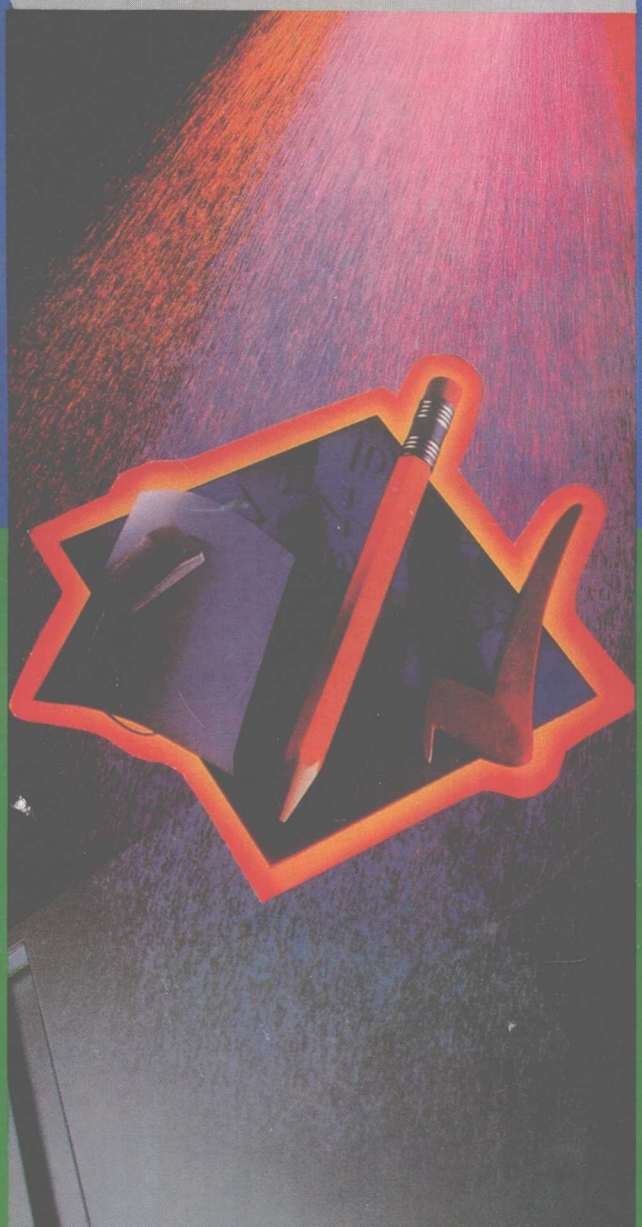
微机操作系统系列丛书(二)

周天爵　高传善　编

Windows 库程序设计和应用



学苑出版社

微机操作系统系列丛书(二)

Windows
库程序设计和应用

周天爵　高传善　编

学苑出版社

(京)新登字 151 号

内容摘要

本书瞄准广大的 Windows 程序员和计算机工作者，使其掌握更良好的方法和技巧，并有效地设计 Windows 的应用。你将从书中发现大量关键知识和背景知识，使你更深层地了解内部结构，更熟练地掌握 Windows 应用技巧。

需要本书的读者，请直接与北京 8721 信箱联系，邮编：100080，电话：2562329

微机操作系统系列丛书(二)

Windows 库程序设计和应用

编　　写：周天爵　高传善

审　　校：希望

责任编辑：甄国宪

出版发行：学苑出版社　　邮政编码：100036

社　　址：北京市海淀区万寿路西街 11 号

印　　刷：常熟市教育印刷二厂

开　　本：787×1092　1/16

印　　张：21.5　　字数：505 千字

印　　数：1－5000 册

版　　次：1994 年 1 月第 1 版第 1 次

ISBN 7－5077－0885－3/TP・27

本册定价：34.00 元

前 言

最近几年以来，我国的计算机应用得到了迅速的发展。计算机进入各行各业和千家万户，产生了巨大的社会和经济效益。如今，掌握一些计算机知识有助于事业的成功已成为人们的共识。

Windows 软件是在 DOS 操作系统基础上发展起来的一种多任务图形操作环境，它突破了 DOS 对内存空间的限制，又可同时运行多重程序，并可直观形象地使用文字或图形来进行各种处理和操作。即使用户对计算机了解甚少，也可使用 Windows 来完成各种各样的文字处理、表格处理、图形设计和数据管理等任务。目前国际上将 Windows 作为微机主流操作环境，深受广大用户的关注和欢迎。

为设计和应用良好的 Windows 系统，新的面向对象和基于库程序的设计方法对于先进的设计应用是十分必要的。使用 Windows 的大多数应用程序，都应建立在库程序基础之上，运用面向对象和动态连接方法，将大大优化 Windows 的应用和控制，成为广大用户方便有效、直观形象的对话手段和环境工具。

