

HOPE

用C语言设计屏幕界面技术

阮江 沈铁齐 编译



北京希望电脑公司

7-18/22
218.1

用 C 语言设计屏幕界面技术

阮江 沈铁齐 编译

JS99 / 11

北京希望电脑公司
一九九二年十二月

目 录

| | |
|-------------------------|----|
| 前言 | 1 |
| 第一章 存取显示适配器 | 4 |
| 1.1 显示适配器 | 4 |
| 1.2 混合显示器 | 5 |
| 1.3 视频缓冲区 | 5 |
| 1.4 正文属性 | 6 |
| 1.5 彩色正文分页 | 8 |
| 1.6 虚拟屏幕 | 9 |
| 1.7 指针和存储模式 | 9 |
| 1.8 直接存取视频缓冲区 | 10 |
| 1.9 BIOS 中断 | 12 |
| 1.10 执行一个中断 | 12 |
| 1.11 ANSI 控制台驱动程序 | 14 |
| 1.12 摘要 | 15 |
| 第二章 窗口和屏幕函数 | 16 |
| 2.1 目的 | 16 |
| 2.1.1 什么是窗口? | 16 |
| 2.2 应用 | 17 |
| 2.2.1 指定编译器 | 17 |
| 2.2.2 函数原型和编码技术 | 18 |
| 2.2.3 使用窗口模块 | 19 |
| 2.2.4 创建一个库 | 20 |
| 2.2.5 外部变量 | 20 |
| 2.2.6 命令行开关 | 21 |
| 2.2.7 窗口函数的类型 | 22 |
| 2.2.8 使用函数 | 32 |
| 2.2.9 优化窗口布置 | 37 |
| 2.2.10 其他技巧 | 41 |
| 2.3 技术 | 42 |
| 2.3.1 窗口的类型 | 42 |
| 2.3.2 更新窗口 | 44 |
| 2.3.3 滑动虚拟窗口 | 44 |
| 2.3.4 外部参数 (数据结构) | 45 |

| | | |
|--------|---------------------|-----|
| 2.3.5 | 窗口结构 | 49 |
| 2.3.6 | 写屏幕 | 49 |
| 2.3.7 | 虚拟屏幕 | 52 |
| 2.3.8 | 写虚拟窗口 | 53 |
| 2.3.9 | 与多任务环境的兼容性 | 54 |
| 2.3.10 | 光标管理 | 56 |
| 2.3.11 | 雪花消除 | 56 |
| 2.3.12 | 雪花和无屏显输出 | 58 |
| 2.3.13 | 最高效的屏幕更新 | 58 |
| 2.4 | 摘要 | 59 |
| 2.5 | 源代码 | 59 |
| 第三章 | 菜单设计 | 110 |
| 3.1 | 简介 | 110 |
| 3.1.1 | 用户观点 | 110 |
| 3.1.2 | 程序员观点 | 112 |
| 3.2 | 弹出菜单的目的 | 113 |
| 3.2.1 | 应用 | 113 |
| 3.2.2 | 技术 | 120 |
| 3.2.3 | 源代码 | 122 |
| 3.3 | 移动亮条菜单的简介 | 129 |
| 3.3.1 | 目的 | 129 |
| 3.3.2 | 应用 | 131 |
| 3.3.3 | 技术 | 132 |
| 3.3.4 | 源代码 | 132 |
| 3.4 | 下拉菜单的目的 | 139 |
| 3.4.1 | 应用 | 141 |
| 3.4.2 | 技术 | 147 |
| 3.4.3 | 结论 | 149 |
| 3.4.4 | 源代码 | 149 |
| 第四章 | 数据输入屏幕 | 158 |
| 4.1 | 简介 | 158 |
| 4.2 | 目的 | 158 |
| 4.2.1 | 字段编辑器 | 158 |
| 4.2.2 | 数据输入屏幕编辑器 | 159 |
| 4.3 | 应用 | 160 |
| 4.3.1 | 数据输入屏幕 (多个字段) | 161 |
| 4.3.2 | 数据检验 | 163 |
| 4.4 | 技术 | 165 |
| 4.4.1 | 屏幕显示 | 166 |

