计算机语言技术系列丛书(二)

学苑出版社

内存管理 C++ MEMORY MANAGEMENT

- Details an object-oriented interface to XMS and EMS 3.0, 3.2, & 4.0
- Complete virtual memory management system
- Disk includes fully documented demonstration programs

THE Len Dorfman PRACTICAL PROGRAMMING SERIES

Len Dorfman
Marc J. Neuberger

C++ 内存管理

Len Dorfman
Narc J. Neuberger 著
熊可宜 译
吴红艳 校

学苑出版社
1994

(京)新登字 151 号

内 容 简 介

本书以丰富的实例详细讨论了内存扩充、内存管理的各种工具，有利于 C++程序员合理地利用高端内存，体会并理解到 EMS、XMS 及内存管理系统的工作过程，书中提供了许多实用程序的源代码，读者可直接调用和借鉴使用。

欲购本书的用户，请直接与北京 8721 信箱联系，邮编：100080，电话：2562329。

版 权 声 明

计算机语言技术系列丛书

C++内存管理

著　　者：Len Dorfman　Narc J. Neuberger
翻　　译：熊可宜
审　　校：吴红艳
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：北京市地矿局印刷厂印刷
开　　本：787×1092　1/16
印　　张：22.625　　**字数**：528 千字
印　　数：1～5000 册
版　　次：1994 年 5 月北京第 1 版第 1 次
ISBN7－5077－0905－1/TP・29
本册定价：49.00 元

学苑版图书印、装错误可随时退换

用 户 请 注 意

欲购本书配套软盘的朋友，请按下列方法汇款：

单价：50.00 **元（含邮费）**

注：从银行电汇款的朋友请按下列帐号和收款单位汇款：

收 款 单 位：北京希望电脑公司
开户银行及帐号：北京海淀工商行中关村城市信用社　　帐号：05079－08
注：要增殖税发票的朋友，请仔细填下表。

购货单位	名称	纳税人登记号
	地址电话	开户银行及帐号

注：本次共订盘　　张　　应收款为￥（小写）：

注：用户填好此单后请连同此单、信汇单一并传真 01－2561057 或 01－2579874，收到传真后即发货。或邮寄 100080 北京海淀 8721 信箱资料部朱红收

联系电话：01－2562329，2541992

用户须知

为了最有效地使用本书，用户需要对C++程序设计语言有初步的了解，并且对其面向对象的特性有一定的认识。本书中所有的程序在Borland C++ 3.1和Microsoft C++ 7.0下都是兼容的。

本书中的汇编语言模块都是在TASM和MASM下汇编的。使用本书中提供的内存管理函数并不需要用户懂得汇编语言程序设计。

“MAKE”文件提供了编译和连接本书中所有程序的工作，每一个库和附带的范例程序均有一个make文件，针对Borland C++ 3.1和Microsoft C++ 7.0，分别有一套make文件，各make文件的文件名如下所示：

MCB.MAK	编译第二章中所有BC++ 3.1的项目文件。
MCBMS.MAK	编译第二章中所有MSC++ 7.0的项目文件。
EMS.MAK	编译第三章、第四章中BC++ 3.1的项目文件。
EMSMS.MAK	编译第三章、第四章中仍有MSC++ 7.0的项目文件。
XMS.MAK	编译第五章中所有BC++ 3.1的项目文件。
XMSMS.MAK	编译第五章中所有MSC++ 7.0的项目文件。
VM.MAK	编译第六章中所有BC++ 3.1的项目文件。
VMMS.MAK	编译第六章中所有MSC++ 7.1的项目文件。

在Borland C++ 3.1环境下，可以在命令行中键入：

```
make -f MCB
```

来启动make创建MCB库和编译范例程序。

在Microsoft C++ 7.0环境下，可以在命令行中键入：

```
nmake -f MCB.MAK
```

来启动make创建MCB库和编译范例程序。

前　言

本书知识性、实践性很强，编写它的目的是用来给C++程序员提供一个将高端内存(扩充内存、扩展内存、硬盘)的动态集合正确地转化成应用程序的实践性工具。本书提供了用来在8086/8088实模式下进行扩充内存、扩展内存管理的工具。