本书瞄准广大的 Windows 程序员和计算机工作者，使你掌握更良好的方法和技巧，去有效地设计 Windows 的应用。你将从书中发现所描述的大量关键知识和背景知识，使你更深层地了解内部结构，你会更理解你的应用和设计，更熟练地掌握 Windows 应用技巧。

本书涉及的主要程序设计环境是 MS－Windows V3.1 和 MS－C V6.0 语言。

本书在编写过程中，得到复旦大学计算机系王琪、林源等同志帮助和多位教授指教，在此特表感谢。由于时间仓促，编写中难免有误，敬请广大读者指正。

编 者

1994 年 3 月

目　录

第一章　Windows 硬件基础 …… (1)
1.1　处理机控制模式 …… (2)
1.1.1　控制模式 …… (2)
1.1.2　虚机器 …… (3)
1.1.3　寄存器和栈 …… (3)
1.1.4　中断机制 …… (5)
1.1.5　保护环层 …… (6)
1.1.6　多任务机制 …… (8)
1.2　存贮器存取模式 …… (8)
1.2.1　存取模式 …… (8)
1.2.2　内存管理器 …… (10)
1.2.3　虚拟存贮 …… (11)
1.2.4　段寄存器 …… (13)
1.2.5　保护机制 …… (13)
1.3　Windows 操作模式 …… (17)
1.3.1　操作模式 …… (17)
1.3.2　模式配置 …… (19)
1.3.3　扩展机制 …… (22)

第二章　Windows 程序基础 …… (24)
2.1　库程序模式 …… (24)
2.1.1　程序模块 …… (24)
2.1.2　库用程序 …… (25)
2.1.3　库程序 …… (27)
2.1.4　程序库结构 …… (28)
2.2　消息通讯处理 …… (30)
2.2.1　消息处理 …… (30)
2.2.2　消息传送 …… (33)
2.2.3　消息通讯 …… (36)
2.2.4　消息优先级 …… (39)
2.2.5　消息分类 …… (39)
2.2.6　消息函数 …… (40)
2.3　面向对象概念 …… (42)
2.3.1　面向对象方法 …… (42)
2.3.2　面向对象特性 …… (43)
2.3.3　对象抽象和封装 …… (43)
2.3.4　对象继承和多态 …… (44)
2.4　动态连接方法 …… (45)

第三章　可装配程序模块 …… (46)
3.1　模块装配结构 …… (46)
3.1.1　代码段 …… (47)
3.1.2　数据段 …… (48)
3.1.3　资源段 …… (50)
3.1.4　任务段 …… (50)
3.1.5　程序运行库 …… (51)
3.2　模块装配过程 …… (51)
3.2.1　装配数据库 …… (52)
3.2.2　代码的装配 …… (56)
3.2.3　数据的装配 …… (61)
3.3　模块装配文件 …… (63)
3.3.1　模块定义文件 …… (63)
3.3.2　Make 文件 …… (66)
3.3.3　编译连接命令 …… (67)
3.3.4　语言约定 …… (71)

第四章　动态存贮管理 …… (73)
4.1　块存贮管理 …… (74)
4.1.1　存贮管理阶段 …… (74)
4.1.2　固定和移动 …… (75)
4.1.3　丢弃和保留 …… (78)
4.1.4　压缩和释放 …… (78)
4.2　段存贮管理 …… (79)
4.2.1　重定义和重分配 …… (79)
4.2.2　动态分配信息 …… (79)
4.2.3　动态分配函数 …… (84)
4.2.4　Stress 管理函数 …… (85)
4.2.5　锁定和解锁 …… (85)
4.3　堆存贮管理 …… (86)
4.3.1　全局堆 …… (86)
4.3.2　局部堆 …… (88)
4.3.3　多重局部堆 …… (89)
4.4　对象存贮管理 …… (91)
4.4.1　原子 …… (91)
4.4.2　资源 …… (92)
4.4.3　字符串表 …… (95)
4.4.4　对象句柄 …… (97)

第五章　动态链接库方法 …… (100)
5.1　DLL 库程序 …… (100)
5.1.1　库程序特点 …… (100)
5.1.2　库程序类型 …… (101)
5.1.3　库程序装入方法 …… (102)
5.2　进出口库程序 …… (103)
5.2.1　进口库函数 …… (103)
5.2.2　出口库函数 …… (105)
5.3　建立 DLL 库程序 …… (106)
5.3.1　多重 DLL …… (106)
5.3.2　仅资源段 DLL …… (106)
5.3.3　无数据段 DLL …… (107)
5.4　DLL 库程序例 …… (108)
5.4.1　窗口类库程序 …… (108)
5.4.2　热键类库程序 …… (110)

