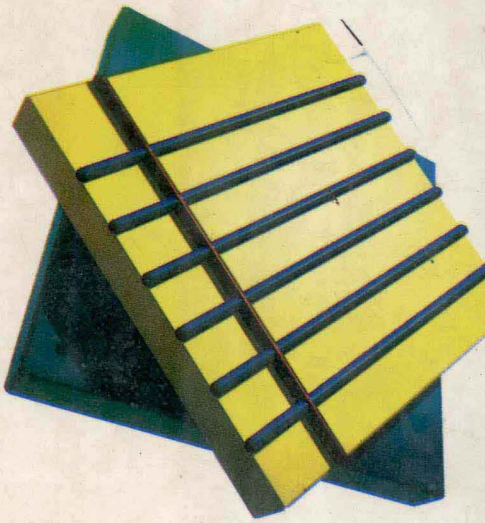
计算机 C/C++语言系列丛书



梁 翎
李爱齐 编著

C 语言用户接口
编程技术与实践

学苑出版社

计算机C/C++语言系列丛书

C语言用户接口编程技术与实践

梁　翎
李爱齐　编著
熊可宜　审校

学苑出版社
1994

(京)新登字 151 号

内 容 提 要

本书全面介绍计算机屏幕用户接口技术，并以目前流行的 Turbo C 和 Quick C 作程序示范，提供了大量的相关源程序，可以直接引用。

全书分为八章：其内容分别是：序言；显示适配器访问技术；窗口及屏幕函数建立；菜单设计方法；数据输入屏幕设计；列表选择技术；目录文件显示；帮助屏幕建立。

本书适合于计算机专业 C 语言软件编程员，用户系统管理员之需要。

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

计算机 C/C++语言系列丛书

C 语言用户接口编程技术与实践

编　　著：梁　翎　　李爱齐
审　　校：熊可宜
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：北京市太和印刷厂
开　　本：787×1092　1/16
印　　张：15.25　　**字数**：351.234 千字
印　　数 1～5000 册
版　　次：1994 年 5 月北京第 1 版第 1 次
ISBN7－5077－0875－6/TP・24
本册定价：18.00 元

学苑版图书印、装错误可随时退换

前　言

用户接口技术是现代计算机程序设计技术一个重要方面，特别是当今提出的“图形用户接口”(GUI)使得这一技术在应用程序中显得越发突出。以往许多软件虽然从功能上来说很符合用户设计要求，但终因软件操作不灵活、画面呆板而不受用户喜欢，导致软件生命期短暂。友好的用户接口可以增强应用软件的活力，更使用户操作耳目一新。本书正是一本介绍用户接口技术程序设计的专著，包括屏幕接口建立、菜单接口设计、数据输入屏幕设计以及帮助屏幕建立等诸多内容。另外还特别对一些实用性技术，如直接写屏技术，雪花防止以及虚拟屏幕操作技术等做了详细介绍。本书以目前流行的 Turbo C 和 Quick C 作程序示范，同时在每章之后均附有与之有关的全部源程序，读者可以直接(或修改部分语句)引用这些源程序到自己的应用程序中。

由于编者的业务水平和时间有限，难免有欠妥之处，深望不吝赐教。

编　者

1994 年 3 月

序　言

用户接口又称用户界面或人机界面是计算机与用户(包括应用软件开发人员和最终用户)之间通讯的重要综合环境。近十几年来随着微型计算机的广泛普及与发展以及高分辨率图形显示技术、窗口、菜单等一系列技术成果的出现,做为人机交互技术的用户接口得到了重大发展,特别是以图形方式进行人机交互的图形用户接口(GUI),要求界面直观、简单、易学易用,因而深受广大用户的喜爱,并且成为用户界面领域的主流和方向。

本书即是帮助C语言程序设计者设计直观的函数性屏幕接口而编写的,主要包括屏幕和窗口函数库的建立,以及基于这些函数库而建立的各种菜单系统。

建立这些系统所涉及到的概念及技术有:

