Windows 平台驱动程序新架构分析

Analysis of Architecture of the Windows Driver

王兰英 居锦武 （四川理工学院计算机科学系 四川自贡 643000）

摘 要： 文章分析了用 Windows 驱动程序模型(WDM)设计驱动程序面临的问题,分析了 Windows 驱动程序架构(WDF)简化驱动程序设计工作的方法,介绍了新构架的概念及基本原理,详细分析了对象及事件回调机制,详细分析了新构架在处理同步问题所采用的线程模型、锁模型、中断同步模型的原理,通过一个基于 WDF 的驱动程序实例来说明了新架构驱动程序的设计方法。

关键词： Windows 驱动程序模型 Windows 驱动程序架构 中断请求级 自旋锁 同步 事件

1 引言

在 Windows NT 系列操作系统上,必须编写内核模式的设备驱动程序才能对硬件设备进行控制。工作于 Windows 2000/XP/2003 上的 Windows 驱动程序模型(WDM)在 NT 驱动程序的基础上,加入了即插即用及电源管理等新的特性,简化了内核模式驱动的设计,但由于其功能的大大加强,WDM 驱动程序的设计仍然是非常困难的。文章介绍了微软为简化驱动程序设计工作而发布的驱动程序开发新架构 – Windows 驱动程序框架(WDF),分析了 WDF 的概念、特点、及简化设计工作的手段,最后给出一个 WDF 驱动程序实例。

2 Windows 驱动程序架构(WDF)

WDM 驱动程序的设计非常复杂,即使对设备的控制只需要交换非常少量的数据,也必须满足 WDM 驱动程序所要求的复杂结构。在编写内核模式驱动程序的过程中,开发者需要解决的问题非常多,这些问题使得驱动程序的设计具有非常高的专业性,这把大量的只是想与设备交换少量数据的程序员排除在驱动程序设计大门之外。

WDF 是微软的新一代驱动程序开发模型,它包括一整套对驱动程序进行开发设计、发布、维护的部件,WDF 支持面向对象的、基于事件驱动机制的驱动程序设计思想。WDF 实现了一些驱动程序所必需的基本特性,为驱动程序提供缺省的处理,并管理与操作系统的交互,这使设计者能够集中精力在特定硬件设备的处理上,而不是操作系统自身,从而大大简化设计。WDF 模型支持内核模式与用户模式两种驱动程序。

2.1 面向对象与事件驱动

WDF 驱动程序基于面对对象的概念。NT 系列操作系统的内核都是面向对象的,一个全局的对象管理器对系统中的对象进行管理,比如驱动程序对象和设备对象。同时,它还在适当的时候负责调用对象的例程。在 WDM 驱动程序的设计过程中,主要工作就是设计各种对象的多种例程,这些例程由操作系统来决定何时调用它们。WDF 驱动程序则是基于事件驱动机制的,系统为各种 WDF 对象提供多种事件,并允许为事件注册自己编写的回调例程,当操作系统中相应的事件发生时,系统将调用事件的回调例程。这种结构简化了 WDF 驱动程序的设计工作,程序员可集中精力在事件回调例程的编写上,而不必去理解何时及为何要调用回调例程。这种程序设计的思想与可视化的应用程序设计思想类似。

2.2 提供缺省行为

一个即使不完成任何具体工作的 WDM 驱动程序也必须具备大量的例程,这些例程使 WDM 驱动程序具有一些标准的结构,完成一些标准的处理,满足 WDM 驱动程序的标准要求。这些繁琐的工作是 WDM 驱动程序复杂的一个重要原因。

微软希望用工程化的方法来编写驱动程序,以结束所要面临的众多编程问题。WDF 体系为普通的驱动程序提供了缺省的行为和简单的接口,用来处理这些每个驱动程序都要做的工作。比如,一个简单的 WDM

驱动程序为了实现 RP_MJ_PNP 和 IRP_MJ_POWER 请求的处理，可能需要一二千行程序，来控制驱动程序的启动、停止，控制计算机的休眠和唤醒。即使对于一个没有即插即用或电源管理能力的老的设备，你也需要这些代码，这是 WDM 体系的要求。而在 WDF 中，通过简化接口和提供缺省行为的代码，将极大的简化这一问题。开发者可以将主要精力放在与设备的交互上。

2.3 克服体系限制

在 WDM 体系中，可以通过使用小端口驱动程序来简化特殊设备驱动程序的编写，但这将使得体系过于僵硬，不灵活，不能跟上创新的步调。例如，网卡的驱动程序要符合网络设备接口标准(NDIS)。NDIS 允许网卡生产商所设计的驱动程序能够适用于所有 Windows 平台。NDIS 是通过为微驱动程序提供完全的设备驱动程序接口(DDI)来实现平台独立性的。但是，由于 NDIS 内部结构的原因，要为一个 NDIS 不支持的网卡写一个微驱动程序是不可能的。比如一个 USB 网卡，除非你能直接访问 USBD 库和内核，你无法对设备进行控制。即便你能为一个 USB 网卡编写驱动程序来控制它，但是由于该驱动程序并不符合 NDIS 结构，将无法通过硬件兼容性测试。WDF 对 WDM 程序设计模型进行了扩展，以便你能为特殊硬件设备编写驱动程序。

