 0                          ; STEP 2
            int  13h
            mov  bp, 5
FT0:        mov  ax, 0509h
            lea  bx, FormatID
            mov  dh, 1
            mov  dl, Drive
            mov  cx, 2401h
            int  13h
            jnc  FT1
            dec  bp
            jnz  FT0
            jmp  FT2
FT1:
            push ds                             ; STEP 3
            xor  ax, ax
            mov  ds, ax
            mov  byte ptr ds:[525h], 6
            pop  ds

            mov  cx, 5                          ; STEP 4
```

```
F0:
        push cx
        mov  ax, 201h
        mov  bx, 1000h
        mov  cx, 2409h
        mov  dh, 1
        mov  dl, drive
        int  13h
        pop  cx
        cmp  ah, 10
        jz   F1
        loop F0
        jmp  ft2
F1:
        mov  dx, offset ok
        jmp  ft3
FT2:
        mov  dx, offset error
FT3:
        push ds                                  ; STEP 5
        xor  ax, ax
        mov  ds, ax
        mov  byte ptr ds:[525h], 2
        pop  ds

        mov  ah, 0
        int  21h

        mov  ah, 4ch
        int  21h

OK      db   'Make key disk ok!', 0dh, 0ah, '$'
Error   db   7, 'Sorry, Make key disk fail!', 0dh, 0ah, '$'

        ends
        end  main
```

9.2.2 反跟踪技术

有了好的密钥技术，若没有强有力的反跟踪手段来保护它，那么软件解密高手使用功能强大的调试器和一些辅助工具来破译，几分钟就可以解开，达不到保护的目的。反跟踪技术是加密软件最关键、工作量最大的部分，同时也是加密技术不断推陈出新、永无止境的原因。

反跟踪的手段主要有以下几种：一种方法是把程序写乱或使用高级语言生成代码，使代码杂乱无章，跟踪者不容易看懂。另一种方法是主动出击，设置陷阱，让调试器不能正常工作。如破坏 INT 3 和 INT 1 的中断向量，让 DEBUG 不能工作；识别调试器的存在情况，让程序走岔道。还有一种方法是做大量变形和跳转，拖垮对手。

比如说，在 BITLOK 使用了加密虚拟机、多层间址多堆栈链接等反跟踪技术。具体方法是：设计一个加密虚拟机，虚拟机指令用 INTEL 80x86 的指令来仿真，加密程序使用虚拟机指令来写，跟踪者自然很难看懂虚拟机指令；而且加上多层间址多堆栈链接等大工作量的

操作，跟踪者即使看懂虚拟机指令，也容易被拖垮。

就未来的加密软件而言，最重要的是如何保护算法，那么反跟踪技术显得更为重要。黄玫瑰软件开发组的 BITSHELL 就是这样一个专门的产品，它不包含密钥部分，主要用于防止跟踪者破译算法。

9.2.3　代码插入技术

加密的核心代码写好以后，剩下的工作是如何把这些代码插入到被加密的程序中。插入加密代码的方式较多，但首先必须了解 DOS 的执行文件格式。上一节中我们讲过，插入方式主要有内含式、外壳式、结合式。

内含方式把插入代码附加在文件尾部，把程序的入口点改到插入代码中，DOS 依然使用 EXEC(INT 21H，功能 4BH)来执行被加密程序(这种方法也是病毒传染的常用方法)。这种方法实现起来简单，但有很大的局限：无法插入到带覆盖模块的程序中，无法有效地解决 EXE 文件的代码变形问题。

外壳方式采用的是另外一种方法。它首先将原来的执行文件，比如说 sample.exe，变形为 sample.ovl，插入代码将 sample.ovl 看成覆盖模块，插入代码给自己取名为 sample.exe，由 sample.exe 调用 sample.ovl，并决定 sample.ovl 的装载和重定位方式(包括在装载过程中将被变形的代码复原)。

假如原来的执行文件 sample.exe 本身带有覆盖，上述方法就会碰到问题。因为象 FoxPro 等编译出的数据库执行文件一般带有覆盖模块，而且它们的文件头内容相同。如果用对待一般执行文件的办法来加密这样的执行文件，跟踪者很容易识别它的文件头和代码部分，跳过被加密的文件头，被加密文件的核心代码部分也就暴露无遗，起不到加密的效果。比较好的做法是：插入代码把控制权转交给被变形的 sample.ovl 之前，先将一个监控代码驻留在内存中，用来接管 INT 21H 操作。如果 sample.ovl 要执行某一个覆盖模块，则该驻留代码截获文件操作，还原将要执行的覆盖代码。

在实现代码插入时，还应该注意非 MZ 格式的问题。目前在 DOS 上 EXE 文件主要是 MZ 格式，但使用 OS/2 格式和 BOSS 格式的程序越来越多，如以 Borland Pascal 保护方式编译出的程序。这些问题增加了开发难度，读者应该深入钻研，具体技巧不再赘述。

9.3　开发 Windows 加密软件——BITLOK for Windows

Windows 加密软件总体结构与 DOS 上的加密软件是一致的，我们仍然把 Windows 加密软件分成三部分来讲。

9.3.1　密钥技术

密钥技术采用与 DOS 上完全一样的方法，但程序在写法上有所不同，必须考虑到 Windows 的兼容性。在 DOS 上可以采用多种方法调用磁盘中断，如：INT 13H 和 INT 40H 及 PUSHF CALL 0F000H:0EC59H 等方法。Windows 在保护模式下运行，不支持 INT 40H，更不能直接调用地址。因此，最好使用 INT 13H 的调用。

另外,还可以使用 DPMI 的功能调用(详细的 DPMI 调用可以参见香港金山公司求伯君主编的《深入 DOS 编程》一书),DPMI 使用的中断号是 31H。

```
INT 31H  AX = 0300H  仿真实模式下的中断
进入:    BL 中断号
         BH 标志(全为零)
         CX 从保护模式栈拷贝到实模式栈的字数
         ES:(E)DI 实模式寄存器数据结构的选择器:偏移量
返回:    ES:(E)DI 修改过的实模式寄存器数据结构的选择器:偏移量
```

如何编写 Windows 加密软件的加密代码部分,这是一个难点。在第 8 章中编写 PACKWIN 的自装载模块时,我们是用一个 HELLOWIN.ASM 来调试的,因为 HELLOWIN 极其简单,可以使我们把注意力集中在需要解决的问题上。这里我们也使用相同的方法来编写加密代码。下面我们逐一讲解编写时遇到的困难,这些困难也就是技术要点。

如何编写多代码段程序

首先遇到的难点是在汇编语言中如何写多个代码段。如果把加密代码放在一个单独的代码段中,就可以很容易把它取出来。由于手头的资料不全,无法解决这个问题。后来只有查一些示范程序,才知道不仅需要修改源程序,还必须修改 DEF 文件。

在程序中这样定义:

```
creatSeg _AppendCode, AppendCode, PARA, PUBLIC, CODE
sBegin AppendCode
  assumes cs, AppendCode
  ...
sEnd   AppendCode
```

在 DEF 文件中需要加入如下说明:

```
Segments
  AppendCode movable, preload
```

这样才能把插入的代码写成两个代码段。用我们第 1 章提到的工具 GetSeg 可以把在 Windows 下编写的供插入使用的加密代码完整地提取出来。

取 Seg _0000 的 Selector

在 DOS 环境中,要取 BIOS 数据区的数据相当方便。但是在 Windows 环境中,要进行同样的操作,需要设置段选择子。重新定义一个 Selector 比较麻烦,幸运的是 Windows 中 Seg _0000 有预定义的 _0000H 段的 Selector,直接使用即可。Selector 是 Windows 中最重要的概念之一,参见第 4 章。

下面示范如何取 BIOS 数据区的磁盘参数表中的扇区大小的值。

在 DOS 操作系统,取[0000:0522H]的程序如下:

```
xor  ax, ax
mov  es, ax
```

```
mov  ax, es:[0522H]
```

在 Windows 操作系统中，完成同样的工作，程序如下：

```
extrn __0000H: abs
; ...
mov  es, __0000H
mov  ax, es:[0522H]
```

置代码段属性为可写

代码段在 Windows 执行程序中是只读的。如果把数据存放在代码段中，或把代码段变形，都需要允许代码段可写。使用未写入文档的 Windows 功能调用 AllocCStoDSAlias 就可以将代码段置为可写，返回一个新的 Selector，将这个 Selector 赋给 DS，就可以直接读写代码段的数据，最后将获得的 Selector 释放掉。

显示错误信息

程序运行时，是先运行加密代码再执行启动功能。由于程序没有初始化，使用 MessageBox 自然会出错，因为程序还没有开始执行呢！因此，我们用另外一个函数 MessageBeep 代替 MessageBox。遇到错误，如读盘错误等，“嘟”一声再退出。

编写可移动的加密代码

在 BITLOK 中，加密代码先是放在一个单独的文件中，然后根据被加密文件的具体情况将附在启动段的后面。注意：加密代码在启动代码的前面运行，但物理位置是在启动段的后面。由于加密代码在被加密文件中的位置不是固定的，所以加密代码要求是可移动的，加密代码中的偏移应该是相对的，不能使用绝对的偏移值。

编写的代码放在启动段的后面，这样原来的偏移值发生了改变，因此不能使用绝对的偏移值。正确的用法如：

```
        call Here
Here:   pop  bx
        sub  bx, offset Here
```

确定代码被插入的位置，然后再调整原来的偏移值。这样，加密代码就可以随意地附在不同程序不同的位置上。

判读密钥的程序实例

下面附上判读密钥程序的源程序——在第 5 章示范程序 HELLOWIN.ASM 的数据段后面插入以下代码：

```
sBegin  CodeAppend
        assumes cs, CodeAppend
        assumes ds, CodeAppend
        include KeyMan.INC
sEnd
```

```
;-----------------------------------
; HELLOWIN.DEF module definition file
;-----------------------------------
NAME            HELLOWIN
DESCRIPTION     'Hello Windows Program'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE MULTIPLE
SEGMENTS
  CODEAPPEND    LOADONCALL MOVEABLE DISCARDABLE
HEAPSIZE        1024
STACKSIZE       8192


# HELLOWIN MAKFILE
hellowin.exe: hellowin.obj hellowin.lnk hellowin.def
  link @hellowin.lnk
hellowin.obj: hellowin.asm hellowin.lnk hellowin.def
              keyman.inc smartkey.inc \
  checksum.inc
  masm -Mx -Zi -DDEBUG hellowin.asm;

;  KEYMAN.INC
;  KEYMAN -- BITLOK for Windows   version 1.0
;  Copyright (c) 1994.6 by Yellow Rose Software Workgroup.
;

        ExternFP <AllocCStoDSAlias>
        ExternFP <FreeSelector>
        ExternFP <MessageBeep>

        Extrn __0000H: abs
        Extrn __RomBios: abs


                call    near ptr WinKey
                db      0e9h
                dw      0               ;  JMP CS:IP

                org     0ch
        db      'ROSE'

                org     10h

AKey            db      4bh, 0ffh, 3, 4, 0ah, 0
                dw      0aef7h, 0cb83h, 94c1h, 0e12ah, 0f2ch

ALocalKey       dw      -1, -1, -1

ShowErrMsg:
        push    -1
```

```
        call    MessageBeep
        ret

MySwitchCStoDS:
        push    cs
        call    AllocCStoDSAlias
        mov     ds, ax
        ret

FreeSel:
        pushf
        push    cs
        pop     ds
        push    ax              ; CurrSel
        call    FreeSelector
        popf
        ret

Get__0000H:
        push    __0000H
        pop     es
        ret

GetRomBiosSeg:
        push    __RomBios
        pop     ax
        ret

cProc   AppendProc, <FAR, PUBLIC>
cBegin  AppendProc
        call    WinKey
cEnd    AppendProc

cProc   WinKey, <PUBLIC>

localW  ReadBuffer
localW  AKeyPos
localW  ALocalKeyPos
localW  FileSpecPos

localD  Int24Save
localD  Pdta
localW  ReadCount
localW  CurrSel
localW  NewPos
localB  Head

cBegin  WinKey

        push    ax
        push    bx
        push    cx
```

```
        push    dx
        push    si
        push    di
        push    ds
        push    es

        call    Here
Here:
        pop     bx
        sub     bx, CodeAppendOffset Here
        mov     NewPos, bx


        mov     ReadBuffer, CodeAppendOffset KeyBuf
        add     ReadBuffer, bx
        .
        mov     AKeyPos, CodeAppendOffset AKey
        add     AKeyPos, bx

        mov     ALocalKeyPos, CodeAppendOffset ALocalKey
        add     ALocalKeyPos, bx

        mov     FileSpecPos, CodeAppendOffset FileSpec
        add     FileSpecPos, bx

        mov     Head, 0

        call    MySwitchCStoDS
        mov     CurrSel, ax

        call    CheckLocalKey
        jnc     @@AppendProc_OK
        call    CheckKey

@@AppendProc_OK:
        mov     ax, CurrSel
        call    FreeSel
        jnc     @@AppendProc_Exit

@@AppendProc_Error:
        call    ShowErrMsg

        mov     ax, 4c00h
        int     21h

@@AppendProc_Exit:

        pop     es
        pop     ds
        pop     di
        pop     si
        pop     dx
```

```
        pop     cx
        pop     bx
        pop     ax

cEnd    WinKey

        include CHECKSUM.INC
        include SMARTKEY.INC
        include LOCALKEY.INC

FileSpec        db      'A:*.*', 0
KeyBuf          db      2*400h dup('*')
```

SMARTKEY.INC 中包含读密钥的核心部分，读者可以把前面给出的 DOS 下判密钥的程序修改成 Windows 下执行的汇编代码即可。本书所附磁盘的 CHAP9 目录下的 LOCALKEY.INC 中包含硬盘安装的部分代码。

9.3.2　反跟踪技术

Windows 下的调试器主要有 SoftICE for Windows、Turbo Debugger for Windows 和 CodeView for Windows。反跟踪的重点在于发现这些调试器并击垮他们。在反跟踪部分可以从 Task Database 或 PSP 中查询这些软件是否存在，如果存在就退出。

9.3.3　加密代码插入技术

如果使用第 6 章的自装载技术把加密代码作为自装载代码插入到程序中，可以把加密做到相当复杂的程度。这种方法存在一些局限性，如：如果已经是自装载的程序，则无法加密。在上一章中我们给出了编写 PACKWIN 的方法，读者可以应用压缩的方法实现对 Windows 执行程序的加密，这种方法可以把加密做得极其复杂。

在这里，我们使用第 4 章的技术把加密代码直接加到启动段中。这种方法完成一个简单的加密软件很容易，但解决复杂的代码变形比较困难。

License.PAS 实现的是加密软件关于密钥的部分。使用这个库非常简单，只需要调用为数不多的几个函数。

```
{生成加密盘}
function MakeFloppyKey (Drive: byte;
var MyKey: KeyInfoStruc; var ABuf: PBuf): boolean;
{安装加密软件}
function InstallFloppyKey (Test: Boolean;
var MyKey: KeyInfoStruc): boolean;
{取硬盘安装的参数}
function GetLocalKey: PBuf;
```

下面所附的是 License.pas 的接口部分。

```
Unit License;

interface
```

```
uses Objects, ExtTools;

type

  {
    BITLOK call 'MakeLicense' and 'GetTransferParameters' ;
    BLINST call 'InstallLicense'
  }

  PLicenses = ^Licenses;
  Licenses = object (TObject)

    function GetLicenseClassName: PChar; virtual;

    function CheckRunLicense: boolean; virtual;
    function MakeLicense: boolean; virtual;
    function GetTransferParameters: PBuf; virtual;

    function InstallLicense: boolean;
    function CheckInstallLicense: boolean;
    function GetLicenseParameters(var Limit: word): boolean; virtual;
    function SetLicenseParameters(Limit: word): boolean; virtual;

  end;