在第六章中，EMS和XMS内存管理函数为支持虚存管理系统提供了基本的模块，整个VMM系统的源代码与EMS和XMS接口函数一起提供给了用户，它们与一个内容丰富的内存管理库都存放在一张磁盘上。

当用户读完本书后，就能够体会并理解到EMS、XMS以及一个虚存管理系统是如何工作的，这会帮助用户在自己的应用程序中充分的吸取EMS、XMS和硬盘内存的精髓，在用户的手头会有一系列使用方便的虚拟动态内存分配函数，可以在程序中用它们在多兆字节的内存空间中获取内存块。

本书一开始就直接了当地讨论一个PC机上的内存管理方案，在简单的叙述完内存管理之后，在第二章中介绍了“内存战场”(在Microsoft这样称呼)或者说内存控制块(MCB，正如在许多系统中提到的那样)，并且提供了一个显示全部内存控制块的程序。

第三章提供了对EMS 3.0和3.2全部函数的详细介绍，并且在本章的演示程序中说明如何使用EMS 3.0和3.2的函数。每个EMS子函数的源代码和函数原型以及演示程序的源代码清单一并提供给了用户。

当用户对使用EMS 3.0和3.2函数有了一定的了解之后，在第四章中通过剖析EMS 4.0扩充内存标准继续讨论EMS，与第三章相似的是，在第四章中同样提供了部分EMS 4.0子函数的源代码和函数原型，以及几个EMS 4.0演示程序的源代码。

第五章介绍了扩展内存规范(XMS) 2.0，与前两章一样，也提供了每个XMS子函数的源代码和原型以及XMS演示程序。

第六章提供了一个虚存管理程序的源代码，此程序可以使用户在自己的应用程序中打开并且使用一个2兆字节的内存空间，这个动态分配内存空间可以被打开、写入、读取并且释放，就如平时进行标准的动态分配内存工作那样方便，在本章中同样提供了VMM接口函数的原型和两个演示程序，本章中源代码的注释均是精心提炼出来的，这样助于使用户加快对VMM的内部工作过程的理解。

目　录

第一章 内存管理

虽然一些程序员感到 80x86 系列 CPU 的段址结构比较繁杂，但是这些 CPU 工作得非常令人满意，用方言来说“它们是好样的！”

让我们简单地叙述一下 80x86 系列 CPU 的内存结构：80x86 系列处理机有三种工作模式，较少使用的是“实模式”，工作在所有 80x86 系列的处理器上，在此模式下，所有 CPU 的工作如同一个 8086/8088 处理器，地址是由一个 16 位的段址和一个 16 位的偏移量组成，将段址值左移 4 位加上偏移量构成一个 20 位的物理地址，它可以寻址的范围是 1 兆字节。

16 位的保护模式可以工作在 80286 以上的处理器上，在此模式下，一个地址是由一个 16 位的选择符和一个 16 位的偏移量构成，与直接计算物理地址的方法不同的是，选择符代表一个描述符表中的描述符。一个描述符包含一个 24 位的基地址，此基地址用来进行地址计算。偏移量与此基地址相加即得到物理地址，这样一个 24 位的地址就可以寻址 16 兆字节的内存空间。

32 位的保护模式工作在 80386 以上的处理器上，在此模式下，一个物理地址由一个 16 位的选择符和一个 32 位的偏移量构成。同样，选择符代表一个描述符，不过现在此描述符包含一个 32 位的基地址，偏移量与此基地址相加，得到一个 32 位物理地址，这样可以寻址 4 兆兆的内存空间，并且，32 位的地址允许使内存形成一个 4 兆兆的线性地址空间，直接通过一个偏移量进行寻址即可。

现有的许多 MS－DOS 和 PC－DOS 程序都是工作在实模式下，本书就是用来处理实模式下的内存管理问题。

在实模式下编程与在 80286，80386，80486 保护模式下编程的不同之处在于，CPU 不能寻址 1 兆以上的内存空间。

在 1 兆字节以下的内存空间可以分成两部分，从 0 到 640K 的内存称作是常规内存，从 640K 到 1024K（1 兆字节）的内存空间称作高端内存（注意：一些资料，尤其是 Microsoft 的一些授权书籍中，将整个内存空间的 0 到 1024K 称作是常规内存），1 兆字节以上的内存被称作扩展内存。扩展内存的第一个 64K（物理地址 0x100000－0x10FFF0）是一个特殊的区域，即人们熟知的高位内存区或 HMA，并不是只有从 0 到 1024K 的内存空间可以正常的寻址，我们可以用第五章中讲述的技巧编写的实模式程序来进入 HMA。