| | |
|------------------|-----|
| 4.5 源代码 | 168 |
| 第五章 列表选择 | 181 |
| 5.1 简介 | 181 |
| 5.2 目的 | 181 |
| 5.3 应用 | 182 |
| 5.4 技术 | 183 |
| 5.4.1 指点 | 183 |
| 5.4.2 快速查找 | 185 |
| 5.5 源代码 | 187 |
| 第六章 目录函数 | 195 |
| 6.1 简介 | 195 |
| 6.2 目的 | 195 |
| 6.3 应用 | 195 |
| 6.4 技术 | 196 |
| 6.5 结论 | 200 |
| 6.6 源代码 | 200 |
| 第七章 求助屏幕 | 205 |
| 7.1 简介 | 205 |
| 7.2 目的 | 205 |
| 7.3 应用 | 206 |
| 7.4 技术 | 210 |
| 7.5 结论 | 213 |
| 7.6 源代码 | 214 |

前 言

本书旨在帮助 C 程序员建立功能强大，直观易学的屏幕接口。
我们将涉及屏幕和窗口函数库的构造，并且要使用它建立与流行的商业软件包中相似的简单系统。

书中给出了具体程序，用户可以用它来开发：

- 基于窗口的屏幕接口。
- 移动亮条式菜单（和 Lotus1-2-3 一样）。
- 多层亮条式菜单。
- 弹出式菜单。
- 对话框。
- 下拉式菜单。
- 字段编辑器。
- 数据输入屏。
- 表选择窗。
- 目录函数（文件选择时使用）。
- 上下文有关的帮助屏。
- 帮助屏编辑器。

在建立这些系统的过程中，本书涉及到了如下的概念和技术：

- 执行 BIOS 中断；
- 直接高速写屏幕缓冲区；
- CGA 显示器中的雪花预防；
- “Instant”（立即）用虚拟更新屏幕；
- 虚屏幕的创建和操作；
- 和 DESQview 等多任务环境的兼容性保证；
- 用户友好菜单的设计；
- 数据输入和验证技术；
- 获取 DOS 目录信息；
- 帮助屏的创建和显式

目标

本书旨在为读者提供：

1. 可以用原样使用或根据不同程序员的需求进行修改的程序。
2. 各个程序中的概念。
3. 用来开发这些程序的编程技术讲解。

可移植性

贯穿本书的始终，力求保证 MS-DOS / IBM-DOS 下的可移植性。

对于程序的用户来说，这意味着程序首先是可以运行的。这些程序不需要任何象 ANSI.SYS 一样的特殊驱动程序及各式各样的 Config.sys 文件，而且也无需购买任何特定的窗口化操作系统或 DOS 外壳。程序可以在所有现行的显示卡上正常运行 (CGA, MDA, Hercules, EGA 和 VGA)。这样的软件才能有最广泛的应用，而且实际上适用于任何一种 IBM PC 及其兼容机。

对于程序员来说，可移植性意味着对编译器的选择。我们没有使用依赖编译器的图形 / 窗口库。编程代码是与存储模式无关的。换言之，所有程序都可以在 Turbo C 和 Quick C 的任一种存储模式下进行编译。

程序全是用 C 编写的，不需要任何汇编程序。注释掉的源代码可以根据程序员的需要进行修改、使用。

使用本书用户需要的东西：

1. 一个 C 编译器。本书使用 Borland 的 Turbo C 编译器开发样例程序。这些源代码全部兼容于 Microsoft 的 Quick C。
2. 一台具有足够内存运行编译器的 IBM PC 或兼容机。
3. IBM 或 MS-DOS
4. 有用 C 编程的经验。无需高级的知识。