第六章　窗口对象类设计 …… (122)
6.1　窗口类对象 …… (122)
6.1.1　窗口和类 …… (122)
6.1.2　窗口属性 …… (127)
6.1.3　窗口状态 …… (127)
6.2　对话设计 …… (128)
6.2.1　对话框窗口 …… (128)
6.2.2　对话框数据 …… (130)
6.2.3　对话框函数 …… (133)
6.2.4　对话框消息 …… (134)
6.3　控制设计 …… (135)
6.3.1　子窗口控制 …… (135)
6.3.2　用户区控制 …… (136)
6.3.3　非用户区控制 …… (143)
6.4　接口设计 …… (145)
6.4.1　接口控制 …… (146)
6.4.2　窗口控制 …… (146)

第七章　窗口类 DLL 库程序 …… (148)
7.1　TestApp 应用程序 …… (148)
7.2　STDWIN 库程序 …… (157)
7.3　BITMAP 库程序 …… (250)
7.4　BUTTON 库程序 …… (265)
7.5　COMBOBOX 库程序 …… (292)

7.6 EDIT 库程序 …… (299)
7.7 LISTBOX 库程序 …… (303)
7.8 SPLIT 库程序 …… (317)

第一章　Windows 硬件基础

Windows 是在 DOS 基础上发展起来的一种多任务图形操作环境，它突破了 DOS 对内存空间的限制，又可同时运行多个程序，并可清晰形象地使用文字或图形来进行各种处理和操作。即使用户对计算机了解甚少，也可使用 Windows 来完成文字处理、表格处理、图形设计和数据管理等任务。目前国际上将 Windows 作为微机主流操作环境，深受广大用户的关注和欢迎。

为设计和应用良好的 Windows 程序，面向对象和基于库程序的设计方法对于先进的设计和应用是十分必要的。大多数的 Windows 应用程序，都应建立在库程序的基础之上，运用面向对象的方法，将有力地提高软件的可重用性和使用方便性。动态连接库技术就是这种十分关键的方法。

动态连接库程序设计的主要目的是为 Windows 的应用提供一个系统的函数库和资源库，运用面向对象和动态存贮管理方法，优化了 Windows 应用程序的设计，且为广大用户提供了更有效的控制和操作环境。

虽然设计良好的 Windows 应用程序要求一种不同于 DOS 编程的思考方法，但一个良好的 DOS 编程者通过同样的努力也能成为一个优秀的 Windows 编程者。因为它们采用了相同的一些技术，有时仅仅是重新编写一下应用程序。当然，它们之间也不是没有区别。首先，操作环境完全不同；而且，在开发过程中，具有许多不同的工具来帮助实现。

Windows 环境为应用程序提供了大量的服务，它们被集成于程序的开发周期之中。在 Windows 下开发应用程序，其最大的不同和困难是：Windows 采取了消息通讯和事件驱动方式，它也很少具有文档资料，本身也还有一些难点。

首先在 DOS 环境下，编程者实际上完全控制着程序的执行。在 Windows 下就有些不同了，因为程序和对象是事件驱动模式，而不是过程驱动模式，故不是每件事都可以假想和 DOS 一样。如果你对任何事都想当然地处理，你的应用程序很可能会被 中止。事件驱动方式可能是最大的困难之一，开发者再也不能通过程序的逐条执行或跟踪执行，来确信他正看着当前执行的线索。因为某些模块调用 Windows 函数，会导致不同模块的代码开始执行，不断改变着执行线索。

其次，Windows 的消息通讯系统是另一种难点。和 DOS 一样，Windows 采用了某些未公开的函数和消息，并限制开发者使用；但它同样也提供了许多特殊的函数和消息，供开发者使用，尤其是在多文档界面 MDI 和用户控制方面，成为另一种难点所在。

为了对 Windows 具有深入的理解，首先必须了解 Windows 的硬件基础和程序基础知识，然后可进一步深入地了解它动态库程序设计思想和方法，掌握这种关键技术，实现其先进的设计和应用。

目前，已存在的大量有关 Windows 的书籍和资料，为学习和了解 Windows 提供了基础，它将有利于进一步深入了解和掌握 Windows 的新进展和新方法。

Windows 硬件基础知识，将主要介绍处理机控制模式、存贮器存取模式和 Windows 操作

模式等三大方面。

1.1 处理机控制模式

1.1.1 控制模式

在 Windows 中，处理机存取信息的控制方式有二种不同的模式，即实模式和保护模式。在实模式中，处理机支持程序访问的地址引用物理的实际内存地址。处理机运行在实模式中，只限于常规内存部分或高内存区 HMA；而在保护模式中，处理机对内存的控制由一系列处理机保护的描述符(段)表来管理。保护模式允许处理机访问整个线性地址空间。虽然在实模式下有时也有一个叫做 Bargermaster 的表可用做与保护模式下描述符表相等价的功能，但实模式提供的仅是内存管理，而处理机的保护模式提供了内存管理和内存保护能力。