(1) 执行BIOS中断。

(2) 对屏幕缓冲区的高速直接写屏。

(3) CGA显示器防止雪花。

(4) 用虚屏产生"瞬息"屏幕修改。

(5) 虚拟窗口的建立与维护。

(6) 多任务环境(如DESQview)下的兼容性。

(7) 友好用户菜单设计。

(8) 数据输入与校对技术。

(9) DOS目录显示功能。

(10) 帮助屏幕的建立与显示。

本书所编写的程序力求在MS－DOS / IBM－DOS范围内可移植。这些源程序不需要像ANSI. SYS这样特定的驱动设备,不需config. sys文件的选择,也不需购买特殊的窗口操作系统或DOS外壳。所编软件将适应于CGA、MDA、Hercules、EGA以及VGA等所有流行的显示卡,这就是说软件适用于几乎所有的IBM PC及其兼容方式。对于程序员来说,可移植性意味着具有编译程序的选择。本书将避免使用特殊编译的图形/窗口库。在所需之处将自行编写建立。

使用本书所需的软硬件环境包括:

(1) 一个具有足够内存运行编译程序的IBM PC或兼容机。

(2) IBM－DOS或MS－DOS。

(3) 一个编译程序。本书采用Borland的Turbo C编译程序。书中源程序也完全兼容于Microsoft的Quick C。

(4) 具有C编程一般经验。不需更高深知识。

全书分为七章:

(1) 第一章是对显示适配器、内存分布与访问的回顾,以及指针的使用和BIOS中断。

(2) 第二章处理基于窗口屏幕函数库的建立。主要涉及窗口的建立、移动以及写入;灵活的屏幕写入例程的产生;以及屏幕缓冲"页"技术。同时还将讨论虚拟屏幕以及DESQview兼容

性问题。

(3) 第三章讨论菜单系统。在此将建立 Lotus 1—2—3 风格菜单条、多级菜单条、上弹菜单、对话框以及下拉菜单。其中特别重要的是上弹菜单部分;同时将建立用于所有菜单中的数据结构、调用协定以及菜单设计的方法。

(4) 第四章处理数据项。在这一章将建立一个字段编辑程序并用它来构造一个 dBASE 风格的输入屏幕,同时讨论数据校验的方法。

(5) 在第五章将建立一个列表选择功能,该功能允许用户使用触激以及急速搜索技术从一个无限项目中来选择。

(6) 在第六章将用第五章的列表选择函数建立一个从目录中选择文件的函数。

(7) 在第七章将建立一个帮助屏幕设计程序,并将展示如何向程序增加指定文本帮助。

目　录

第一章　显示适配器访问技术 …… 1
1.1　显示适配器 …… 1
1.2　黑白复合显示 …… 1
1.3　视频缓冲区 …… 2
1.4　文字属性 …… 2
1.5　彩色文字页及虚拟屏幕技术 …… 4
1.6　指针及内存模式 …… 6
1.7　BIOS 中断 …… 6
1.8　视频缓冲区直接访问 …… 7
1.9　ANSI 控制驱动程序 …… 9
1.10　小结 …… 10
第二章　窗口及屏幕函数建立 …… 11
2.1　窗口的概念 …… 11
2.2　函数原型 …… 12
2.3　外部变量 …… 12
2.4　窗口函数的类型 …… 14
2.5　窗口函数使用实例 …… 20
2.6　窗口类型 …… 24
2.7　窗口位置的优化 …… 26
2.8　保存初始屏幕技术 …… 30
2.9　窗口的隐藏 …… 30
2.10　窗口修改技术 …… 31
2.11　滑动虚拟窗口 …… 31
2.12　外部参数 …… 32
2.13　窗口结构 …… 35
2.14　直接写屏技术 …… 35
2.15　虚拟屏幕 …… 38
2.16　多任务环境兼容性问题 …… 39
2.17　雪花排除 …… 41
2.18　提高屏幕修改性能 …… 43
2.19　小结 …… 43
2.20　源程序 …… 43
第三章　菜单设计方法 …… 104
3.1　菜单的概念 …… 104