运行软件需要的其他东西：

1. 一台 IBM PC 或兼容机；
2. 64K 可用 RAM。若用户的应用程序很大时可能需要更多空间；
3. IBM 或 MS-DOS；

本书的层次：

本书共分成七章。

1. 第一章回顾了视频适配器、存储器的层次和存取，以及指针和 BIOS 中断的使用。
2. 第二章讨论如何创建一个基于窗口的屏幕函数库。涉及的论题有创建，删除和写窗口；灵活的屏幕写例程的建立；以及屏幕缓冲区的“分页”技术。本书还讨论了虚拟

屏幕和 DESQview 的兼容性问题。

- 3.第三章讨论菜单系统。我们将创建Lotus1—2—3风格的菜单亮条、多层菜单亮条、弹出式菜单、对话框以及下拉式菜单。最重要的一节是弹出式菜单。在这里我们给出了数据结构、调用规则和在所有菜单中都要用到的菜单设计原理。
- 4.第四章讨论数据输入。我们将建造一个字段编辑器并用它构成一个dBASE风格的输入屏，而且具有编码颜色字段和数据验证功能。
- 5.第五章将建立一个表选择函数，允许用户在不定数的选项中进行选择。这里我们使用了指点和快速搜索技术。
- 6.在第六章我们建立了一个从某目录中选择文件的函数。它用到了前面建立的表选择函数。
- 7.在第七章，我们建立了一个帮助屏设计器并且讨论了如何将上下文有关的帮助增加到我们的程序中。

每个主论题都分成四个子部分：

- **目标 (Goals):**

讨论代码中将要实现的主要目标和特性。这部分描述了产品的总体概貌和特点。

- **应用 (Appliation):**

它是本书的实践部分。讨论每个函数及其在程序中的使用。为便于说明给出了样本程序。还描述了最佳使用这些函数的一些技巧。

- **技术 (Techniques):**

本节讨论编码时的各种设计考虑因素和技术。我们讨论了各种可能的编码方案、优缺点以及解决方法。重点放在如何更直观测试源代码的方面。

- **源代码 (Source Code):**

尽量给出代码注释。并竭尽所能使注释和代码实用，易读。尽可能使用有含义的变量名（例如，top=1而不是用t=1）。

这种模块化实现方案便于用户更多地将注意力放在最有用的部分。假如用户对建立一个屏幕和菜单工具集感兴趣，那么就可能会常看“应用”部分。假如只对代码如何工作有兴趣，则就看“技术”和“源代码”两部分。

获取源代码

请参见附录 A 给出的获取本书中代码的建议。

第一章 存取显示适配器

本章将要回顾的是：

- 显示适配器的类型；
- 视频缓冲区存储器映象；
- 访问视频存储区时的指针使用；
- 各编译器存储模式的缺省指针类型；
- BIOS 中断；
- ANSI 控制台驱动程序；