在每个版本的 Windows 中，实模式都用作进入保护模式的开始，只有当创建了一个全局描述符并初始化相应结构(中断描述符表，OS 代码/数据段，局部描述符表，和任务状态段)后，才能进入保护模式。保护模式可以以二种方式运行——16 位和 32 位方式，用于代码和数据段的选择类型。如果运行 16 位保护模式代码，可利用 64K 代码和数据段(USE 16)。如果执行 32 位保护模式代码，所有段都认为是 32 位宽(4G)。

Windows 可使用处理机保护模式来突破 1M 的限制，实现简单的内存保护。在保护模式下的内存寻址涉及了段查找表的使用，即描述符表。在保护模式下，处理机所有对段和线性内存的访问都是由这些表来管理的。如果访问到不在表中的段，应用程序就会终止。

保护模式有二种方式，即 16 位和 32 位方式。很自然，16 位保护模式是由 80286 处理机运行其上使用的，而 32 位保护模式是针对 80386 和 80486 处理机使用的。主要的区别是可分配段的大小；在 16 位保护模式下，一般是 64K 段；在 32 位保护模式下，可以是 4G 段。为了和大量的 80286 处理机相容，大多数 Windows 本身所有的应用程序和库程序都运行在 16 位保护模式下。这二种方式很容易共存，因为 16 位保护模式的描述符只是 32 位保护模式的描述符的子集。在 Windows 中，虚拟内存管理程序和所有的虚拟设备驱动程序都是运行在 32 位保护模式下的。

处理机控制模式能分为实模式和保护模式，这与内存分段有关。段是 Intel 808x 和 80x86 系列处理机的基本概念，每件事都涉及到段。段只是一块内存的逻辑映射，可能会几个段相互重叠，映射到同一片内存。段由起始地址、长度、一些属性组成，属性包括只读、读/写、只执行、执行/读和其它组合。段还可以表示为代码、数据和资源段(仅对 Windows)，它们间接地说明了相应段的属性(即，代码段是只执行的，或执行/读的等)。在分段情况下，段被映射到线性地址空间中。

在保护模式下，如果段没有被映射，或如果地址的位移量部分超过了段的长度，就认为地址是非法的。而在实模式下缺少由保护模式的段描述表所提供的内存存取和保护服务，所有 1M 以下的内存对所有运行的应用程序都可用，没有明确的段分配或映射要求。

在实模式下，地址由一个 16 位段值和一个 16 位位移量组成。段值左移 4 位加上位移量，得到一个 20 位的地址，可达到物理内存前 1M 的任意位置。由于这种生成地址的方式，64K 段可以存取。64K 至 1M 的那部分内存被 80x86 处理机用来生成叫做 HMA(高位内存区)的内存，运行在实模式下。当 808x 处理机运行在实模式下时，地址仅仅紧靠内存的开始处，而不使

用额外的 RAM。

可被处理机操作使用的可寻址物理内存(RAM)总量由处理机的地址线总量决定。808x CPU 有 20 根地址线,使用 1M(2^{20})的直接可寻址内存。80286 有 24 根地址线,使用 16M 的内存,80386 CPU 有 32 根,提供 4G 的可寻址物理内存。一个处理机的物理地址空间(它的 RAM)代表了可以存取而不必让处理机完成任何虚拟内存技巧的内存总量。

在实模式下,逻辑地址(由软件产生的地址)是静态地映射到线性地址中间,而在保护模式下,在返回一个线性地址前经过了一个描述符转换。但是在 80386 的保护模式下,当使用分页时,发生了另一层的解释,线性地址通过处理机的分页机构映射到物理地址。当存取未映射的线性地址时产生一个缺页错误,导致处理机挂起应用,交换 4K 的页到物理 RAM 中,然而在出错的地方重新开始执行应用。

在保护模式下,80x86 处理机使用了描述符表来管理可用段的存取。在表中的描述符是一个描述映射段的 8 位值。80286 的段描述符有一个 24 位基地址(指向 2^{24}或 16M 内存的任何地方)和一个 16 位的限制(使段局限于 64K),而 80386 的段描述符则有一个 32 位基地址(指向 4G 内存的任何地方)和一个 20 位的段限制。

80386 运行在 32 位保护模式下,多个段可以允许映射到 4G 的线性地址空间,或者处理机可以有效地切换到一个线性地址空间的一般编址模式。在那时,所有的系统寄存器(CS,DS,SS,ES)被映射到了同一线性地址空间范围。这有效地消除了所有分段的问题,一般模式实际上与把所有程序寄存器都映射到同一 64K 段的微小内存模式十分相似。