内存范围	名称
0～640K	常规内存
640K～1MB	高端内存
1MB 以上	扩展内存
1MB～1MB＋64K	高位内存

常规内存可被 PC BIOS 的数据区、DOS、中断向量表、设备驱动程序以及 TSR 使用，剩余的可被用户程序占用的内存称作暂存程序（TPA——Transient Program Area）。

高端内存是为 BIOS 代码和硬件设备所保留的，视频显示适配器需要不同数量的高端内

存来当作它们的屏幕刷新缓冲区，系统 BIOS(基本输入输出系统)，提供了低级的硬件接口，驻留在高端内存，占据高端内存的还有磁盘控制器，用来提供与操作系统的接口。

任何了解如何利用 EMS 和/或 XMS 规范的程序员都可以使用扩展内存。

常规内存被分割成许多内存块，每一个这样的内存块都由一个内存控制块(MCB——Memory Control Block)来描述。在这些内存块结构中提供了如下的信息：标识内存块的大小，内存块是自由的还是被用户程序所占用。

其它各种需要都必须占用低端内存区域，这样整个空间就会用尽，在用户使用 DOS 5.0 以前的版本时，会发现只有 450K(或更少)的空间供他们使用(TPA)。

虽然 DOS 5.0 和其它的一些商业性的内存管理实用工具通过允许用户安装 TSR(中断驻留内存程序)、设备驱动程序以及将 DOS 安装到高端内存或扩展内存中等一些手段相当大的促进了对内存的管理，用户还是受限制于微不足道的 640K 暂驻内存区的界限。

为什么要用“微不足道”来描述 640K 的限制呢？假设用户需要动态的分配 1M 字节的内存供程序使用，简而言之，640K 空间不可能达到需求。随着程序的不断的增大和复杂，软件设计者们迫切需要一种使用 640K 内存以外内存空间的手段。

在 EMS 和 XMS 问世之前，一个解决办法是用磁盘存储来代替 RAM；虽然这种手段能够进行，但是它却带来许多不方便，磁盘存取的速度比 RAM 存取的速度慢得多，利用磁盘来解决 640K 空间的限制使程序的执行变得非常慢，以致于软件设计者们说“时间就是金钱”。

但是，硬件设计者们经过努力，冲破了 640K 的界限，于是出现了扩充内存(EMS)，EMS 是驱动程序的软件规范，它使得在实模式下可以寻址超了 1M 字节来存取内存，EMS 提供了一个页切换方案，使得大块的扩展内存可以被映像到高端内存地址，被称作扩充内存管理(EMM——Expanded Memory Managers)系统的程序就是为了支持这种页切换方案而设计的，或是通过使用硬件上的页映像特性来实现。

这种调整证明是相当有效的，因为 Lotus，Intel 和 Microsoft(LIM)一起创建了 EMS 标准，EMS 标准存在的必要性在于它对于编写一个运行良好而不会破坏其它程序的 EMS 数据的程序起了极大的推动作用，虽然页切换机制并不如线性地址那样清晰明了，但它确实改进了使用硬盘的缺陷。

80286 芯片对市场的巨大冲击预示着现在允许 CPU 存取 1M 字节界限以外的内存的保护模式编程的来临。1M 字节以上的内存被称作扩展内存，但是 DOS 并不是一个工作在实模式下的操作系统，并且它没有提供能够存取 1M 字节以上内存空间的手段。一时间，在用户程序中使用扩展内存被证明是危险的，因为并没有一个调整扩展内存的使用的规范。

然而，到了 80 年代，出现了扩展内存规范(XMS)，这个 XMS 标准伴随着 Microsoft 公司的 HIMEM.SYS XMS 设备驱动程序的产生，使得程序员们可以以一个有序的风格来存取扩展内存。

现在，DOS 内存管理应用程序可以使用廉价的扩展内存来实现类似于扩展内存的功效，这是一种经济的解决办法，许多新的 80x86 母板允许安装许多兆字节的 XMS 内存条，内存管理实用程序可将此扩展内存分割成 XMS 和 EMS 的各种组合。