3.2 上弹菜单 …… 106
3.3 完全与部分菜单技术 …… 111
3.4 菜单工作过程 …… 113
3.5 get_key 函数作用 …… 113
3.6 源程序 …… 114
3.7 移动光条菜单 …… 122
3.8 源程序 …… 126
3.9 下拉菜单 …… 134
3.10 下拉菜单数据结构 …… 136
3.11 下拉菜单工作过程 …… 142
3.12 源程序 …… 144
第四章 数据输入屏幕设计 …… 155
4.1 绪论 …… 155
4.2 字段编辑程序特性 …… 155
4.3 数据输入屏幕编辑程序 …… 156
4.4 有关函数说明 …… 156
4.5 多字段数据输入屏幕 …… 157
4.6 字段编辑过程 …… 159
4.7 数据校对技术 …… 160
4.8 字段编辑屏幕显示 …… 162
4.9 源程序 …… 163
第五章 列表选择技术 …… 181
5.1 绪论 …… 181
5.2 列表选择函数 …… 181
5.3 源程序 …… 185
第六章 目录文件显示 …… 195
6.1 绪论 …… 195
6.2 目录文件选择函数 …… 195
6.3 源程序 …… 198
第七章 帮助屏幕建立 …… 205
7.1 帮助屏幕概念 …… 205
7.2 向应用程序增加帮助屏幕 …… 205
7.3 帮助系统 …… 207
7.4 建立帮助文件 …… 209
7.5 模块 readhelpc …… 211
7.6 源程序 …… 211
附录 函数索引 …… 234

第一章　显示适配器访问技术

1.1　显示适配器

目前适应于 IBM PC 机的各种显示器及文本/图形卡分为单色及彩色两部分，如图 1－1。

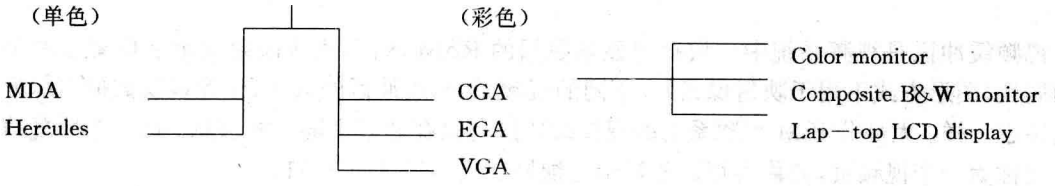


图 1－1　主要显示适配器类型

主要显示适配器说明如下：

（1）MDA ＝单色显示适配器。其文本字符显示非常好，但不能显示图形。文本以 80×25 格式（25 行 80 列）显示。这种卡用于分辨率为 720×350 的单色显示器。

（2）Hercules＝又称 HGC，它在单色显示卡上加上绘图功能，从而使得在同一个显示器上既能显示清晰的文本，又能显示图形。这种卡也需要一个单色显示器。

（3）CGA ＝彩色图形适配器。这种卡可同时显示彩色文本及图形。适用于 640×200 的彩色图形显示器，文本显示限于 80×25。

（4）EGA ＝增强图形适配器。这种卡与 MDA 或 CGA 兼容，提供较高的文本与图形分辨率，而且还可以做一些前两者所做不到的事情，故称增强型。它支持的图形方式为 640×350，文本方式大于 80×25。

（5）VGA ＝视频图形适配器。与 EGA 兼容，但具有更多的文本与图形方式，文本方式为 720×400，图形方式为 640×480。

（6）LCD ＝液晶显示。用于膝上计算机，这种卡通常等价于 CGA 适配器。

新型显示卡支持早期卡的功能与模式。例如，VGA 卡可运行为 CGA 编写的程序，Hercules 卡仿真 MDA。如果要使编写的软件兼容于 CGA 和 MDA，我们就要增强新适配器的兼容性。