1.1.2 虚机器

虚机器由实和保护模式代码组成。虚机器 VM 的实模式部分完全仿真 808x 系统,直到段寻址、中断、视频内存部分;而虚机器保护模式的主要不同是对所有中断和硬件的访问都是虚拟的,这包括串行口,并行口,和视频内存的访问,硬件不能象在实模式下那样由应用程序直接访问。例如,在一个虚机器中,处理机通过使用在任务状态段中的 I/O 许可位来限制对 I/O 端口的访问,这个位决定了虚机器可以访问哪些端口和设备。

80386 实现多任务机制,Windows 就增加了一个虚机器管理器,用来管理和协同 VM 和其余系统的访问。在 Windows 下,所有的应用程序和库程序都运行在唯一一个系统虚机器的保护模式下;而对于 DOS 应用程序则运行在它自己虚机器实模式下。当创建一个新的虚机器时,系统虚机器的实模式部分被拷贝。这使对每个 DOS,VM 都需要新的 DOS 拷贝。

Windows 3.1 支持一种特殊的批处理文件 WINSTART,这个文件可用来预装入 Windows 系统虚机器,而不能装 DOS 虚机器。一般来讲,当常驻内存软件(如网络驱动程序),在 Windows 之前先装入内存时,它将影响所有运行的应用程序——Windows 的和非 Windows 的。这是因为在 Windows 进入保护模式之前,记下了实模式的状态,这些状态将用来预装入所有新的 DOS 虚机器,并且被拷贝到系统虚机器中。使用 WINSTART.BAT 可使用户能够指定应用程序只装入到系统虚机器中。如想预装入一个 DOS 虚机器,则需要创建另一个批处理文件,在进入 DOS 前预装入 VM。

1.1.3 寄存器和栈

处理机中具有许多不同的寄存器,如 AX,BX,CS 等。当程序执行时,不同的处理机寄存器被用于不同的工作。常常需要了解具体寄存器和寄存器对,以及它们如何使用的基本知识。当

一个程序出错时或变量未被装配时，若没有源代码存在，就必须求助于处理机和汇编语言知识。需要排错调试时。

寄存器分为二类：用户可修改的和系统保留的。系统保留的寄存器是指诸如虚存控制、中断控制、I/O 控制等系统结构的寄存器。可修改寄存器指可以自由使用，包括数据寄存器和索引/基寄存器等。80386 寄存器只是 80286 的超集。在寄存器名前加上前缀 E，就可得到所有的 32 位值(如 EAX、EBX 等)。

数据寄存器用来保持各种静态值，或指向数据的指针。所有的数据寄存器都有高字节和低字节二部分，每个都可用各自的名字引用(即 AH/AL，BH/BL，CH/CL，DH/DL 等)，其它的寄存器不能这样分开。

各种寄存器及它们的用途如下：

寄存器	类型	描述
AX	数据	用于算术运算和 I/O 接口，以及过程返回字。
BX	数据	用于数据结构的起始地址
CX	数据	用于循环和位移操作的计数器
DX	数据	用于算术操作数据
SI	索引	用于字符串源索引
DI	索引	用于字符串终索引
BP	基	用于栈底部指针
SP	基	用于栈顶部指针
CS	段	用来保存当前执行的代码段
DS	段	用来保存当前数据段
SS	段	用来保存当前栈段
ES	段	用于后备数据寄存器

寄存器配对及用途如下：

配　对	描述
CS:IP	当前任务的下一个指令段和位移量
SS:SP	栈段和位移量
DS:SI	指向字符串的值
ES:DI	同上
DX:AX	保存返回字和过程返回值

Windows 汇编语言代码可分成二部分——操作码和操作数。操作码通常是命令码；操作数紧跟其后，它可以是一个寄存器、静态数据、或一个间接内存地址。下列是不同寻址方式的汇编语言例子：

```
MOV AX,BX;        寄存器到寄存器
MOV AX,1;         数到寄存器
MOV AX,[BX];      在 BX 中的内存地址到寄存器(间接)
```

间接地址可以通过下列几种方法之一描述；

```
MOV AX,[BX+2];      (指向 ES:BS 的值)+2
MOV AX,[BX]+2;      和上面一样
```

MOV AX,〔BX〕.Name;　内存到寄存器(对结构)

索引地址方式可提供简便的矩阵结构存取。索引地址的格式,类似于C语言中对矩阵的存取:

MOV SI,2

MOV A;,String〔SI〕;　从字符矩阵中得到第3字节。

当象BX和BP这样的寄存器用于一个间接地址时,编译器假设某些缺省数值:BX是DS,BP是SS。在某些情况下,为了生成一个正确的地址要求重新定义,看下面例子:

MOV AX,〔BP〕;　假设为SS

MOV AX,DS:〔BP〕;　使用DS而不是SS

间接地址还可以指定其它寄存器用来帮助生成终止地址,生成基于索引方式寻址。看下列几个等价例子:

MOV AX,〔BX+DI+2〕

MOV AX,〔BX+2〕〔DI〕

MOV AX,2〔BX+DI〕

MOV AX,〔BX〕〔DI〕〔2〕

当跟踪编译代码时,可留意到一些特殊的缩写指令。它们要么是优化的结果,或者仅仅是常规编译生成的代码:

XOR AX,AX;　比MOV AX,0更快

PUSH DS;　时常使用它而不是用MOV AX,DS

POP AX;

栈是与应用任务模块相联系的临时数据存在区域。栈用于函数调用期间的存贮(保存参数和返回地址)和其它时候临时变量值的存贮(如局部变量)。所有的Intel处理器都具有指令把字长的值压入或推出这个先进后出队列;处理器还包含类似矩阵存取的指令来存取堆栈。

栈通常包含在一个程序的自动数据段(DGROUP)中,术语SS==DS,指的是栈段等同于数据。在任何时候,处理器维护着一个栈段寄存器,用来找到当前栈的段地址;和一个栈指针,用来索引当前的栈段。

栈最通常的使用是在函数调用期间,它用来保存调用函数的参数和返回值。调用的函数使用栈来建立一个框架指针,可能保存一些寄存器;最后,也为局部变量在栈中保留空间。尽管全局变量自动地被初始化为0(或null),而对于栈的局部变量并不遵循这个规则,这部分归结于它们创建的方式不同。当需要创建局部变量时,并不是把"0"值压入栈中,而仅仅把栈指针减小某个数量。由于局部变量引用的是这个未初始化的一部分栈,所以并不知道那里会是什么值。

尽管栈可用一个64K段,但它通常限制在大约5K或者10K。如果少于5K的话,Windows会自动转成5K的栈。当程序中含有许多嵌套和函数递归调用时,就需要较大的栈。在Windows 3.1提供的对话操作时,应该使用8K的栈,8K的栈往往要比一般的5K更加适合。

1.1.4　中断机制

实模式的中断向量表IVT和保护模式的中断描述符表IDT在Windows下起着重要的作用。二者都用来控制特殊中断的过程地址。IVT位于实模式的开始400h位处或虚机器中。IDT则由IDTR寄存器定位。它的位置由它的描述符决定,并不象IVT那样在内存中固定。

当发生一个中断或例外时,当前的CS:IP内容和标志被保存下来。然后控制转到一个中

断句柄，在IVT中或在IDT中。采用哪一个要依据当前Windows的执行模式——实、保护、或虚机器。当中断发生在实模式下，控制转换到IDT中的一个项。当中断发生在一个VM中，它以二种可能方式之一得到服务。首先，中断被传到VM的第0层核心控制程序，它决定了对中断干些什么。如果它是一个DOS，BIOS，或用户定义中断，它被转到虚机器的INT。如果中断或例外是响应一个缺页错或某些其它非DOS错而产生的，则使用一个在IDT中的向量。为了定位在内存中的IDT，保护模式使用了IDTR系统寄存器。

处理机还会产生处理机、软件、和硬件的例外。例外是指当处理机不能解释程序中的一条指令时，强制切换到一个例外处理过程或请求的任务。例外包括出错、捕获和异常中止。而中断是指程序的正常中止，它可以包括硬件和软件中断。软件中断可以进一步分成二类——预定义的(象INT21)和用户定义的。例外返回控制给产生例外的出错指令，而中断返回控制给中止指令的下一条指令。下面是几类例外和中断：

处理机	硬件	软件
被零除	软/硬盘驱动器	INT 21
段不存在	串行/并行口 (所有IRQ驱动器)	(其它所有INT)
缺页错	Mouse	INT 9

80386及其以上处理机有一个很好的特性是当产生一个例外时，操作码的可重新启动性。这有效地使Windows能固定住产生例外的位置，并且返回控制给任务，使之能再试一次。在Windows下有几种产生中断的方法。下面就是中断相关函数和它们的使用方法：

函数	描述	使用
DebugBreak()	产生一个INT 3h中断，切换到排错程序	C
DOSSCall()	产生一个DOS INT 21h中断	汇编
INT##	产生一个常规中断	汇编
NetBIOSCall()	产生一个NetBIOS INTsch中断	汇编
SetErrorMode()	控制是由Windows来处理DOSINT24h错误，还是由应用程序来处理错误。	C

