文章编号:1005-0523(2003)02-0086-04

Windows2000驱动程序的设计与实现

钟海峰

(华中科技大学 计算机学院,湖北 武汉 430074)

摘要:扼要介绍了Windows2000设备驱动程序的原理、结构和运行环境.然后着重介绍了WDM驱动程序的构成和实现方法并给出一个USB设备的驱动程序作为实例.

关 键 词:Windows2000;内核模式驱动程序;WDM;USB设备的驱动程序
中图分类号:TP391.6 **文献标识码**:A

1 引 言

设备驱动程序是连接应用程序、硬件以及操作系统的桥梁,是硬件设备连接到计算机系统的软件接口.Windows 2000以其界面友好、广泛支持新一代硬件设备、更好的可管理性及具有工业级的可靠性而日渐流行.而在Windows2000操作系统下,执行于用户态的应用程序代码不能直接访问硬件,而是通过调用执行于核心态的设备驱动程序提供的各种服务间接地对硬件资源进行访问.这一机制在确保了系统的安全、稳定性的同时,也使编写windows2000下的驱动程序也变得更为复杂.为方便广大用户编写Windows2000下的驱动程序,微软公司提出了Windows驱动程序模型WDM(Windows Driver Model)的概念.为Windows2000/98操作系统提供了I/O服务的统一结构模型和一些二进制兼容的类驱动程序.

2 Windows2000驱动程序简介

在最高层上Windows2000支持两种基本模式的驱动程序类型,即用户模式和内核模式.用户模式驱动程序是按用户模式运行的系统级代码,它们不能直接访问硬件,必须依赖于内核模式驱动程序.内核模式驱动程序则由运行于内核模式的系统级代码组成,是作为Windows 2000执行体部分来运行,被用来直接控制硬件.

Windows2000内核模式驱动程序采用了一种特有的分层体系结构,整个系统是由"包"驱动的,并且像操作系统自身,也由一些按照需求设计的单独的、模块化的组件组成.如图1所示.

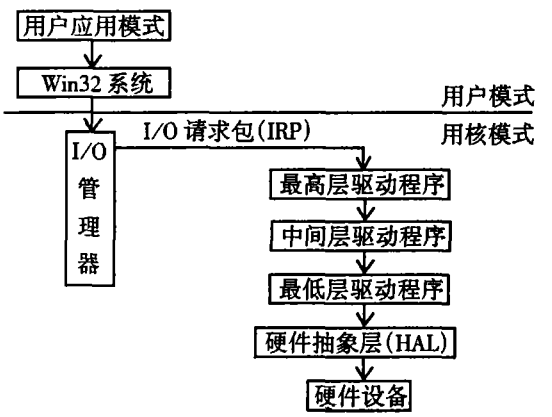

图1 内核模式驱动程序的体系结构

三种基本类型的内核模式驱动程序构成了Windows 2000驱动程序的层次结构,每一种都有稍微不同的结构和完全不同的功能:(1)高层驱动程序,例如系统支持的FAT、NTFS和CDFS文件系统驱

收稿日期:2002-12-15
作者简介:钟海峰(1970-),男,江西南昌人,华中科技大学在职硕士.

动程序，为服务请求者提供非物理的抽象.(2)中间层驱动程序，例如类、微型和过滤器驱动程序，将非物理的抽象转换为特定的设备请求，但通常不负责对硬件资源进行直接的、寄存器级的操纵.(3)最低层驱动程序，例如 PnP 硬件总线驱动程序，直接控制物理外围设备.不依赖于较低层驱动程序.

Windows2000 系统一般是由内核中的 I/O 管理器来调用驱动程序进行硬件操作.I/O 管理器创建 I/O 请求包(I/O RequestPacket IRP)，通过把用户程序对硬件的操作需求转变成相应的 IRP 来控制硬件驱动程序的启动、运行、关闭和相互协调.通常 IRP 先被送到驱动程序堆栈的最上层，然后逐渐过滤到下面的驱动程序.每一层驱动程序都可以采用以下三种方法之一处理 IRP:(1)仅仅是向下层传递该 IRP.(2)直接处理完该 IRP，不再向下传递.(3)驱动程序既处理了 IRP，又把 IRP 传递下去.

