硬件接口开发系列

Writing Windows VxDs and Device Drivers

Windows VxD与设备驱动程序权威指南

（第二版）

【美】KAREN HAZZAH 著
孙喜明 译

R&D BOOKS
中国电力出版社
www.infopower.com.cn

硬件接口开发系列

Writing Windows VxDs and Device Drivers

Windows VxD
与设备驱动程序权威指南

（第二版）

【美】KAREN HAZZAH 著
孙喜明 译

中国电力出版社

内 容 提 要

本书是讲述 VxD 和设备驱动程序的权威指南。主要分为两部分。第一部分（第 2 到第 12 章）讲述 Windows 执行环境和 VxD。第二部分（第 13 到第 19 章）讲述基于 DLL 的驱动程序。本书几乎涵盖了设备驱动程序开发的各个层面，内容全面翔实，可读性较高。

本书适合程序员及大专院校师生阅读。

图书在版编目（CIP）数据

Windows VxD 与设备驱动程序权威指南/（美）哈兹编著；孙喜明译-北京：中国电力出版社，2001

ISBN 7-5083-0558-2

Ⅰ.W… Ⅱ.①哈…②孙… Ⅲ.窗口软件，Windows-程序设计-指南 Ⅳ.TP316.7-62

中国版本图书馆 CIP 数据核字（20001）第 12493 号

北京版权局著作权登记号 01-1999-3844

本书英文版原名：Writing Windows VxDs and Device Drivers

Published by R&D Books, an imprint of Miller Freeman, Inc.

1601 West 23rd Street, Suite 200, Lawrence, KS 66046, USA

本书由美国 Miller Freeman, Inc.公司授权出版

中国电力出版社出版、发行

（北京三里河路 6 号 100044 http://www.infopower.com.cn）

实验小学印刷厂印刷

各地新华书店经售

2001 年 5 月第一版 2001 年 5 月北京第一次印刷

787 毫米×1092 毫米 16 开本 27.5 印张 617 千字

定价 55.00 元

译 者 序

——VxD技术漫谈

说起 VxD 技术，很多接触电脑的人都会感到陌生，不过提起 CIH 病毒，我相信您一定不再摇头。

有过一定 Windows 编程经验的人可能听说过 VxD 这个词。80386 的保护模式给了我们非常多的“保护”，对于一个不懈地追求自由的人来说，“镀金的笼子也是笼子”，翻遍 Windows 编程的书籍，也许最终只找到这样一扇通向自由的门：试一试 VxD。

VxD 是什么?为什么 CIH 会如此猖獗?

VxD = Virtual x Device，这里的 x 代表 Something。比如说 VKD 代表 Virtual Keyboard Device（键盘驱动），VPICD 代表 Virtual Programmable Interrupt Device（可编程中断控制器驱动）。VxD 是 Windows 3.x、Windows 9x 的驱动程序模型。

在很多人的印象中，VxD 是与硬件设备打交道的，对于软件来说没有太大的作用，其理由之一是：用来开发 VxD 的 DDK 一般来说都分发给硬件开发商，对于搞软件开发的人来说，SDK 才是看家本领，而 SDK 中对于 VxD 却是言及甚少。这种想法可是大错而特错了。

由于 VxD 运行在系统的 Ring 0 级，拥有与操作系统同等的级别，所以，VxD 可以最大程度地帮助我们走出 80386 保护模式强加给我们的“镀金的笼子”。我们就拿曾经轰动一时的 CIH 病毒来说一说 VxD 这柄利剑的强大。

CIH 运行在 Ring 0，挂接 IFS（Installable File System）服务，截获 File Opening 操作，然后把自己“传染”到其他 PE 格式的文件中，如果到了该发作的时间，它就攻击主板的 Flash Memory（闪存）。

CIH 之所以能破坏硬件资源（通过攻击主板的闪存，达到破坏硬件的目的），正是因为它利用了 VxD 技术。对于一般的运行于 Ring 3 的 Windows 应用程序来说，想破坏主板的闪存是不可能的，因为这是 80386 保护模式对于代码权限的盘查所不允许的，其结果最多是引起操作系统异常，也就是在 Windows 中经常出现的、令用户心跳加速的、蓝底白字的“General Protection Fault（一般保护性错误）”，在这里，我们该感谢那个“镀金的笼子”，至少它使我们免受邻居家那个只学过 Turbo C 编程的小孩的搔扰。

除了搞破坏，VxD 还可以用来做些什么呢?