第一节里讨论显示适配器，并且着重介绍视频缓冲区和用来访问它的技术。

1.1 显示适配器

适用于 IBM PC 的各种现行的监视器和文本 / 图形卡大体有两类，即单显和彩显。

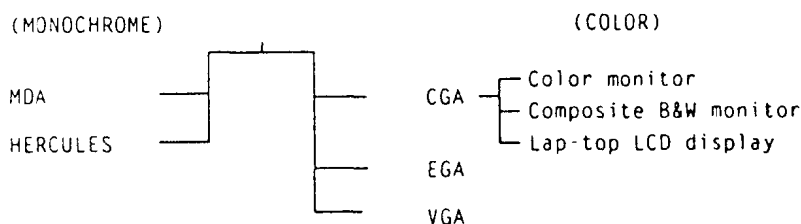


图 1-1 主要显示适配器的演变图

其演变树如图 1-1 所示。如下描述各种主要的适配器：

- MDA = 单色显示适配器。这种显示卡主要用于商业应用。它只能显示严格定义的文本字符而不能显示图形。正文以 80×25 格式（80 列，25 行）显示。该卡要求使用单色监视器。
- Hercules = 该卡的设计是与 MDA 兼容的，但它能显示特定的高分辨率图形。这种卡也要求使用单色监视器。
- CGA = 彩色图形适配器。这种卡即可显示彩色正文又可显示彩色图形。正文显示的限制为 80×25（最大范围）。
- EGA = 增强型图形适配器，该卡与 CGA 兼容但其正文和图形分辨率更高。它支持大于 80×25 的正文模式。
- VGA = 视频图形适配器。与 EGA 兼容，但有更多的正文和图形模式。

LCD = 液晶显示器。只在便携机上使用，这种卡通常是与CGA适配器等价的。

更先进的高级显示卡都支持早期显示卡的所有功能和模式。例如，VGA卡可以运行CGA写的程序，而Hercules卡可以模拟MGA。如果开发的软件与CGA和MDA兼容，那么自然可以与更先进的显示卡兼容。

1.2 混合显示器

可以注意到CGA适用于两种显示器：彩显和黑白显示器（B&W）。黑白混合显示器以灰色阴影表示颜色，而且颜色很有限。

混合显示器上显示的彩色字符通常是不可读的。现有便携机使用的LCD大都可归类为混合显示器。因为大多LCD监视器使用灰度值而不是颜色。

对这种B&W显示器用户常使用DOS的“mode”命令来取消其颜色，只允许显示正常的或高亮的正文。在DOS命令行中打入“mode bw80”就把显示模式设置为B&W（黑白），80列。

好的软件应该使用BIOS调用来检查显示模式以确定用户选取的模式。以该模式为基础，软件中可以为显示器选择适当的正文属性（例如，有无颜色）。不幸的是，如果某应用程序直接写视频缓冲区，它就绕过了BIOS并且不受“mode”命令的影响，因此在不适合用户监视器时仍然可能写入带有颜色属性的正文。

虽然软件能够检测现有显示适配器的类型，但它无法“辨别”与该适配器连接的显示器的类型是彩色的还是混合型的。用软件只能确定设置的模式是否为B&W模式。

在后面章节中，将讨论确定显示适配器类型和当前选取的显示模式的技术并给出相应代码。这些信息对正确进行显示是很必要的。

1.3 视频缓冲区

IBM PC使用一部分内存作为视频缓冲区。CGA和CGA兼容卡的缓冲区从内存b800段开始。单色卡（MGA，Hercules和其他）使用的缓冲区从b000开始。第一个存储单元放显示的第一个字符，而下一个单元包含该字符的显示属性。这种安排方式称为Memory mapped display（内存映象显示），视频缓冲区内容的任何变化都立即反映在显示器上。

如果要在CGA显示器的左上角显示“hello”，则其相应的存储映象如图1-2所示。整屏80列，25行显示包括2000（80×25）个字符和相应的2000字符属性。使用行列（x，y）的屏幕定位系统的话，左上角就是（1，1）而右下角就是（25，80）。有关视频存储器映象的知识对后面开发屏幕写技术是十分必要的。

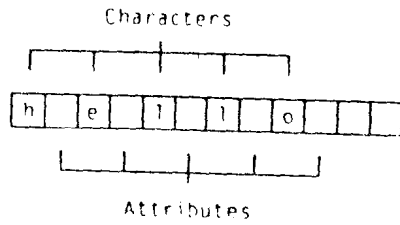


图 1-2 存储器映象

1.4 正文属性

正文属性随使用的显示卡的不同类型而变化。单显中可以使用的属性有标准字符、加亮、反显和下划线。本书中的程序在 `mylib.h` 中定义了这些属性以便使用。

```
#define UNDERLINE 1 /* ATTRIBUTES FOR MONOCHROME CARDS
*/
#define NORMAL 7
#define HI_INTEN 15
#define REVERSE 112
```