在编写本书的时候，工作在实模式下的程序员们已经掌握了三种为进行动态内存分配而冲破 640K 界限的独特方法，他们可以使用：

EMS 和硬件内存映像

- 扩展内存和 XMS 或 EMS 软件
- 硬盘代替 RAM

1.1 使用扩充内存进行动态内存分配

实现了 EMS 标准的软件被称作扩充内存管理员(EMM—— Expanded Memroy *Manager*),一个 EMM 程序分配位于 640K 和 1024K 地址空间之间的一个 64K 空间用做页面,4 个 16K 的位于 1M 字节以上的 RAM 块即可通过调用 EMM 程序来映像到此页面。

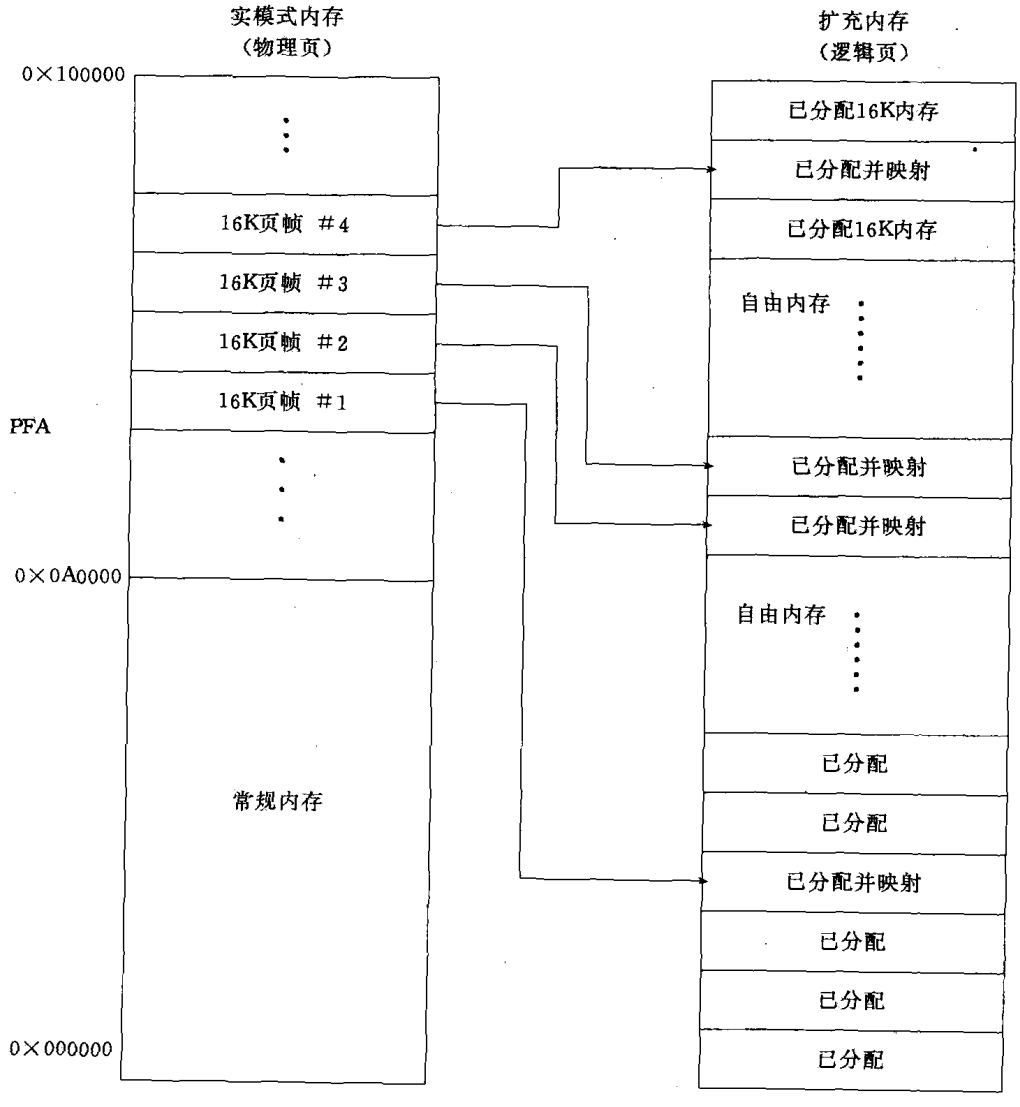

图 1—1

这个 64K 的页面是由 4 个 16K 的物理页组成的,这四个物理页是从一个称作逻辑页的庞大的集合中取(实质上是映像)来的(熟悉操作系统的用户可以注意到,逻辑页这个术语的一些背景,在这里程序引用物理页映像到逻辑页而不是其它方式),让我们来更加形象化的说明 16K 逻辑页,16K 物理页与 64K 页面之间的关系。图 1—1 所示为 EMS 物理和逻辑页之间的

关系。

表1—1提供了与EMS V3.0、V3.2、V4.0相关的功能清单。

表1—1 EMS V3.0、V3.2、V4.0相关的功能清单

版本	功能	子功能	描述
3.0	40h		取得EMS状态
3.0	41h		取得EMS页帧地址
3.0	42h		取得EMS 16K页的数目
3.0	43h		分配EMS句柄和16K页
3.0	44h		将EMS逻辑页映射到物理页
3.0	45h		自由的EMS句柄和逻辑页
3.0	46h		取得已安装的EMS版本
3.0	47h		存储EMS页映射
3.0	48h		恢复EMS页映射
3.0	49h		(保留以便将来使用)
3.0	4Ah		(保留以便将来使用)
3.0	4Bh		取得EMS句柄数
3.0	4Ch		取得EMS句柄页
3.0	4Dh		取得所有句柄的所有EMS页
3.2	4EH	00h	存储EMS页映射
3.2	4EH	01h	恢复EMS页映射
3.2	4Eh	02h	存储和恢复EMS页映射
3.2	4EH	03h	取得EMS页映射信息大小