在这里我简单谈一下 VxD 的应用。

由于 VxD 可以虚拟根本不存在的硬件，因此，可以利用 VxD 虚拟硬件狗来破解一些有版权的软件的保护。

由于 VxD 工作在操作系统的最底层，所以掌握它能使你具备扩展操作系统的能力。有没有想过在 Windows 9x 里按下“Ctrl + Alt + Delete”键，而后弹出的是你自己设计的对话框（聪明的你一定可以发挥你天才的想像力，比如说显示一张你女朋友的照片，以提醒自己关了电脑之后别忘了去赴约会），VxD 可以帮助你跟 Windows 9x 开开玩笑。

好吧，说点正经的应用。想不想利用 VxD 来做一些“实时”控制之类的东西？当然，作实时工控，首选的操作系统还是 DOS，但是如果你经不住 Windows 那张漂亮面孔的诱惑，也可以在 Windows 下作“准实时”控制应用。一般来说，通过 VxD 的协助，在很多情况下，我们还是可以获得比较满意的实时效果的。据测试，486/66 的 PC 机上运行 Windows 9x，在没有其他应用程序干扰的情况下，VxD 可以处理 10kHz 的硬件中断，不会漏掉一个中断！精心设计你的应用程序和 VxD（最好再买台好点的电脑），相信你可以获得满意的“实时”效果。

再讲一个有趣的应用。如果你有一堆 16 位的 DLL（假设你曾在 Windows 3.1 下作过应用程序开发，并曾写过许多功能不错的 DLL），而你又懒得通过重写代码或者利用烦琐的 Thunk 移值到 Windows 9x 下，那么，试一试 VxD，可能会给你“它山之石可攻玉”的经历。当然，我的前提是你对于 VxD 已经有了比较多的认识并有一些实战经验，否则你大概会因此导致神经衰弱的。

再说一个比较有趣的应用。想不想使 DOS TSR、Win16 应用程序、Win32 应用程序之间产生一些协作？想不想打破 Win32 应用程序 4GB 独立线性地址空间带来的限制？通过 VxD 作中介，你会发现原来“万物通于一理”。别相信坐在你对面的那个权威的“No”，相信你自己。

对于 VxD，我们已经说得够多了，也许你已经兴奋得想立刻去找本《21 天轻轻松松掌握 VxD 技术》来一读为快了。先安静会儿，现在开始泼冷水。

VxD 的明天依然美好吗？

1996 年的 Windows Hardware Engineering Conference（WinHEC）会议上，Microsoft 宣布了一种新的 Windows 设备驱动程序模型——Win32 Driver Model（WDM）。这种新的设备驱动程序模型是 NT4 Kernel Mode Driver + Power Management + PnP，它将成为 Windows 2000 的核心。

在阅读本书的时候，有的读者已经在使用 Windows 2000 了，在安装硬件驱动的时候，你会发现驱动程序的后缀名已经不再是.VxD 而是.SYS 了。

这个事实令所有从事 Windows 设备驱动程序（VxD）开发的人感到沮丧（虽然大家早已预料到 Windows 系列与 Windows NT 系列最终将走到一起）。WDM 的浮出水面把一个吃尽苦头的 VxD 开发者又推到了一个新的起跑线上。如果你曾去 DDK 的汇编代码里观光过，你一定可以体会这个消息对 VxD 开发者是个沉重的打击，而对于 Windows NT 设备驱动程序（Kernel Mode Driver）者来说，却是另一番心情——因为 WDM Kernel Mode Driver 太相似了。

VxD 将让位于 WDM，即使到了 WDM 一统天下之时，也不用灰心，因为无论是 VxD 还是 WDM，都要求开发人员对计算机硬件有全面而细致的了解。通过 VxD 的锻炼，你至

少熟悉了计算机的硬件资源并对保护模式有了比较深刻的认识，这些东西都是将来从事WDM开发的硬功夫。可以说，如果掌握了VxD，那么WDM的开发将大为省力，至少我自己和我认识的WDM开发者是这样一种情况。

本书的作者在Windows 系统开发领域里享有盛誉，也是现在世界上为数不多的真正对Windows内部运行机制了如指掌的人之一。因此本书的权威性是毋庸置疑的。

本书深入浅出地讲解了VxD开发的方方面面，相信读者在仔细读过本书后会有醍醐灌顶之感，能深刻体会到作者的大家风范和敬业精神。

孙喜明
2001年4月

译者简介

多年从事Windows 设备驱动程序的开发和研究，早在Windows 3.1 SDK的年代就开始涉足DDK的丛林。曾为《中国计算机报》撰写VxD专栏，连载多期。曾参加多个国家重点项目的开发，负责底层硬件驱动程序的开发。对于Windows系列操作系统内部运行机制有深刻认识。

