HOPE

北京希望电脑公司

用MicrosoftCV6.0、QuickC语言V2.50

进行系统程序设计的技巧与实例

主要介绍系统程序设计的技巧，键盘和控制台函数，打印函数，文件管理函数，视频函数，交互式屏幕设计，独立于设备的图形函数，中断处理程序，内存驻留程序，扩展内存接口，鼠标接口，实用函数等。

用Microsoft C V6.0、Quick C V2.50
进行系统程序设计的技巧与实例

丹兵　静泽

北京希望电脑公司
一九九一年五月

■ 北京市新闻出版局
准印证号：3193—90193
■ 订购单位：北京8721信箱资料室
■ 邮　　码：100080
■ 电　　话：2562329
■ 传　　真：01—2581057
■ 乘　　车：320、332、302路
车至海淀黄庄下车
■ 办公地点：希望公司大楼一楼
往里走101房间

前　　言

本书是为使用 Microsoft C 的编程人员而写的，可以满足两种不同的需要：首先，对应用程序员来说，书中包含一个可随时调用的函数扩展集合，这极大地扩充了 Microsoft C 的运行时库；其次，对于系统程序员而言，本书提供了完整的源代码，并通过举例说明，对每个函数进行了详细的解释，用以体现 Microsoft C 的先进特点，并详述了如何在系统级使用 Microsoft C 以获得满意的效果。

本书展示出的函数和技巧可以在 MOCROSOFT C 优化编译或 Microsoft QuickC 中使用。文中把使用函数的说明内容同函数内部工作机制的分析内容分开描述，只对使用函数感兴趣的编程人员可跳过有关内部工作机制的分析内容，不必深入了解函数具体是如何实现的。所有的函数均设计了简单的用户接口，这样可以使用户在调用函数时不必关心其实现细节。

那些对 Microsoft C 的高级特性感兴趣的编程人员，以及希望使用 Microsoft C 作为系统编程工具的人，将会在第一章中找到关于诸如存储模式、汇编语言接口、系统服务调用及中断使用等有关内容的介绍，后续各章将依次展示各种有益的方法和诀窍、易犯的错误及防止误解的警告等内容。对于这些编程人员，函数起到了抛砖引玉的作用，而不仅仅是一个可调用的软件包了。

本书提供的软件工具是专门为 Microsoft C 环境开发的，利用该编译工具的许多独一无二的优点设计的函数易于使用，填补了 Microsoft C 运行时库的空白，并且为实用程序提供了重要的服务。本书提供了多组函数，它们的共同目标是通过直接访问 MS-DOS 机器中的许多重要的硬件和软件资源来提高 Microsoft C 的功能。

我们组织编译本书的目的是，希望能藉此给大量的使用 Microsoft C 的用户和编程人员提供有益的帮助，在本书的编译过程中得到梁卡、沈海以及谭军安同志的全力协助，尤其是秦人华老师的大力支持，在此一并致谢。书中不当之处，敬请指正。

译者

一九九一年四月

目　　录

第一章　技巧

本章主要讨论使用本书提供的函数库的基本技巧，同时也介绍一些用于扩展这些函数的 Microsoft C 编程方面的高级技巧和概念。在阅读和探究本章的各项主题之前，读者必须已经安装了 Microsoft C 编译器，5.0 版以后的优化编译，Quick C 集成环境，或者命令行系统；而且还必须熟悉它们的基本操作。(本书的使用者须熟练使用 C 语言与 Microsoft C 编译器；如果需要了解这方面的有关背景知识，请参阅相应的参考书)。

本章首先描述本书所有函数的总体组织结构。其次总述了在 C 语言程序中装配和使用这些函数的各种方法。然后讨论如何在 Microsoft C 各种内存模式中工作以及 C 语言程序如何与汇编语言函数接口。最后描述了使用系统服务（由 MS-DOS 和 BIOS 提供）的方法。