  { Key inforamtion }

  PKeyInfoStruc = ^KeyInfoStruc;
  KeyInfoStruc = record
    Track,
    Sector,
    SecType,
    Media,
    Limit,
    Reserved: Byte;
    K: array [0..4] of word;
  end;

  PSmartKey = ^SmartKey;
  SmartKey = Object (Licenses)

    PDisk: PBytes;
    ADrive: byte;
    AKey: KeyInfoStruc;

    constructor Init (KeyDrive: byte; KeyInfo: KeyInfoStruc);
    destructor Done; virtual;

    function GetLicenseClassName: PChar; virtual;

    function CheckRunLicense: boolean; virtual;
    function MakeLicense: boolean; virtual;
```

```
    function GetTransferParameters: PBuf; virtual;

    function GetLicenseParameters(var Limit: word): boolean; virtual;
    function SetLicenseParameters(Limit: word): boolean; virtual;

  private

    function ReadSectors (Drive, Track, Head, BeginSec, SecNum: byte;
      var P: PBytes): boolean;
    function WriteSectors (Drive, Track, Head, BeginSec, SecNum: byte;
      P: PBytes): boolean;
    function WaitDiskReady (Drive: Byte): boolean;
    function DriveType(Drive: byte): byte;
    function BiosDriveType (Drive: byte): byte;
    function MediaType(Drive: byte): byte;
    function GetDiskState (Drive: Byte): Byte;
    function GetUsableKeyDrive (Media: Byte): Byte;
    procedure SetDASA (MyDrive, MyDriveType, MyMediaType: Byte);

    function ReadKeySector (Drive, KTr, KHead, KSec, KSecType: byte;
      var K1, K2: word): boolean;
    function CreateKeySector
      (Drive, KTr, KHead, KSec, KSecType: byte): boolean;

  end;

  function GetLocalKey: PBuf;

  function MakeFloppyKey (Drive: byte;
    var MyKey: KeyInfoStruc; var ABuf: PBuf): boolean;
  function InstallFloppyKey (Test: Boolean;
    var MyKey: KeyInfoStruc): boolean;

  { 具体实现不赘述 }
```

下面所附的是 BLW 的主程序，我们可以从中领略精妙的加密代码插入技术。

BLW 的主要的实现方法是把加密代码插入到启动段的尾上，并把程序的入口点定向到改在加密代码中。BLW 的加密代码放在一个段的任何地方都可以，所以不论启动段有多长，把启动段代码与重定位部分分开，直接把加密代码插入即可。这种方法与大部分 DOS 上的加密程序是一样的。代码变形我们用 PACKWIN 来实现，也就是说，我们把一个软件用 BLW 加密后，再用 PACKWIN 来压缩，可以保证不容易被解开。

BLW 的实现方法与我们第 4 章中提到的在一个段后面附加代码的情况是完全一样的。

Step 1. 修改文件首部：如果文件存在快装区域(Gangload Area)，需要重新计算快装区的开始地址和长度。一个段加长了，首先需要修改段表中该段及以后段的偏移，其次需要修改资源表中记录的该段之后的资源所在的偏移；

Step 2. 复制首部之后、该段以前的其他内容；

Step 3. 先写该段的代码或数据，再写入加密代码，最后写该段的重定位的部分；

Step 4. 把文件其它部分复制过来即可。

这个例子只是演示了大致的功能，还有一些不完备的地方，如没有考虑如果被加密的程序没有 KERNEL 和 USER 模块时应添加这两个模块等。读者在阅读时要抓住本质的东西去体会。参照例子，可以仔细思考如何实现加密软件。

```
{ BLW.PAS }
{ BITLOK for Windows -- Copyright (c) 1994.6 by KingSoft Beijing Co. }

Uses FileDef, ExtTools, Dos, License, SetKey;

procedure WinKey; external; { $L WINKEY.OBJ}

{ File Struct Object }

type
  PAppendFile = ^TAppendFile;
  TAppendFile = Object(TNeHeader)
    { 加密程序的对象}
    F2: File;
    Key: PBuf;
    CurrOfs: LongInt;
    AddPages, csLen,
    KernelModuleNo, UserModuleNo,
    OldKernelModuleNo, OldUserModuleNo: word;
    PWinKey, PProc, PReloc: PBuf;

    constructor Init(FName, FName2: PathStr; ABuf: PBuf);
    destructor Done; virtual;

    procedure LoadWinKey;
    procedure FreeWinKey;
    procedure Run; virtual;
  end;

{ TAppendFile Object }

constructor TAppendFile.Init;
begin
  Key := ABuf;
  PWinKey := nil;
  PProc := nil;
  PReloc := nil;

  {初始化文件名}
  if Pos('.', FName) = 0 then FName := FName + '.EXE';
  { Default name is TEMP.EXE }
  if FName2 = '' then FName2 := 'TEMP.EXE';
  inherited Init(FName);
  Assign(F2, FName2);
  Rewrite(F2, 1);

  {取启动段的段号}
  csLen := GetSegmentEntry(PSeg, neHeader._CS)^.lenSeg;
```

```
  {取模块号}
  KernelModuleNo := GetModuleIdx('KERNEL');
  UserModuleNo   := GetModuleIdx('USER');
  LoadWinKey;
end;

destructor TAppendFile.Done;
begin
  FreeWinKey;
  Close(F2);
  DisposePBuf(Key);
  Inherited Done;
end;

{遍历 WINKEY 的重定位表}
{ LoadWinKey:  Browse relocation table }
procedure NewRelocList
  (PSelf: PNeHeader; P: PRelocItem; Body: PBytes); far;
begin
  With PAppendFile (PSelf)^ do begin
    Inc(P^.Ofs, csLen);
    if (P^.RelocType and 3 = 1) then begin
      if (P^.Idx = oldUserModuleNo) then
        P^.Idx := UserModuleNo
      else if (P^.Idx = oldKernelModuleNo) then
        P^.Idx := KernelModuleNo
      else Write('S');
    end; { if }
  end;
end;

{装入加密代码}
procedure TAppendFile.LoadWinKey;
const
  WinKeyName = 'WINKEY.BIN';
var
  TmpF: File;
  PA: PBuf;
begin
  if FileExists(WinKeyName) then begin
    Assign(TmpF, WinKeyName);
    Reset(TmpF, 1);
    NewPBufEx(PWinKey, FileSize(TmpF), nil);
    BlockRead(TmpF, PWinKey^.P^, PWinKey^.Len);
    Close(TmpF);
  end
  else begin
    NewPBufEx(PWinKey,
      PWord(@PBytes(@WinKey)^[$8])^ + $10,
      PBytes(@WinKey) );
  end;
```

```
  Move(Key^.P^, PWinKey^.P^[$20], Key^.Len);
  OldKernelModuleNo := PWord(@PWinKey^.P^[$C])^;
  OldUserModuleNo   := PWord(@PWinKey^.P^[$E])^;

  NewPBufEx(PProc, PWord(@PWinKey^.P^[2])^, @PWinKey^.P^[$10]);
  NewPBufEx(PReloc, PWinKey^.Len-PProc^.Len-$10-2,
    @PWinKey^.P^[$10+PProc^.Len+2]);

  BrowseAsegRelocList(PReloc, 0, NewRelocList);

end;

{释放加密代码的空间}
procedure TAppendFile.FreeWinKey;
begin
  if PWinKey <> nil then DisposePBuf(PWinKey);
  if PProc <> nil then DisposePBuf(PProc);
  if PReloc <>nil then DisposePBuf(PReloc);
end;


{调整段表值}
{ Browse: IncSegOfs }
procedure IncSegOfs(PSelf: PNeHeader; P: Pointer; Index: word); far;
begin
  with PAppendFile(PSelf)^, PSegmentEntry(P)^ do
    if ofs > CurrOfs then Inc(Ofs, AddPages)
end;

{调整资源表}
{ Browse: IncResOfs }
procedure IncResOfs (PSelf: PNeHeader; P: Pointer); far;
begin
  with PAppendFile(PSelf)^, PEachRes(P)^ do
   if (CurrOfs * GetPageSize <
     LongInt(NameInfo^.rnOffset) * GetResPageSize) then
     Inc(NameInfo^.rnOffset,
         AddPages * GetPageSize div GetResPageSize);
end;

procedure TAppendFile.Run;
var
  PCode: PBytes;
  Len: LongInt;
  P: PSegmentEntry;
  PMySeg, PBSeg, PBRes: PBuf;
  AddLen, N, oldIP, RelocNum: Word;
begin
  AddLen := PProc^.Len + PReloc^.Len;
  N := neHeader._CS;                    { Get boot segment number }
  P := GetSegmentEntry(PSeg, N);        { Get boot segment entry  }
  { Get boot segment relocation items entries }
```

```
RelocNum := GetRelocItemCount(N);
{ Calculate pages that need be appended }
Len := AlignSize((P^.lenSeg + 2 + RelocNum * 8 + AddLen),
       GetGeneralPageSize)-AlignSize((P^.lenSeg + 2 + RelocNum * 8),
       GetGeneralPageSize);
AddPages := Len div GetPageSize;
CurrOfs  := P^.Ofs;

oldIP := neHeader._IP - P^.lenSeg - 3 - 3;
neHeader._IP := P^.lenSeg;

{ Reset GangLoad area 计算快装区}
with neHeader do
  if GangLoad then begin
    if P^.Ofs < ofsGangLoad then Inc(ofsGangLoad, AddPages);
    if (P^.Ofs >= ofsGangLoad) and
      (P^.Ofs <= OfsGangLoad + lenGangLoad) then
      Inc(lenGangLoad, AddPages);
end;

{ Modify current segment entry  修改当前的段表项 }
Inc(P^.lenSeg, PProc^.Len);
Inc(P^.lenMem, PProc^.Len);

{ Modify all segment entry 整理整个段表}
NewPBufEx(PBSeg, neHeader.lenSegmentTab, PSeg);
BrowseSegmentTab(PBSeg, IncSegOfs);

{ Modify all resource table entry 整理整个资源表}
if neHeader.lenResourceTab <> 0 then begin
  NewPBufEx(PBRes, neHeader.lenResourceTab, PResource);
  BrowseResourceTab (PBRes, IncResOfs);
end;

{ Copy WINSTUB 复制 WINSTUB}
CopyFileBlockEx(F, 0, F2, 0, ofsNE);

{ Write header 复制 NE 首部}
BlockWrite(F2, neHeader, $40);

{ Write segment table 写段表}
BlockWrite(F2, PBSeg^.P^, PBSeg^.Len);
DisposePBuf(PBSeg);

{ Write resource table 写资源表}
if neHeader.lenResourceTab <> 0 then begin
  BlockWrite(F2, PBRes^.P^, PBRes^.Len);
  DisposePBuf(PBRes);
end;

{ Copy some codes between resource table and boot segment }
{复制从资源表到启动段之间的数据}
```

```
  Seek(F, FilePos(F2));
  CopyFileBlock(F, F2, LongInt(CurrOfs) * GetPageSize-FilePos(F2) );

  { Restore current segment entry }
  Dec(P^.lenSeg, PProc^.Len);
  Dec(P^.lenMem, PProc^.Len);
  PMySeg := ReadASeg(N);
  BlockWrite(F2, PMySeg^.P^, PMySeg^.Len);
  DisposePBuf(PMySeg);

  { * * * Write new append codes * * * 写加密代码}
  PWord(@PProc^.P^[4])^ := oldIP;
  BlockWrite(F2, PProc^.P^, PProc^.Len);

  { Copy relocation list 写重定位项}
  PBSeg := ReadARelocList(N);
  Inc(RelocNum, PReloc^.Len div 8);
  BlockWrite(F2, RelocNum, 2);
  BlockWrite(F2, PBSeg^.P^, PBSeg^.Len);
  BlockWrite(F2, PReloc^.P^, PReloc^.Len);

  { Align two files 写对齐的填充数据}
  Seek(F, AlignSize(FilePos(F), GetPageSize));
  WriteSeek(F2, AlignSize(FilePos(F2), GetPageSize));

  { Copy left codes 拷贝剩下的数据}
  CopyFileBlock(F, F2, FSize-FilePos(F));
end;

const
  FirstSetName: boolean = true;

var
  ABuf: PBuf;
  Drive: Byte;
  PAppend: PAppendFile;
  FName, FName2, S: String;

{生成密钥盘}
procedure MakeKey;
var
  AKey: KeyInfoStruc;
begin
  if KE.Source = '' then Halt(31);
  FName := KE.Source;
  FName2 := KE.Target;
  if FName2 = FName then FName2 := '';
  Drive := KE.Drive;

  ABuf := nil;
  AKey.Track   := KE.Track;
  AKey.Sector  := 255;
```

```
    AKey.SecType := 3;
    if (KE.InstNumber > $ff) or (KE.InstNumber = 0) then
      AKey.Limit := $ff
    else AKey.Limit := KE.InstNumber;

    Writeln('Make key... ');

    if MakeFloppyKey(Drive, AKey, ABuf) then
      Writeln('Build key successful.')
    else begin
      Writeln('Error: create key diskette failure. ');
      Halt(24);
    end;

  end;

  {加密主过程}
  begin
    Writeln;
    Writeln('BITLOK for Windows  version 1.0');
    Writeln('Copyright (c) 1994.6 by Yellow Rose Software Workgroup');
    Writeln;

    if ParamCount = 0 then begin
      Writeln;
      Writeln('Usage: BLW <SourceExe> <TargetExe>');
      Writeln;
      Halt(1);
    end;

    MakeKey;

    PAppend := New(PAppendFile, Init(FName, FName2, ABuf));
    if PAppend = nil then
      DisplayRtError
    else begin
      PAppend^.Run;
      Dispose(PAppend, Done)
    end;
    Writeln;
    Writeln('Encryption completed successfully.');

  end.
```

9.3.4　密钥安装技术

由于软盘容易磨损，而且使用软盘密钥特别麻烦，目前都趋向把密钥安装到硬盘上。我们用安装程序把机器的 BIOS 校验和记录在文件中，如果安装软件被拷贝到其他的机器上就无法使用。我们这里只是简单演示了安装方法。

在计算 BIOS 的 ROM 校验和时，装入 BIOS 段地址应该使用 RomBios 函数。

```
{BLWINST.PAS BLW 硬盘安装程序}

Uses Dos, License, FileDef, ExtTools;

const
  BLWFlag = $45534f52;  { ROSE BLW 的加密标志}

var

  F: File;
  ABuf: PBuf;
  P: PNeHeader;
  FName: PathStr;
  AKey: KeyInfoStruc;
  Flag, KeyPos: LongInt;
begin
  Writeln('BLWINST  Copyright (c) 1994.6 by
    Yellow Rose Software Workgroup.');

  FName := ParamStr(1);
  if not FileExists(FName) then begin
    Writeln('Error: File not exists.');
    Halt(1);
  end;

  P := New(PNeHeader, Init(FName));
  if P = nil then
    DisplayRtError
  else begin
    {计算密钥存放的位置}
    KeyPos := P^.GetAbsFilePos(P^.neHeader._cs, P^.neHeader._ip) + $10;
    Dispose(P,Done)
  end;

  {读标志}
  assign(F, FName);
  reset(F, 1);
  Seek(F, KeyPos-4);
  BlockRead(F, Flag, 4);
  if Flag <> BLWFlag then begin
    Writeln('Error: the file is not encrypted by BLW. ');
    Halt(2);
  end;
  BlockRead(F, AKey, SizeOf(AKey));

  {安装软盘的密钥}
  if InstallFloppyKey(false, AKey) then begin
    Writeln('OK!');
    ABuf := GetLocalKey;
    BlockWrite(F, ABuf^.P^, ABuf^.Len);
    {把机器的特征写到程序中去}
  end;
```

```
  close(F);

end.
```

9.4　BITLOK for Windows 的使用

BITLOK for Windows 主要由三个程序构成的。

BLW　　— BITLOK for Windows 的主程序
BLWINST — BITLOK for Windows 的硬盘安装程序
PACKWIN — Windows 的执行程序压缩后仍能执行

用法如下：

BLW

```
BLW SAMPLE.EXE W.EXE A:
```
把 A 驱动器中的盘做成密钥盘，再把 SAMPLE.EXE 加密到 S.EXE 中。
```
BLW SAMPLE.EXE B:
```
把 B 驱动器中的盘做成密钥盘，再把 SAMPLE.EXE 加密。
```
C:\ > BLW

BITLOK for Windows   Version 1.0
Copyright (c) 1994.6 by Yellow Rose Software Workgroup

Usage: BLW <SourceExe> <TargetExe>

C:\ > BLW WINMINE.EXE W.EXE

BITLOK for Windows   Version 1.0
Copyright (c) 1994.6 by Yellow Rose Software Workgroup

Make Key...
Making a new fingerprint... OK
Build key successful.