3 WDF 架构的特点

3.1 事件，方法，属性

WDF 对象支持方法和属性，对象能够产生使注册的回调函数被调用的事件。比如对于 AddDevice 和 DriverUnload 例程，在 WDM 体系中，它们的函数指针被放到驱动程序对象中，它们在特定的时刻被操作系统调用，用于管理硬件设备的变化。在 WDF 体系中，完成同样功能的例程是与驱动程序对象的事件相关联的两个回调函数。当创建一个 WDF 驱动程序对象时，将 EvtDriverDeviceAdd 函数注册为事件" 检测到新设备" 的回调函数，将 EvtDriverUnload 函数注册为事件" 驱动程序被卸载" 的回调函数。当相应的事件在驱动程序对象上发生时，这两个函数被调用。操作系统在检测到有新设备加入时，即调用已注册的 EvtDriverDeviceAdd 回调函数。当驱动程序卸载事件发生时，系统会调用已注册的 EvtDriverUnload 回调函数。

使用事件回调机制的优点是灵活性，这些事件的处理是可选的，如果注册了回调函数，则对事件进行处理，如果没有注册回调函数，系统会采用缺省行为来处理事件。

3.2 同步

在内核模式下，驱动程序的编写者很难正确使用内核所提供的同步原语。在 Windows 的多任务机制下，或是在多处理器机器上，由于不同的目的，程序可能对共享资源发出多重访问请求，很难搞清楚哪些请求要进行同步，哪些请求则不需要。例如编写一个标准的通用异步串行接口的驱动程序时，串行接口用中断来通知 CPU 所发生的事件，CPU 在中断服务程序(ISR)中对硬件寄存器进行访问。驱动程序要给应用程序提供大量的操作接口，如设置串行接口的工作方式、波特率、读数据、写数据等。一些操作要在 ISR 中完成，一些操作在其它例程中完成，它们可能都要对同一个寄存器进行读写访问。这种情况下，访问逻辑变得非常复杂。这些访问寄存器的例程可能工作于不同的中断请求级(IRQL)上，ISR 工作于 DIRQL 级上，其它的例程可能工作于 DISPATCH_LEVEL 级或是 PASSIVE_LEVEL 级上。驱动程序可能需要在任何 IRQL 上接受来自应用程序的读、写或是其它的请求，而在高于 DISPATCH_LEVEL 级上是不能访问分页内存的，一些内核函数也不能用，在这种情况下，同步所有访问串行接口硬件寄存器的请求将是一件非常复杂的工作。

在原来的 WDM 驱动程序中，标准做法是，对于工作于 DIRQL 级的 ISR，可采用 KeSynchronizeExecution 或是 KeAcquireInterruptSpinLock 进行同步，而对于工作于 DISPATCH_LEVEL 级的其它请求，可采用自旋锁来实现同步。但系统的同步对象包括自旋锁、互斥对象、快速互斥对象、信号量及其它，它们似乎都可以完成同样的同步工作，这些同步对象都可以用来在多个线程访问共享资源时进行同步，如何选择正确的同步对象呢？这里的关键问题是中断请求级 IRQL，同步原语的使用与 IRQL 密切相关。在 DIRQL 这样的高 IRQL 级上，系统不允许线程被阻塞。对于 IRP_MJ_READ 和 IRP_MJ_WRITE 之类的 IRP 请求，可以采用队列的方法来进行串行访问。

WDF 驱动程序试图在各种不同 IRQL 级提供相同的形式来实现同步，这包括线程、锁与中断同步模型。

这些模型与 WDM 驱动程序的行为完全不同。

3.3 线程模型与锁模型

WDF 实现同步的第一种方式是线程模型，在 WDF 驱动程序的 DriverEntry 例程中，通过传入的参数，来声明驱动程序的工作方式是同步还是异步。当驱动程序工作于同步方式时，工作于 PASSIVE_LEVEL 级的大部分驱动程序的回调函数都允许在同一个系统线程的上下文环境中被阻塞。当驱动程序工作于异步方式时，驱动程序的回调函数将在 DISPATCH_LEVEL 级上被调用。在 DISPATCH_LEVEL 级上系统不允许线程被阻塞，因此如果使用异步线程模型，就不允许事件的回调例程被阻塞。这意味着，如果驱动程序采用同步线程模型，应用程序对它的访问自然是同步的，因为如果驱动程序的回调函数因暂时无法完成请求而阻塞，则应用程序也将阻塞，直到当前访问完成，回调函数才会返回。