DOS 3cALL()和NetBIOSCall()指令比INT##执行得更快些，因为控制是自动地被切换到一个中断门，而不是由Windows通过各种机制来反映中断。

各种门使得在不同的特权代码段之间切换。不同类型的门包含在LDT、GDT、或IDT中，每个都引用在GDT、LDT、或IDT中的一个段选择符。例如，一个调用门定义为二个项——一个过程的入口点和访问过程的特权级。调用门包含一个指向控制将被转换到可执行代码段的选择符；一个任务门包含着它所表示的任务的信息，包括在全局描述符表中的一个任务状态段的选择符；中断和捕获门包含一个服务于中断过程的可执行代码段描述符。所有的门使用它们自己的栈来实现保护。

1.1.5 保护环层

Intel的PC处理机通过分段(所有处理机)和分页(80386/80486)来实现保护。保护层可以用4-环图表示，见图1.1。越是低的数字，特权级越高。环0通常被核心和其它操作系统内核使用，环1和环2被所有的设备驱动程序所用，环3被应用程序所用。Windows在第3环层运行

所有的应用程序,DLL 库程序和 DOS 块程序,系统层的代码在第 0 环层运行。

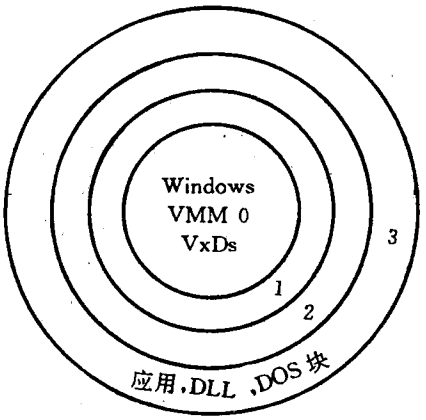


图 1.1 Windows 环保护层

保护可通过几种机制实现,这些都由处理机的地址翻译并行实现。下面包括 5 种基本的保护检查:

(1). 用类型来控制段的访问方式(只执行是 x-0,只读是 r-0,和可读写是 r-w)。

(2). 限制建立在线性内存中段的边界。

(3). 由 CPL,RPL 和 DPL 提供的各特权保护层。

(4). 由门描述符提供了过程入口点限制。

(5). 由处理机来监控指令集的限制。

段在几个方面支持保护。首先,把段作为私有地址空间的概念提供了任务保护。在缺省情况下,若没有表示许可,任务不能访问另一个任务的段。如果所需的段不在程序的局部或全局描述表中,不可以对该内存访问;其次,代码、资源、和数据段有各自的只执行,只读,和可读写属性。这些属性定义了对段的基本访问方式;第三,处理机捕获所有的对段界外的读写操作,产生一个例外。这包括一个越界地址,以及读一个字(它的一个字节位于段结束之前)或读一个双字(它的一个,二个或三个字节位于段结束之前)。

几种处理机构结有它们相关的特权层,包括:

(1). 任务。它由一个与之相关的当前特权层(CPL),一个位于当前执行代码段的属性。

(2). 段。它有一个与之相关的描述符特权层(DPL),它代表着对该段访问所需的特权层。

(3). 段选择符。它有一个与之相关的要求特权层(RPL),它代表着生成该选择符的任务的特权层。

还有另一个 EPL,叫有效特权层。特权层通常用一个 2 字节值表示,它产生一个相应的保护环层。

处理机还以受限操作码形式实现保护。所有处理全局、局部和中断描述符表的寄存器和操作码都是受限制的。保护还经由调用、捕获、中断、和任务门描述符,提供它们转换到不同段控行前检查的保护层。

80386 的内存分页机制也实现了保护,因为每个页表都包含了指出它保密级和可读写状态的一个 2 位值。这个保密级指出该页是属于一个超级用户级任务还是一个用户级任务。应用程序不能访问标上超级用户所拥有的页面。这个超级用户位等价于 CPL 的 0—2。在将来 32

位模式版本的 Windows 中，环层保护模式将被废除，因为分段将不再存在了。但是，页级保密和保护将仍然需要。

1.1.6 多任务机制。

Intel 的 80386 和 80486 保护模式处理机提供了在虚机器之间的基于硬件的抢占多任务能力。在任何时刻，处理机可以强制执行任务切换。而 Windows 实现的仅是非抢占多任务，活跃任务必须自愿地放弃控制给 CPU，当不这样的话，系统马上会停止。Windows 实际上把整个系统当作任务虚机制。所有的 Windows 应用程序，DLL 库程序和设备驱运程序都运行在单个系统 VM 上，而每个 DOS 块都拥有自己的 VM。这就是为什么 DOS 块是抢占的多任务而应用程序不是的原因。一旦每个应用程序都拥有了它自己的 VM 和分离地址空间时，就将有完全的抢占多任务了。