4.0	4FH	00h	存储部分EMS页映射
4.0	4Fh	01h	恢复部分EMS页映射
4.0	4Fh	02h	取得部分EMS页映射信息的大小
4.0	50h	00h	按数目映射多个EMS页
4.0	50h	01h	按地址映射多个EMS页
4.0	51h		按句柄重定位EMS页
4.0	52h	00h	取得EMS句柄属性
4.0	52h	01h	设置EMS句柄属性
4.0	52h	02h	取得EMS属性容量
4.0	53h	00h	取得EMS句柄名
4.0	53h	01h	设置EMS句柄名

版本	功能	子功能	描述
4.0	54h	00h	取得所有 EMS 句柄名
4.0	54h	01h	查找 EMS 句柄名
4.0	54h	02h	取得所有 EMS 句柄
4.0	55h	00h	按数目和 JMP 映射 EMS 页
4.0	55h	01h	按地址和 JMP 映射 EMS 页
4.0	56h	00h	按数目和 CALL 映射 EMS 页
4.0	56h	01h	按地址和 CALL 映射 EMS页
4.0	56h	02h	取得 EMS 映射页和 CALL 的大小
4.0	57h	00h	移动内存区域
4.0	57h	01h	交换内存区域
4.0	58h	00h	取得可映射 EMS 页的地址
4.0	58h	01h	取得可映射 EMS 页的数目
4.0	59h	00h	取得硬件配置信息
4.0	59h	01h	取得原始 16K 页的数目
4.0	5Ah	00h	分配句柄和标准 EMS 页
4.0	5Ah	01h	分配句柄和原始 16K 页
4.0	5Bh	00h	取得其它 EMS 映射寄存器
4.0	5Bh	01h	设置其它 EMS 映射寄存器
4.0	5Ch		为热启动预备 EMM
4.0	5Dh	00h	禁止 EMM 操作系统功能
4.0	5Dh	01h	允许 EMM 操作系统功以
4.0	5Dh	02h	释放 EMS 访问键

EMS 的页映像过程可以理解成以下三个步骤：

1. 使用 EMM 将四个不同的 16K 逻辑页映像成 4 个不同的 16K 物理页。
2. 从物理页中读取数据，或向其中写入数据。
3. 将新的 16K 逻辑页映像到物理页。
4. 转到步骤 2。