Encryption completed successfully.
```

PACKWIN

加密完成后，可再用 PACKWIN 压缩。压缩后再检查压缩是否改变了程序的执行性。

```
C:\ >PACKWIN W.EXE
PACKWIN DOS and Windows Executable File Compressor Version 1.0
Copyright (c) 1993,1994 Yellow Rose Workgroup. All Rights Reserved.

Compressing: W.EXE
Original: 30624 Compressed: 24000 Ratio: 21.63 %
```

BLWINST

安装的方法很简单，把加密过的程序拷贝到硬盘中，用 BLWINST 程序处理一下即可。

```
C:\ > BLWINST W.EXE
BLWINST Copyright (c) 1994.6 by Yellow Rose Software Workgroup.
OK!
```

用户可以用 BLWINST 工具把 W.EXE 安装到硬盘中，在执行 W.EXE 时不需要插入密钥盘。把 W.EXE 拷入其它机器里，必须插入密钥盘，W.EXE 才能执行。

用 BITLOK for Windows 加密后，建议再使用 PACKWIN 做压缩。可能你会发现有的程序不能被压缩：如果应用程序检查自身的完整性，加密后的程序已发生改变，执行该程序的，Windows 会提示出错；还有一类程序会把配置信息写入自身程序，由于该程序被加密，这样写入配置信息会带来不能预计的错误。

附在本书中的 BITLOK for Windows 功能比较简单，可能不能满足读者加密商品软件的要求。若需要 BITLOK 的商业版，请与作者及代理商联系。当读者看到这本书时，BITLOK for Windows 的商业版可能已经提高到 2.0 版，希望读者能够关心 BITLOK 的新动向。

第 10 章

Windows NT 及 Chicago 执行文件格式

Microsoft 设计了一种新的文件格式 Portable Executable File Format(即 PE 格式),该格式应用于所有基于 Win32 的系统:Windows NT、Win32s 及 Chicago。在不久的将来,PE 格式将在所有基于 Win32 的系统中扮演重要角色。如果你用过 Win32s 或 Windows NT,你就已经在使用 PE 文件了。本章主要介绍 PE 这种新的文件格式。

10.1 PE 简介

操作系统的可执行文件的格式从多种意义上讲是操作系统本身执行机制的反应。虽然研究可执行文件格式并非一个程序员的首要任务,但这种工作能积累大量的知识。在本章中将要介绍 PE 文件格式,它应用于所有基于 Win32 的系统:Windows NT、Win32s 及 Chicago。在不久的将来,PE 格式将在所有基于 Win32 的系统中扮演重要角色。即使你只是在 Windows 3.1 中使用 Visual C++ 编程,你实际上还是用到了 PE 文件(Visual C++ 的 MS-DOS 32 位扩展内容用到了这种格式)。总之,不久以后,PE 格式会越来越普遍,应该引起大家的重视。我们现在应该开始研究这种新的文件格式及在操作系统中的作用。

由于很多读者具有使用 16 位 Windows 的各种经验,所以本章中把 Win32 的 PE 文件格式和 16 位的 NE 文件格式对照起来讲解。

除了新的 PE 执行文件格式之外,Microsoft 还相应引入了新的目标模块格式,新的编译器和汇编程序生成这种格式。新的 OBJ 文件格式和 PE 文件格式有很多相似之处,本章在 PE 格式之外还描述了新的 OBJ 格式。

众所周知,Windows NT 继承了 VAX、VMS 及 UNIX 的某些思想,因为许多 Windows NT 的创始者在来 Micorsoft 之前都设计并编写过上述平台。当设计 Windows NT 时,他们借助了以前编写的并经过测试的工具以减少项目设计的启动时间,这也是很自然的。他们的这些工具所产生或使用的可执行文件和目标文件格式称为 COFF(Common Object File Format),COFF 格式本身是一个好的起点,但需要予以扩充以适应现代操作系统(如 Windows NT 或 Chicago)的需要,其结果便产生了 PE 格式。之所以称为"Portable",是因为 Windows NT 在各种平台(x86, MIPS、Alpha 等等)上都使用同样的执行格式。当然,在 CPU 指令的二进制译码等方面会存在着差异,但最重要的是操作系统装载器和编程工具无需对任何一种新出现的 CPU 进行重写。

Microsoft 为了将精力集中在 Windows NT 上,使之快速发展起来,放弃了现存的 32 位工具及文件格式。比如,在 Windows NT 出现之前,16 位 Windows 的虚拟设备驱动程序使

用的是32位文件格式——LE格式;更重要的是OBJ格式的改变:在Windows NT C编译器之前,所有的Microsoft编译器使用的都是Intel的OMF(Object Module Format)规范。在前面讲过,基于Win32的Microsoft编译器生成的是COFF格式的目标文件。一些Microsoft的竞争者,如Borland及Symantec继续使用Intel OMF格式,抛弃了COFF格式。其结果是OBJs和LIBs必须有针对编译器发送不同的版本。

PE格式在WINNT.H头文件中有不甚精确的描述(Borland编译器也有类似的文件,叫NTIMAGE.H)。WINNT.H中有一节叫"Image Format",该节前面的一小段给出了大家熟知的MS-DOS MZ格式和NE格式头,之后就是PE格式的新内容。WINNT.H提供了PE文件中要使用的原始数据结构,但关于结构和标记含义的注释很有限,要深入理解PE格式,这些注释还远远不够。

要条理化地理解WINNT.H中的信息,你还需要以CD-ROM形式提供的Microsoft Developer Network(MSDN)或Windows NT版的Win32 SDK中的PE.TXT文件。很奇怪的是,Microsoft没有在NT版的Visual C++ 1.0中提供PE.TXT(Visual C++是以CD-ROM形式发售的,不存在"磁盘空间不够"的借口)。

MSDN CD-ROM或Win32 SDK中的PE.TXT比WINNT.H详细得多,但是仍然难以读懂,而且留下了一些"黑洞",我们在此将这些东西补充进来。MSDN CD-ROM中还有另外一个文档——Randy Kath写的《The Portable Executable File Format from Top to Bottom》,这篇文章比PE.TXT及WINNT.H更详尽一些,但仍然忽略了某些主题,在其它地方又比较哆嗦,并且不够准确。本章目的之一就是要介绍PE格式的各个方面,而且同你的日常经验及使用联系起来。

除了了解PE文件的组成之外,读者或许还希望dump一些PE文件来验证PE结构。基于Win32的Microsoft开发工具中有一个DUMPBIN程序,它能够以一种清晰可读的形式剖析PE文件和COFF格式的OBJ/LIB文件。在所有的PE文件dump程序中,DUMPBIN是最详尽的。它还有一个很精巧的选择功能,能够反汇编从文件取出的代码片段。Borland用户能用TDUMP查看PE执行文件,但TDUMP不能识别COFF OBJ文件。但这并不要紧,因为Borland编译器并不产生COFF格式的OBJ文件。

我们可以扩展前面章节介绍过的PDUMP,使之能显示PE格式。具体的方法是编写一个新的对象TNEHeader,使之能分析PE格式的文件。读者在学习完本章之后可将PDUMP改进得更完善些。

```
{ PEFMT.PAS 分析 PE 执行文件格式 }

unit PEFMT;

interface

uses Dos, FileDef, ExtTools, Strings;

{ Windows NT EXE Struct Object ---------------------------- }

type

  Image_File_Header = Record
```

```
    Machine,
    NumberOfSections: Word;
    TimeDateStamp,
    PointerToSymbolTable,
    NumberOfSymbols: LongInt;
    SizeOfOptionalHeader,
    Characteristics: Word;
  end;

  Image_Data_Directory = Record
    VirtualAddress,
    Size: LongInt;
  end;

const

  Image_NumberOf_Directory_Entries = 16;

type

  Image_Optional_Header = Record
    { Standard fields }
    Magic: Word;
    MajorLinkerVersion,
    MinorLinkerVersion: Byte;
    SizeOfCode,
    SizeOfInitializedData,
    SizeOfUninitializedData,
    AddressOfEntryPoint,
    BaseOfCode,
    BaseOfData: LongInt;

    { NT additional fields }
    ImageBase,
    SectionAlignment,
    FileAlignment: LongInt;
    MajorOperatingSystemVersion,
    MinorOperatingSystemVersion,
    MajorImageVersion,
    MinorImageVersion,
    MajorSubsystemVersion,
    MinorSubsystemVersion: Word;
    Reserved1,
    SizeOfImage,
    SizeOfHeaders,
    CheckSum: LongInt;
    Subsystem,
    DllCharacteristics: Word;
    SizeOfStackReserve,
    SizeOfStackCommit,
    SizeOfHeapReserve,
    SizeOfHeapCommit,
```

```
    LoaderFlags,
    NumberOfRvaAndSizes: LongInt;
    DataDirectory:
      array [0..Image_NumberOf_Directory_Entries-1] of
      Image_Data_Directory;
  end;

  Image_NT_Headers = Record
    Signature: LongInt;
    FileHeader: Image_File_Header;
    OptionalHeader: Image_Optional_Header;
  end;

  Image_Section_Header = record
    Name: array [0..7] of char;
    VirtualSize,                          { OBJ PhysicalAddress }
    VirtualAddress,
    SizeOfRawData,
    PointerToRawData,
    PointerToRelocations,
    PointerToLinenumbers: LongInt;
    NumberOfRelocations,
    NumberOfLinenumbers: Word;
    Characteristics: LongInt;
  end;

  PImportDirectoryEntry = ^ImportDirectoryEntry;
  ImportDirectoryEntry = Record
    HintNameTable,
    TimeDateStamp,
    FowarderChain,
    ImportLookupTable,
    ImportAddressTable: LongInt;
  end;

  PImageResourceDirectory = ^ImageResourceDirectory;
  ImageResourceDirectory = record
    Characteristics,
    TimeDateStamp: LongInt;
    MajorVersion,
    MinorVersion,
    NumberOfNamedEntries,
    NumberOfIdEntries: Word;
  end;

  PImageResourceDirectoryEntry = ^ImageResourceDirectoryEntry;
  ImageResourceDirectoryEntry = record
    Name,
    OffsetToData: LongInt;
  end;

  PPEHeader = ^TPEHeader;
```

```
  TPEHeader = object(TMzHeader)
    peHeader: Image_NT_Headers;
    constructor Init(FName: PathStr);
    procedure DisplayPE;
  end;

implementation

{ Windows NE EXE File Struct Object ---------- }

constructor TPEHeader.Init (FName: PathStr);
begin
  inherited Init(FName);
  if rtError <> 0 then fail;
                    ●
  if (wSign <> (WinNTSign)) then begin
    rtError := $EE01;
    fail;
  end;
  {$I-}
  seek(F, ofsNe);
  blockread(F, peHeader, sizeof(peHeader));
  {$I-}
  if IOResult <> 0 then begin
    rtError := $EE02;
    fail;
  end;
end;

procedure TPeHeader.DisplayPE;
var
  I, J: Word;
  K: LongInt;
  PS: PChar;
  Objects: Image_Section_Header;
  RsrcRVA,
  RsrcPointerToRawData,
  RsrcSizeOfRawData,
  ImportRVA,
  ImportPointerToRawData,
  ImportSizeOfRawData: LongInt;
  P: PBytes;
  PImport: PBuf;
  PA: PImportDirectoryEntry;
begin
  Writeln(#10#10'Portable Executable (PE) File'#10);
  Writeln('Header base: ', lhStr(OfsNE), #10);

  with peheader.FileHeader, peHeader.OptionalHeader do begin
    Write  ('CPU type                    ');
    case Machine of
      $14d: Writeln('Intel i860');
```

```
    $14c: Writeln('80386');
    $162: Writeln('MIPS R3000');
    $166: Writeln('MIPS R4000');
   else Writeln('Unknown');
end;

Write  ('Flags                    ', whStr(Characteristics), ' [ ');
{
   0x0001  // Relocation info stripped from file.
   0x0002  // File is executable  (i.e. no unresolved externel references).
   0x0004  // Line nunbers stripped from file.
   0x0008  // Local symbols stripped from file.
   0x0010  // Reserved.
   0x0020  // Reserved.
   0x0040  // 16 bit word machine.
   0x0080  // Bytes of machine word are reversed.
   0x0100  // 32 bit word machine.
   0x0200  // Debugging info stripped from file in .DBG file
   0x0400  // Reserved.
   0x1000  // System File.
   0x2000  // File is a DLL.
   0x8000  // Bytes of machine word are reversed.
}
if Characteristics and $0002 <> 0 then Write('executable ');
if Characteristics and $0040 <> 0 then Write('16bit ');
if Characteristics and $0002 <> 0 then Write('32bit ');
if Characteristics and $2000 <> 0 then Write('DLL ');
Writeln(']');

Write  ('DLL flags                ', whStr(DllCharacteristics), ' [ ');
case DllCharacteristics of
   1: Write('Call when DLL is first loaded');
   2: Write('Call when a thread terminated');
   4: Write('Call when a thread starts up');
   8: Write('Call when DLL exits');
end;
Writeln(']');

Writeln('Linker Version           ',
   VerStr(MajorLinkerVersion, MinorLinkerVersion));

Writeln('Time stamp               ', lhStr(TimeDateStamp));

Writeln('O/S Version              ',
   VerStr(MajorOperatingSystemVersion, MinorOperatingSystemVersion));

Writeln('User Version             ',
   VerStr(MajorImageVersion, MinorImageVersion));

Writeln('Subsystem Version        ',
   VerStr(MajorSubSystemVersion, MinorSubSystemVersion));
```

```
Write  ('Subsystem                ', whStr(SubSystem), ' [ ');
case Subsystem of
  1: Write('NATIVE');
  2: Write('Windows GUI');
  3: Write('Windows character');
  5: Write('OS/2 character');
  7: Write('Posix character');
end;
Writeln(' ]');

Writeln('Optional header size     ', whStr(SizeOfOptionalHeader));
Writeln('Code size                ', lhStr(SizeOfCode));
Writeln('Init Data size           ', lhStr(SizeOfInitializedData));
Writeln('Uninit Data size         ', lhStr(SizeOfUninitializedData));
Writeln('Entry RVA                ', lhStr(AddressOfEntryPoint));
Writeln('Image base               ', lhStr(ImageBase));
Writeln('Code base                ', lhStr(BaseOfCode));
Writeln('Data base                ', lhStr(BaseOfData));
Writeln('Object/File align        ',
  lhStr(SectionAlignment),'/',lhStr(FileAlignment));
Writeln('Image size               ', lhStr(SizeofImage));
Writeln('Header size              ', lhStr(SizeOfHeaders));
Writeln('Stack reserve/commit     ',
  lhStr(SizeOfStackReserve), '/', lhStr(SizeOfStackCommit));
Writeln('Heap reserve/commit      ',
  lhStr(SizeOfHeapReserve), '/', lhStr(SizeOfHeapCommit));
Writeln('Number interesting RVAs  ', lhStr(NumberOfRvaAndSizes));
Writeln('Name                     RVA       Size');
Writeln('-------------  ------  ------');
Writeln('Exports             ',
  lhStr(DataDirectory[0].VirtualAddress),'  ',
  lhStr(DataDirectory[0].Size));
Writeln('Imports             ',
  lhStr(DataDirectory[1].VirtualAddress),'  ',
  lhStr(DataDirectory[1].Size));
Writeln('Resources           ',
  lhStr(DataDirectory[2].VirtualAddress),'  ',
  lhStr(DataDirectory[2].Size));
Writeln('Exceptions          ',
  lhStr(DataDirectory[3].VirtualAddress),'  ',
  lhStr(DataDirectory[3].Size));
Writeln('Security            ',
  lhStr(DataDirectory[4].VirtualAddress),'  ',
  lhStr(DataDirectory[4].Size));
Writeln('Fixups              ',
  lhStr(DataDirectory[5].VirtualAddress),'  ',
  lhStr(DataDirectory[5].Size));
Writeln('Debug               ',
  lhStr(DataDirectory[6].VirtualAddress),'  ',
  lhStr(DataDirectory[6].Size));
Writeln('Description         ',
  lhStr(DataDirectory[7].VirtualAddress),'  ',
```