1.2　黑白复合显示

CGA 可工作在彩色与黑白两种显示监视器中。黑白复合显示用灰度来控制。

在复合监视器上显示彩色字符通常是不可见的。许多膝上的 LCD 属于复合显示器，因为大多数 LCD 监视器使用灰级而不是颜色。

使用黑白显示器的用户通常使用 DOS 的“mode”命令来取消彩色，而只允许正常或增强

文本显示(其更加易读)。在DOS命令行输入"mode bw80"设置显示方式为黑白80列。

写得好的软件应通过BIOS调用检查显示方式以查明用户的选择,以便为那台监视器选择适合的文本属性(如彩色或非彩色)。

但是,假如一个应用程序采用直接视频缓冲区写入,由于它绕过BIOS而不受"mode"命令影响,因此即使该程序不适合于用户的监视器,也可以用彩色属性写文本。

虽然软件可以通过一定的手段探测到目前的显示适配器是什么类型,但它不能"知道"与适配器连接的监视器的类型是彩色还是黑白。软件只能查看是否已设置黑白方式。

1.3 视频缓冲区

视频缓冲区是视频系统中一块存放显示数据的RAM区间,用于映射显示在屏幕上的字符和属性(图形方式下用于映射像素)。不同的视频系统,缓冲区的大小、位置以及数据组织方法往往不一样。大部分IBM视频系统的视频缓冲区可以存储多个屏幕的信息。每一个屏幕信息空间称为一个视频页,简称为页。各显示适配器起止地址如表1-1。

表1-1 各显示适配器起止地址

MDA	B000:0000~B000:FFFF
HGC	B000:0000~B000:FFFF
CGA	B800:0000~B800:3FFF
EGA/VGA	A000:0000~B000:FFFF A000:0000~A000:FFFF B000:0000~B000:7FFF B800:0000~B800:7FFF

缓冲区第一个内存位置是在显示器上显示的第一个字符,接下来的位置是那个字符的属性。程序对视频缓冲区的读写与对RAM中其它区域的读写完全一样,而且对视频缓冲区的改写在监视器上只是一闪而过的。

图1-2是在CGA显示器上方显示"hello"的内存映射。整个80列25行显示器将包括2000个(80×25)字符加上2000个属性。显示屏幕编号左上角为(1,1)右下角为(25,80)。视频内存映射是开发写屏技术的基础。

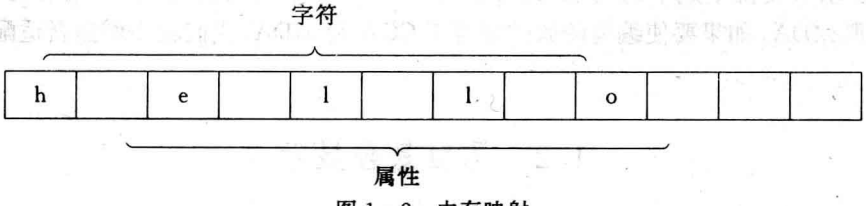


图1-2 内存映射

1.4 文本属性

文本属性根据所使用视频卡的类型有所不同。单色系统的属性包括正常、增强、反相以及

下划线。

在头文件 mylib.h 中定义了这些单色卡属性。

```
#define    UNDERLINE    1
#define    NORMAL       7
#define    HI_INTEN     15
#define    REVERSE      112
```

彩色适配器中属性字节用于设置前景/后景颜色以及前景增强和字符的闪烁。彩色属性如图 1－3。

三种基本添加色——红色、蓝色及绿色——相结合后产生所谓的复合色。例如，红色与蓝色加在一起产生洋红色；红色与绿色产生黄色。洋红色、青蓝色以及黄色可同样相结合以形成基本色。例如，青蓝色与黄色产生绿色。其关系如图 1－4 的色彩搭配。