所有的 Windows2000 内核模式驱动程序，包括 WDM 驱动程序，都包含一组系统定义的标准驱动程序例程和一些由设计者决定的内部例程.这使它们对于操作系统来说都呈现相同的结构，一个驱动程序可以不经过转换当前的驱动程序或 I/O 系统，就能容易地被插入到分层结构中，从而实现了驱动程序的灵活性和健壮性.

3　WDM 的结构和实现

3.1　WDM 简介

WDM 驱动程序是一种 PnP 驱动程序，同时还遵循电源管理协议，它直接对硬件抽象层 HAL 操作，能在 Windows 98 和 Windows 2000 间实现源代码级兼容.

WDM 模型包括两个方面：描述驱动程序标准结构的 WDM 模型和微软公司为标准类型的设备实现的一系列模块化、分层的总线驱动程序和类驱动程序.WDM 模型描述了驱动程序的加载、如何响应用户应用程序的请求和与硬件打交道.而 Microsoft 提供的总线和类驱动程序简化了设备驱动程序的编写：用户的驱动程序可以不必直接跟硬件打交道，而只需与下层的总线或类驱动程序打交道，由它们完成烦琐的硬件控制任务.

3.2　WDM 模型构造

WDM 模型使用了如图 2 的层次结构.图中左边是一个设备对象堆栈.设备对象是系统为帮助软件控制硬件设备而创建的数据结构.处于堆栈最底层的称为物理设备对象(physical device object，PDO).向上依次为下层过滤器设备对象(filter device object，FIDO)、功能设备对象(functional device object，FDO)和上层过滤器设备对象(filter device object，FIDO).它们分别对应总线驱动程序、下层过滤器驱动程序、功能驱动程序、和上层过滤器驱动程序.

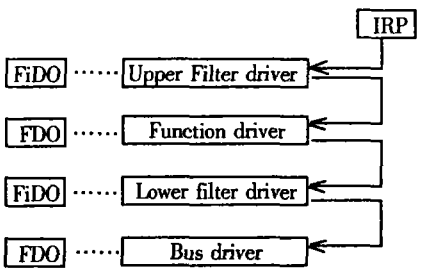

图 2　WDM 驱动程序的层次结构

在 WDM 驱动程序堆栈中，不同位置的驱动程序扮演了不同的角色：功能(function)驱动程序，即硬件设备驱动程序，负责实现设备的功能，包含了使硬件工作的所有细节.总线(bus)驱动程序，负责发现总线上的全部设备，管理设备与计算机之间的连接，检测设备何时添加和删除.过滤器(filter)驱动程序，多数情况下用于修改现有功能驱动程序的行为和为用户提供额外的处理功能.

3.3　WDM 驱动程序的构成

一个 WDM 驱动程序外部表征为操作系统软件(通常是 I/O 软件)调用的一个函数集，它包括五个必要的、系统定义的标准驱动程序函数，即 DriverEntry，AddDevice，DispatchPnp，DispatchPower，DipatchWmi，加上一些可选的标准函数与内部函数，例如，需要对 IRP 排队的驱动程序一般都有一个 StartIo 函数，执行 DMA 传输的驱动程序应有一个 AdapterControl 函数.操作系统正是调用这些函数来执行针对 IRP 的各种 I/O 操作的.图 3 说明了这一概念.