我们推荐程序员将 EMS 当作打破 640K 内存界限进行动态内存分配的第一个位置。

1.2 使用扩展内存进行动态内存分配

我们将扩展内存 XMS 的使用作为继 EMS 之后的第二个选择对象是有以下原因的：

1. EMS 允许将页映像成可寻址的内存空间，而 XMS 却需要将数据传送到扩展内存，或

者从扩展内存中读出数据，两者相比较，映像比起传送 16K 的数据来说快得多。

2. EMS 提供了比 XMS 更加丰富的函数集。

表 1－2 提供了 XMS 2.0 规范的功能清单。

表 1－2

版本	功能	描述
2.0	00h	取得 XMS 版本号
2.0	01h	申请高端内存
2.0	02h	释放高端内存
2.0	03h	全局允许 A20
2.0	04h	全局禁止 A20
2.0	05h	局部允许 A20
2.0	06h	局部禁止 A20
2.0	07h	查询 A20
2.0	08h	查询自由扩充内存
2.0	09h	分配扩充内存块
2.0	0Ah	自由扩充内存块
2.0	0Bh	移动扩充内存块
2.0	0Ch	锁定扩充内存块
2.0	0Dh	解锁扩充内存块
2.0	0Eh	取得句柄信息
2.0	0Fh	重新分配扩充内存块
2.0	10h	申请上端内存块
2.0	11h	释放上端内存块

1.3 使用硬盘进行动态内存分配

在 C++中使用文件 I/O 是一个相对来说比较容易的工作。在主机没有可用 EMS 或 XMS 内存的时候，用基于 DOS 的文件 I/O 函数建立的进行扩展内存管理的函数，为使用扩展内存提供了一个有用的备份手段。

我们将用磁盘进行动态内存分配管理的工作称作“备份”是因为它与基于 RAM 的手段相比较，速度太慢了。

1.4 小结

DOS 是一个工作于实模式的操作系统，它最多只能给用户程序提供 640K 字节的内存空

间，一个程序可占用的内存空间称作可用内存区(TPA)

许多程序对内存的需要远远超过 640K 的限制，于是出现了各种各样的内存管理方法。

使用 EMS 进行动态内存分配被证明是可靠的，并且速度也很快。EMS 的可靠性是随着标准的出现而逐渐成熟的，程序设计标准保证了程序的正确运行，EMS 作为我们进行动态内存分配的第一个选择对象的主要原因在于它的功能与 EMS 3.0、3.2 和 4.0 规范的紧密结合，以及与数据传输相比较，进行页映像的速度要快得多。

使用 XMS 进行动态内存分配同样也具有速度快，可靠性高的优点，正因为 XMS 2.0 提供的功能少于 EMS 规范所提供的，所以我们才将其作为进行动态内存分配的第二个选择对象。

鉴于以上的原因，我们可以顺利地列出下面的内存管理特征：

1. EMS(扩充内存)
2. XMS(扩展内存)
3. 硬盘驱动

第二章 内存控制块

要想全面的了解内存管理机制，首先必须了解一个 DOS 如何在低端内存进行内存分配。简单地说，DOS 将 640K 的低端内存分割成一系列相邻的内存块，这些内存块的大小是 16 字节的整数倍，并且称这些内存块为段落，这样做是很有意义的，因为段寄存器(DS,ES,CS,SS)是以段落做边界，而不是以单个字节做边界(这使我们想起在计算一个物理地址数的时候，是将段地址值左移四位，或者说，乘以 16 与偏移地址值相加而实现的)。

DOS 在每个内存块的开始处设置了一个段头(16 字节)。在此段头中描述了本内存块的一些特殊的属性。在本书中将一个段落内存块描述符称作内存管理块(MCB)，其它一些书籍中将它称作内存区，以及区头。

在用户计算机的低端内存中，有许多内存块，它们用来描述内存被用户程序占用的情况。这些内存控制块的集合称作内存控制块链(MCB Chain)。

在内存中，紧接在内存控制块之后的就是它们描述的那块内存。让我们了解一下在 DOS 4.0 以前的内存控制块结构大概是如何工作的。请注意，我们说“大概”这个词是因为在早期的 Microsoft 文档中并没有提及 MCB(内存控制块)。事实上也是这样，如何定位第一个 MCB 地址的 DOS 中断调用仍然是一个 DOS 尚未公开的服务。在我们的 C++类 MCB_Mcb 中提供了一个方便的周游 MCB 链的方法。

```
struct MCB_Mcb {
  char chain_status;
  WORD owner_psp;
  WORD size_paragraphs;
  BYTE dummy[3];
  char reserved[8];

  };
```