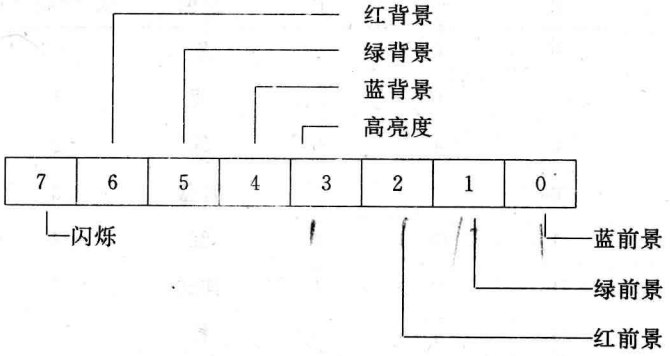


图 1－3　彩色属性位映射

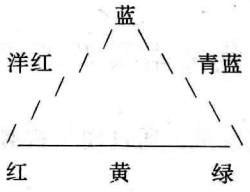


图 1－4　色彩搭配

色彩搭配只是一个基本说明。实际颜色产生决定于红、蓝、绿及增强的结合。假如没有增强设置，红色与绿色产生棕色。增强设置为 ON，结果是黄色。表 1－2 显示命名颜色位映射，其结果是基本颜色位被打开(增强置 OFF)。在 mydef.h 中有说明彩色属性的有关定义。

```
#define    BLACK      0
#define    BLUE       1
#define    GREEN      2
#define    CYAN       3
#define    RED        4
#define    MAGENTA    5
#define    BROWN      6
#define    WHITE      7
```

```
#define     YELLOW    14    /* 设置高亮度 */
```

以下宏定义函数设置前景/背景颜色：

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

在这个宏中首先移动背景4位，然后将其与前景相结合(逐位运算符“or”)。例如，通过以下调用使用这个宏，变量attribute设置成蓝色前景、黑色背景：

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

当第4位置为ON(置为1)对于任何属性增强被置为HIGH。通过使用宏set_intense()还可以对任何属性强制这位设为ON，该宏执行一个与数字8(相当于二进制的00001000)的逐位或(“or”)。使该位强制变为1。

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

表1－2 主要颜色位映射

R	G	B	颜色	属性(数字)
0	0	0	黑	0
0	0	1	蓝	1
0	1	0	绿	2
0	1	1	青蓝	3
1	0	0	红	4
1	0	1	洋红	5
1	1	0	棕	6
1	1	1	白	7

1.5 彩色文本页及虚拟屏幕技术

CGA以80列方式显示4个文本页。通过使用视频中断，可以在计算机屏幕上显示任何页。

前面提到过80×25显示器屏幕缓冲区占4000字节。实际上屏幕缓冲区中的内存位移不是4000，而是4096(十六进制数)。从表1－3和图1－5看出，页以十六进制数1000增加，每页占未用内存的一小块。采用这种段的开始页边界使得它更容易计算所需的缓冲区。

表1－3 文本页实际占据字节及段地址

	十进制	十六进制	段地址
0页	0到3999	0到f9f	B800h
1页	4096到8095	1000到1f9f	B900h
2页	8192到12191	2000到2f9f	BA00h
3页	12288到16287	3000到3f9f	BB00h

计算机最开始对0页进行读写。一些编程员使用的巧门是文本页触发；也就是说，对某一页进行写的同时显示另一页。这就使得修改屏幕显得几乎是瞬间。图1－6显示了这个例子。

当然，对间隔页的写入要稍微长些，但用户看不到通常发生的那种逐行写现象。即刻“弹”入屏幕的一个完整页给用户一个错觉。

字处理程序使用文本页来存储当前页、后一页和前一页的资料。当用户触及一个翻页键，相应的文本页就被显示出来。正当用户观看新的页时，另一页被重写。用户查觉不到脱屏现象。