任务切换不同于函数调用，因为大量的处理机状态信息将被保留。一个正常的过程调用使用栈来实现，以致于允许该过程再一次调用自己。但是任务并不是把所有数据都压入栈中，也不能重入。任务切换保存和装入了整个寄存器组值、LDT。和用于映射任务的线性内存的不同分页结构。

80386 和 80486 处理机提供了几个结构和寄存器来完全支持多任务环境，包括：

(1). 每个任务一个局部描述符表(位于 GDT)，提供任务地址空间保护。
(2). 任务状态段(在 GDT 中的一个系统段)包含了任务相关信息。
(3). 一个任务寄存器，它包含了一个指向当前任务的任务状态段的 GDT 选择符。
(4). 任务门描述符，提供了切换任务的一个间接方法。
(5). 任务状态段的一个特殊描述符，它包含了一个位，表示任务是忙，或闲。

1.2 存贮器存取模式

1.2.1 存取模式

在 Windows 中，存贮器按照存取方式的不同，可以分成三种不同类型的内存，即常规内存、扩充内存和扩展内存。见图 1.2。

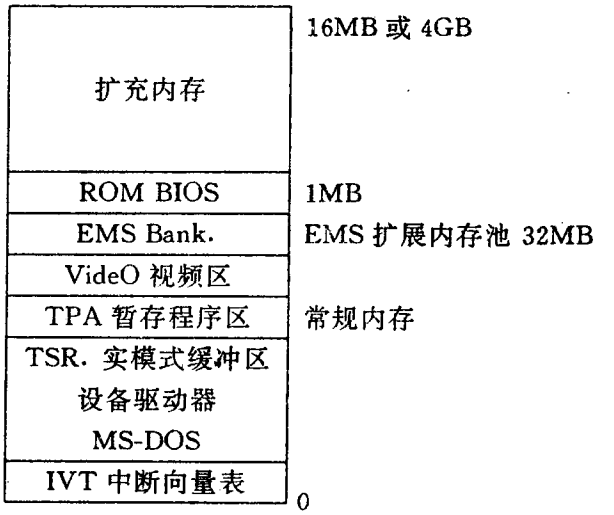


图 1.2 Windows 内存类型

(1)常规内存

常规内存范围在 0—1MB 之间,它代表了可以被所有模式所能存取的内存。在这种常规内存的存取模式中,其中低位 RAM 的前 10—20K 用来保存中断向量表 IVT、实模式内存缓冲区 TSR、设备驱动程序和 MS-DOS,较上面的 384K 用作视频 RAM/ROS 和系统 ROM BIOS。全局堆由小于 640K 的内存空间减去已用去的低位 RAM 组成,这个空间称为暂存程序区 TPA,它包括所有可能的网络驱动程序软件和 Windows 的小部分,以及每个库程序和工作台上的应用程序。

Windows 可以调用 Global DOS All0(1)来分配常规内存,不然的话,就无法保证内存来自哪一部分。除非绝对需要,不应使用常规内存。常规内存太珍贵,以致于只有在 Windows 装入之前时,才可以创建一个实模式内存缓冲区 TSR。

挤在视频内存区和 ROM BIOS 之间的内存称为高内存块(UMB),这一段内存通常是为 loadhi 工具保留的,在 MS_DOS5 和 QuarterDeck 的 QEMM 中可以找到这些工具。高内存块一般不能被开发者直接分配。

(2)扩展内存

扩展内存 EMS 能够给应用程序 32M 的额外内存。Windows 内部也使用扩展内存,但它只能存取小的 13K 块,这只对某些类程序有效。另外,扩展内存通过位于常规内存中的窗口框架存取,这使得常规内存比原先更加珍贵了。

扩展内存实附上仅对数据存贮有用,而不是代码存贮。不同的 EMS 规范原先是为 808x 计算机设计的。由于 Windows 3.1 已经不支持实模式,应用程序对扩展内存的支持同样也会慢慢地减弱。

(3)扩充内存

扩展内存由所有位于常规内存的 1M 界线以上的内存组成,它包括 HMA 空间和其余的物理内存。这意味着一个 80286 处理机可有最大 15M 的扩充内存,以及 80386 和其以上处理机可有最大 4G(减去 1M)的扩充内存。Windows 和它的应用程序几乎独占地使用扩充内存。

存取扩充内存要求使用保护模式,具有许多其它内存模式不能提供的好处。首先,扩充内存可以相当充足。即使没有这么多,80386 系统还可以 Pageout 扩充内存,来给出大于实际地址空间的内存,可以仿真扩展内存(如果需要的话)。更重要的是扩充内存还容易去控制、运行、