原 书 序

本书主要针对那些需要为非标准设备编写驱动程序（VxD 或 DLL 形式）的程序员。所谓非标准设备是指显除了显示器、键盘、鼠标、串行接口或打印机的任何其他设备。第二版与前一版相比，扩展了 VxD 的范围，重点讲述了 Windows 9x 的新特性，如即插即用所引起的问题。

虽然本书不是为初级程序员编写的，但我还是尽量使本书对更多读者有所裨益并易于接受。如果你曾为 DOS 或其他操作系统编写过设备驱动程序或设备接口代码，应该不会对本书的内容感到陌生。阅读本书，应该有很强的 C 语言实践知识。你还应该能够阅读 80x86 汇编程序，虽然第二版使用汇编大大减少了。如果能够充分理解 DOS 编译和汇编程序对段的使用方式，将会很有帮助。你不用成为 Windows 程序员。实际上，你会发现书中的代码不同于普通的 Windows 程序，而更像传统的 DOS 代码。

循序渐进的方法

Windows 是极为复杂的环境。本书的目的是帮助你理解这个环境中那些部分真正对各种驱动程序而言至关重要。每章会介绍一个新的驱动程序，并且仅仅介绍理解新例子所需的新内容。我将使驱动程序范例尽可能简单，这样主要的特性就能不言自明。大部分范例程序都在 VC++中用 C 和内嵌汇编语言写成。必要时，还会使用 MASM v6.1 编写汇编。本书代码包括一个封装函数库，借助它，你可以几乎完全用 C 语言编写 VxD 程序。

Windows 版本

本书涵盖了 Windows 9x 和 Windows 3.x （增强模式）。重点是 Windows 9x，但大多数内容都适用于 Windows 3.x。在大部分章节中，两个版本区别很小，有关 Windows 3.x 的内容将单独出现在各章的末节。在某些区别较大的章节，我将完整地分节讲述两个版本。

关于本书

本书主要分为两部分。第一部分（第 2 到第 12 章）讲述 Windows 执行环境和 VxD。第二部分（第 13 到第 19 章）讲述基于 DLL 的驱动程序。每一部分中，每一章都是以前面各章为基础的。读完第 1 章以后，根据你创建的对象是 VxD 还是 DLL，可以决定怎样读本书的其他内容。也许每个人都应该读一读第 2 章和第 3 章。讲述了 Windows 系统体系中对驱动程序编写者非常重要的部分。所包括的主题对 VxD 和 DLL 程序员都很重要。对 80x86 系列处理器的选择器、描述符、页表和其他系统体系细节不熟悉的读者，在阅读这两章的时候可能需要参考附录 A。我假定你对附录 A 中的内容很熟悉。如果发现有什么没有讲到的程序细节影响了你的理解，也许在学习第一个驱动程序范例时，应该看一看附录 B。这个附录主要是全书中使用的汇编语言封装库的参考。

目　录

第一部分　Windows 执行环境和 VxD

第二部分　DLL 驱动程序

第1章 入 门

什么是驱动程序

从广义上说，“驱动程序”是指一些函数的集合，这些函数能够对硬件设备进行操作。对驱动程序的分类方法有多种，其中有一种就是根据这些函数集合的组织方式。在DOS模式下，“驱动程序”可以是一个模块（module），这个模块可以被链接（link）到.EXE文件中；“驱动程序”还可以是与应用程序完全分离的程序（例如DOS device driver或者TSR）。在Windows下，“驱动程序”可以是一个能被动态地链接进应用程序.EXE的模块（我们称之为DLL）；Windows下的驱动程序还可以是与应用程序完全隔离的，即不需要在运行时与应用程序.EXE进行动态链接（我们称之为VxD）。

基于优先权的区分

另外一种对驱动程序进行分类的方法就是根据其权限的优先级别。一些操作系统，如UNIX和Windows NT，禁止应用程序直接访问硬件设备。在这些操作系统下，只有那些具有很高优先权的代码（如设备驱动程序）才被允许直接操作硬件设备。如果应用程序需要控制硬件设备，那必须通过设备驱动程序提供的服务接口来实现。