1.1.函数的组织

本书的函数分成几个独立的模块。模块意指一个内容相关的函数、数据结构和常量定义的集合。函数由其名字中的前三个字母标识其所属模块。下面是本书提供的所有功能模块：

前缀	章	用途
kdb	2	管理键盘
Ioc	2	管理控制台 I／O
Prt	3	管理打印机
Fil	4	管理磁盘，目录和文件
Scr	5	文本模式的屏幕 I／O
Buf	6	管理在窗口中显示数据的逻辑显示缓冲区
Gra	7	产生图形显示输出
Int	8	处理硬件和软件中断
Tsr	9	安装和管理内存驻留程序
Ems	10	分配和管理扩展内存
Mou	11	管理鼠标
Uty	12	实用服务

举个例子来说，函数 kdbGetC 属于 Scr 模块，用于从键盘读入一个字符，函数 ScrPutS 属于 Scr 模块，用于在屏幕的指定位置写一个字符串。

每个模块都由相关的文件组成。例如，kdb 模块由下列文件构成：

kdb 模块文件	内容
KBD.C	用C写的函数源代码，局部数据声明和常量定义。
KBDA.ASM	用汇编语言写的函数的源代码，局部数据声明和常量定义。
KBD.H	全局函数的原型，所有的全局数据声明和常量定义。

另外，本书为演示如何使用模块中的函数，在每一个模块中都提供了一组范例程序。例如 Kdb 模块有下列范例程序：

KBDDEMO1.C

KBDDEMO2.C
KBDDEMO3.C
KBDDEMO4.C
KBDDEMO5.C
KBDDEMO6.C

同样，就象下节所述，每个范例程序都有一个 MAKE 文件（用于在 Microsoft C 中用 MAKE 组织程序）。为范例程序还提供一个 Quick C 的程序目录（用于为 Quick C 集成环境生产 MAKE 文件）。

在某些模块中（如 Kdb，Scr，Tsr 和 Uty 模块中），有一个或多个模块是用汇编语言编写的。注意：在本书中的相关文件不管是 C 语言文件还是汇编语言文件都属于同一模块。

值得注意的是，本书中的函数在设计时支持抽象与隔离代码和数据的概念。因此，函数实现的细节和内部数据结构对函数的使用者（也就是调用程序）是隐藏的，给用户只提供了一个简单的通用接口，用户不需要知道函数的内部工作，也不必直接使用函数的内部数据。代码和数据的抽象代表着一种劳动的分工：系统程序员写函数，应用程序员使用它们。这样系统程序员可以在任何时候自由的改变函数的具体实现，同时通用接口（调用协议和共享变量）仍维持不变。

由此而来，函数和变量的声明、常量的定义、如果只由模块内部的函数自己使用，那么它们就只在由模块代码内部定义。进一步来说，这些局部函数和数据项使用 static 关健字，限制其它文件存取它们，（汇编语言文件中定义的数据不使用 PUBLIC 语句使之保持局部）。

相反，全局函数和数据声明以及常数定义，都与调用函数共享，并且放置在头文件里。同样，全局函数和数据项的声明不用 static 关健字，自动使它们可以由其它源文件使用。（汇编语言文件中，所有共公项目用 PUBLIC 声明）。例如，头文件 EMS.H 包含有全部的 Ems 模块的函数声明，可以由应用程序调用，同时全部的变量和常数定义也可以被调用程序使用。模块源文件（.C 和.ASM）代表函数模块的具体部分，头文件（.H）代表共公使用的接口。

1.2.使用函数

要使用本书提供的函数，必须经过下面三个简单的步骤使函数可以在程序中引用：

1.包含正确的头文件。正如前文所见，头文件按照相关模块的头三个字母命名。例如，要想调用函数 KdbReady，必须包含头文件 KBD.H，如果想调用 ScrGest，就必须包含 Scr.H。包含正确的头文件是绝对必要的前题，否则函数就不能正确的工作。正确地包含头文件的一个重要原因是：它包含了模块内部和可以外部调用的所有函数的原型。这些原型告诉编译器函数的数据类型和它们的参数。这些信息，例如，使编译器产生正确的 far 函数调用指令，并且使之正确的把一个 near 地址转换成 far 地址，并传送给声明为 far 的指针（见本章后文"使用内存模式"一节）。