```
    lhStr(DataDirectory[7].Size));
  Writeln('Machine Specific    ',
    lhStr(DataDirectory[8].VirtualAddress),'  ',
    lhStr(DataDirectory[8].Size));
  Writeln('TLS                 ',      { Thread Local Storage }
    lhStr(DataDirectory[9].VirtualAddress),'  ',
    lhStr(DataDirectory[9].Size));
  Writeln('Callbacks           ',
    lhStr(DataDirectory[10].VirtualAddress),'  ',
    lhStr(DataDirectory[10].Size));
  Writeln('reserved            ',
    lhStr(DataDirectory[11].VirtualAddress),'  ',
    lhStr(DataDirectory[11].Size));
  Writeln('reserved            ',
    lhStr(DataDirectory[12].VirtualAddress),'  ',
    lhStr(DataDirectory[12].Size));
  Writeln('reserved            ',
    lhStr(DataDirectory[13].VirtualAddress),'  ',
    lhStr(DataDirectory[13].Size));
  Writeln('reserved            ',
    lhStr(DataDirectory[14].VirtualAddress),'  ',
    lhStr(DataDirectory[14].Size));
  Writeln('reserved            ',
    lhStr(DataDirectory[15].VirtualAddress),'  ',
    lhStr(DataDirectory[15].Size));
  Writeln;
end;

Writeln('Object table:');
Writeln('#   Name       VirtSize    RVA      PhysSize  Phys off  Flags');
Writeln('-- ------ ------ ------ ------ ------ ------');

Seek(F, OfsNE + SizeOf(peHeader));
for I := 1 to peHeader.FileHeader.NumberOfSections do begin
  Write(bhStr(I), '  ');
  BlockRead(F, Objects, SizeOf(Objects));
  with Objects do begin
    PS := Name;
    if StrIComp(PS, '.idata') = 0 then begin
      ImportRVA := VirtualAddress;
      ImportSizeOfRawData := SizeOfRawData;
      ImportPointerToRawData := PointerToRawData;
    end
    else if StrIComp(PS, '.rsrc') = 0 then begin
      RsrcRVA := VirtualAddress;
      RsrcSizeOfRawData := SizeOfRawData;
      RsrcPointerToRawData := PointerToRawData;
    end;

    Write(PS, '':10-StrLen(PS));
    Write(lhStr(VirtualSize), '  ',
          lhStr(VirtualAddress), '  ',
```

```
            lhStr(SizeOfRawData), ' ',
            lhStr(PointerToRawData), ' ',
            lhStr(Characteristics), ' [');
        if (Characteristics and $ 00000020) <> 0 then Write('C');
        if (Characteristics and $ 00000040) <> 0 then Write('I');
        if (Characteristics and $ 00000080) <> 0 then Write('U');
        if (Characteristics and $ 20000000) <> 0 then Write('E');
        if (Characteristics and $ 40000000) <> 0 then Write('R');
        if (Characteristics and $ 80000000) <> 0 then Write('W');
        Writeln(']');
      end;
    end;

    Seek(F, ImportPointerToRawData);
    NewPBuf(PImport);
    PImport^.Len := ImportSizeOfRawData;
    GetMem(PImport^.P, PImport^.Len);
    BlockRead(F, PImport^.P^, PImport^.Len);
    P := PImport^.P;

    I := 0;
    Repeat
      PA := @(PImport^.P^[I * SizeOf(ImportDirectoryEntry)]);
      if (PA^.HintNameTable <> 0) then begin
        Writeln('Imports from ',
          PChar(@P^[PA^.ImportLookupTable-ImportRVA]));
        J := 0;
        Repeat
          K := PDword(@P^[PA^.HintNameTable + J * 4 - ImportRVA])^;
          Inc(J);
          if K <> 0 then begin
            Dec(K, ImportRVA);
            Write('    ', PChar(@P^[K+2]));
            if PWord(@P^[K])^ <> 0 then
              Write('(hint = ', DowncaseStr(whStr(PWord(@P^[K])^)), ')');
            Writeln;
          end;
        until (K = 0);
      end;
      Inc(I);
      Writeln;
    Until (PA^.HintNameTable = 0);

    DisposePBuf(PImport);

  end;

  end.
```

10.2 Win32 及 PE 的基本概念

让我们看一下渗透着 PE 设计思想的基本概念(参阅图 10.1)

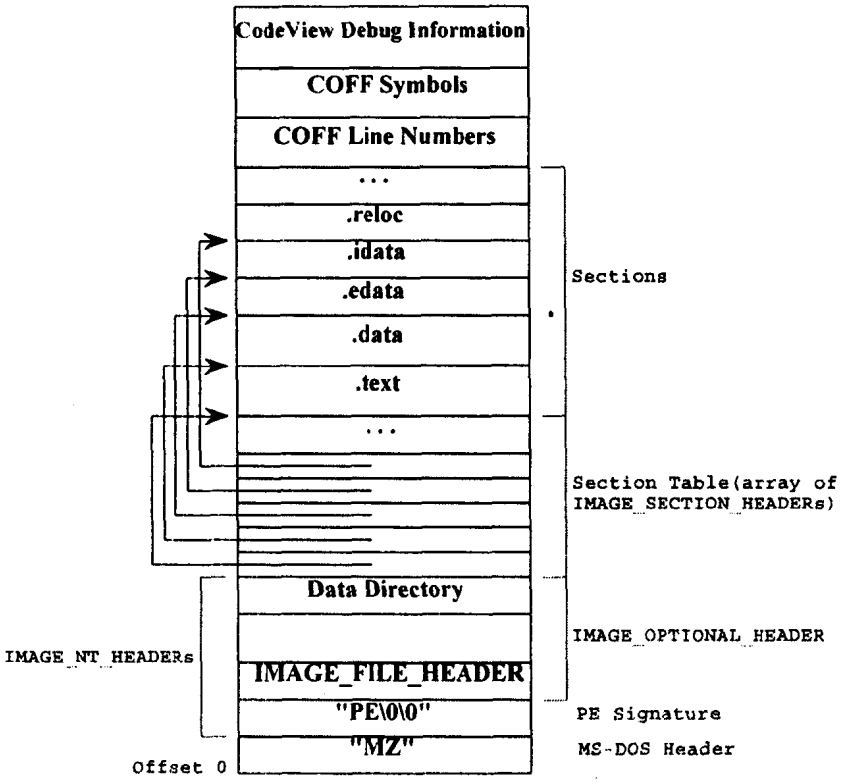

图 10.1 PE 文件格式

我们将用"Module"这一术语来表示已装入内存的执行文件或 DLL 的代码、数据及资源。除了程序中直接用到的代码和数据以外,一个 Module 也指 Windows 中用于判定代码及数据在内存中位置的支撑数据结构。在 16 位 Windows 中,支撑数据结构在 Module Database 中(HModule 指向这个段)。而在 32 位 Windows 中,这些数据结构放在 PE 首部中,我们在这里简要说明一下。

对于 PE 文件,首先应该知道,磁盘中的执行文件与其被 Windows 装载程序装入后的 Module 看起来非常相似。Windows 装载程序把磁盘文件变成实际执行代码的工作相当简单,装载程序使用文件内存映象机制将磁盘文件块映射到虚拟地址空间。打个比方,PE 文件象一个活动的房子,装载时只需将某个块放在某个地方,再把它和其它部分拧合好即可(比如说,把它和 DLL 连接在一起)。加载 PE 格式的 DLL 也是同样简单。一旦 Module 被装入,在 Windows 中就同其他已装入内存的文件一样了。

这种方式与 16 位 Windows 明显不同。16 位 NE 格式文件装载程序读取部分磁盘文件,并生成一个完全不同的数据结构,在内存中建立 Module。当代码或数据段需要装入时,装载程序必须从全局内存中分配出一块,查找原始数据在文件的什么地方,找到位置后再读

取原始数据，最后再进行一些修整。还有，每一个 16 位的 Module 要负责记住它现在使用的所有 selector、该 selector 表示的段是否已经被抛弃等等。

对于 Win32，某 Module 中的代码、数据、资源、输入表、输出表以及其它有用的数据结构等使用的内存都放在一个连续的内存块中，编程人员只要知道装载程序文件映象到内存后的地址即可。通过映象后面的各种指针可以轻易地找到 Module 中所有内容。

读者还应该熟悉 Win32 中的“相对虚拟地址”(Relative Virtual Address, RVA)这个名词。PE 格式中的许多项是以 RVA 方式指定的，RVA 就是某个项相对于文件映象地址的偏移。例如：装载程序将一个 PE 文件装入到虚拟地址空间中从 0x10000 开始的内存中，如果 PE 中某个表在映象中的起始地址是 0x10464，那么该表的 RVA 就是 0x464。将 RVA 转换为可用的指针，只要将 RVA 的值加上 Module 的基地址即可。基地址是指装入到内存中的 EXE 或 DLL 程序的开始地址，它是 Win32 中的一个重要概念。为了方便起见，Windows NT 和 Chicago 将 Module 的基地址作为 Module 的实例句柄 Instance Handle(HInstance)。在 Win32 中称基地址为 HInstance，似乎有些混淆，因为“Instance Handle”一词来源于 16 位的 Windows，在 16 位的 Windows 中每一个执行实例都有自己的数据段，依此来相互区分，这就是“Instance Handle”的来历。在 Win32 中，因为不存在共享地址空间，所以应用程序无需加以区别。当然，Win16 和 Win32 中的 HInstance 还是有些联系：在 Win32 中可以直接调用 GetModuleHandle 以取得指向 DLL 的指针，通过指针访问该 DLL Module 的内容。

读者还需要知道 PE 文件中的“section”(块)。在 PE 文件中，section 大致相当于 NE 格式中的段或者资源。块中包含代码或数据，与段不同的是，块没有大小限制，是一个连续内存块。有些块中包含程序中申明及直接使用的代码或数据，而另外一些数据块由连接器或库管理程序产生，包含对操作系统极其重要的信息。在一些 PE 格式文件的描述中，section 也称为 object，因为 object 具有太多的意义，所以我们坚持称之为 section。

10.3　PE 首部

象其他可执行文件格式一样，PE 格式文件也有一个首部信息集合，描述了文件的基本内容。这个首部包含了诸如代码和数据区的地址、长度及该文件适合何种操作系统、堆栈的初始大小以及其他一些重要信息。PE 文件首部并不在文件的最开始部分，最开始的几百个字节是 MS-DOS stub(WINSTUB)，这个极小的 DOS 程序显示“This program cannot be run in MS-DOS mode.”之类的信息，以提示用户应在 Win32 环境中运行。

PE 首部的起始地址隐含在 MS-DOS 首部之中。在 WINNT.H 文件包含了 MS-DOS stub 的结构定义，从中很容易找到 PE 头。e_lfanew 字段就是真正 PE 首部的相对偏移(或 RVA)，要得到内存中 PE 首部指针，只需用基地址加上 e_lfanew 即可。

```
pNTHeader = dosHeader + dosHeader->e_lfanew;
```

得到了 PE 首部指针后，我们接下来看一下 PE 首部结构。PE 首部在 WINNT.H 中定义为结构 IMAGE_NT_HEADERS，这个结构是由一个双字的标志和两个子结构组成：

```
DWORD Signature;
```

```
IMAGE_FILE_HEADER Fileheader;
IMAGE_OPTIONAL_HEADER OptionalHeader;
```

标志项由“PE\0\0”ASCII 文本构成。如果根据 MS-DOS 首部中的 e_lfanew 找到的标志是“NE”，而不是“PE\0\0”，这说明该文件是 NE 格式的。同样，如果为 LE 标志，即指出该程序是 Windows 3.x 的设备驱动程序，而 LX 标志是 OS/2 2.0 的执行文件。

PE 首部 PE 标志之后是一个“IMAGE_FILE_HEADER”，这个结构只包含了文件的一些基本信息，显然是由原先的 COFF 未加修改生成过来的。除了作为 PE 首部的一部分之外，它也是 Micorsoft Win32 的编译器所产生的 COFF OBJs 最前面的部分。

IMAGE_FILE_HEADER 格式如下：

```
WORD Machine
```

这个文件所适用的 CPU 类型。CPU 标志符是如下定义的：

```
0x14d Intel i486
0x14c Intel I386 (same ID used for 486 and 586)
0x162 MIPS R3000
0x166 MIPS R4000
0x183 DEC Alpha AXP
```

```
WORD Numberofsections
```

文件中块的个数

```
DWORD TimeDataStamp
```

连接器(或 OBJ 文件编译器)生成文件的时间，这一字段保存从 1969 年 12 月 31 日下午 4:00 以来的秒数。

```
DWORD PointerToSymbolTable
```

COFF 调试符号表的偏移地址。这一字段只用于带 COFF 格式 debug 信息的 OBJ 文件和 PE 文件。PE 文件支持多种 debug 格式，所以 debug 程序应该引用 Data Directory(后面将定义)中的 IMAGE_DIRECTORY_ENTRY_DEBUG。

```
DWORD NumberofSymbols
```

COFF 符号表中符号个数。

```
WORD SizeofOptionalHeader
```

在 IMAGE_FILE_HEADER 结构之后的可选首部的大小。在 OBJs 中，该字段为 0。在执行文件中，是指 IMAGE_OPTIONAL_HEADER 结构的长度。