MCB 结构的第一项是一个 8 位的值，用来标识内存链的状态。若其值是 ASCII 字符“M”，则表示在内存块链中还有许多内存块，若其值是 ASCII 字符“Z”，表示本 MCB 块是整个内存块链的最后一块。

MCB 结构中的第二项是占有本内存控制块的用户程序的 PSP 段值，在装入一个用户程序(.EXE 文件)的时候，DOS 自动创建一个 256 字节的 PSP(程序段前缀)，在本章中，我们将使用用户程序 PSP 的以下数据：

PSP 段值可用做一个用户程序 ID

PSP:[0x2C]存放程序的环境段

PSP:[0x80]存放程序的命令行长度

PSP:[0x81]是命令行(ASCE)字符串的开始

MCB 结构中的第三项是一个 16－bit 的整数，其值为紧接着本 MCB 的内存块的大小，并

且其最小单位是段落(即 16 个字节),当需要以字节数计算一个内存块的大小时,简单地用此整数值乘以 16(当然,对于编制汇编语言代码的程序员来说就是将此整数值左移四位)即可。

清楚了 MCB 所描述的内存块的大小之后,就很容易找到在 MCB 链中的下一块 MCB。用户可按以下公式计算:下一个 MCB 块地址=本 MCB 地址+相对于本地址的内存块大小+1(MCB 本身的大小)。你瞧,很容易就得到了下一个 MCB 的地址。

程序 MAPMEM,MEM,TDMEM 以及 PROG2－3.CPP(在图 2－3 提供了一个显示内存的实用程序),分别是用来得到第一个 MCB 的地址,显示出一些信息,再得到下一个 MCB 的地址,再显示出一些信息,就这样一直循环到此 MCB 链的结。当掌握了获取 MCB,PSP 以及程序环境段的途径后,用户可以提供内存块占用者的许多信息。

在 DOS 4.0 以后的版本中,第五项一共占用 8 个字节,存放的是占用此 MCB 的内存块的文件的文件名(不包括后缀)这样就减轻了寻找占用 MCB 的程序的文件名的负担,因为用户不需要再从占用内存程序环境段中寻找占用者的文件名了。

这里有一个 DOS 4.0 以后版本中使用的 MCB 结构

```
struct MCB_Mcb {
  char chain_status;
  WORD owner_psp;
  WORD size_paragraphs;
  BYTE dummy[3];
  char file_name[8];
  };
```

现在我们已经介绍完了 MCB,下面可以编写一些程序了。

2.1　预备性的内存管理例程

图 2－1 提供了 MCB.H 的源代码。这些文件中包含了用于显示内存控制块链信息的有用的定义。

```
////////////////////////////////////
//
// mcb.pas
//
// Unit file for the MCB functions
//
////////////////////////////////////

//
//  Define the Memory Control Block Structure
//
struct MCB_Mcb {
    char                chain_status;
    WORD                  owner_psp;
```

```
    WORD                size_paragraphs;
    BYTE                dummy[3];
    char                file_name[8];
    };

//
//  Define the MCB_McbPtr class. Which gives convenient access
//  to the MCB chain.
//
class  MCB_McbPtr {

    MCB_Mcb             * mcbPtr;

public:

        MCB_McbPtr() {
            mcbPtr= NULL;
            }

        MCB_McbPtr(WORD);

    void first();

    int  more() {
            return (mcbPtr ! = NULL);
            }

    void next();

    MCB_Mcb  * operator->() {
            return mcbPtr;
            }

    WORD segment();

    };

DWORD       getVecPointer(BYTE);
```

图 2-1 MCB.H 的源代码清单

仔细分析一下 MCB.H 的源代码，除了定义 MCB 结构以外，我们还定义了一个类 MCB_McbPtr，这使得更容易获取 MCB 的信息。此类的对象看上去像一指向 MCB 的指针，实际上还包含了三个成员函数，它们用来更加方便的周游 MCB 链，这三个函数一起用在一个“for”循环