HOPE

用C语言编写DOS设备驱动程序

迪克 编译

用C语言编写DOS设备驱动程序

迪克　译

中国科学院希望高级电脑公司

一九九一年二月

■ 北京市新闻出版局
准印证号：3068—900068
■ 订购单位：北京8721信箱资料部
■ 邮　　码：100080
■ 电　　话：2562329
■ 传　　真：01—2561057
■ 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

目　录

第一部分　导论

第一章　引言

设备驱动程序是今天任何一个操作系统的必不可少的部分．它们实现了计算机系统所有附属设备的一个标准接口．换句话说，设备驱动程序提供了存取各种不同的硬件设备的公用机制．

执行 DOS(PC-DOS 或 MS-DOS)的个人计算机从设备驱动程序获益匪浅．例如，DOS 设备驱动程序让你用同样的命令和实用程序存取各式各样设备上的文件，如软盘、硬盘、CD-ROM 和 WORM．

没有 DOS 设备驱动程序，则每一 DOS 应用程序都要支持标准 DOS 所不支持的每一设备．这样，每当出现一种新设备，我们就得买喜爱的 DOS 应用程序的新版本，以获取对新设备的支持．

由于 DOS 已经发展了好几年，在设备驱动程序领域出现了许多重要的变化．在 DOS 2.0 版之前，不存在用户安装的设备驱动程序的概念．这样使得早期的个人计算机不支持许多设备．

DOS 2.0 版对以前版本作了许多改进．对于熟悉 UNIX 的人们，它减轻了痛苦，支持许多 UNIX 中的功能．最为相似的是设备驱动程序．它采用了 UNIX 的字符和块设备的输入／输出思想．每一设备族支持少量数目的设备族命令，如 READ 和 WRITE．

DOS 2.0 版还采用了设备专用控制的概念，即 I／O 控制(IOCTL)接口．IOCTL 接口即设备命令，允许应用程序给设备提供更为专用的控制信息，而不仅仅是通常可用的标准 READ 和 WRITE 命令．

DOS 设备驱动程序似乎能做所有的事情．事实上并非如此．DOS 设备驱动程序在个人计算机中只执行很特定的功能并且需要符合很严格的结构和格式．有关设备驱动程序的功能，结构和格式的细节在 DOS 技术参考手册中提供．我们在附录 B 中给出其压缩了的说明．

§1.1 背景

编写设备驱动程序是令人激动和富有挑战的工作．本书将教你如何设计和实现自己的 DOS 设备驱动程序．同时提供正确分析已有的 DOS 设备驱动程序以及设计自己的 DOS 设备驱动程序的知识和工具程序．

随着对本书的阅读，可以了解 DOS 设备驱动程序的细节．同时也可能更多地了解个人计算机的全部性质．这是因为设备驱动程序位于 DOS 核心和个人计算机的硬件之间．因此，对 DOS 设备驱动程序接口了解越多，则你的设备驱动程序的功能也越强．

类似地，对个人计算机硬件和设备驱动程序之间的接口了解越多，则你的设备驱动程序的功能也越强．

用从阅读本书、运行程序和提供的程序获得的知识和理解，就可以开始分析、设计、实现和调试自己的 DOS 设备驱动程序.

§1.2 预期读者

本书是为要加深对 DOS 设备驱动程序的理解的程序员而写的，而不是作为参考文献. 因此你应能阅读本书并使用所提供的例子.

本书中的所有例子都是用 C 程序设计语言编写的，因此要求对 C 语言有较好的理解. 此外，要尝试本书给出的例子，还要使用 C 编译程序和个人计算机的汇编程序. 我们是用 TURBO C 2.0 版和 TURBO Assembler 建立这些例子的.

由于 DOS 设备驱动程序主要处理附属于个人计算机的硬件设备的特性，因此使用个人计算机的硬件，包括外部设备的知识将有助于对本书的学习. 一般地，你需要随个人计算机来的、描述其全部细节的硬件技术参考手册. 不必记住手册中的信息，甚至不必理解全部信息. 但是，如果对所讨论的主题有所了解，将使对 DOS 设备驱动程序的理解容易一些.