在彩色适配器中，属性字节用来设置前景/背景颜色以及前景高亮或闪烁字符等特性。颜色属性图见图 1-3。

用三种原始颜色（即红、蓝和绿）可以合成所谓的“复合”颜色。例如，红和蓝可以组合（相加）成粉色；红和绿组合为黄。粉色、青色和黄色也可以同样进行组合（此时是相减）形成原始色。例如，青色和黄色相减形成绿色。“复合”色系统在图象放大时使用。其相互关系参见图 1-4 的颜色轮。

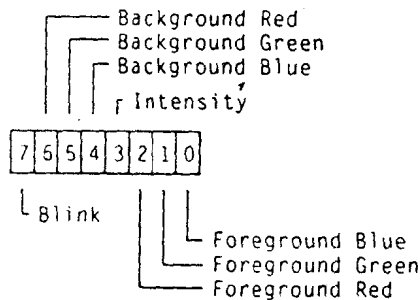


图 1-3 颜色属性的位图

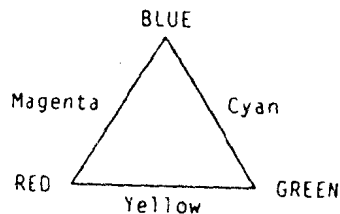


图 1-4 颜色轮

颜色轮仅供参考。实际产生的颜色取决于红，蓝，绿，和高亮度的组合情况。例如，如果没有置高亮位时红和绿产生棕色。若高亮位置 1 时其结果就是黄。表 1-1 给出了由原色组合（高亮位置 0）出的颜色的位图。为简化颜色属性的使用，在 mydef.h 中定义如下：

```
#define BLACK    0 /* THESE ARE FOR COLOR CARDS */
#define BLUE     1
#define GREEN    2
#define CYAN     3
#define RED      4
#define MAGENTA  5
#define BROWN   6
#define WHITE    7
#define YELLOW  14 /* intensity set on */
```

如下函数模样的宏设置前景/背景颜色：

```
#define set_color (foreground, background) (((background) << 4) |
                                           (foreground))
```

注意，该宏是将背景值左移 4 位，再“或”上前景值。

例如，利用这个宏，变量 attribute 将前景色置为 BLUE，背景色置为 BLACK，其调用如下：

```
attribute = set_color (BLUE, BLACK);
```

当任何属性的第四位为 ON（置为 1）时，高亮位就置成了 HIGH。我们可以用宏 set_intense（）来置高度位为 ON，该宏将属性值“或”上一个十进数 8（即二进制的 00001000）。这就把高亮位置为 1 了。

```
#define set_intense (attribute) ((attribute): 8)
```

表 1-1 主要颜色的位图

| R | G | B | Color | Attribute (decimal) |
|---|---|---|---------|---------------------|
| 0 | 0 | 0 | Black | 0 |
| 0 | 0 | 1 | Blue | 1 |
| 0 | 1 | 0 | Green | 2 |
| 0 | 1 | 1 | Cyan | 3 |
| 1 | 0 | 0 | Red | 4 |
| 1 | 0 | 1 | Magenta | 5 |
| 1 | 1 | 0 | Brown | 6 |
| 1 | 1 | 1 | White | 7 |

1.5 彩色正文分页

CGA 允许显示四页 80 列模式的正文。使用视频中断就可以在计算机屏幕上显示任一页正文。

前面提到 80×25 的屏幕显示缓冲区占 4000 字节，但该缓冲区的偏移量不是所希望的 4000，而是 4096。如果不是用 16 进制表示数值这点是毫无道理的。从表 1-2 和图 1-5 可以看出页以 1000H 的步长依次排列，每页都有一小块内存区没有使用。这样划分页边界使访问缓冲区时的计算更方便、简单。