Windows也支持有特权的驱动程序（译注：本书中Windows专指Win 3.x和Win 9x）。在Windows下，这些享有特权的驱动程序被称作VxD。但是，令人吃惊的是，在Windows下，对于硬件设备的访问接口没有必要一定由VxD提供，许多DLL直接提供了访问硬件的接口，从而充当了“驱动程序”的角色。

驱动程序接口

还有一种对驱动程序进行分类的方法是根据其与应用程序和OS kernel（操作系统内核）的接口。所有的Windows NT 驱动程序都对NT kernel使用同样的接口。这样一来，NT kernel就相当于提供了一个标准的接口，应用程序可以用这套标准接口（如open、read等等）调用任何驱动程序。与之相对，Windows下的VxD却有所不同。虽然所有的Windows VxD都使用同样的kernel接口，但是从应用程序这一层看来，VxD没有并提供一套标准的接口。相反，每个VxD都定义了自己的应用程序接口。

在Windows下，有些DLL充当驱动程序的角色。有时这些DLL需要向kernel提供指定的接口。这样的DLL被称作“系统驱动程序”。但是，需要注意的是，键盘的系统驱动程序使用的接口与显示设备的系统驱动程序使用的接口是有很大不同的。还有一些充作驱动程序的DLL可以不向kernel提供任何特定的接口，开发者可以在这些DLL中自由设计任何kernel和应用程序的接口。

需要编写哪种驱动程序

很显然，在 Windows 下，有很多种“驱动程序”。究竟需要编写哪一种，要根据如下的几个因素：

- Windows 的版本号（3.x、9x）；
- 硬件设备的种类（键盘、网卡、第三方的 A/D 卡）；
- 硬件设备接口种类（I/O 端口、中断）；
- 性能需求（吞吐量、中断延时）；

总之，根据上述四个因素，我们可以决定是写一个 DLL 还是一个 VxD。

哪一类设备

决定我们选择的第一个因素是根据我们的硬件设备种类。表 1-1 显示了 Windows 直接支持的硬件设备种类和相应驱动程序的种类。如果我们的硬件设备是其中的一种，那我们就不用再做更多的考虑了。

表 1-1 需要特殊驱动程序的设备

设备分类	Windows 3.x		Windows 9x	
	16位 DLL	VxD	16位 DLL	VxD
显示器	DISPLAY.DRV	VDD.VXD	DISPLAY.DRV	VDD.VXD
打印机	PRINTER.DRV		PRINTER.DRV	
键盘	KEYBOARD.DRV	VKD.VXD	KEYBOARD.DRV	VKD.VXD
鼠标	MOUSE.DRV	VMD.VXD	MOUSE.DRV	VMD.VXD
串行/并行端口	COMM.DRV	VCD.VXD		VCOMM 端口驱动程序
多媒体	可安装的驱动程序 DLL		可安装的驱动程序 DLL	
网络	不是Windows 驱动程序，而是 DOS 设备驱动程序或TSR (如 NDIS 2.0 或 ODI)			NDIS 3.0 MAC 驱动程序
块设备(硬盘、CD-ROM)	不是Windows 驱动程序，而是 DOS 设备驱动程序			分层块设备驱动程序

如表 1-1 所示，对于大多数的设备种类，Windows 3.1 和 Windows 9x 需要的驱动程序的种类是相同的。有两个例外，网卡和块设备（例如硬件、CD-ROM 等）。在 Windows 3.1 下，这两种设备都不被直接支持（都是用的 DOS 驱动程序），但是，在 Windows 9x 下，这两种设备都需要 VxD 来作为驱动程序。

大多数的硬件设备都需要 DLL 和 VxD 作为驱动程序，其中大多数都可以通过 DLL 来

实现。值得注意的是，DLL 驱动程序通常是 16 位的——即使在 Windows 9x 下，虽说正常情况下的应用程序和 DLL 都是 32 位的。

多媒体设备的驱动程序是第一次在 Windows 3.1 下被引入的，大多以 DLL 的形式存在，并遵守一套新的基于消息的接口。如果充当驱动程序的 DLL 遵守这个标准，那么该 DLL 就被称作“可安装的驱动程序（installable driver）”。这种“可安装的驱动程序”输出一个 DriverProc 的接口（类似于 Windows 应用程序的 WindowProc），并对 DRV_LOAD、DRV_OPEN、DRV_INSTALL 和 DRV_CONFIGURE 等消息进行响应。这套接口给用户提供了一种通过“控制面板”来安装多媒体驱动程序的标准机制。同时，这套接口还为操作系统提供了加载、启用、禁止多媒体设备的标准方式。

