Windows NT 4.0 下设备驱动程序的开发

吴奇 范大鹏 彭丽 (长沙国防科技大学机械电子工程系 410073)

摘要:在 Windows 环境下运行是当前测控软件发展的主流,设备驱动程序作为联系外部设备与计算机之间的接口程序则是开发测控软件的关键,本文将要介绍的就是在 Windows NT 4.0 这一特定环境下怎样生成设备驱动程序以及正确安装它的方法。

关键词:Windows NT4.0 设备驱动程序

1. Windows NT 4.0 下设备驱动程序开发的必要性

目前 Windows 环境下测控软件的发展在国外方兴未艾,如大家所熟悉的 LabView、RealTime 等著名软件,它们都因为界面的形象直观、操作简单方便等特点,而在市场有着较强的竞争力,反观我们国内却还有太多同类软件运行于 DOS 平台之上,所以尽快发展自己 Windows 环境下的测控软件已是大势所趋,而设备驱动程序作为测控软件中不可缺少的关键部分更应受到重视,因为没有它做基础整个测控软件将无法完成任何实际的设备操作。另一方面,由于 PC 机上 Windows NT 4.0 网络操作系统和近乎同期发布的 Windows95 个人操作系统相比有着网络功能强、安全性高及可靠性好等优点,目前国内外已有越来越多的科研、生产单位采用了 Windows NT 4.0 作为自己的应用软件开发平台,整个测控软件开发业也有着向 Windows NT 领域拓展的趋势,因此根据上述两方面原因推断,开发 Windows NT 4.0 下的设备驱动程序对国内测控软件业已经很有必要。

2. 设备驱动程序的开发条件

开发 Windows NT 4.0 设备驱动程序有其特定的软硬件条件。最低条件为:

Windows NT 4.0 零售版本;

DDK(Device Development Kits) for WindowsNT 4.0;

Win32 SDK(Software Development Kits) for Windows NT 4.0;

Visual C/C++4.1 或其以上的版本;

装有以上软件并能正常运行的微机,建议至少用 586 档次的机器。

上述条件中 Visual C/C++ 为开发工作提供编译器,DDK、SDK 则包含开发中所需要的各种头文件和库文件,二者联合形成驱动程序的链接环境。另外,开发工作最好能在两台微机上联合进行以提高开发速度,其中一台微机用于程序的编辑、编译和链接,另一台则模拟驱动程序的实际运行,两台微机之间用网络线或一般串口线实现连接。

3. 设备驱动程序的生成

设备驱动程序简单讲是指管理实际数据传输和控制特定物理设备的一段代码。由于在 Windows NT 下设备驱动程序运行于内核模式,和应用程序相比它能绕开硬件检查机制从而获得对 I/O 设备的全部访问权,所以不管是从编写的内容还是从编译链接的方法上驱动程序都和应用程序有着很大不同。

(1)驱动程序源文件的编写。Windows NT 4.0 设备驱动程序的编写一般遵循模块化设计原则(模块是指由若干例程函数组成并具有完整的功能的一段代码),虽然不同的驱动程序包含的模块不尽相同,但以下几个模块却是它们所共有的,即装载和卸载模块、I/O 请求散转模块和 I/O 数据传输模块,而这些模块对开发一般的控制、测量设备驱动程序已能满足要求。

为了说明这些模块,这里需要先介绍一下 DDK 中 IRQL、IRP 队列中和 DPC 队列的简单概念。

IRQL(Interrupt Request Level)称为中断请求级,它定义了 CPU 当前运行任务的重要性,此值越高表示当前任务的重要性越大且级别高的任务优先执行,在驱动程序中表现为各例程函数的执行次序。

IRP 是 I/O 请求包(I/O Request Packet)的简写。当应用程序发出 I/O 的请求后,由 Windows NT 内核的 I/O 管理器为这一请求打包并形成 IRP 传给驱动程序,而这个包能否被驱动程序处理要视驱动程序繁忙程度而定。假如当前驱动程序空闲且此前 IRP 队列为空,那么这个 IRP 将被驱动程序接受,否则它只能被排入队列等待。

DPC 则是推迟过程调用(Deferred Procedure Call)的简写,它配合硬件设备中断服务例程 ISR(Interrupt Server Routine)进行工作。由于 ISR 的 IRQL 比较高(IRQL = DIRQLs),通常只把必要的工作放进 ISR 中处理,以便 CPU 有更多时间来处理一些 IRQL 较低但更为紧要的任务,另一些中断收尾工作则放入 DPC 例程中延缓执