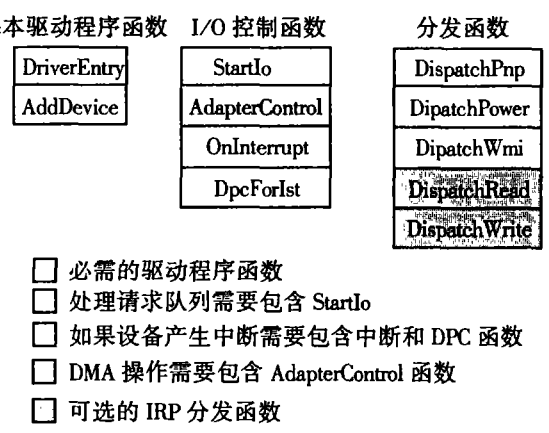

图 3　WDM 设备驱动程序函数集

3.4 WDM 驱动程序的实现

WDM 驱动程序是 PE 格式的、以 SYS 为文件扩展名的动态链接库.和一般的 C 语言程序不一样,WDM 驱动程序没有作为入口的 main 函数或 WinMain 函数,而是与 DLL 相类似,它向操作系统显露一个名称为 DriverEntry 的入口函数,在启动驱动程序的时候,操作系统将调用这个入口函数来初始化驱动程序范围的数据结构和资源,包括输出其他驱动程序入口点、初始化驱动程序使用的特定对象并设置每个驱动程序系统资源,其主要工作是把各种函数指针填入驱动程序对象.WDM 驱动程序中只有 DriverEntry 函数的名字是固定的,其他所有函数名都可变且都要由这个函数向系统注册.

下面以一个 USB 设备驱动程序的主要代码为例来说 WDM 驱动程序的实现.

```
//USB Driver 的入口函数,第一个参数指向驱动程序对象,第二个参数指相关设备配置参数在注册表中的位置
extern "C" NTSTATUS DriverEntry(IN
    PDRIVER_OBJECT DriverObject,IN
    PUNICODE_STRING RegistryPath)
{//为驱动程序的其它入口点设置了函数指针.在这里,用了 DriverUnload、AddDevice、StartIo 函数
  DriverObject->DriverUnload = UsbUnload;
DriverObject->DriverExtension->AddDevice =
    UsbAddDevice;
DriverObject->DriverStartIo = UsbStartIo;
//以下为 IRP 处理函数入口指针
//即插即用管理,电源管理,系统配置管理
DriverObject->MajorFunction[IRP_MJ_PNP]
= UsbPnp;
  DriverObject->MajorFunction[IRP_MJ_POWER] =
    UsbPower;
DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTRL]
= UsbWmi;
//设备的打开与关闭
DriverObject->MajorFunction[IRP_MJ_CREATE] =
    UsbCreate;
    DriverObject->MajorFunction[IRP_MJ_CLOSE] =
UsbClose;
//基本 I/O 请求处理,读写
DriverObject->MajorFunction[IRP_MJ_READ] =
UsbRead;
DriverObject->MajorFunction[IRP_MJ_WRITE] =
UsbWrite;
//为 Wmi 备份 RegistryPath 串的内容
//servkey 是一个 UNICODE_STRING 类型的全局变量
servkey.Buffer = (PWSTR)
    ExAllocatePool(PagedPool,
    RegistryPath->Length + sizeof(WCHAR));
  if(! servkey.Buffer)
return STATUS_INSUFFICIENT_RESOURCES;
servkey.MaximumLength = RegistryPath->Length +
    sizeof(WCHAR);
  RtlCopyUnicodeString(&servkey,
RegistryPath);
//返回 STATUS_SUCCESS 指出函数成功
  return STATUS_SUCCESS;
}
```