在Windows 3.1下，可安装的DLL驱动程序很快就吸引了别的非多媒体的设备。但是，Microsoft现在正在大力推行VxDs作为驱动程序。有趣的是，在Windows 9x下，多媒体设备驱动程序还是16位可安装的DLL驱动程序。幸运的是，多媒体设备驱动程序的开发者们无需像别的16位驱动程序开发者那样考虑“转换层（thunking）”的问题，因为Windows自己包含了所需的转换层（就像对待别的Windows模块一样，如USER、GDI等）。详见第18章关于转换层的讨论。

哪种硬件接口

如果准备开发驱动的硬件设备不在表 1-1 中，那么 Windows 并没有直接说明应该选择何种驱动程序（DLL 还是 VxD）和接口。由于无论对于哪种驱动程序，都需要自己设计接口。这时，需要考虑的因素就是硬件设备的接口。

- 该设备是 I/O 映射的还是内存映射的？
- 该设备是否产生中断？
- 该设备是否使用 DMA？

无论是在 Windows 3.1 还是在 Windows 9x 下，通过 DLL 都很容易访问 I/O 映射的硬件设备。如果准备开发的硬件设备是 I/O 映射的并不产生中断或者 DMA，那么，最好的选择是开发 DLL 形式的驱动程序。

另一方面，从 DLL 中访问内存映射的设备，或者处理硬件中断、进行 DMA 操作都是“可能”的。在 Windows 3.x 下，这是容易实现的。但是在 Windows 9x 下，只有 16 位的 DLL 才可能进行这三种操作。Windows 9x 应用程序当然是 32 位的，所以，如果在 Windows 9x 下使用 16 位的 DLL，必须同时开发一个“转换层”DLL。这个“转换层”负责在 32 位的 Windows 9x 应用程序与 16 位的 DLL 驱动程序之间进行转换。

由于开发“转换层”DLL 是件额外的工作，所以，在 Windows 9x 下，只有两个原因可以考虑使用 DLL 驱动程序而不是 VxD。第一，如果需要支持的是非常简单的 I/O 映射设备并且不使用中断。在这种情况下，可以写一个很简单的 32 位的 DLL 来访问硬件。第二，如果已经有了 16 位的 DLL 驱动程序。在这种情况下，可以加上一个“转换层”就可以在 Windows 9x 下使用了。

在开发驱动程序的时候，还要考虑对于更新的总线的支持。Windows 9x 内置了对于“即

插即用”设备的支持，如 PCI、PCMCIA 和 VL-Bus。如果想提供对“即插即用”设备的充分支持，那就必须是 VxD，由 VxD 来与 Plug and Play Configuration Manager（这是 Windows 9x 的一个模块，也是一个 VxD）通信。

如果选择了写一个 DLL 驱动程序而不是 VxD 作为“即插即用”设备的驱动程序，那就不得不使用与总线相关的 BIOS 调用来获得设备的配置信息。由于大多数的 BIOS 调用需要软件中断，而 32 位的代码是不允许支持软件中断的（详见第 13 章），这样一来，相应的 DLL 必须是 16 位的，同时还需要一个“转换层”。关于“转换层”的讨论见第 18 章。

什么是性能需求

对于 I/O 映射和内存映射的设备来说，DLL 驱动程序和 VxD 对于硬件的实际访问时间是差不多的。但是对于中断响应（即中断延时）来说就不同了，VxD 对于硬件中断的响应要远快于 DLL 驱动程序。所以，如果硬件设备产生大量的中断并且没有足够大的缓存，那最好还是写个 VxD。

总结

通过本章提供的信息，读者应该能够判断自己需要开发哪种驱动程序。如果 DLL 可以满足需要，那么读者可以跳过从第 4 章至第 12 章的所有章节，并且把精力主要放在第二部分的 DLL 信息上，如果读者想开发 VxD，那就应该把精力集中在第一部分。

无论是何种情况，读者都应在读完第 3 章之前阅读一下附录 A。本书从始至终都假定读者对于附录中的内容已经相当熟悉了。

不论是何种情况，无论是打算开发VxD还是DLL，接下来的两章都是很重要的基础知识。第2章对于虚拟机的基本概念作了讲解。第3章讲解了Windows如何使用80×86体系结构实现虚拟机。