由于我们接下来将讨论 DOS 和个人计算机硬件之间的接口，最好有一些使用 DOS 和个人计算机的知识. 有关 DOS 的最好信息来源是 DOS 技术参考手册(DOS Technical Reference Manual). 如同硬件技术参考手册提供计算机的详细信息，它详细讨论 DOS. 当然也不必把手册中的内容记在脑子里，但是如果对 DOS 的概念稍有了解，就可以使对本书的了解容易一些.

但是有一点必需明确，即使你从来没有读过个人计算机的硬件技术参考手册或者 DOS 技术参考手册，随着对本书的阅读也可以加深对硬件和 DOS 的理解. 当你读了本书的每一节并试了所提供的例子，对 DOS 和设备驱动程序一定会有较深入的了解.

§1.3 预期的好处

通过对本书内容的阅读和思考，你将获益非浅. 首先，可以了解 DOS 的详细信息，而这如果自学是很困难的. 另外，本书以简单易懂的方式给出接口结构的概念.

本书最突出的优点是可以通过使用书中给出的例子，一步一步地了解 DOS 设备驱动程序结构和接口. 从现在开始就可以阅读和尝试了. 等读完本书时，将对设备驱动程序有较深的理解，并拥有功能很强的工具，帮助你以后分析了解 DOS 设备驱动程序.

§1.4 所用的约定

本书中所有程序都是用 C 语言编写的. 更具体点，是用 TURBO C 2.0 版编写的. 虽然 TURBO C 提供了对 ANSI C 的改进，但是在我们给出的程序仍遵守标准. 不过，由于是系统程序(确切地说，是 DOS 设备驱动程序)的缘故，必须使用 TURBO C 的直接寄存器存取功能.

TURBO C 的直接寄存器存取功能允许 INTEL 处理器的每一寄存器都可以被直接存取. 直接存取需要使用一套由下划线和寄存器名称的大写形式组成的预定义名；即

__AX，__BX 等.

这一程序设计语言的功能可能是很危险的容(易出错). 由于这个原因，我们把直接寄存器存取限制在绝对需要其使用的代码段内，如服务中断时.

注意，虽然我们使用了 TURBO C 的这一特性，用其他任何的 C 编译程序也可以达到同样的效果，这时需要把使用直接寄存器存取的 TURBO C 程序段用汇编语言改写. 例如，下面的 TURBO C 程序语句:

```
__AX = 0xFFFF;
__BX = __CX;
```

用汇编语言改写后是下面代码段:

```
mov ax, 0FFFFh;
mov bx,cx
```

它们得到同样的效果.

简而言之，我们选择 TURBO C 程序设计语言是为了减少本书中给出的例子的复杂程度，同时使尝试本书中的例子所需的材料最少.

§ 1.5 内容概述

本书中由三部分组成. 第一部分，也即你现在正在读的部分，给出阅读本书其余的内容的指导信息.

第二部分给出有关 DOS 设备驱动程序的信息. 它试图给出包含在 DOS 技术参考手册中的信息. 这些信息是用 C 程序术语以及最终组成 DOS 设备驱动程序样板(用以开发自己的 DOS 设备驱动程序)的程序段给出的.

第三部分扩展你的眼界. 它试图通过从开始到结束开发一个 DOS 设备驱动程序来解决一个现实世界的问题. 我们实现一个 WORM 设备驱动程序.

每一部分中的每一章都小结该章所学的内容，并且每一章都包含几个在阅读该章之后应能回答的问题.