```
WORD Characteristics
```

文件的信息标志，一些重要字段为：

0x0001 文件中没有重定位
0x0002 是一个可执行文件(区别于 OBJ 和 LIB)
0x2000 文件是一个动态连接库(不是一个程序)

其它字段在 WINNT.H 中有定义。

PE 首部的第三部分为“IMAGE_OPTIONAL_HEADER”结构。对于 PE 文件而言，这一部分不能称为可选项(Optional)。COFF 格式允许在标准的 IMAGE_FILE_HEADER 之外定义附加的信息结构，IMAGE_OPTIONAL_HEADER 的内容就是 PE 设计者对上述信息的说明。

IMAGE_OPTIONAL_HEADER 中的大部分字段都不重要。应该注意其中的 Image-

Base 和 Subsystem 这两项，读者可以跳过对其它字段的描述。IMAGE_OPTIONAL_HEADER 的格式如下：

IMAGE_OPTIONAL_HEADER 格式

WORD Magic
好象是标志字。总是被设为 0x010B。

BYTE MajorLinkerVersion
BYTE MinorLinkerVersion
生成这个文件的连接器版本号。数字以十进制数显示，而不是十六进制。比如 2.23。

DWORD SizeofCode
所有 Code Section 总共的大小(只入不舍)。通常情况下多数文件只有一个 Code Section，所以这个字段和.text section 的大小匹配。

DWORD SizeofInitializedData
估计是指由已初始化的数据构成的块的总的大小(不包括代码段)。但是它似乎与文件中实际情况不一致。

DWORD SizeofUninitializedData
块的大小，装载程序要在虚拟地址空间中为这些块约定空间。这些块在磁盘文件中不占用空间，如同"UninitializedData"这一术语，这些块在程序开始运行时没有指定值。未初始化数据通常是在.bss 块中。

DWORD AddressofEntryPointer
装载程序开始执行的地址。这是一个 RVA(相对虚拟地址)，通常能够在.text 块中找到。

DWORD BaseofCode
文件 Code Section 开始的 RVA。在内存中，Code Section 通常是在 PE 首部之后、Data Section 之前。在 Microsoft 连接器生成的执行文件中，RVA 通常是 0x1000；Borland 的 TLINK32 是将 Image Base 加上第一个 Code Section 的 RVA，并将结果存入该字段。

DWORD BaseofData
文件 Data Section 开始的 RVA。Data Section 通常是在内存中的末尾，即 PE 首部和 Code Section 之后。

DWORD ImageBase
当连接器生成一个可执行文件时，假定文件是被直接映象到指定的内存地址中。把这个地址就存放在本字段中，以允许连接器优化装载地址。如果文件真的被装入到这个地址，在运行之前代码就不需要做变动了。在 Windows NT 生成的可执行文件中，缺省的 Image Base 为 0x10000；对于 DLLs，缺省值为 0x400000。在 Chicago 中，0x10000 不能用来装入 32 位可执行文件，因为该地址处于所有进程共享的线性地址区域，因此 Microsoft 将 Win32 可执行文件的缺省基地址改变为 0x400000。连接时假定为 0x10000 的老程序在 Chicago 中装入时，用的时间要长一些，因为装载程序要重定位。

DWORD SectionAlignment
当装入内存时，每一个块的起始虚拟地址都要保证是本字段的倍数。为了分页的目的，这个缺省的块对齐值是 0x1000。

DWORD FileAlignment
在 PE 文件中，组成块的原始数据必须保证从本字段的倍数地址开始。缺省值是 0x200 字节，这也许是为了保证块总是从磁盘的扇区头开始(也是 0x200 字节)。这个字段的功能等价于 NE 格式中的段/资源对齐因子。PE 格式不象 NE 格式中的段那样有数以百计的块，所以因对齐而浪费的空间很小。

WORD MajorOperatingSystemVersion
WORD MinorOperatingSystemVersion
使用这一个可执行文件所需要的操作系统最低版本，该字段似乎含义不明确和不必要，因为下面一些字段有相同的意义。

WORD MajorImageVersion

WORD MinorImageVersion

这是用户自定义字段,定义可执行文件或 DLLs 的不同版本。通过连接器的/VERSION 参数来设置。

例如:"LINK/VERSION2.0 myobj.obj"。

WORD MajorSubsystemVersion

WORD MinorSubsystemVersion

包含了运行这一可执行文件所需要的子系统最低版本,一般为 3.10(即 Windows NT 3.1)。

DWORD reserved

似乎总为 0。

DWORD SizeofImage

似乎是装载程序应该保证的装入文件各部分的总长度。它是指装入文件从 Image Base 到最后一个块的大小。最后一个块根据对齐大小往上取整。

DWORD SizeofHeaders

PE 首部及 Section Table 的大小。首部之后就是块的原始数据。

DWORD CheckSum

好象是文件的 CRC 校验和。在 Microsoft 的其它格式中,该字段被忽视并置为 0。为了提高可靠性才需要设置文件的有效校验和。

WORD Subsystem

该可执行文件用户界面使用的子系统种类。WINNT.H 中定义了如下值:

NATIVE	1	不需要子系统(如设备驱动程序等)
WINDOWS_GUI	2	运行于 Windows GUI 子系统
WINDOWS_CUI	3	运行于 Windows 字符子系统
OS2_CUI	5	运行于 OS/2 字符(只限于 OS/2 1.x 应用系统)子系统
POSIX_CUI	7	运行于 POSIX 字符子系统

WORD DLLCharacteristics

一组标志位,指出了 DLL 的初始函数(例如 DLLMain)被调用的环境。这个值好象总被设为 0,操作系统总是在以下四种情况下调用初始函数:

1 当 DLL 首次被装入一个进程的地址空间时调用
2 当一个线程终止时调用
4 当一个线程建立时调用
8 当 DLL 退出时调用

DWORD SizeofStackReserve

为线程的初始堆栈保留的内存大小。这一值并非一定要设定,其缺省值为 0x100000(1MB)字节(参见下一字段)。如果你 CreateThread 时设定栈的大小为 0,仍然会生成一个 1M 大小的栈。

DWORD SizeofStackCommit

初始线程栈的内存大小。Microsoft Linker 的这个字段缺省值为 0x1000(一页),当 TLINK32 的缺省值是二页。

DWORD SizeofHeapReserve

保存初始进程队列的虚拟内存数量。这个队列的句柄能够在 GetProcessHeap 调用中找到。这一值并非一定要设定(参见下一字段)。

DWORD SizeofHeapCommit

在进程队列中,内存大小初始设定,缺省值为一页。

DWORD LoaderFlags

从 WINNT.H 的说明来看,好象是与调试支撑数据相关的一个字段。笔者从来没有看到一个可执行文件中设置它们,也不清楚如何让连接器设置它们。该字段的定义如下:

1 进程开始之前,调用一个断点指令

2　进程装入之后，调用一个调试器

DWORD NumberofRvaAndSizes

数据目录(Data Directory)中数组的项数。流行的工具软件总是设为 16。

IMAGE_DATA_DIRECTORY

DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]

初始数组项目包括起始 RVA 和可执行文件的重要部分的长度。数组末尾的一些项基本上是没有用的。数组的第一项总是 Exported Function Table 的地址及长度(如果有的话)，第二个项是 Imported Function Table 的地址及长度，……。在 WINNT.H 中有 IMAGE_DIRECTORY_ENTRY_XXX 数组项的完整清单。有了这个数组，装载程序能够在文件中快速找到某一个块(如 Imported Function Table)，而不需要来回查找比较。多数数组项描述了整个块的数据，但 IMAGE_DIRECTORY_ENTRY_DEBUG 项只讲了一小部分关于.rdata 块。

在讲述完 PE 首部的各个部分之后，我们用上一节中改造的 PDUMP 来显示 Windows NT 中的 TASKMAN.EXE 首部信息。

```
PDUMP DOS and Windows Executable File Dumper Version 1.0 Shareware
Copyright (c) 1993, 1994 Yellow Rose Workgroup. All Rights Reserved.

                    Display of File TASKMAN.EXE

Old Executable Header

DOS File Size                                   6974h  ( 26996. )
Load Image Size                                 450h   (  1104. )
Relocation Table entry count                    0000h  (     0. )
Relocation Table address                        0040h  (    64. )
Size of header record       (in paragraphs)     0004h  (     4. )
Minimum Memory Requirement (in paragraphs)      0000h  (     0. )
Maximum Memory Requirement (in paragraphs)      FFFFh  ( 65535. )
File load checksum                              0000h  (     0. )
Overlay Number                                  0000h  (     0. )

Initial Stack Segment  (SS:SP)                  0000:00B8
Program Entry Point    (CS:IP)                  0000:0000

Portable Executable (PE) File
Header base: 00000080
CPU type                80386
Flags                   838E [ executable 32bit ]
DLL flags               0000 [ ]
Linker Version          2.39
Time stamp              2C376A3D
O/S Version             1.00
User Version            0.00
Subsystem Version       3.10
Subsystem               0002 [ Windows GUI ]
Optional header size    00E0
Code size               00002600
Init Data size          00003C00
Uninit Data size        00001000
Entry RVA               00001750
Image base              02210000
```

```
Code base                 00001000
Data base                 00004000
Object/File align         00001000/00000200
Image size                0000D000
Header size               00000400
Stack reserve/commit      00100000/00001000
Heap reserve/commit       00100000/00001000
Number interesting RVAs   00000010
```

10.4 块 表

在 PE 首部与原始数据之间存在一个块表，块表包含每个块在映象中的信息。块在映象中是按起始地址(为 RVA)来排列的，而不是按字母表顺序。

NE 文件中的代码及数据存储的在明确的“段”中，在 NE 文件首部有一个“段表”数组结构，存储了程序中段的信息，每一项对应一个段。这些信息包括段的类型(代码或数据)、大小及地址等。在 PE 文件中，块表就类似于 NE 格式中的段表，但又与段表有区别，块表不存储代码及数据的 selector，而是存储原始数据装入到内存之后的地址，虽然块可以说成是 32 位的段，但却不是分离的段，它们是进程在虚拟地址空间中的一段存储范围。

PE 文件与 NE 文件另外一个不同之处在于它们管理支撑数据的方式有区别。支撑数据由操作系统使用，程序中用不着。例如：PE 和 NE 管理执行文件中要用到的 DLL 列表或各种表的位置的方式有差异。在 NE 文件中，虽然资源也有 selector，但它们不被当作段。资源的信息不是存放在段表中，而是在 NE 首部的资源表中。还有，NE 首部有输入输出名表，但没有给出输入输出函数具体是在哪个段中。PE 文件的情况就不同了，任何重要的代码或数据都存储在一个完备的块中。输入函数、输出函数及重定位数据都存储到相应的块中。无论是程序中要用到的还是系统要用到的数据都存储在相关的块中。

在讨论特定块之前，需要说明操作系统管理这些块所需要的数据。在内存中，紧接 PE 首部的是一个“IMAGE_SECTION_HEADER”数组，该数组的大小已经在 PE 首部中给出。我们可以用前面介绍的 PDUMP 输出块表和块的各字段、属性等信息，下面是一个典型的执行文件的块表。

```
Name                      RVA       Size
------------              ------    ------
Exports                   00000000  00000000
Imports                   0000A000  0000099E
Resources                 00007000  000011E0
Exceptions                00000000  00000000
Security                  00000000  00000000
Fixups                    0000B000  000002A0
Debug                     00005010  00000054
Description               00000000  00000000
Machine Specific          00000000  00000000
TLS                       00000000  00000000
Callbacks                 00000000  00000000
```

```
reserved            00000000  00000000
reserved            00000000  00000000
reserved            00000000  00000000
reserved            00000000  00000000
reserved            00000000  00000000

Object table:
#   Name      VirtSize  RVA       PhysSize  Phys off  Flags
--  --------  --------  --------  --------  --------  --------
01  .text     00000000  00001000  00002600  00000400  60000020 [CER]
02  .bss      00000000  00004000  00001000  00000000  C0000080 [URW]
03  .rdata    00000000  00005000  00000200  00002A00  40000040 [IR]
04  .data     00000000  00006000  00000800  00002C00  C0000040 [IRW]
05  .rsrc     00000000  00007000  00001200  00003400  40000040 [IR]
06  .CRT      00000000  00009000  00000200  00004600  C0000040 [IRW]
07  .idata    00000000  0000A000  00000A00  00004800  40000040 [IR]
08  .reloc    00000000  0000B000  00001400  00005200  42000040 [IR]
```

下表给出了 IMAGE_SECTION_HEADER 的格式。有趣的是，PE 格式的块信息中没有那些我们在 NE 结构中很熟悉的信息，比如预装载属性(PRELOAD)。NE 文件允许指定段在模块装入时的预装载属性。OS/2 2.0 LX 格式和 NE 格式有些类似，允许指定最多 8 页的预装载，PE 格式中没有这回事。Microsoft 对 Win32 中的请求式分页装载(Demand-paged Loading)的性能充满信心。

另外，PE 格式中也没有页表说明。在 OS/2 的 LX 格式中与 IMAGE_SECTION_HEADER 相当的部分，没有直接指出怎样在文件找到块的数据或代码，但它有一个页查找表，该表确定了块中页的属性和地址。PE 文件没有这些，并保证一个块的数据是连续地存放在文件中。比较这两种格式，LX 的方法可能更灵活一些，但 PE 的风格更是出奇的简捷。

PE 格式的一个重大改变是，任何偏移地址都是直接给出的。在 NE 格式中，几乎所有的地址都是以类似磁盘中扇区值的形式表示，需要先乘以对齐因子；极个别的情况是以相对于 NE 首部的偏移值给出。因为 NE 首部并不在文件的最开始，所以需要先找到首部的偏移。总之，PE 格式比 NE、LX、LE 等格式都简单得多。

下面是 IMAGE_SECTION_HEADER 格式。

```
BYTE Name[IMAGE_SIZEOF_SHORT_NAME]
```

这是一个 8 位 ANSI 名(不是 UNICODE 内码)，用来定义块名。多数块名以一个"."开始(如.text)。尽管许多 PE 文档都这样认为，实际上这并不是要求的。要给块命名，在汇编语言中可以直接用段的名字，在 Microsoft C 或 C++ 编译器中可用"#pragma data_seg"或"#pragma code_seg"。值得注意的是如果块名超过了 8 个字节，则没有最后的终止标志"NULL"字节。

```
UNION {
  DWORD  PhysicalAddress
  DWORD  VirtualSize
  } MISC;
```

这个字段在 EXE 和 OBJ 文件中有不同的意思。在可执行文件中，它保存的是代码或数据的真实长度，这个长度是对齐以前的长度，所以称为真实长度。后面的 SizeofRawData 字段是调整后的值。Borland 连接器将这一字段中 VirtualSize 和后面的 SizeofRawData 的意思颠倒过来，似乎更准确一些。对于 OBJ 文

件,这一字段指出了块的物理地址。第一个块的开始地址为 0,要找到 OBJ 中下一块的物理地址,要加上当前块的 SizeofRawData。

DWORD VirtualAddress

在可执行文件中,这一字段包含装入该块的 RVA 地址。要计算出一个给定块在内存中的起始地址,应将该块的 VirtualAddress 加上文件映象的基地址。在 Microsoft 工具中,第一个块的缺省 RVA 为 0x1000。在 OBJ 中,这个字段没有意义,并被设为 0。

DWORD SizeofRawData

在可执行文件中,该字段包含经过 FileAlignment 调整后的块的长度。例如,指定 FileAlignment 大小为 0x200,如果 VirtualSize 中块的长度为 0x35A 字节,这一字段应保存的长度为 0x400 字节。在 OBJ 中,这一字段包含由编译器或汇编程序生成的精确长度,换句话讲,该字段等于可执行文件中 VirtualSize 字段。

DWORD PointertoRawData

程序经编译或汇编后生成原始数据,这个字段用于给出原始数据在文件中的偏移。如果程序自装载 PE 或 COFF 文件(而不是由操作系统装入),这一字段比 VirtualAddress 还重要。在这种状态下,必须完全使用线性映象方法装入文件,所以需要在该偏移处找到块的数据,而不是 VirtualAddress 字段中的 RVA 地址。

DWORD PointertoRelocations

在 OBJ 中,这是块的重定位信息在文件中的偏移。OBJ 中块的重定位信息就放在块的原始数据之后。在可执行文件中,这个字段无意义,被设置为 0。当连接器生成可执行文件时,已经将许多块都连接起来了,只剩下重定位的基地址和输入函数(Imported Function)需要在装入时决定。重定位的基地址和输入函数的信息都存储在各自的块中,所以生成可执行文件时不需要在块的原始数据之后加上每一个块的重定位数据。

DWORD PointertoLinenumbers

行号表在文件中的偏移。行号表将源文件中的行号与该行生成的代码地址联系起来。现代的 debug 格式中,如 CodeView 格式,行号信息是作为 debug 信息的一部分存储的。在 COFF debug 格式中,行号信息与符号名字/类型信息分开存储。通常只有代码块(如.text)具有行号信息。在可执行文件中,行号集中起来放于文件末尾,即块的原始数据之后。在 OBJ 中文件中,行号表就放在该块的原始数据和重定位表之后。

WORD NumberofRelocations

该块中重定位表中的重定位数目。这个字段似乎只与 OBJ 文件有关。

WORD NumberofLinenumbers

该块行号表的数目。

DWORD Characteristics

多数程序员称之为标志,COFF/PE 格式称之为 characteristics。该字段是一组指出块属性(如代码/数据、可读、可写等等)的标志。完整的块属性参见 WINNT.H 中的 IMAGE_SCN_XXX_XXX 定义。比较重要的标志如下:

0x00000020 该块包含代码,通常与可执行标志(0x80000000)同时设置。

0x00000040 该块包含已初始化的数据。除了可执行块及.bss 块之外,所有块都设置该标志。

0x00000080 该块包含未初始化数据(如.bss 块)。

0x00000200 该块包含注解及其他一些类型信息。比较典型的是编译器产生的.drecve块包含连接器命令。

0x00000800 该块的内容不能放入最终的可执行文件中。这类块用于编译器或汇编程序向连接器传递信息。

0x02000000 该块可被丢弃,因为它一旦被装入之后,进程就不再需要它。常见的可丢弃块为基地址重定位(.reloc)。

0x10000000 该块为共享块。如果在 DLL 中设置这类共享块,则表示其中的数据可被所有使用该 DLL 的进程共享。数据块的缺省设置为不能共享,即每一个进程使用 DLL 时,都要有单独的数据拷贝。

更专业一点来说，共享块通知内存管理器为该块设置页面映象，以使所有进程使用 DLL 时均指向内存中的同一个物理页。为实现块的共享，在连接的时候使用 SHARED 属性，例如：