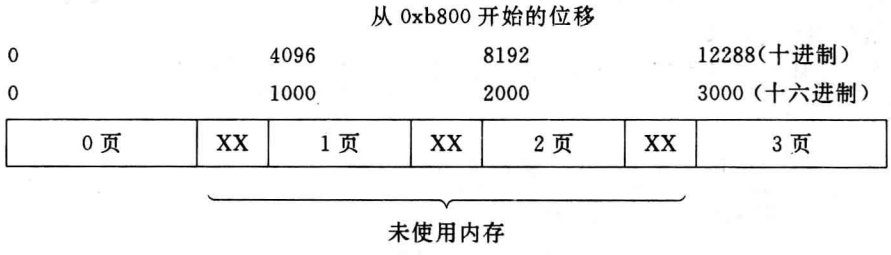


图 1－5　屏幕缓冲区

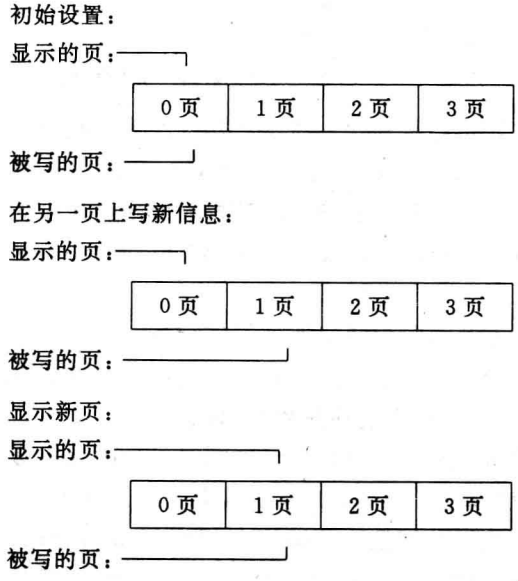


图 1－6　页滑动的例子

如果一个字处理程序能迅速处理文本翻页，它可能就使用了这一技术。通过按住向下翻页键(PgDn)过载来查看修改是否是瞬间。如果程序在按键之间没有时间去取下一页，那么在下一页被取入之前会有一段暂停。

由于文本触发技术不支持单色显示器，故我们在程序中不使用它。

除了页技术，我们将使用一个称为虚拟屏幕的技术。虚拟意味着在计算机中“虚构”或“做假”。例如，一个 RAM 磁盘设备被认为是一个虚设备。

虚拟屏幕可用于完成页触发同样的作用。虚拟屏幕是一个已被分配内存的连续区段，其大小与物理屏幕缓冲区相同。屏幕写入是被直接写到这个缓冲区而不是屏幕缓冲区。当完成写屏操作，虚拟屏幕缓冲区拷入物理屏幕缓冲区。

使用虚拟屏幕有一点不利那就是需要一些额外的内存和时间来建立和操作缓冲区。然而，“活泼的”屏幕修改补偿大于额外开销。

1.6 指针及内存模式

直接内存缓冲区操作对于执行屏幕函数提供了最佳条件。要操作内存缓冲区，重要的是要对指针的使用以及它们与编译内存模式的关系有一个全面地了解。内存模式的选取决定了代码和数据指针的缺省类型。

IBM PC及其兼容机中用专用的16位寄存器指定哪个段用于程序及数据。段寄存器如表1－4所示。

表1－4 段寄存器

寄存器	名 称	作 用
CS	程序段	程序代码
DS	数据段	数据存储
SS	栈 段	栈定位
ES	外部段	根据需要，通常用于大量数据

汇编语言程序员要比C程序员更多地关心段寄存器。在此，我们只需了解段是如何作用于编译内存模式的选择。C语言规定了三种指针：near指针(16位)、far指针(32位)和huge指针(32位，仅Turbo C有)。对于操作不同的程序段与数据段要指定相应的指针类型。有关指针的使用在此不再赘述。另外还有6种内存模式：tiny，small，medium，compact，large以及huge，所选择的模式决定了段的使用。

tiny模式各段(CS，DS，SS，ES)均取相同的地址值(即CS＝DS＝SS＝ES)；程序及数据占据相同的64K范围。也就是说tiny模式可用做一个near指针进行数据访问。所有数据在程序所访问到的相同64K段中，near指针只需包括对数据的位移。

large内存模式对数据使用far指针。far指针必须同时包含段及位移，因此要比near指针大。large模式可以访问1兆位的数据，因此适用于非常大的应用程序。

tiny、small以及medium模式对数据使用near指针，而compact、large以及huge模式使用far指针。除非应用程序需要许多数据空间，否则使用较小的内存模式情况更佳，因为它们占据较少内存而且速度较快。