第二章　基本原理

用 C 语言开发 DOS 设备驱动程序最困难的一点是理解编译程序如何处理 Intel 系列处理器的分段结构．本章介绍分段结构的基本概念．

§2.1 Intel 8086／8088 结构

Intel 8086／8088 结构是基于分段内存模式并支持十三个 16 位寄存器：四个通用累加器，两个索引寄存器，两个指针寄存器和一个标志寄存器．

§2.1.1 内存寻址

四个段寄存器中的每一个，其功能有点类似基寄存器，被指定用于特定的段：一个用于代码段(CS)，一个用于数据段(DS)，一个用于堆栈(SS)以及一个附加段(ES)寄存器被指定用于数据移动．

处理器用两步来计算内存地址．首先，它选择适当的段寄存器并把寄存器的内容左移四位．选择段寄存器的原则是很简单的．如果内存存取是由于指令调用，如要转移到指定的程序位置，则处理器选择 CS 寄存器．如果内存存取是由于指令要求内存操作数，则处理器选择 DS 寄存器．如果内存调用是堆栈操作的结果，则选择 SS 寄存器．

第二步，处理器把段偏移量加到所选寄存器的左移了的值中．这一操作得到 20 位的地址，这一地址最多可以存取 1 兆字节内存．

§2.1.2 通用和索引寄存器

四个 16 位通用累加器 AX，BX，CX 和 DX 中的每一个都可以作为两个 8 位的累加器使用．这是通过指定 16 位累加器的高位部分(AX 的 AH)或低位部分(AX 的 AL)来进行的．

两个 16 位索引寄存器被记为 SI 和 DI．这些寄存器对于展开数据表操作的索引地址非常有用．

另外，两个附加的 16 位寄存器被记为指针寄存器．基指针(BP)寄存器对实现要求激活记录和递归变量框架指针的高级语言非常有用．堆栈指针(SP)寄存器作为硬件和软件堆栈指针．

最后，Intel 的结构还支持一个 16 位标志寄存器，维护来自前一条指令的条件代码．

注意，由于采用高级语言方式来实现 DOS 设备驱动程序，这种低级处理器结构的许多细节对于你来说被屏蔽了．这是很重要的，因为这样你可以花更多时间考虑算法细节，而不必再去考虑应使用哪个寄存器或哪条指令来完成手头的任务．

可以说，如果你遵循本书中的原则，就不必集中在处理器结构的低级细节上．看了本书中给出的所有例子，你可以用 C 语言开发自己的设备驱动程序算法；如同开发其他 C 程序一样．

§2.2 段和 C 编译模式

你会发现许多与分段内存模式有关的术语，而在讨论支持平台内存模式的处理器时是找不到它们的．由于个人计算机的结构是基于分段内存模式的，我们必须定义这些术语，并说明它们是如果影响高级语言的编译程序为个人计算机生成代码．

要掌握的第一个概念是个人计算机的整个地址空间—1 兆字节—如果不修改段寄存器 CS、DS、SS 和 ES 中的一个或多个是不能被直接寻址的．处理器被设计为在某一时刻只能寻址 64K 字节的内存．每一 64K 的内存块被称作段．从它我们得到术语分段内存模式．随之，高级语言的编译程序推出一种寻址内存的统一方法也是很重要的．

Intel 提出了编译程序可以用于生成代码的各种内存的概念．换句话说，Intel 提出了可以寻址他们的处理器的几种不同的方法，而让你去告诉编译程序你的程序要使用哪一种内存寻址方式．

Intel 把这些内存模式记为 SMALL、MEDIUM、LARGE 和 HUGE 模式．BORLAND 国际公司在他们的 TURBO C 编译程序中还引入了其他两种模式，它们被称为 TINY 和 COMPACT 模式．下面我们逐一对这些内存模式作一简单介绍．

§2.2.1 TINY 模式

TINY 模式是编译程序可以生成的内存模式中最小的一个．四个段寄存器(CS,DS,SS 和 ES)都被初始化为同一个值．因此整个程序，包括代码、数据和局部堆栈变量，只有 64K 字节的地址空间．该模式使用近指针—16 位值．近指针也被称作偏移量．

§2.2.2 SMALL 模式

SMALL 模式程序的可寻址内存是 SMALL 模式程序的两倍．这是通过把一个段分配给代码、把另一个段分配给数据来完成的．因此，SMALL 模式程序最大可以是 128K 字节的内存．该模式对于代码引用和数据引用都使用近指针．