2.按本书所描述的方式调用所需函数。

3.用正确的目标文件或文件链接程序。附录 A、B 列出的所有模块都需要目标文件。范例程序演示模块的功能，都附有 MAKE 文件，它们也列出了所需要的目标文件。例如，如果要调用函数 KdbGetSbift，就必须把 KBD.OBJ（目标文件由 KBD.C 产生）和 UTYA.OBJ（目

标文件由 UTYA.ASM 生成）链接入用户的程序，需要 UTYA.OBJ 的原因是因为源文件 KBD.C 中有几个调用函数在 UTYA.ASM 中定义，需要用 UTYA.OBJ。

注意，把所有函数的目标代码都置于一个库文件可以极大地简化上述的三步操作。链接程序能够不必在链接命令行指定每一个需要的目标文件，而找到相关的代码。准备这样一个库文件的过程见附录 B。

下面章节中的程序，MAKE 文件，和 Quick C 程序目录演示了使用这些函数模块的实际方法。同样，本章稍后的"装配函数和范例程序"一节，提供了更多的细节。

1.2.1.错误处理

许多（并非全部）本书提供的函数，支持调用函数有一个指示函数调用是否成功的错误代码。0 错误码总是表示函数调用是否成功，而一个非 0 的错误码总是表示发生了错误，错误代码使用下面介绍的两种机制之一传送给调用程序。

1.函数可以简单地把错误代码直接返回。例如：KbdInsert 如果成功则返回 0，如果有错误则返回 1。返回值可以用下面的方法测试：

```
int   Error;
.
.
.
Error = KbdInsert (65, 0)
if (Error)
/ * KbdInsert 失败，调用错误处理程序 * /
```

这种方法适合用于函数的返回值只反映错误，而不反映有错误状态。

2.函数返回一个特殊值，指明有错误发生；然后给全局的错误变量赋一个实际的错误码。例如函数 EmsAlloc，通常返回一个分配内存的基地址指针，如果函数失败，就返回一个 Null，并且把实际的错误码赋给全局变量 EmsError。EmsError 在头文件 EMS.H 中定义，可由调用函数使用。调用函数可用下面的方法测试错误状态：

```
char far  * PtrEms
if((ptrEms = EmsAlloc(4)) = = Null)
    switch(EmsAlloc)
{
    case NOTINSTALLED;
            / *  EMS 内存未安装  * /
           / *  调用错误处理程序  * /
    case ALLOCATED;
            / *  EMS 存储器好 * /
            / *  分配  * /
            / *   调用错误处理 * /
             .
             .
```

这种方法适用于函数不仅仅返回错误码的情况．特殊的整数值在指示错误时必须是在允许的范围之内，而不是在整数有意义的范围内．

不论是在描述函数的正文，还是在函数汇总的附录 A，错误报告协议对全部函数都一致的，注意，当函数有一个以上的非零错误代码时，模块在头文件中定义这些代码的常数（象刚才例子中定义于 EMS.H 中的 NOTINSTALLED 和 ALLOCATED）．

1.3.装配函数和范例程序：

如本章前文所述，本书为每个模块提供一个或多个范例程序．而且为每个范例程序都提了用 Microsoft MAKE 实用程序装配程序的 MAKE 文件（MAKE 文件带有.M 扩展名）和用 Quick C 集成环境装配程序的程序目录．MAKE 文件或 Quick C 程序目录，不仅装配相应的范例程序，而且还自动编译或汇编最新加入或修改的模块源文件．下例 MAKE 文件装配第五个 Kbd 模块范例程序 KBDDEMO5.C．

```
KBDDEMO5.OBJ:KBDDEMO5.C KBD.H
    Cl  /C  /W2 /ZP KBDDEMO5.C

KBD.OBJ:KBD.C  KBD.H
    Cl /C  /W2 /ZP  KBD.C

UTYA.OBJ:UTYA.ASM
    Masm /MX UTYA.ASM

KBDDEMO5.EXE: KBDDEMO5.OBJ  UTYA.OBJ
    Link /N01 /NOD  KBDDEMO5+KBD+UTYA,NULL,SLIBCE
```