WDF 实现同步的第二种方式是锁模型，包括 4 种架构：驱动程序、设备、队列与无锁。驱动程序根据请求来决定在何种 IRQL 上何时加锁及采用何种锁来实现同步。如果驱动程序指定锁模型为" no locking"，同时指定异步线程模型，则 WDF 驱动程序就象 WDM 驱动程序一样，在任意 IRQL 上都可能调用回调函数。对于其它三种锁模型，系统在调用驱动程序的事件回调例程之前，将首先对对象进行加锁处理，这个锁由系统维护。根据驱动程序所采用的线程模型，该锁可能是一个自旋锁或是快速互斥对象。

4 实例

最后，通过一个 WDF 驱动程序的例子来了解其结构与编写方法。这是一个可以在系统上安装、加载、卸载的 WDF 驱动程序，但它并不完成任何功能，这个程序只是一个框架。

```
extern " C" NTSTATUS DriverEntry( PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath ) //驱动程序入口函数
{ PAGED_CODE( );
DfwTraceDbgPrint ( DRIVERNAME " - Version % d. % 2. 2d. % 3. 3d, % s % s\n",
VERMAJOR, VERMINOR, BUILD, __DATE__, __TIME__ );//DfwTraceDbgPrint 打印出调试信息
DFW_DRIVER_CONFIG config = { sizeof ( DFW_DRIVER_CONFIG) } ;//此结构用于驱动程序对象的创建
config. DeviceExtensionSize = 0;
config. RequestContextSize = 0;
config. Events. EvtDriverDeviceAdd = EvtDriverDeviceAdd;//注册新增设备事件回调函数
config. Events. EvtDriverUnload = EvtDriverUnload;//注册设备卸载事件回调函数
config. DriverInitFlags = 0;
config. LockingConfig = DfwLockingDevice;//指定对设备加锁
config. ThreadingConfig = DfwThreadingAsynchronous;//指定采用异步线程模型
config. SynchronizationConfig = DfwSynchronizationNone;//指定对硬件的访问无需同步
DFWDRIVER Driver;
DFWSTATUS status = DfwDriverCreate ( DriverObject, RegistryPath, NULL, &config, &Driver ) ;//创建驱动程序对象
if ( ! NT_SUCCESS ( status ) ) DfwTraceError ( DRIVERNAME " - DfwDriverCreate failed - % X\n", status) ;
return status;
}
DFWSTATUS EvtDriverDeviceAdd ( DFWDRIVER hDriver, DFWDEVICE hDevice )
{//新增设备事件回调函数
DfwTraceDbgPrint ( DRIVERNAME " - EvtDriverDeviceAdd entered - IRQL is % d\n", KeGetCurrentIrql ( ) ) ;//打印调试信息
DFWSTATUS status;
status = DfwDeviceInitialize ( hDevice ) ;//初始化设备对象
if ( ! NT_SUCCESS( status ) )
{ DfwTraceError ( DRIVERNAME " - DfwDeviceInitialize failed - % X\n", status) ;
return status; }
DFW_FDO_EVENT_CALLBACKS callbacks;
DFW_FDO_EVENT_CALLBACKS_INIT ( &callbacks) ;
status = DfwDeviceRegisterFdoCallbacks ( hDevice, &callbacks) ;//注册 PnP IRP 的处理函数
```

```
    if ( ! NT_SUCCESS( status ) )
      { DfwTraceError ( DRIVERNAME " - DfwDevice-
RegisterFdoCallbacks failed - %X\n", status);
        return status; }
    return status; }
    VOID EvtDriverUnload( DFWDRIVER Driver ) // 设备
卸载事件回调函数
    { PAGED_CODE( );
    DfwTraceDbgPrint ( DRIVERNAME " - Unloading
driver - IRQL is %d\n", KeGetCurrentIrql( ) );
    }
```

5 结语

文章对 windows 驱动程序新构架进行了详细的介绍。WDF 驱动程序模型在 WDM 驱动程序模型的基础上工作，通过向驱动程序编写者提供一个以对象为中心的接口，以及事件与其回调函数的思想，WDF 简化了特定类型设备的驱动程序的编写工作。

参考文献

1 郭艳、苗克坚，Windows 2000 下 WDM 驱动程序的研究与开发[J]，计算机工程，2006:32(22).

2 王兰英、居锦武，Windows 内核模式驱动程序运行环境的分析[J]，微计算机信息，2005:21(11X)201-202,197.

3 刘鸿、王平、俞伟，WDM 驱动程序开发疑难分析[J]，计算机应用，2003:23(6)112-113,116.

4 [美] CHRIS CANT Writing Windows WDM Device Drivers [M]，北京:机械工业出版社，2000.1.

5 [美]David A Solomon Windows 2000 内部揭密[M]，北京:机械工业出版社，2001.10.

6 [美]Microsoft Corporation Windows 2000 驱动程序开发大全[M]，北京:机械工业出版社，2001.8.

7 郭益昆，VC++.NET 开发驱动程序详解[M]，北京希望电子出版社，2002.4.

8 武安河，Windows 2000/XP WDM 设备驱动程序开发[M]，北京:电子工业出版社，2005.