§2.2.3 MEDIUM 模式

MEDIUM 模式程序允许的代码空间最大可达 1 兆字节．编译程序是通过为所有的指令引用生成远指针—由 16 位的段值和 16 位的偏移量值组成的 32 值—来完成的．MEDIUM 模式的程序如同 SMALL 模式一样，其数据空间被限制为一个段(64K 字节)．该模式为代码引用使用远指针，为数据引用使用近指针．

§2.2.4 COMPACT 模式

COMPACT 模式程序与 MEDIUM 模式程序正好相反．换句话说，其代码空间被限制为一个段(64K)字节，而数据则最多可占 1 兆字节的内存空间．该模式为代码引用使用近指针，为数据引用使用远指针．

The first line
§2.2.5 LARGE 模式

LARGE 模式程序为数据和代码引用都使用远指针．这就允许代码和数据最多都可占据处理器的 1 兆字节寻址限制．

§2.2.6 HUGE 模式

HUGE 模式是 LARGE 模式的扩展．它允许一个以上静态分配的数据段存在于程序中．如同 LARGE 模式的程序，HUGE 模式程序在所有情况都使用远指针．

§2.3 仔细观察 TINY 模式

DOS 设备驱动程序必须遵循 DOS 技术参考手册中所定义的结构化的体系结构．在下一节将详细讨论这个问题．但是，在继续学习要内容之前必须先讨论其中的几点．

DOS 设备驱动程序必须是一个.COM 文件．.COM 文件是不声明专用的堆栈段的 TINY 模式程序．由于 TURBO C 把堆栈段包含在程序的数据段中，因此可以用 DOS 的 exe2bin 实用程序把 TINY 模式程序的.EXE 文件转换为.COM 文件．

DOS 设备驱动程序必须在地址 0 有一个特殊的设备驱动程序头．该设备驱动程序头包含各种数据和指针．

DOS 设备驱动程序的初始化过程通常被放在它的最后．这样使得设备驱动程序在初始化过程结束之后释放该过程所占的内存．

开发人员必须控制 DOS 设备驱动程序的各个组成元素的物理位置以及它们的次序．这在汇编语言中显然是件很容易的事情，但是对于象 C 这样的高级语言就不太容易了．

最后，当使用 C 编译程序时，理解它在编译代码时所作的假设是很重要的．例如，TURBO C 假设当你连接编译了的程序时，你会在连接过程中引入启动模块．启动模块负责在调用名为 main 的函数之前，把处理器的寄存器初始化为指定的值，并且建立 C 运行环境．

DOS 设备驱动程序有其自己一套初始化原则，这些原则在大多数情况下都与 TURBO C 的启动模块的相冲突．因此，在本书中的 DOS 设备驱动程序的开发过程中，我们不引入启动过程，你也不必命名 main 过程的任何东西．

§2.4 第一个 TINY 模式程序

理解 TINY 模式程序的含义的最好办法是建立一个 TINY 模式的程序，然后检查编译程序的输出．下面程序 first.c 使用了我们将在 DOS 设备驱动程序例子中使用的许多 C 语言的特性．

```
/*--------------------------------------------------------------*/
/*                                                              */
/*  PROGRAM :       First                                       */
/*                                                              */
/*  REMARKS :       First is a program that is designed to be   */
/*        compiled in TINY model by the TURBO C Version 2.0     */
/*        compiler.  Once compiled the assembler output is      */
```

```
/*        reviewed to identify the structure and problems that  */
/*        will be encountered when developing a DOS device      */
/*        driver in this language.                              */
/*                                                              */
/*--------------------------------------------------------------*/

#include        <stdio.h>


/*--------------------------------------------------------------*/
/*                                                              */
/*      Global Data Required For This Program                   */
/*                                                              */
/*--------------------------------------------------------------*/

unsigned int    global_int;
unsigned char   global_byte;

/*--------------------------------------------------------------*/
/*                                                              */
/*  FUNCTION:       Function                                    */
/*                                                              */
/*  REMARKS:        Function is a function responsible for      */
/*        accessing the supplied parameters and assigning the   */
/*        global data variables to the current values of the    */
/*        parameters.                                           */
/*                                                              */
/*--------------------------------------------------------------*/

Function (int param_int, char param_byte)
{
    global_int  = param_int;
    global_byte = param_byte;
}

/*--------------------------------------------------------------*/
/*                                                              */
/*  FUNCTION:       main                                        */
/*                                                              */
/*  REMARKS:        main is the main program function that is   */
/*        responsible for initializing its local data variables */
/*        and then calling Function with them as parameters.    */
/*                                                              */
/*--------------------------------------------------------------*/
```

```
void main (void)
{
    int          local_func_int;
    char         local_func_byte;

    local_func_int = 0;
    local_func_byte = 0;

    Function (local_func_int, local_func_byte);
}
```