对于特定的C应用程序没有必要关心缺省指针的类型。专用于内存模式的缺省指针通常十分适合于定位内存或变量指针。然而，当处理视频缓冲区，情况就相当不同了(详见1.8节)。

1.7 BIOS中断

BIOS是一组低级软件程序，被称为基本输入输出服务或BIOS。BIOS程序本身直接和外设通讯，它为软件人员提供了一个简单的与外设打交道的接口，其中包括处理键盘读取以及控制屏幕输入输出等操作。程序员只需简单地调用BIOS例程而不必关心内存在如何工作。BIOS例程通过8088中断访问。每个BIOS功能有一个与其相联的中断号，除了知道例程的物理位置，以及产生一个对那个位置的调用，程序员只要知道中断号。

BIOS例程的实际位置存储在一个中断向量表中。表中的每个位置包含有相关例程的地

址。当执行一个软件中断，即指定 BIOS 例程号。8088 在中断向量表中找到例程地址，并调用该例程。

使用 BIOS 中断对于保证与多任务系统及未来 DOS 版本的兼容性是一个好的方式。例如，一个像 DESQview 这样的多任务、基于窗口的系统可以重定向 BIOS 中断并根据所需处理中断。屏幕显示文本的 BIOS 调用可以修改为直接向 DESQview 虚拟屏幕输出。通过 BIOS 写入应用程序因此可以被控制及限制为对 DESQview 窗口，并可以在后台操作。

在 2.16 节将讨论如何对 DESQview 缓冲区直接内存访问。

1.8　视频缓冲区直接访问

存取视频适配器有三种方法。第一是通过 DOS 调用，但对窗口操作来说速度太慢。第二是 BIOS 调用，速度相对快些，但仍然不能进行大量操作。第三种方法是直接读写视频 RAM，速度非常快，但编程较为复杂。要实现快速屏幕操作，就必须直接存取视频 RAM。

读写视频缓冲区要求使用 far 指针。使用 far 指针有两种方法：一是使用 far 关键字来定义一个指针是 far 指针；二是使用 large 模式编译，这时所有指针均缺省地默认为 far 指针。在以下程序中缺省指针用于向屏幕写一个字符 'a'：

```
main()
{
        char *screen;
        char ch;
            screen=0xb0000000; /* 单色屏幕地址 */
            ch='a';
            *screen=ch;
}
```

通过改变屏幕指针说明可以使该程序在任何模式下正常工作，如下：

```
char far *screen;
```

访问视频缓冲区的另外一个方法是 C 函数库的使用。一些函数使用缺省内存模式指针。例如，函数 movmem()用于从内存的一个位置移动字节到另一个位置。它有三个参数：

(1) 一个指向原地址的指针
(2) 一个指向目标地址的指针
(3) 要移动的字节数

movmem()库函数可用于即刻移动大块的数据(如，拷贝视频缓冲区到另一个地方)。这种以高效汇编语言写成的例程比用 C 程序循环一个字节一个字节地访问要快得多。movmem()函数也可用于向视频缓冲区移动一个字符。例如：

```
#include "mem.h" /* movmem 头文件 */
main()
{
        char *screen;
        char ch;
```

```
        screen=0xb000000; /* 单显地址 */
        ch='a';
    movmem(&ch,screen,1);
}
```

该程序在large模式下编译。由于movmem()函数只接受缺省指针，改变指针为far指针不会有所帮助。

移动大块数据到视频缓冲区的一个较好选择是movedata()函数。它给出源段及位移、目标段及位移以及要移动的字节数。通过movedata()向屏幕传送字符，必须首先用属性字节填充。调用movedata()形式如下：

```
movedata(source_seg,source-off,dest_seg,dest_off,size);
```