这个 MAKE 文件首先编译范例程序 KBDDEMO5.C．（如果需要加入最新的 C 源文件和头文件）．因为 KBDDEMO5.C 调用文件 KBD.C（KbdSetShift 和 KbdGetShift）中的函数）MAKE 文件下一步生成目标文件 KBD.OBJ．而且，因为 KBD.C 文件调用 UTYA.ASM 中的几个函数．MAKE 也装配目标 KBDA.OBJ．最后，引用链接程序把所有最新的目标文件装配到一起，生成可执行文件 KBDDEMO5.EXE．更详细的有关 MAKE 实用程序的信息请参阅相关编译器的资料．

下面是本书中 MAKE 文件中命令行标志的小结：

标志	程序	用途
/c	CL	编译程序但不执行链接
/W2	CL	提供内部的编译警告，警告在传递给函数的参数和函数原型之间的数据转换的不匹配(类型或数量)　(原型丢失也警告)

/ZP	CL	压缩所有的结构也就是说，每个结构域的起始地址都处于第一个可用字节地址，而不是排到偶地址（压缩对于使用结构和操作系统交换数据时是很重要的，本书有几个模块是这样的）
/MX	MASM	保护公共和外部名的大小写情况（如果加这个选项，汇编程序把这些名字都换成大写字母，如果选择了链接程序/NOI选项就必须包含这一标志）
/NOI	LINK	引起链接程序区分大小写字母的情况（通常用于C语言，C语言是强调字母状态的语言，所以如果选择这一项的话，就必须选择/MX汇编选择项）
/NOD	LINK	阻止链接程序查找缺省库（编译程序在目标文件中写下缺省的库名）

注意，如果C语言运行时库使用标准名字（例如，小模式的库SLIBCE.LIB），就不需要选择/NDD选项，也不必在LIKE的命令行定义C库的名字，如果库不用标准名字，就必须象本书给出的文件一样，定义/NOD标志，并且包含库的名字。（这些MAKE文件定义库SLIBCER.LIB，系统使用装配函数和范例程序的小模式C库；库名加上R是区别MS-DOS的实模式库版本和OS/2保护模式版本）。如果不使用缺省的名字，就在MAKE文件中使用实际C库的名字。

1.3.1.使用 Quick C

如果准备用 Quick C的命令行编译器装配函数，可使用本书提供的MAKE文件，只是把每个CL命令换成qcl。所有的命令行选择项都能在Quick C编译器和链接器下正常工作。而且本书列出所有函数和程序都能用于Quick C命令行编译器。

如果使用标准的Microsoft MAKE实用程序（MAKE.EXE，带优化C编译器和宏汇编），就只需要在引用MAKE实用程序时指定MAKE文件。例如下面一行使MAKE实用程序装配范例程序KBDDEM01.EXE。

```
MAKE    KBDDEM01.M
```

如果，使用Quick C版本2.0支持的NMAKE程序，就需要指明MAKE文件（带/f标志）和要装配的目标程序的名字。例如：用下面的命令建立程序KBDDEM01.EXE。

```
NMAKE /f KBDDEM01.M KBDDEM01.EXE
```

如果希望在Quick C集成环境中装配范例程序，可以使用与程序一起提供的Quick C程序目录。例如，下面的程序目录提供给范例程序KBDDEM05.C（这个目录的作用和本书

前面提供的 MAKE 文件是一致的):

KBDDEM05.C

KBD.C

UTYA.OBJ

一但启动了 Quick C 集成环境（敲入 QC)。并且装入了范例程序，就可以开始装配这个程序和任何需要的函数模块.