行。因为 DPC 的 IRQL 值为 Dispatch _ lEVEL<DIRQLs >,所以当 ISR 完成后,CPU 要等到程序的 IRQL 低于 Dispatch _ Level 且没有更多的 DPC 要处理时,此 DPC 才被响应,此前它只能被挂入 DPC 队列等待。

下面便对上述几个驱动程序共有模块的内容和作用做一介绍。

①装载和卸载模块。装载模块是 Windows NT 驱动程序加载系统的入口点,装载便意味着驱动程序获得进入内核模式执行的许可,驱动程序将被系统赋予访问硬件的特权。这一模块的编写流程如图 1 所示:

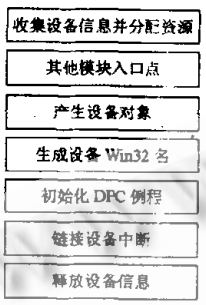


图 1

图中设备信息与资源是指设备占用的端口地址、DMA 通道以及中断号等情况,信息的收集对应系统注册表的查询,资源分配则要求对注册表进行修改。根据设备类型的不同,注册表查询分为两类情况。如果设备在 Windows NT 4.0 启动时没有自检信息,注册表中的查询地址为 HKEY _ LOCAL _ MACHINE/SYSTEM/CURRENTCONTROLSET/SERVICE,这类设备如大部分老式的 ISA 设备;当设备有自检处理时,如现在流行的 PNP 设备,查询地址则改为 HKEY _ LOCAL _ MACHINE/HARDWARE/DESCRIPION/SYSTEM。至于资源分配时对注册表的修改要在 HKEY _ LOCAL _ MACHINE/HARDWARE/RESOURCEMAP 目录下进行。另外,图中的 Win32 名是为了设备被 Windows NT 系统识别而生成。初始化 DPC 则要视设备中断而定,中断工作简单时可舍去 DPC。

卸载模块是帮助驱动程序退出系统内核的一段代码,具体工作为释放并删除驱动程序中存在的内核存储块,需要指出的是这种释放工作一定要彻底,否则整个 NT 系统极易崩溃,这是因为当驱动程序退出系统内核后,这些未经释放的存储块中所包含的内容对系统来讲已不可用,但系统却仍会把它当作自身的内部代码予以征用从而导致不可预料的运行错误。模块中通常所需删除的内容及其顺序为:设备中断、设备 Win32 名、设备硬件资源和设备对象。

②I/O 请求散转模块。这个模块承担应用程序和驱动程序之间的通信接口任务,它提供一系列功能代码供应用程序通过 Microsoft Win32 API 函数作为参数来调用,这些函数有 CreateFile( )、CloseFile( ) 以及 DeviceIoControl( )等,应用程序的 I/O 请求也将由这些函数的调用而产生,由于这些 I/O 请求将被系统打包形成 IRP 传给驱动程序,在此模块中驱动程序的首要工作就是解析 IRP 包并获得这些函数传递的功能代码,再根据解析结果转向相应的处理函数,具体编程时通常用 C 语言的 switch 语句来完成这一过程。需要说明的是本模块中的 I/O 请求不一定是对端口的实际操作,它只是应用程序和驱动程序之间所有交流信息的一种概括说法,真正的端口操作将由下一个模块完成。本模块编写流程如图 2 所示。

图 2

上面附图中的最后部分,即 IRP 的完成不容忽视,它是驱动程序在完成一个 I/O 请求后送回系统的反馈信息,系统要根据这一信息释放当前的 IRP 包,如果没有得到反馈,当下一次应用程序 I/O 请求来临时,系统会因为在上一个 IRP 地址空间建立新的 IRP 而导致运行出错。

③I/O 数据传输模块。设备驱动程序将在这个模块中实现硬件端口读写和硬件中断请求服务功能,此模块是设备驱动程序的功能核心,它由两个主函数组成,分别为端口传输和中断服务函数。其中端口传输函数是当一个 IRP 包含端口操作功能代码时,才会被上述 I/O 请求散转模块的 I/O 请求处理部分调用,程序员在这个函数中可用 DDK 提供的多个宏函数,如 READ _ PORT _ USHORT( )、WRITE _ PORT _ USHORT( )等实现对端口的直接读写;至于中断服务函数则由外部硬件中断触发,不被应用程序直接调用,内容由硬件操作目的决定。这两个函数的具体编写和以前在 DOS 下的同类函数相差不大,只是 DOS 和 Windows NT DDK 端口操作函数的替换问题,所以这里不再赘述。唯一特殊的是出于安全

性考虑，在中断服务函数中应有异常中断处理部分，这是因为在 Windows NT 下一个中断常常被共享，因此当一个属于其他设备的中断闯入本服务程序时，程序需通过读一些端口状态来摒弃此异常中断，显然这对防止设备误动作极为必要。