表 1-2 各页实际所占的空间

| | Decimal | Hex |
|--------|----------------|--------------|
| Page 0 | 0 to 3999 | 0 to f9f |
| Page 1 | 4096 to 8095 | 1000 to 1f9f |
| Page 2 | 8192 to 12191 | 2000 to 2f9f |
| Page 3 | 12288 to 16287 | 3000 to 3f9f |

初始情况时，计算机读写第 0 页 (Page 0)。一些程序员常使用的一种技巧是正文页切换，也就是说，写一页同时显示另一页。这样就使屏幕更新几乎是新瞬即毕。图 1-6 给出了一个例子。当然，各页所需的写时间是相同的，但用户不再感到屏幕象通常那样一行行地重写。整页立即“弹出”到屏幕上给用户一个快速的错觉。

字处理器使用正文页保存当前，后一页或上一页的正文。当用户按一个换页键时就显示相应的正文页。因用户读新一页的同时写另一页。软件利用了“空闲”时间，这时反应迟钝的我们还在决定下一步干什么呢。用户根本没有意识到幕后的活动。

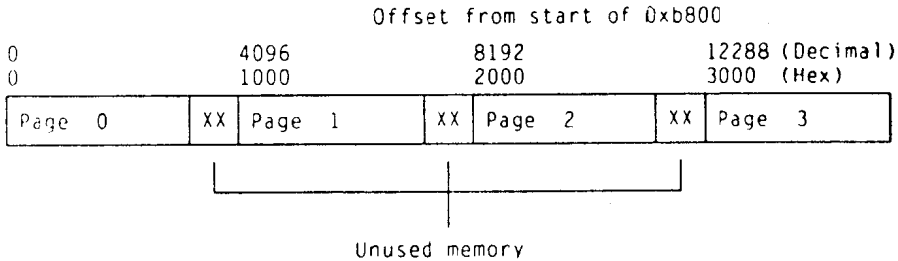


图 1-5 屏幕缓冲区

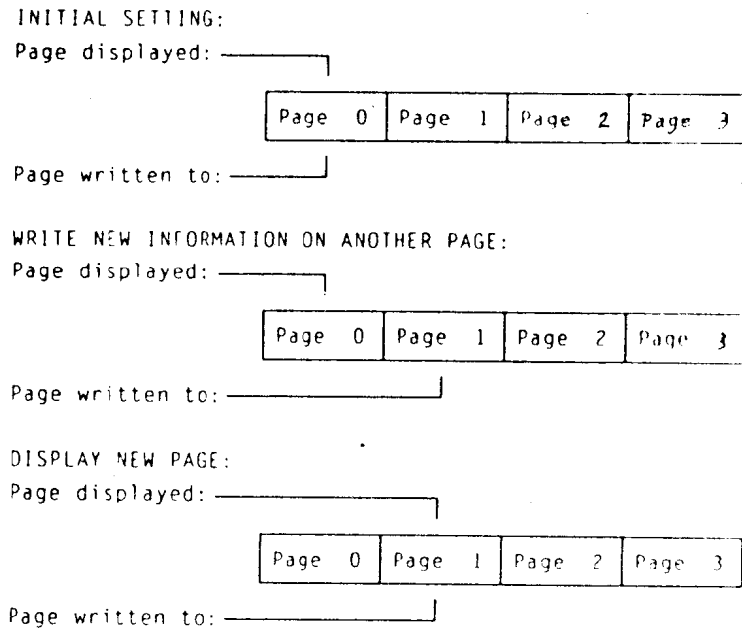


图 1-6 页切换的样例

如果用户的字处理器看似可以立即换页，那它很可能采用了这种技术。为了验证这点，用户可以调入它，按住 PgDn 键并观察屏幕更新是否仍是即时的。如果该程序在按键期间没有时间写下一页，那么在写下一页时就会产生一个停顿。