```
LINK /SECTION:MYDATA,RWS ...
```

通知连接器称为"MYDATA"的块是可读、可写、共享的。

0x20000000　该块可执行。通常当 0x00000020 处的标志被设置时，也设置该标志。

0x40000000　该块可读。可执行文件中的块总是设置该标志。

0x80000000　该块可写。如果可执行文件中的块没有设置该标志，装载程序就会将内存映象页标记为只读或只执行。.data 和 .bss 设置该标志。有趣的是，.idata 也设置这种属性。

10.5　各种块的描述

了解了块之后，我们来看一下在可执行文件或目标文件中的经常遇到的各种常见块。

10.5.1　.TEXT

.text 是在编译或汇编结束时产生的一种块。因为 PE 文件是运行在 32 位方式下，不受 16 位段的约束，所示不必要将不同文件产生的代码再分成分离的块。连接器将所有目标文件的 .text 块连接成一个大的 .text 块。如果使用 Borland C++，其编译器将其产生的代码存于名为 CODE 的区域中，Borland C++ 产生的 PE 文件中有一个名为 CODE 的块，不是称为 .text，对此将在下面进行解释。

有趣的是，.text 中有并非编译器产生的或运行时间库函数使用的代码。在 PE 文件中，当调用其他模块中的某一功能时（比如 USER32.DLL 中的 GetMessage），编译出的 CALL 指令不能直接调用 DLL 中的功能，而是转换成这样一种控制指令：

JMP DWORD PTR [XXXXXXXX]　（参见图 10.2）

这一指令也存在于 .text 中。JMP 指令通过双字变量间接进入 .idata 中，这个 .idata 中的双字包含了操作系统功能入口点的真正地址。仔细考虑一下，就可明白了为什么要用这种方式实现 DLL 调用：将针对某一给定的 DLL 的所有 DLL 调用集中在一个地方，装载程序就不必逐一修订其它的调用指令了——PE 装载程序要做的只是将目标功能的正确地址放入 .idata 中，不必修补任何调用指令。这与 NE 文件中的方法明显不同，在 NE 文件中，每一个段中都包含一个重定位表，如果在该段中调用一个 DLL 功能 20 次，装载程序就必须将这一功能的地址在该段中重复写 20 次。PE 中这种方法的缺陷是不能把一个变量的地址初始化为 DLL 功能的真正地址。例如：你也许想用下面的方式

```
FarPROC pfnGetMessage = GetMessage;
```

把功能 GetMessage 的地址写入变量 pfnGetMessage 中。在 16 位 Windows 中是可以的，但在 32 位 Windows 中就不行了。如果在 Win32 中这样赋值，变量 pfnGetMessage 的值最后变成微码（thunk）"JMP DWORD PTR[xxxxxxxx]"的地址，这一点从图 10.2 中可以看出。如果希望通过指向功能的指针来调用是可以的，但如果要读取功能 GetMessage 开始处的字块就不行了（除非自己在 .idata 中指针的后面再作一些处理）。在后面的 .import 中还会讲这个问题。

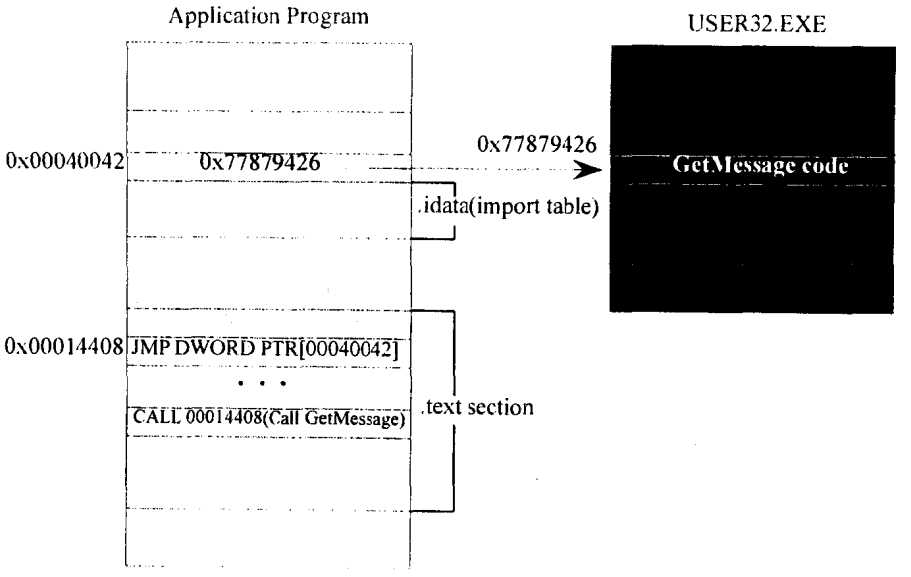

图 10.2 调用其他模块中的函数

虽然 Borland 能够使编译器产生名为 .text 的块，但它的默认名为 CODE。为了在 PE 文件中给某个块命名，Borland 链接器取目标文件名中的前 8 个字符。

块名不同无关紧要，重要的是 Borland PE 文件如何链接到其他模块中去。如在 .text 所描述的，所有对 OBJ 的调用都生成一个 JMP DWORD PTR[xxxxxxxx]微码(thunk)。在 Microsoft 系统中，这一微码通过输入库的 .text 进入可执行文件。因为当链接外部 DLL 时，库管理程序(LIB32)生成输入库(以及微码)，链接程序无需知道到底该如何生成微码。输入库实际上就是链接到 PE 文件的另外一些代码及数据。

Borland 系统处理输入函数的方法其实只是扩展了 16 位 Windows 中的方法。Borland 链接器所使用的输入库实际上只是 DLL 中的功能的列表。TLINK32 负责决定哪一个重定位点是外部 DLL，并且生成一个类似 JMP DWORD PTR[xxxxxxxx]的微码，并将微码存储在 .icode 中。

10.5.2 .DATA

如同 .text 是默认的代码块一样，.data 中是初始化数据。这些数据包括编译时被初始化的 global 和 static 变量，也包括文字串。链接器将 OBJ 及 LIB 文件中的 .data 结合成一个大的 .data。local 变量放在一个线程的堆栈中，不占用 .data 或 .dss 的空间。

10.5.3 .BSS

.bss 存放未初始化的 static 和 global 变量。链接器将 OBJ 及 LIB 文件中的 .bss 块结合成一个大的 .bss。在块表中，.bss 的原始数据偏移字段被设为 0，指示这个块在文件中不占空间。TLINK 不产生这个块，而是扩展了 .data 的虚拟长度。

10.5.4　.CRT

.CRT 是另外一个初始化数据块，是 Microsoft C/C++ 运行时间库产生的。为什么不将它放入 .data 块中，我们不得而知。

10.5.5　.RSRC

.rsrc 包含模块的全部资源。在 Windows NT 的早期，16 位的 RC.EXE 产生的资源文件不能形成一种 Microsoft PE 链接器可以识别的格式。CVTRES 可将资源文件转换成 COFF 格式的目标文件，并将资源数据放入 OBJ 目标文件中的 .rsrc 块。这样，链接器能够将资源 OBJ 当作普通的 OBJ 链接进来，不用知道它的具体内容。最新的一些 Microsoft 链接器可以直接处理资源文件了。

10.5.6　.IDATA

.idata 包含其他外来 DLL 的函数及数据信息。该块的功能与 NE 文件的模块引用表类似，关键的差异在于 PE 文件中的每一个输入函数都明确地列于该块中，要在 NE 格式中找到相同的信息，必须从各个段的重定位数据中查找。

10.5.7　.EDATA

.edata 是该 PE 文件输出函数和数据的列表，以供其它模块引用。它与 NE 文件中的入口表、驻留名表及非驻留名表的综合功能等价。与 16 位的 Windows 不同，PE 执行文件没有理由输出一个函数，所以通常只是在 DLL 文件中才见得到 .edata 块。当使用 Microsoft 工具时，.edata 中的数据通过 EXP 文件进入 PE 文件。换一种说法，也就是链接器就不能自己生成这些信息，它必须依赖库函数管理程序(LIB32)搜索 OBJ 文件并生成 EXP 文件，以便将要链接的模块加入列表中。那些烦人的 EXP 文件只不过是 OBJ 文件的一个扩展罢了。

10.5.8　.RELOC

.reloc 保存基地址重定位表。当装载程序不能按链接器所指定的地址装载文件时，需要对指令或已初始化的变量值进行调整，基地址重定位表包含调整所需要的数据。如果装载程序能够正常装载文件，它就忽略 .reloc 中的重定位数据。如果希望装载程序总能够按链接器假定的地址装入，可以加上/FIXED 选择项，告诉链接器跳过这些信息。虽然这种方法可以节省执行文件空间，但可能导致该执行文件在其他 Win32 系统中不可执行。例如：假设构造了一个 Windows NT 下的可执行程序，其基地址是 0x10000，如果你通知链接器去掉重定位，该执行文件将不能在 Chicago 系统下运行，因为 Chicago 中地址 0x10000 已被其它程序使用。

应该注意到，编译器生成的 JMP 和 CALL 指令使用的是相对相对于该指令的偏移量，而不是 32 位段中的真实偏移。如果要把文件装入到某个并非链接器假定的地址，这些指令不需要做改变，因为他们使用的是相对地址。这样，也就没有所想象的那么多重定位信息。只有那些使用 32 位偏移地址数据的指令才需要重定位。举例来说，假设定义了如下的全程

变量:

```
int i;
int * ptr = &i;
```

假设链接器假定的基地址是 0x10000,变量 i 的地址为 0x12004,那么链接器会在内存中"* ptr"处写入 0x12004。如果由于某种原因,装载程序将该文件从基地址 0x70000 处装入,那么 i 的地址就变成 0x72004。.reloc 中存放链接器假定的装入地址和实际的装入地址之间的差值。

10.5.9 .TLS

使用编译器伪指令__ declspec(thread),则用户定义的数据被放入了.tls 块中,而不是.data或.bss 中。TLS 的意思是"thread local storage"(线程局部存储器),它与 Win32 中的 TlsAlloc 系列功能有关。当处理一个.tls 块时,内存管理程序建立了一个页面表,一旦要在进程中切换线程时,一组新的物理内存页面就映象到这个.tls 的地址空间,这样,每一个线程都有自己的全程变量。有了.tls,就不用先给每一个线程基分配内存,然后把内存指针存放在用 TlsAlloc 分配的地址中,因为多数情况下,这种处理.tls 的机制更简单。

关于.tls 块和__ declspec(thread)变量,还有一点需要补充说明。在 Windows NT 和 Chicago 中,如果 DLL 是由 LoadLibrary 动态加载的,则上述 TLS 机制无效。对执行文件或隐式装载的 DLL,上述机制工作正常。如果既不能隐式装载 DLL,又要使各个线程基有自己数据,那就只有回到使用 TlsAlloc 和 TlsGetValue 动态分配内存的老路上去。

10.5.10 .RDATA

.rdata 块通常是在.data 或.bss 中间,但程序中很少用到该块中的数据。至少在如下两种情况下要用到.rdata,其一是在 Microsoft 链接器产生的 EXE 中,用于存放调试目录,只有执行文件中有调试目录(在 TLINK32 产生的 EXE 中,调试目录在.debug 块中)。调试目录是一个 IMAGE_DEBUG_DIRECTORY 结构数组,该结构存储文件中各种 debug 信息的类型、地址及长度等等。debug 信息类型主要有:CodeView,COFF 及 FPO。

比如,一个典型的 Debug Directory 有如下信息:

Type	Size	Address	FilePtr	Charactr	TimeData	Version
COFF	000065C5	000000000	00009200	0000000	2CF8CF30	0.00
???	00000114	000000000	0000F7C8	0000000	2CF8CF30	0.00
FPO	000004B0	000000000	0000F8DC	0000000	2CF8CF30	0.00
CODEVIEW	0000B0B4	000000000	0000FD8C	0000000	2CF8CF30	0.00

调试目录不是在.rdata 块的开始部分。要找到调试目录表,需使用 Data Directory 第 7 项(IMAGE_DIRECTORY_ENTRY_DEBUG)中存放的 RVA,Data Directory 在 PE 首部的末尾。要判定 Microsoft 链接器生成的 Debug Directory 的项数,需要将调试目录的大小(在调试项的大小字段中找到)除以结构 IMAGE_DEBUG_DIRECTORY 的大小。TLINK32 只输出一个简单的计数,通常为 1。

.rdata 块中另一个有用部分为说明字符串。如果在程序的 DEF 文件中指定一个 DE-

SCRIPTOIN 项，则字符串就会出现在 .rdata 中。在 NE 格式中，说明字符串总是非驻留名表的第一项。说明字符串用于描述文件，但笔者没有找到发现它的简单方法，因为有的 PE 文件的说明字符串在调试目录之后，而有的则在调试目录之前。或许根本就不存在某种方法。

10.5.11　.DEBUG $ S 和 .DEBUG $ T

.debug $ S 及 .debug $ T 只出现在 OBJ 文件中，它们存储 CodeView 的符号及类型信息。它们的名称来源于 16 位编译器产生的调试段名 $ $ SYMBOL 和 $ $ TYPES。.debug $ T 唯一的用途是存储 PDB 文件的路径名，PDB 文件包含了所有 OBJ 的 CodeView 信息，链接器读入 PDB 文件并用它组成部分 CodeView 信息，最后将这些信息放在 PE 文件的末尾。

10.5.12　.DRECTIVE

.drective 只出现在 OBJ 文件中，它包含链接器命令行的文本描述。例如：用 Microsoft 编译器编译出的 OBJ 的 .drective 块中总会出现以下信息：

```
-defaultlib:LIBC -defaultlib:OLDNAMES
```

在代码中使用 __ declspec(export)时，编译器将等价的命令行输出到 .drective 中（例如：-export: MyFunction）。

还有两点应该记住。首先，不要被编译器和汇编程序产生的标准块约束住了。如果需要使用其它特殊的块，可以毫不犹豫地自己生成。在 C/C++ 编译器中可用 #pragma code_seg 及 #pragma data_seg；在汇编程序中，创建一个与标准块名放在不同的段即可。如果用 TLINK32，必须使用不同的类，或者关闭代码段压缩（code segment packing）。另外一点是，不同寻常的块名肯定有更深层次的意义和目的。

10.6　PE 文件的 IMPORT

前面已描述过，在调用外部 DLL 时，CALL 指令实际上被转化成 EXE 文件 .text 块中的 JMP DWORD PTR[xxxxxxxx]指令（如果是使用 Borland C++，则在 .icode 块中），JMP 指令要跳转到的地方才是真正的目标地址。装载程序判定目标函数的地址并将该函数插补到执行映象中，所需要的信息都放在 PE 文件 .idata 块中。

.idata 块（习惯称之为 import 表）以一个 IMAGE_IMPORT_DESCRIPTOR 数组开始。每一个被 PE 文件隐式链接进来的 DLL 都有一个 IMAGE_IMPORT_DESCRIPTOR。在这个数组中，没有字段指出该结构数组的项数，但它的最后一个单元是 NULL，可以由此计算该数组的项数。IMAGE_IMPORT_DESCRIPTOR 的格式如下：