在 DriveEntry 中注册的各 IRP 处理函数中,设备的打开与关闭函数及基本 I/O 请求处理函数等在实现上与在 NT 环境中基本相同,限于篇幅,在此仅给出设备加载函数 UsbAddDevice 和即插即用函数 UsbPnp 的主要代码实例.UsbAddDevice 函数将创建和初始化一个功能设备对象(FDO),并把该对象连接到设备栈中供当前驱动程序使用.UsbPnp 函数则支持启动设备、撤除设备、停止设备等等多种功能.

```
//设备加载函数 UsbAddDevice
NTSTATUS UsbAddDevice(IN PDRIVER_OBJECT
    DriverObject, IN PDEVICE_OBJECT pdo)
{//调用 IoCreateDevice 函数创建设备对象
PDEVICE_OBJECT fdo;
NTSTATUS status = IoCreateDevice(DriverObject,
    sizeof(USB_DEVICE_EXTENSION),NULL,FILE_
    DEVICE_UNKNOWN, 0, FALSE, &fdo);
//如创建失败,返回一个错误代码,不改变 fdo 中的值
    if (! NT_SUCCESS(status)) return status;
//在 ux 中注册功能设备对象
PUSB_DEVICE_EXTENSION ux = (PUSB_DEVICE_
    EXTENSION)fdo->DeviceExtension;
ux->fdo = fdo;
//注册设备接口,
IoRegisterDeviceInterface(pdo, &Usb_GUID,
    NULL, &ux->ifsymLinkName);
// 如果发现错误,释放刚创建的设备对象并返回状态码
if (! NT_SUCCESS(status))
{  IoDeleteDevice(fdo);
    return status;
```

```
}
//启用注册的设备接口
IoSetDeviceInterfaceState(&ux->
    ifsymLinkName, TRUE)
//将功能设备对象加入到设备对象栈
ux->NextDevice=
    IoAttachDeviceToDeviceStack(fdo,pdo)
    ……
return STATUS-SUCCESS;
}
//即插即用函数 UsbPnp
NTSTATUS UsbPnp (IN PDEVICE_OBJECT fdo,
                 IN PIRP Irp)
{  NTSTATUS status=STATUS_SUCCESS;
PUSB_DEVICE_EXTENSION ux=(PUSB_DEVICE_
    EXTENSION)fdo—>DeviceExtension;
// 用UsbIrpstack 指向记录 IRP 的所有参数及副功能码的堆栈
    单元
PIO_STACK_LOCATION UsbIrpstack =
    IoGetCurrentIrpStackLocation(Irp);
ULONG MinorFunction = UsbIrpstack->MinorFunction;
//处理 PNP 副功能码的例程
switch(MinorFunction)
{//处理设备启动例程
    case IRP_MN_START_DEVICE:
      status=PnpStartDeviceHandler(fdo,Irp);
      break;
//关闭设备
case IRP_MN_STOP_DEVICE:
      status=PnpStopDeviceHandler(fdo,Irp);
      break;
      //其它 PNP 副功能码处理例程
      ……
      //处理不认识的副功能码
      default:
      status=PnpdefaultHandler(fdo,Irp);
      }
    return status;
}
```

3.5　实现 WDM 驱动程序应注意的几个问题

除了使用 Windows DDK 进行开发外,也可以借助一些辅助工具如 RiverStudio 等.

DDK 的 src \ 目录下有一个庞大的模板代码,几乎覆盖了所有类型的设备驱动程序、高层驱动程序和过滤器驱动程序.在开始开发之前,可以先在这个样板库下面寻找类似的例程以减少工作量.

在分阶段来设计驱动程序,如果从一开始就确定好最终要创建的所有 Device 对象将有助于解决同步问题.如果把与设备有关的任何需要保持的信息都放到 Device Extension 里,而尽量不要使用全局变量和静态变量,则将有助于保证驱动程序是完全可重入的.

由于 WDM 是跨平台和跨操作系统的的驱动程序模型,所以在编写时一定不要使用汇编.且要想在这两个平台上使用,就必须分别在这两个平台下编译.

参考文献:

[1] Document, Windows2000 DDK, http://www.microsoft.com/ddk/.

[2] Walter Oney. programming the Microsoft Windows Driver Model[M]. Microsoft press,2000.

[3] 尤晋元,史美林. Windows 操作系统原理[M]. 北京:机械工业出版社,2001.

[4] Art Baker. The Windows 2000 Device Driver book (second edition) [M]. Prentice Hall PTR,2001.

Design and Implementation of Windows2000 Driver

ZHONG Hai-feng

(Institute of Computer, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: Fristly, this paper introduces simply the theory, structure and running environment of Windows2000 Device Driver. And then, it introduces particularly the organization and implementation of WDM device driver and gives an implementation of USB device driver on WDM.

Key words: Windows2000; Kernel-Mode Driver; WDM; USB Device Driver