(2)驱动程序的编译和链接。DDK 中用 BUILD 命令来实现驱动程序的编译链接，但只有源文件还不能执行此命令，开发者还需做一些准备工作。

首先把所有的源文件放到一个目录下，接着编写两个小文件和这些源文件放在一起。第一个文件为 MAKEFILE，它只含一行代码"! INCLUDE $ (NTMAKEENV) MAKEFILE. DEF"，MAKEFILE 作用是为 BUILD 命令提供编译链接需要的一个标准 makefile. def 文件。另一个文件为 SOURCE，为 BUILD 执行提供一系列命令关键字，要注意的是这两个文件都没有后缀。下面是一个 SOURCE 文件的模板，实际应用中要用具体内容代替以下的中文说明。

TARGETNAME = 驱动程序名

TARGETTYPE = DRIVER //告诉 BUILD 让它按生成驱动程序的方式来编译链接源文件

TARGETPATH = 将要放置驱动程序可执行文件的起始目录

INCLUDES = 编译链接所需头文件的路径，一般为 (BASEDIR) \ INC; . . \ INC

BASEDIR 为 DDK 安装的路径

SOURCES = 驱动程序所有源文件列表，每个文件之间用空格隔开

第二步是在资源管理器中 TARGETPATH 所指目录下建立放置驱动程序执行文件的完整路径——TARGETPATH \ 1386 \ CHECKED(FREE)。

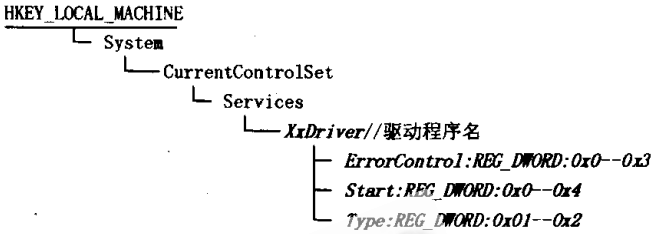
最后一步是从 Start Menu 中的 Windows NT DDK 子菜单进入 Checked Build 或 Free Build 环境(这两个环境类似于 Windows 95 下的 MS - DOS 方式)，并来到 MAKEFILE、SOURCE 和驱动程序各源码文件所在的目录下，键入 BUILD 后就可实现源文件的编译和链接并生成形式为 * . SYS 驱动程序可执行文件。

4. 设备驱动程序的安装

Windows NT 设备驱动程序的可执行文件生成后需要按一定步骤安装后才能运行。以下便是这种安装步骤的说明：

首先把 * . SYS 文件拷贝到 Windows NT 系统盘的 \ SYSTEM32 \ DRIVERS 目录下，一般为 C: \ WINNT \ SYSTEM32 \ DRIVERS。

随后按照下面所示路径修改系统注册表 Regedt32，加粗斜体字表示要添加的内容。



最后重新启动系统。

上面注册表中添加内容的意义说明如下：

ErrorControl 规定当驱动程序装载失败时系统响应方式，设备驱动程序一般取 0x1 或 0x2，0x1 指系统将记录这次装载失败并为用户弹出一个消息框，0x2 则表示系统记录失败后将用上次成功启动的配置重新启动系统；

Type 定义该驱动程序类型，0x1 表示内核型，0x2 表示文件系统型，设备驱动程序是内核驱动程序，所以要选择 0x1；

Start 则规定驱动程序装载时机，用户自己开发的设备驱动程序一般取 0x2 或 0x3，0x2 表示驱动程序将在 NT 系统启动后自动装载，而 0x3 则需用户手工装载，手工装载使用控制面版中的设备一项来完成。

5. 结束语

本文通过对设备驱动程序主要编写内容、编译链接过程和安装方法的介绍，力图让读者对这类程序的开发工作有一个整体的了解，其中阐述的源文件应遵循模块化编写的方法对实际编程最为重要，灵活运用这一方法将对提高开发工作的速度有着很大帮助。笔者曾在 Windows NT 4.0 下开发了数个 A/D 数据采集板、数字量输入输出隔离板等测控设备的驱动程序，并利用本文介绍的四大模块实现了模块化编程，从最后效果来看既省时又省力，所以希望这一方法对本文的每个读者也会有用。至于本文没有涉及到一些细节内容，比如各模块的具体代码编写，如果读者对它们感兴趣并想深入学习驱动程序编码，笔者推荐您参阅本文后附的参考书目 1。

参考文献

[1] 《Windows NT 4.0 设备驱动程序设计指南》机械工业出版社 1997

[2] 《Windows NT 技术内幕》清华大学出版社 卡斯特(美) 1993

(来稿时间：1998 年 11 月)