程序 first.c 声明了两个对于整个程序都可见的全局变量，包含一个函数 main 和另一个有两个形参的函数 Functin.

你会注意到即使象 first.c 这样一个小而简单的 C 程序，也执行下列几种操作:

- ☐ 全局变量的存取
- ☐ 局部(堆栈)变量的存取
- ☐ 参数传递给函数
- ☐ 函数参数存取
- ☐ 函数调用

上述操作的每一个对于 C 程序的操作都是必须的．因此，理解这些项在开发用 C 语言编写的 DOS 设备驱动程序的过程中是很重要的．

first.c 是用 TURBO C 2.0 版编译的．用下面命令编译:

```
tcc -mt -y -M first.c
```

该命令要求 TURBO C 生成一个 TINY 模式程序(-mt)，包括行号信息(-y)和连接／装入映象(-m)．下面是该编译过程生成的连接／装入映象(link ／load map):

```
 Start  Stop   Length Name                 Class

 00000H 00659H 0065AH _TEXT                CODE
 00660H 007E7H 00188H _DATA                DATA
 007E8H 007EBH 00004H _EMUSEG              DATA
 007ECH 007EDH 00002H _CRTSEG              DATA
 007EEH 007EEH 00000H _CVTSEG              DATA
 007EEH 007EEH 00000H _SCNSEG              DATA
 007EEH 00837H 0004AH _BSS                 BSS
 00838H 00838H 00000H _BSSEND              STACK

  Address         Publics by Name

 0000:02D6       DGROUP@
 0000:07CF       emws_adjust
 0000:07D3       emws_BPsafe
 0000:07CB       emws_control
 0000:07D1       emws_fixSeg
 0000:07B5       emws_initialSP
```

```
0000:06F5       emws_[illegible]
0000:07C5       emws_nmiVector
0000:07C1       emws_saveVector
0000:07D5       emws_stamp
0000:07C9       emws_status
0000:07CD       emws_TOS
0000:07D9       emws_version
0000:02C0       _abort
0000:046F       _atexit
0000:063A       _brk
0000:06CF       _environ
0000:06DB       _errno
0000:0305       _exit
0000:02D8       _Function
0000:07EE       _global_byte
0000:07EF       _global_int
0000:02E9       _main
0000:0574       _malloc
0000:0648       _sbrk
0000:06DD       __8087
0000:06CB       __argc
0000:06CD       __argv
0000:07E6       __atexitcnt
0000:07F2       __atexittbl
0000:06ED       __brklvl
0000:06D1       __envLng
0000:06D3       __envseg
0000:06D5       __envSize
0000:0220       __exit
0000:07DC       __exitbuf
0000:07DE       __exitfopen
0000:07E0       __exitopen
0000:06E9       __heapbase
0000:07E2       __heaplen
0000:06F1       __heaptop
0000:06BB       __Int0Vector
0000:06BF       __Int4Vector
0000:06C3       __Int5Vector
0000:06C7       __Int6Vector
0000:06D9       __osmajor
0000:06DA       __osminor
0000:06D7       __psp
0000:07EE       __RealCvtVector
0000:0283       __restorezero
0000:07EE       __ScanTodVector
0000:033A       __setargv
0000:0425       __setenvp
0000:06DF       __StartTime
0000:07E4       __stklen
```