```
DWORD   Characteristics
```

该字段是一个指针数组的偏移（RVA），其中每一个指针都指向一个 IMAGE_IMPORT_BY_NAME 结构。

```
DWORD   TimeDataStamp
```

文件建立的时间及日期标记。

DWORD ForwarderChain

正向链接索引。比如说,在 Windows NT 中,NTDLL.DLL 中的某个功能调用的是 KERNEL32.DLL 中的功能。如果应用程序调用 NTDLL.DLL 中的这个功能,最后实际调用的是 KERNEL32.DLL。这种方式称之为正向链接。本字段包含 FirstThunk 数组的索引。由该字段索引的函数将实际上是在另一个 DLL 中。遗憾的是,文档上没有函数被正向链接的格式,也很难找到这样的例子。

DWORD Name

以 NULL 结尾的 ASCII 字符串的 RVA,该字串包含输入的 DLL 名,比如"KERNEL32.DLL"与"USER32.DLL"。

PIMAGE_THUNK_DATA FirstThunk

该字段是在 IMAGE_THUNK_DATA 联合结构中的偏移(RVA)。大多数情况下,IMAGE_THUNK_DATA 是指向 IMAGE_IMPORT_BY_NAME 结构的指针。如果不是一个指针的话,那它就是该功能在 DLL 中的序号。是否真的如 NE 结构那样,能通过序号成功地调用输入函数,文档中没有明确说明。

IMAGE_IMPORT_DESCRIPTOR 中较重要部分有输入的 DLL 名字及两个 IMAGE_IMPORT_BY_NAME 指针数组。在执行文件中,这两个指针数组(由 Characteristics 和 FirstThunk 指出)彼此平行,末尾的 NULL 指针表示数组结束。两个数组中的指针都指向 IMAGE_IMPORT_BY_NAME 结构。图 10.3 给出了这种关系的图形描述。

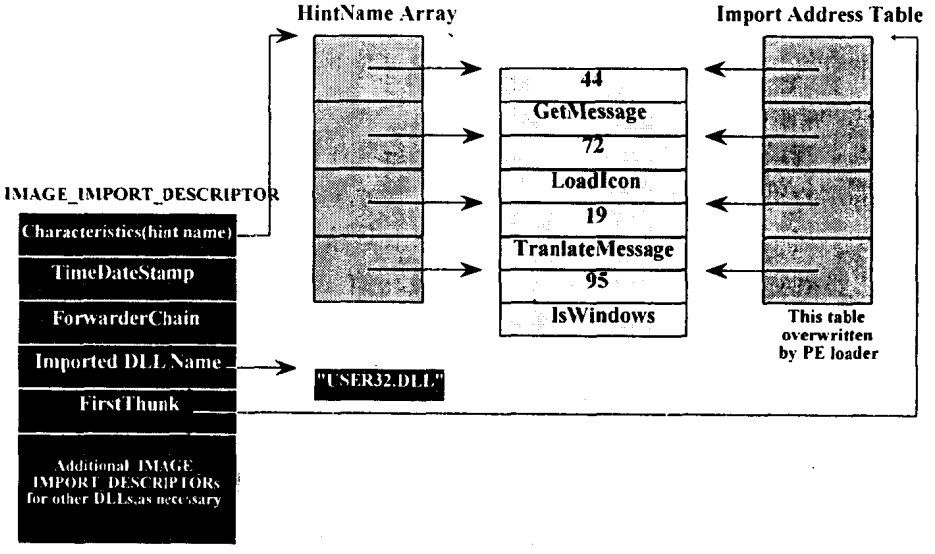

图 10.3 两个并行的指针数组

PDUMP 分析 TASKMAN.EXE 输入表时的显示结果:

```
Imports from ADVAPI32.dll
    OpenProcessToken(hint = 009f)
    RegCloseKey(hint = c00b1)
    RegCreateKeyExW(hint = 00b6)
    RegQueryValueExW(hint = 00cf)

Imports from KERNEL32.dll
    CloseHandle(hint = 0016)
    CreateThread(hint = 0039)
    ExitProcess(hint = 0053)
```

```
ExitThread(hint = 0054)
...
```

PE 文件调用的每一个输入函数都有一个 IMAGE - IMPORT - BY - NAME 结构。IMAGE - IMPORT - BY - NAME 很简单，结构类似于：

```
word  Hint;
byte  Name[?];
```

Hint 被猜测是被调用的输入函数的输出序号。与 NE 格式文件不同，这一值不一定要是准确的，它只是用来帮助 NE 装载程序进行输出函数的二分检索(Binary Search)时确定初始值。Name[]是输入函数名的 ASCIIZ 字符串。

为什么有两个并行的指针数组指向 IMAGE - IMPORT - BY - NAME 结构呢？第一个数组(Characteristics 所指)是单独一项，而且不可改写，它有时称为提示名表(Hint-nameTable)。第二个数组(FirstThunk 所指)是由 PE 装载程序重写的。装载程序迭代搜索数组中的每一个指针，找到每一个 IMAGE - IMPORT - BY - NAME 结构所指的输入函数的地址，然后装载程序用找到的地址改写 IMAGE - IMPORT - BY - NAME 指针。JMP DWORD PTR [xxxxxxxx]中的[xxxxxxxx]是指 FirstThunk 数组中的一个入口。因为 FirstThunk 所指的数组由装载程序改写后，实际上装入的是所有输入函数的地址，所以它被称为输入地址表(Import Address Table)。

对于 Borland 用户，则与上述叙述有一些细微的差别。由 TLINK32 产生的 PE 文件缺少一个数组，在可执行文件中，IMAGE - IMPORT - BY - NAME 中的 Characteristics 字段是 0，也就是说，在 PE 文件中，只有 FirstThunk 所指的数组(输入地址表)。有一个与此相关的十分有趣的问题：在不断的优化过程中，Microsoft 优化了 Windows NT 的 DLL 系统中的 thunk 数组，使该数组中的指针不是指向 IMAGE - IMPORT - BY - NAME 结构，而是包含输入函数的地址。换句话讲，装载程序不再需要寻找输入函数的地址然后将它改写到 thunk 数组中。这就给 PE 的转储(dump)处理带来了一个问题，因为转储程序需要该数组包含指向 IMAGE - IMPORT - BY - NAME 结构的指针。你也许想："没关系，为什么不使用提示名表(Hint-name Table)呢"？这只是一个臆想的方案，因为 Hint-name Table 不存在于 Borland 文件中。当然，转储程序还是可以处理这些情况的，但处理过程复杂得多。

因为输入地址表是可写的，所以截听 EXE 或 DLL 对另一个 DLL 的调用就相对容易些，只需将输入地址表项修改成指向希望调用的截取功能即可，不用修改任何调用和被调用模块的代码。很容易吧？

值得一提的是，在 Microsoft 产生的 PE 文件中，输入表并不是完全由链接器综合生成的。调用一个外来 DLL 功能所需的所有信息都存于输入库中。链接一个 DLL 时，库管理程序搜索将要链接的 OBJ 文件，并生成一个输入库，这种输入库完全与 16 位 NE 文件中所使用的完全不同：32 位 LIB 产生的输入库有一个.text 块和几个.idata $ 块。输入库的.text 包含 JMP DWORD PTR [XXXXXXXX]指令，它的名字存储在 OBJ 的符号表(symbol table)中，表中 symbol 的名字与该 DLL 中要输出函数的名字是相同的(例如：_Dispatch Message@4)。输入库中有一个.idata $ 包含微码(thunk)指令间接引用的 DWORD 值，另外一个.idata 块中含有输入函数名序号的空间，其后是输入函数名，这两个字段就构成了前面所说的 IMAGE - IMPORT - BY - NAME 结构。此后链接一个需要使用输入库的 PE 文件时，输入库

的各个块就被加入到 OBJ 的块列表中。因为输入库的 thunk 微码(指 JMP 指令)中含有与要输入函数相同的名字,链接器就把该微码看作是真正的输入函数,并把针对该输入函数的调用修改成指向该微码的指针。输入库中的 thunk 微码就可以被看作真正的输入函数。

除了生成输入函数微码的代码部分之外,输入库还提供 PE 文件.idata 块(输入表)的部分内容。这些内容来自于各种.data $ 块,它们是由库管理程序放入的。总而言之,链接器不能分辨输入函数和 OBJ 中出现的功能,链接器只是根据他们的预置规则来建立或合成块。

10.7 PE 文件的 EXPORT

与输入一个函数相对应的就是为 EXE 或 DLL 提供输出函数。PE 文件将输出函数的信息存储在.edata 块中。通常,Microsoft 链接器生成的可执行文件是不被调用的,所以没有.edata 块;Borland TLINK32 生成的执行文件中至少输出一个符号。多数 DLL 文件提供的输出存放在.edata 块。一个.edata 块的主要内容是函数名表、入口点地址、输函数序号值。在 NE 文件中,与其对应的功能是出口表、驻留名表及非驻留名表,这些表存储在 NE 头中,而不是作为一个单独的段或资源存放。

.edata 块的开始部分是 IMAGE_EXPORT_DIRECTORY 结构,该结构内容如下:

DWORD Characteristics
该字段好象总是被设为 0。

DWORD TimeDateStamp
文件生成时,建立时间日期标志。

WORD MajorVersion
WORD MinorVersion
该字段好象总是设置为 0。

DWORD Name
该 DLL 的名字 ASCIIZ 字串的 RVA。

DWORD Base
输出函数序号的开始值。例如,如果文件输出序号为 10、11、12,则该字段的值为 10。为了得到函数的输出序号,需要将该字段的值加入到 AddressOfNameOrdinals 数组的相应项中。

DWORD NumberofFunctions
数组 AddressofFunctions 的项数,该值也就是这一模块中输出函数的数目,从理论上讲,它与 NumberofNames 可以不同,但实际上总是一致的。

DWORD NumberofNames
数组 AddressofName 的项数,这一变量似乎与 NumberofFunctions 字段总是一致的,即输出函数的数目。

PDWORD * AddressofFunctions
这是一个 RVA,指向函数地址数组。函数地址是本模块中每一个输出函数的入口点(RVA)。

PDWORD * AddressofNames
这是一个 RVA,指向字符串指针,字符串是本模块中每一个输出函数的名字。

PWORD * AddressofNameOrdinals
这是一个 RVA,指向 WORD 类型数组。该 WORD 类型数组是本模块中所有输出函数的输出序号。不要忘

记加入 Base 字段中的起始序号。

在 IMAGE_EXPORT_DIRECTORY 结构之后就是该结构指针所指的数据。输出表的结构有些奇特，如图 10.4 所示。

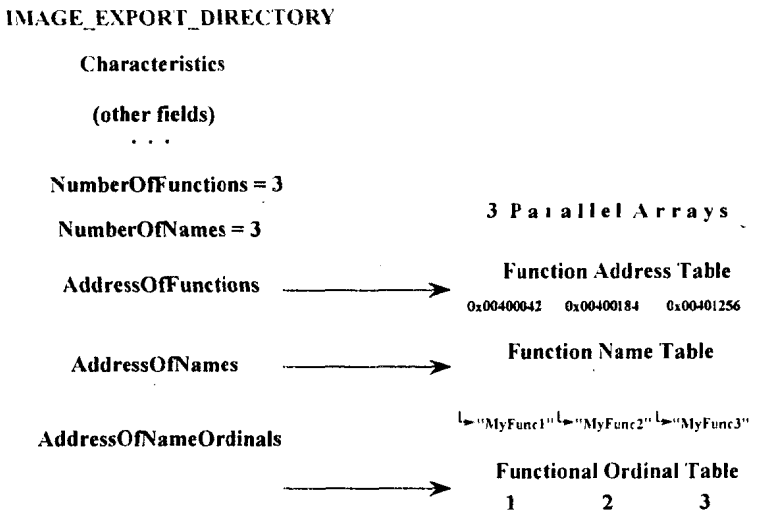

图 10.4　输出表的层次图

下面是一个典型的 EXE 文件的输出表。

```
    Name:              KERNEL32.dll
Characteristics:       00000000
TimeDataStamp:         2C4857D3
Version:               0.00
Ordinal Base:          00000001
# of Functions:        0000021F
# of Names:            0000021F

Entry Pt  Ordn    Name
00005090     1  AddAtomA
00005100     2  AddAtomW
00025540     3  AddConsoleAliasA
00025500     4  AddConsoleAliasW
00026AC0     5  AllocConsole
00001000     6  BackupRead
00001E90     7  BackupSeek
00002100     8  BackupWrite
0002520C     9  BaseAttachCompleteThunk
00024C50    10  BaseDebugDump
// 其它省略...
```

如刚才所讲，输出函数需要的是名字、地址及输出序号。你一定认为 PE 格式的设计者可能会将这三项放入一个结构中，然后给出一个这种结构的数组。事实并非如此，而是每一个输出项都作为数组，共有三个这种数组（AddressofFunctions、AddressofNames、

AddressofNameOrdinals)，而且他们彼此平行，互不影响。如果需要找第 n 个函数的信息，就需要在每一个数组的第 n 项中进行寻找。

顺便说一下，如果你要转储(dump)Windows NT 系统 DLL(如 KERNEL32.DLL 和 USER32.DLL)，多数情况下，你可能会注意到两个函数，它们只是在名的末尾有一个字符的区别。如 CreateWindowsExA 与 CreateWindowsExW。这是为了实现 UNICODE 支持。如果函数以 A 结束，则是 ASCII(或 ANSI)的兼容函数；如果是以 W 结尾，则为 UNICODE 版本的函数。在源程序中无需指定要调用哪一个函数，在 Windows NT 的 Windows.h 中，通过执行 #ifdefs 预处理语句就可以选择相应的函数。请看下面的例子：

```
#ifdef UNICODE
  #define DefWindowProc DefWindowProcW       // UNICODE
#else
  #define DefWindowProc DefWindowProcA      //! UNICODE
#endif
```

10.8 PE 文件资源

在 PE 文件中寻找资源比起在 NE 文件中要复杂一些。各种资源的格式没有多大的变化，但是需要绕过一层又一层结构才能找到它们。

PE 文件中的资源层次目录如同硬盘存放数据的方式。有一个主目录(Master Directory 或 Root Directory)，主目录下面又有其他子目录，子目录下面的子目录才可能指向原始资源，比如对话模板等。在 PE 格式中，无论是根目录还是子目录，所有层次资源的格式都是一个 IMAGE_RESOURCE_DIRECTORY 类型结构，其格式如下：

```
DWORD  Characteristics
```
理论上讲，该字段存储资源标志，但好象经常被置为 0。

```
DWORD  TimeDateStamp
```
资源生成时的时间和日期。

```
WORD  MajorVersion
WORD  MinorVersion
```
理论上讲，该字段存储了资源的版本号，但经常置为 0。

```
WORD  NumberofNamedEntries
```
本结构后面的数组中以名字代表资源的项数。

```
WORD  NumberofIDEntries
```
本结构后面的数组中以 ID 号代表资源的项数。

```
IMAGE_RESOURCE_DIRECTORY_ENTRY  DirectoryEntries[]
```
该字段并非 IMAGE_RESOURCE_DIRECTORY 结构的一部分，它是一个 IMAGE_RESOURCE_DIRECTORY_ENTRY 结构的数组，数组中项数是 NumberofNamedEntries 及 NumbersofIDEntries 字段的总和。在这个数组中，以名字指定的资源项在前面。

一个目录项既能指向一个子目录(即另一个 IMAGE_RESOURCE_DIRECTORY 结构)，又能指向资源的原始数据。一般说来，在进入真正的原始资源之前，至少有三层目录，

其中只有第一层目录能在.rsrc 块的前端找到,子目录数据资源都要在文件中才能找到。例如包含一个对话、字符串表及菜单的 PE 文件,就必须有三个子目录:一个对话框目录、一个字符串表目录及一个菜单目录。每一个这种子目录都依次包含 ID 子目录。对于一种给定的资源类型实例,都有一个 ID 子目录,在每一个 ID 目录中,或者有一个字符串名(如"MyDialog"),或者有一个用来在 RC 文件中指定资源的 ID 数。

下面是 Windows NT 中 CLOCK.EXE 的资源层次目录的例子。