虽然正文页切换很有用，但我们没有采用这种技术，有如下两个原因：

1. 我们的代码必须支持单色显示器，它不能采用这种技术。
2. 有些时候，需要多于四页的正文页。

不采用分页技术，我们将采用一种叫虚拟屏幕的技术。

1.6 虚拟屏幕

虚拟的意思是“想念”或“虚构”。例如，一种 RAM 磁盘驱动器也称为虚拟驱动器。

使用虚拟屏幕技术可以实现与页切换同样的效果。一个虚拟屏幕是分配的一片连续存储空间。通常与物理屏幕缓冲区大小相同。屏幕写直接对该缓冲区操作而不是写屏幕缓冲区。写完成后再将虚拟屏幕缓冲区拷贝到物理屏幕缓冲区。

使用虚拟屏幕的一个小缺点是需要额外的存储空间而且需要时间来创建和操作该缓冲区。但是，“即时”的屏幕更新效果是可以弥补这些额外的开销。

1.7 指针和存储模式

直接操作内存缓冲区可使屏幕函数的效率得以优化。为操作内存缓冲区，透彻了解指

针的使用和它与编译器存储模式的关系是至关重要的。

特定的 16 位寄存器用来指定代码段和数据段。这些段寄存器在表 1-3 中列出。

段寄存器通常是汇编语言程序员要考虑的而 C 程序员很少注意它们。但是，我们确实应该了解段如何影响我们选取的编译存储模式。Turbo C 有 6 种模式：tiny, small, medium, compact, large 和 huge，所选取的存储模式决定了段的使用。

Tiny 模式对所有四个段寄存器使用同一起始地址；代码和数据占据相同的 64k 的空间。这意味着 tiny 模式可以使用 near 指针来访问数据。所有数据都存放在程序已知的同一 64k 的段内。near 指针只需包含该数据的偏移量即可。

大存储模式使用 far 指针访问数据。far 指针必须同时包含段址和偏移量，因此要比 near 指针大。大模式可以访问 1 兆空间内的数据。

tiny, small 和 medum 模式使用 near 指针访问数据而 compact, large 和 huge 模式使用 far 指针。除非程序中需要大量数据空间。用户最好使用小存储模式，因为它占的空间少且速度快。

表 1-3 段寄存器

| Register | Name | Usage |
|----------|---------------|----------------------------------|
| CS | Code segment | Program code |
| DS | Data segment | Data storage |
| SS | Stack segment | Stack location |
| ES | Extra segment | As needed, usually for more data |

对某些 C 应用程序来说无需考虑缺省指针的类型。缺省指针由存储模式指定，通常可以正常进行存储分配，或作变量指针使用。但是在处理视频缓冲区时，情况却大不相同。

1.8 直接存取视频缓冲区

视频缓冲区必须用 far 指针存取。我们必须确实使用 far 指针。如下程序中使用缺省指针向屏幕写一个字符：

```
main()
{
    char *screen;
    char ch;

    screen=0xb0000000; /* Monochrome screen address */
    ch='a';

    *screen=ch;
}
```

这个程序在大存储模式下可以编译运行，因为此时的缺省指针类型就是 far。但对小模式该程序不能正常运行因为小模式使用 near 指针并且不能访问定义的数据段外的空间。

为使程序适用于任何存储模式，可以作如下修改屏幕指针的声明：

```
char far * screen;
```

描述符 `far` 不是标准 C 语言提供的，但许多编译器包括 `Turbo C` 和 `Quick C` 都支持这种声明。

另一个必须多加小心的地方是 C 库函数的使用，有些函数使用缺省存储模式指针而且可能不象用户希望的那样工作。例如函数 `movmem()`，它用来从一个存储单元移动字节到另一存储单元。它要有三个参数：

- 源指针
- 目标指针
- 移动的字节数

象 `movmem()` 这样的库函数可以用来一次移动大批量数据（比如拷贝视频缓冲区到另一存储区）。这些函数是用效率很高的汇编语言写成的，比在 C 程序循环中逐个访问数据要快的多。函数 `movmem()` 也可以用来将单个字符送入视频缓冲区。例如，

```
#include "mem.h"      /* header file for movmem */

main( )
{
    char *screen;
    char ch;

    screen=0xb0000000; /* Monochrome screen address */
    ch = 'a';

    movmem(&ch,screen,1);
}
```