1.进入 MAKE 菜单选择 Set Program List 项。(对于 Quick C 1.0 版，进入文件菜单选择此项)。

2.Quick C 请求输入文件名，敲入范例程序的文件名（不带.C 扩展名)。Quick C 询问是否产生 MAKE 文件（带.MAK 扩展名)，必须作肯定的答复。

3.下一步，按程序目录中的名字敲入全部的文件名，选择 Save List 块使 Quick C 生产并保存 MAKE 文件.

4.使用刚产生的 MAKE 文件装配和运行应用程序，进入 MAKE 菜单选择 Build Program 项，或 Run 菜单选择 Go (Quick C 1.0 版，进入 Run 菜单选择 Start 或 Compile)。

注意，Quick C 集成环境，对于不在 Quick C 核心库的相关函数都到 C 库中支找。它还假定这个库有标准名字。例如，安装一个小模式的程序，就去找库 SLIBCE.LIB（必须在 LIB 环境变量定义的路径中)。如果给这个库起了另外一个名字，就必须在程序目录中增加完整的路径名（包括.LIB 扩展)。(注意，在 Quick C 1.0 版集成环境中，所有程序都用中模式编译，因而 Quick C 就要寻找库 MLIBCE.LIB)。

另外还要注意汇编语言文件不能在 Quick C 集成环境中编译。然而，这些文件（象例子中 UTYA.OBJ)。用户在进入集成环境前预先处理目标文件:

masm /MX UTYA.ASM

可供选择的另外一个办法是，用户可以把汇编语言文件转换成带行间汇编指令的 C 源文件，这些程序只能在 Quick C 集成环境或 Quick C 命令行编译器中编译（优化编译不支持行间汇编)。关于行间汇编的进一步的更多的信息，请参见有关编译器的资料.

本书中大部分程序都能在 Quick C 环境中装配（除第 10 章 EMSDEM02.C，如果使用 Quick C 1.0 版就不行)。另外有些程序不能在环境中运行（第九章的内存驻留应用程序)。

1.3.2.产生库文件

如本章前文所述，装配范例程序最简单的办法是：把全部的函数模块的目标代码文件都放到一个单独的库文件中去，使用的办法在附录 B 中给出。一但生成了这个库文件，装配使用了任何含有库中函数的程序都可以只用一条命令。例如，函数的目标代码都置于名为 MSCTOOLS.LIB 的库文件中，可以用下面的命令装配范例程序 KBDDEMO5.C.

CL /W2 /ZP KBDDEMO5.C /LINK /NDD SLIBCE.LIB MSCTOOLS.LIB

注意，如果 C 库是标准名字、SLIBCE.LIB，可以省略命令行参数的 /LINK /NOD SLIBCE.LIB 部分.

在阅读本书的过程中，可在需要的时候把函数加入库中。另外，如果有与本书配套的磁

的库连接起来，就可以使用本书的函数。

1.4.存储模式

Microsoft C 和其它为 8086 系列所写的 C 编译器一样，把数据指针和函数分成 near 和 far，其根据是目标的寻址方法。这种划分对系统编程很重要，其来源是 8086 系列处理器的寻址方式。8086 系列的数据和代码的寻址包含有 16 位段地址和 16 位段内偏移量。(因为 16 位偏移量不能表达大于 64K 字节的值，所以 64K 是代码或数据段的最大值)。

如果某个程序的所有目标数据都在一个段内，就不必为每个变量保持段地址；因而数据指针也只需 16 位字节长 (使用数据时也不必每次都装入数据段的寄存器)。同样，如果所有函数都在一个段内，函数的寻址只需 16 位。函数调用也不必装入代码段寄存器 (这些用于 near 调用和返回)。这些数据指针和函数是 near 类型，以示区别。

如果某程序的数据大于一个段，那么数据指针必须包含段地址和偏移量，因而需要 32 位长。同样如果需要用多于一个段的空间来容纳某个程序的函数代码，函数的地址也必须是 32 位长，函数在调用和返回时需要装入和保存代码段寄存器 (这用于 far 调用和返回) 这些数据指针和函数是 far，以示与 near 区别。通常使用的程序都有多于 64K 的代码和多于 64K 的数据。