```
ResDir (0) Named:00 ID:06 TimeDate:2C3601DB Vers:0.00 Char:0
  ResDir (ICON) Named:00 ID:02 TimeDate:2C3601DB Vers:0.00 Char:0
    ResDir(1) Named:00 ID:01 TimeDate:2C3601DB Vers:0.00 Char:0
      ID:00000409    offset:00000200
    ResDir(2) Named:00 ID:01 TimeDate:2C3601DB Vers:0.00 Char:0
      ID:00000409    offset:00000210
  ResDir(MENU) Named:02 ID:00 TimeDate:2C3601DB  Vers:0.00 Char:0
    ResDir (clock) Named:00 ID:01 TimeDate:2C3601DB Vers:0.00 Char:0
      ID:00000409    offset:00000220
    ResDir (Genericmenu) Named:00  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
      ID:00000409    offset:00000230
  ResDir(DIALOG) Named:01  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
    ResDir(aboutbox) Named:00  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
      ID:00000409    offset:00000240
    ResDir(64) Named:00  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
      ID:00000409    offset:00000250
  ResDir(STRING) Named:00  ID:03  TimeDate:2C3601DB  Vers:0.00 Char:0
    ResDir(1) Named:00  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
      ID:00000409    offset:00000260
    ResDir(2) Named:00  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
      ID:00000409    offset:00000270
    ResDir(3) Named:00  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
      ID:00000409    offset:00000280
  ResDir(GROUP_ICON) Named:00  ID:02  TimeDate:2C3601DB  Vers:0.00 Char:0
    ResDir(CCKK) Named:00  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
      ID:00000409    offset:00000290
  ResDir(VERSION) Named:00  ID:02  TimeDate:2C3601DB  Vers:0.00 Char:0
    ResDir(1) Named:00  ID:01  TimeDate:2C3601DB  Vers:0.00 Char:0
      ID:00000409    offset:000002A0
```

就如以前提到的:每一个目录项都是一个 IMAGE_RESOURCE_DIRECTORY_ENTRY 结构,它格式如下。(IMAGE_RESOURCE_DIRECTORY_ENTRY 格式)

DWORD　Name

这个字段要么包含资源的 ID,要么包含指向名串结构的指针。如果高位是 0,这个字段为 ID;如果高位非 0,则其余 31 位表示 IMAGE_RESOURCE_DIR_STRING_U 结构的偏移地址(相对于资源的开始位置)。该结构包含一个 WORD 的字符计数,紧跟着是一个 UNICODE 字串(资源名)。要注意,即使对非 UNICODE 的 Win32 执行方式,这里使用的还是 UNICODE 方式。要将 UNICODE 字串转换为 ANSI 字串,使用 WideCharToMultiByte 函数。

DWORD　OffsetToData

该字段或者是另一个资源结构的偏移,或者是一个指向特定资源实例的指针。如果高位为 1,该目录项指向其子目录,低 31 位表示另一个 Image_Resource_Directory 的偏移;否则低 31 位为一个 IMAGE_RESOURCE_DATA_ENTRY 结构的指针,该结构存储着资源的原始数据、大小、代码页等的位置。

要全面了解资源格式,需要查阅相关的资料。Win32 SDK 中的 RESFMT.TXT 文件对各种资源类型格式有详尽的介绍,如果感兴趣可以参考,我们在这里就忽略不讲,因为内容太庞杂。

10.9 PE 文件的 Base Relocations

当链接器生成一个可执行文件时,它就假定好了文件应装入内存的什么地方。所以,链接器会将代码及数据的真实地址放入 EXE 中。比如由于某种原因,可执行文件被装入到了另外的一个地址空间,则链接器装入的地址就错了,此时,存储在.reloc 块中的信息可以帮助装载程序修正原来的地址。如果装载程序能够将文件装入到链接器指定的基本地址,则.reloc中的数据就没用了。因为.reloc 块中数据的使用与被装入映象的基地址相关,所以称其内容为 Base Relocations。

与 NE 文件格式的重定位不同,在 PE 格式中,Base Relocations 是相当简单的,在内存的地址目录中加入一个变量就可以了。Base Relocations 数据格式有些奇特:Base Relocations 各项被放入到一串长度不定的小块(chunk)中,每一个块都描述了一个 4KB 页面的重定位。我们来看一下 Base Relocations 是怎样工作的:链接一个可执行文件,基地址被假定为 0x10000,在偏移 0x2134 处有一个包含某个串地址的指针,这个串的物理地址为 0x14002,也即该偏移处的值为 0x14002。当载入文件时,如果装载程序将文件的基地址定为物理地址 0x60000,链接器假定的装入地址与实际装入地址之差称为 delta,在本例中,delta 是 0x50000,因为整个文件都上移了 0x50000 字节,该字串也不例外(新地址为 0x64002),这样指向字串的指针现在就不对了。为了解决 Base Relocations 的问题,装载程序将 delta 值加入到原来的 Base Relocations 地址上,在本例中,装载程序将 0x50000 加上原来的指针变量 0x14002,并将字串的真实地址 0x64002 存回,这样一切都正常了。

每一块基地址重定位数据都以一个 IMAGE_BASE_RELOCATION 结构开始,该结构格式如下。

```
DWORD   VirtualAddress
```

该字段包含了重定位数据块开始的 RVA,这个数值加入到其余的重定位数据的偏移中,形成重定位所需要的真正 RVA。

```
DWORD   SizeofBlock
```

该字段的值为本结构的长度加上其后所有重定位的长度。要确定这一块数据中重定位的数目,首先从 SizeofBlock 中减去 IMAGE_BASE_RELOCATION 的长度(8 字节),然后除以 2。例如:SizeofBlock 的值为 44,则重定位的长度为 18。

```
(44-Sizeof(IMAGE_BASE_RELOCATION))/Sizeof(WORD) = 18
```

```
WORD TypeOffset
```

这不只是一个 WORD,而是一个 WORD 数组,保存了按照上面的公式计算出的数目。每个字的低 12 位是一个重定位偏移,需要加上重定位块首中 Virtual Address 字段的值;字的高 4 位是一个重定位类型。运行在 Intel CPU 上的 PE 文件只有两种重定位类型:

```
0 IMAGE_REL_BASED_ABSOLUTE
```

这种重定位没有意义,只是用于调整重定位数据块为双字的倍数

```
3 IMAGE_REL_BASED_HIGHLOW
```

这种重定位意味着将 delta 的高、低 16 位加入到由 RVA 计算出的 DWORD 中

下面是一个 EXE 文件中的 Base Relocations。由分析结果可以看出，在 IMAGE_BASE_RELOCATION 字段中 RVA 的值已经被虚拟地址所代替。

```
Virtual Address:  00001000   Size: 0000012C
  00001032   HIGHLOW
  0000106D   HIGHLOW
  000010AF   HIGHLOW
  000010C5   HIGHLOW
  // ...
Virtual Address:  00002000   Size: 0000009C
  000020A6   HIGHLOW
  00002110   HIGHLOW
  00002136   HIGHLOW
  00002156   HIGHLOW
  // ...

Virtual Address:  00003000   Size: 00000114
  0000300A   HIGHLOW
  0000301E   HIGHLOW
  00003038   HIGHLOW
  0000306A   HIGHLOW
  // ...
```

10.10　PE 和 COFF 目标文件的区别

PE 文件中有两个部分操作系统没有使用，这就是 COFF 符号表和 COFF 调试信息。为什么有那么多完备的 CodeView 信息可用，人们还需要 COFF 调试信息呢？如果你想用 Windows NT 系统的 debugger(NTSD)或 kernel debugger(KD)，那么只能和 COFF 打交道。

在前面的讨论中我们已经多处提到，COFF OBJ 与由其生成的 PE EXE 的许多结构和表是相同的。在文件的头或靠近头的地方都有一个 IMAGE_FILE_HEADER，随后就是一个包含了文件中所有块的信息的块表(section table)。OBJ 和 EXE 有相同的 line number 和符号表格式，尽管 PE 文件也能有非 COFF 格式的符号表。

这两种文件格式的相似绝非偶然：这种设计格式的目的就是为了使链接工作尽可能容易。从理论上说，由一个 COFF OBJ 文件生成一个 PE EXE 只需插入一些表或修改一些偏移即可。如果这样想，你就可以认为一个 COFF 文件就是一个准 PE 文件，只是有某些欠缺或有些差异。将这些内容列出来：

☐ COFF 格式在 IMAGE_FILE_HEADER 之前没有 MS-DOS 的 STUB 及“PE”标志。

☐ OBJ 文件没有 IMAGE_OPTIONAL_HEADER。在 PE 文件中，它是紧随在 IMAGE_OPTIONAL_HEADER 之后的。有趣的是，COFF LIB 文件中又确实有 IMAGE_OPTIONAL_HEADER。由于篇幅所限，不再叙述 LIB 文件了。

☐ OBJ 文件没有 Base Relocations，但是有通常的基于符号的修订项。COFF 目标文件的重定位格式太费解了，如果读者想专门研究一下，在块表中的 PointerToRelocations 和 NumberOfRelocations 指出了各块中的重定位情况。重定位数据是个 IMAGE_RELOCATION 结构数组，该结构在 WIN.H 中有定义。

□ OBJ 中的 CodeView 信息存储在两个块中(.debug $ S 和 debug $ T)。当链接器处理 OBJ 文件时,它并不是将这两个块放在 PE 文件中,而是将它们集中起来,构成一个单独的符号表放于文件末尾。这个符号表不是一个正式的块,在 PE 块表中没有它的入口地址。

10.11 总 结

随着 Win32 的产生,Microsoft 在 OBJ 和可执行文件的格式上进行了很大的改进,这些文件格式的主要目标就是增强在不同的平台上的可移植性。

本章主要参考 Matt Pietrek 的《Peering Inside the PE: A Tour of Win32 Portable Executable File Format》,Microsoft Systems Journal, March 1994。

参 考 文 献

1.《深入 DOS 编程》,求伯君主编,北京大学出版社,1993 年 1 月

2.《数据压缩经典》,钱国祥等编,学苑出版社,1994 年 8 月

3.《Microsoft Windows 3.1 程序员参考手册》,清华大学出版社,1993 年 6 月

4. Andrew Schulman, Dave Maxey&Patt Pietrek. *Undocumented Windows*, Addison－Wesley, 1992

5. Charles Petzold, *Programming Windows* 3.1. Redmond, Wash.: Microsoft Press, 1992

6. Mark Nelson. The *Data Compression Book*, Prentice Hall, Englewood Cliffs NJ, 1991

附

金山公司计算机系列丛书目录

《WPS NT 循序渐进》

学苑出版社 1994 年 6 月出版，全书 45 万字。介绍方正 Super Ⅵ型汉卡及 WPS NT 1.0 的全部功能及使用方法。方正 Super Ⅵ型汉卡及全新的 NT 系列软件较之以前的版本有较大的提高和改进，本书在详细介绍功能及用法的过程中，侧重于新增加的功能和用法。以一篇具体文章为例，从输入开始到以各种风格面貌输出为止，深入浅出、循序渐进，是学习 WPS NT 的一本好的工具书。

《深入 DOS 编程》

北京大学出版社 1993 年 1 月出版，全书 50 万字。本书全面、深入地剖析了 MS-DOS5.0 的功能调用，包括未公开的功能调用，分析了各种 DOS 版本的差异，介绍了基于 DOS 的 EMS、XMS、DPMI 等先进技术以及新版 MOUSE 的强劲功能，给出了各种编程语言调用 DOS 功能的方法和丰富实例；书后还附有程序员经常要查阅的各种表格。

《深入 Windows 编程》

清华大学出版社出版，全书 60 万字。本书是《打开 Windows 这扇窗》系列丛书之一，是所有想深入了解 Windows 系统的读者首选的工具书。本书主要讲述 Windows 核心运行机制及通过 Windows 加密软件和压缩软件编程的技巧与方法。以 Windows 的可执行文件格式为基本点进行全方位展开，介绍了众多的 Windows 分析工具并给出了完整的源代码，以清晰的逻辑关系展现了复杂的 Windows 运行机制——Windows 程序自装载技术，以及如何直接修改 Windows 执行文件、如何用汇编语言编写 Windows 应用程序等。本书还解决了编写通用 Windows 加密软件的技术难题，并对 Windows NT 和 Chicago 的可执行文件格式及分析工具做了介绍。本术带软盘发行，将大量精心编写的源程序及不可多得的工具奉献给读者。

本书是金山公司 Windows 多年开发经验的积累，不可不看。

《WPS 轻松学习》

中国环境科学出版社 1993 年 7 月出版，全书 48 万字。本书介绍方正 Super Ⅴ型汉卡及 WPS 3.0F 的全部功能及使用方法，以帮助用户在最短时间内掌握 WPS 的操作方法为宗旨，内容广泛，涉及到微机基础知识、指法练习、各种常用输入法等等。本书是 WPS 最优秀的通俗教材。

《SPDOS NT 1.0 编程指南》

科学出版社 1993 年 12 月出版，全书 30 万字。本书详细介绍 SPDOS 汉字系统功能，提供汇编、C++、PASCAL 语言的调用例程。本书是一本程序员手册，为有经验的程序员编

写的，旨在帮助程序员更好地理解和使用 SPDOS 提供的新颖使用功能。

《WPS 使用经验与技巧》

清华大学出版社 1994 年 3 月出版。本书收集七十多篇有关 WPS 应用的文章，都出自近几年内在计算机行业有影响的报刊。内容涉及到 WPS 密码破译、WPS 系统的安装和运行、提高 WPS 的显示速度、打印机的使用和打印输出质量的提高、图文混排系统 SPT 的使用方法、WPS 系统与其他系统之间的转换与调用等方面以及应用技巧若干例。内容丰富，指导性强。

《WPS 工具箱》

学苑出版社 1994 年 8 月出版，全书 14 万字。本书提供了很多 WPS 及金山汉卡上使用的工具软件，有 WPS 文件格式转换、其它汉字系统使用金山汉卡、软件版 WPS 的改进、WPS 文件口令解除、WPS 文件修复、SPDOS 下的数据更新等，对 WPS 及金山汉卡的用户有很大帮助。

《数据压缩经典》

学苑出版社 1994 年 9 月出版，全书 65 万字。本书系统地介绍了桌面计算机上采用的各种压缩技术，描述典型压缩算法的原理及算法，并给出所有算法的 C 语言实现细节。全书共分五篇十二章。从数据压缩的基本概念和发展简史讲起，讲解了三种类型的压缩技术，并详细介绍了文件归档。本书还配一张软盘，包含书中讲到的所有程序和源代码，读者可以参考或直接引用。

《新编深入 DOS 编程》

本书由学苑出版社 1994 年 6 月出版，全书 68 万字，是在 1993 年 1 月北京大学出版社出版的《深入 DOS 编程》的基础上，根据最新版本 MS-DOS 的发展，增加了硬件环境资料，如奔腾芯片的识别、VESA 标准的使用、实模式与保护模式切换、读取 IDE 硬盘参数等。是任何 DOS 版本下的编程人员不可缺少的工具书。