该程序可以在大模式下正确运行，但在小模式下不行。和前面的例子不同，即使把指针改成 `far` 也毫无用处，`movmem()` 函数只接收缺省大小的指针。除非想使用大存储模式，用户不要使用该函数来存取视频缓冲区。

为移动大批量数据最好选用 `movedata()` 函数。它要求提供源段址和偏移量，目标段址和偏移量，以及要移动的字节数。`movedata()` 的调用格式如下：

```
movedata (source__seg, source__off, dest__seg, des__off, size);
```

用户可能会认为使用 `movedata()` 函数可以很容易将一个串直接送入视频缓冲区。但问题是视频缓冲区要求其中的字符由属性字节分开。任何要利用 `movedata()` 送入屏幕的串必须首先装配上属性字节。拷贝屏幕缓冲区到另一存储区的函数无需进行装配工作。一个这样的例子就是拷贝虚拟屏幕到屏幕缓冲区。

1.9 BIOS 中断

IBM PC 及其兼容机在 ROM 中固化了许多实用的函数例程。这些基本输入输出服务程序（或称 BIOS）可以处理读键盘和控制屏幕输出的许多细节。

BIOS 中包含的例程有清屏、移动光标和向屏幕上写正文。BIOS 旨在使程序员不必了解硬件的细节。例如，BIOS 光标移动例程适用于任何一种显示卡，无论是单显卡，彩色图形卡，EGA 或 VGA 都适用。程序员可以不考虑其内部实现细节而很好地使用 BIOS 例程。在新机器中为处理新的硬件 BIOS 例程进行更新。为保证新的 BIOS 和旧版本的兼容性需要做大量工作。基于 BIOS 开发的程序在理论上讲应该与新机器兼容。BIOS 例程通过 8088 中断调用。每个 BIOS 函数都有一个相应的中断号。无需了解该例程的物理位置再调用该例程，用户只需知道其中断号即可，新的 IBM 机及兼容机只要保证各函数的中断号一样，可以自由安排各函数的物理位置。

BIOS 例程的实际位置保存在一张中断向量表中。表中的每个位置包含相应例程的地址。执行一个软中断时，就指定了一个 BIOS 例程的中断号。8088 在中断表中找到该例程的地址，然后调用该例程。

使用 BIOS 中断是保证与多任务系统和未来 DOS 版本兼容性的一种好方法。以多任务系统为例，象 DESQview 等基于窗口的系统可以“窃取”（重定向）中断并根据自己的需要执行该中断。在屏幕上显示正文的 BIOS 调用可以被改成重定向输出到 DESQview 虚拟屏幕。因此通过 BIOS 完成的应用程序可以被控制并连接到某个 DESQview 窗口，然后在后台运行。

如果应用程序直接写视屏，而不是用中断调用，它就可能“冲掉”其他窗口。DESQview 无法截取直接写入视频缓冲区的正文。在第二章我们将看到直接访问 DESQview 缓冲区的方法。

1.10 执行一个中断

现在来看看在 C 程序中使用 BIOS 中断的技术。

标准 C 语言不提供对 DOS 中断的支持。幸运的是，大多数为 IBM PC 设计的 C 编译器都有一个特定的函数来完成这项工作，在 Turbo C 和 Quick C 中该函数是 `int86()`。它是许多基于 MS-DOS 编译器提供的一种扩展功能。

在调用 BIOS 例程前，必须为 8088 通用寄存器装入相应的值。这些值指定申请调用的特定功能，该函数所需的参数。我们用一个类似 8088 寄存器的数据结构来指示 `int86()` 如何加载寄存器。让我们仔细观察一下这些寄存器，以便更好地理解其结构。