在Turbo C以及Quick C中，BIOS调用功能是int86()。int86功能提供三个自变量：中断号(一个整数值)、保存功能引入时装入寄存器的值的地址域(union REGS型)、接收从功能返回寄存器的值的地址域(union REGS型)。在调用BIOS例程之前，8088通用寄存器必须装入标识哪个特定功能要被调用以及该功能所要使用参数的值。通过一个类似于8088寄存器的数据结构来告知int86()如何装载寄存器。

表1-5 BIOS寄存器

	8位(字节)	
	MSB	LSB
16位(字)	高位	低位
ax	ah	al
bx	bh	bl
cx	ch	cl
dx	dh	dl

每一个通用寄存器是16位宽(一个字)，可以分为两个8位"字节"。最左端字节被称为高字节，最右端字节被称为低字节。高字节和低字节有时分别叫做MSB(最高有效位)和LSB(最低有效位)。例如，十六进制字22ffH有一个MSB为22和一个LSB为ff。

为执行BIOS中断所需的寄存器是ax、bx、cx、dx，如表1-5所示。ah寄存器表示高位，al表示低位。

Turbo C和Quick C同时有代表这些寄存器字与字节的结构定义。结构以REGS组成一个联合。例如，Turbo C按以下方式定义联合：

```
struct    WORDREGS    {
          unsigned    int        ax, bx, cx, dx, si, di, cflag, flages;
};
struct    BYTEREGS    {
          unsigned    char       al, ah, bl, bh, cl, ch, dl, dh;
};
union     REGS        {
```

```
        struct      WORDREGS x;
        struct      BYTEREGS h;
};
```

可以看出，REGS 是一个两个结构的联合。使用 WORDREGS 就可以把 CPU 的寄存器当作 16 位数存取，使用 BYTEREGS 就可以存取单个的寄存器。

联合 REGS 在程序中用“union REGS regs;”用于说明变量 regs。以下是装载字节和字寄存器的例子。

例：

(1) 在程序中，使用语句“regs. h. ch＝5;”向一个寄存器的低位部分装载数字 5。也可以说是“regs. h. ah ＝(unsigned char)5;”，因为一个无符号字符相当于一个字节。

(2) 对于 ax 的字结构为 regs. x. ax。可以把寄存器看作为一个字，并以语句“regs. x. ax＝5;”同时装入两个字节。

int86()功能需要三个参数：中断号、要装入计算机中的数值以及存储中断返回寄存器值的地方。一些像读取光标位置的功能通过寄存器返回值。在装载寄存器之后，调用要执行的函数。

```
int86(0x10,&regs,&regs);
```

下面是一个通过 BIOS 调用移动光标的程序。移动光标的 BIOS 例程为视频中断号 10H。通过存储在寄存器 ah 中的值实现功能的指定。

光标功能还必须知道要移动光标的行列位置。通过在寄存器 dh 中装入行，在 dl 寄存器中装入列来完成这一工作。寄存器 bh 被装入文本页号，设定为‘0’。

注意，BIOS 以 0 开始计算行列；所以对于 BIOS 左上角(0,0)，在屏幕系统中是(1,1)。

```
void        gotoxy(int x, int y)
{
union       REGS        regs;
  regs. h. ah＝2;          /* ah＝2,光标位置函数 */
  regs. h. dh＝y－1;       /* 行 */
  regs. h. dl＝x－1;       /* 列 */
  regs. h. bh＝0;          /* 假设 0 页 */
int86(0x10, &regs, &regs);      /* 执行中断 */
}
```

BIOS 功能的确非常有用。但是，它们执行速度慢，而且 BIOS 在虚拟屏幕下工作不太灵活，并且有些功能只能工作在活动文本页中。像清屏、卷屏以及写屏这样的大多数屏幕功能通过视频缓冲区的直接内存访问可以以更快更加灵活的方式执行。以后讨用这种直接访问技术。

1.9　ANSI 控制驱动程序

ANSI 驱动程序是美国国家标准学会(ANSI)为标准屏幕控制而设计的规范，它允许软件在各种计算机系统中进行操作。

在 MS－DOS 计算机中，ANSI 驱动程序用在 CONFIG. SYS 文件中的"DEVICE＝ANSI.