管理 near 类型目标的开销 (指代码段的大小和程序的执行速度) 要比 far 目标小。出于便于程序员最有效地使用存储模式的目的，Microsoft 优化 C 编译和 Quick C 提供了四种存储模式：Small，Medium，Compact 和 Large。(Microsoft 还提供 huge 存储模式，Quick C 1.0 版不兼容，参见编译程序的资料中的范例)。这些模式的通过命令行中 /AX 选择，X 是模式的第一个字母 (S，M，C 或 L)，选择的模式决定数据指针和函数的类型 (near 或 far)，以及程序段在内存中的整体组织。表 1.1 列出了每种内存模式的数据指针和函数类型以及段组织。

表 1.1Microsoft C 四种内存模式的特征

内存模式	CL 或 QCL 表标	数据指针	函数调用	段
small	/AS	near(16 位)	near	1 个代码段 1 个数据段
medium	/AM	near(16 位)	far	多个代码段 1 个数据段
Compact	/AC	far(32 位)	near	1 个代码段 多个数据段
Large	/AL	near(32 位)	far	多个代码段 多个数据段

注意，本书中的 MAKE 文件不指明内存模式，生成 Microsoft 优化 C (CL) 和

Quick C (QCL) 命令行编译的缺省模式：small 模式程序。如果想要从 Compact、medium 或 large 模式中调用本书提供的函数，必须重新编译函数的 C 源代码，指明与调用函数相同的内存模式（使用 /AX 标志）。下节将会看到，汇编语言的函数已设计成可以被任何模式的 C 语言函数调用，在改变存储模式时可以不必须改或重新汇编这些汇编语言文件。

注意，Quick C 1.0 版集成环境把程序都编译成 medium 内存模式，在环境中不能调整内存模式。

在选择内存模式之后，可以用关键字 near、far 改变单个数据指针或函数的缺省寻址址类型，例如表达式：

```
char far * video;
```

产生一个 far（32 位）指针，而不管是何种内存模式。这种指针对于小模式数据存取缺省的 C 数据段外的数据，（如显示缓冲）是很有用的，在本书中有许多函数和范例程序采用了这种方式。

注意，在 Quick C 集成环境（2.0 版）中编译的程序都使用 far 指针存取程序外的数据，用户必须关闭 Pointer Check 选择项（通过 options / Make / ComplierFlags 菜单项），否则结果程序在执行寻址指令时会产生一个运行时消息。

下面的函数演示了 far 关键字的两种其他的使用方法：

```
void far FarFunction (char far * FarParm);
```

首先，far 关键字置于函数名前强迫使用 far 调用该函数，而不管目前是什么内存模式。（如果 FarFunction 写在小模式 C 语言程序中，实际函数开始定义的函数类型也必须包含 far 关键字，这样编译器知道产生一个 far 返回指令。如果 FarFunction 是用汇编语言编写、就必须声明为 PROC FAR，或者带有 .MODEL MEDIUM 或 .MODEL LARGE 汇编指示）。

```
 1: .MODEL LARGE
 2: .CODE
 3:
 4: PUBLIC _KbdReady
 5: _KbdReady PROC
 6: ;
 7: ;      Prototype:
 8: ;           int far KbdReady
 9: ;                (void)
10: ;
11: ;    This function returns a non-zero value if a key is ready to be read, or
12: ;    zero if the BIOS keyboard buffer is empty.
13:
14:       mov  ah, 01              ;Specify the BIOS keyboard status function.
15:       int  16h                 ;Invoke BIOS keyboard services.
16:       jz   a01                 ;ZF set means keyboard buffer empty.
17:       mov  ax, 1               ;ZF not set; key is ready; therefore return 1.
18:       ret
19: a01:
20:       mov  ax, 0               ;Return 0 indicating no key ready.
21:       ret
22:
23: _KbdReady ENDP
24:
25: END
```

图 1.1 一个简单的汇编语言源文件