ISA数据采集卡的WDM驱动程序实现

张 龙

清华大学工程物理系(100084)

2008-11-18

摘 要： ISA设备在WINDOWS新的驱动程序模型WDM(WINDOWS DRIVER MODEL)中没有获得很好的支持,使用WDM实现需要一些特殊处理。主要讨论老式ISA设备的WDM驱动程序的设计与实现。

关键词： Windows驱动程序模型(WDM) 工业标准结构(ISA) 数据采集

WDM作为微软为WINDOWS 2000及以后版本准备的新驱动模型,它可以在WINDOWS 98及以后的操作系统上共享。这种具有跨平台性的设备驱动模型可以大大简化驱动程序的开发工作。WDM支持PNP(即插即用),为用户的PNP设备驱动完成了大量的底层工作。这也使得WDM对各种老设备,特别是那些不具有即插即用特性硬件的支持明显不足。在科研和工控中,最常用的数据采集卡通常都是基于PC总线的不能为PNP提供硬件支持的ISA设备。为这类硬件编写非WDM驱动程序只能局部支持PNP特性,而且需要做大量额外的工作如:必须检测硬件,为硬件创建设备对象(用于代表硬件),配置并初始化硬件使其正常工作,这些工作非常复杂。利用WDM可以避免这些麻烦。

1 WDM的结构

WDM实际上是一个编写驱动程序的规范。其驱动程序结构的特点和WINDOWS程序设计的消息驱动机制很相像,采用IRP驱动机制。WINDOWS 98和WINDOWS 2000处理IRP的方式一样,本质却完全不同。

WINDOWS 2000主要由I/O管理器来管理驱动程序的行为,管理的方法就是给驱动程序发送各种IRP,同时I/O管理器还负责传递这个设备的用户请求给驱动程序。驱动程序不能直接操作硬件设备,而是通过一个叫做HAL(硬件设备抽象层)的层来访问底层设备。HAL完成对各种硬件差异的屏蔽。Windows 98内核与Windows 2000内核有很大的差别。Windows 98操作系统内核称为虚拟机管理器(VMM)。虚拟设备驱动程序(VxD)则使设备虚拟化,从而与虚拟机管理器形成虚拟机。Windows 9x的内核不支持IRP,它通过模拟的方法支持WDM和IRP。Windows 98包含了NTKERN.VXD (VMM32.VXD)系统模块,该模块含有大量Windows NT内核支持函数的Windows实现。NTKERN.VXD使用与Windows 2000相同的方式创建IRP并发送IRP到WDM驱动程序。也就是说,Windows 98是使用VxD技术实现WDM的。尽管Windows 98和Windows 2000的内核完全不同,但是WDM的结构在Windows 98和2000下可以认为是完全相同的。

WDM驱动程序面对的主要内容是一系列内核驱动对象。这些对象包括驱动对象、设备对象,还有一系列的资源抽象对象如中断对象、适配器对象(处理DMA操作)、内核模式派发器对象、控制器对象、推迟过程对象、定时器对象、设备队列对象、回调对象等。其中驱动对象由I/O管理器负责创建和管理。设备对象代表用户需要操作的物理或者是逻辑设备。设备对象负责管理自己设备所需要的各种资源对象。这些资源对象有的代表真正的物理设备如中断对象、适配器对象、定时器对象;有的对象则是逻辑对象,它们是操作系统实现自己的特性所必需的,如推迟过程调用就是与操作系统的抢占式多任务的实现有关。WDM驱动程序利用IRP和这些对象完成硬件设备的抽象化,并对应用程序提供统一操作接口。这就是所谓的WDM规范。

WDM驱动程序的结构很简单。它的主体是一个入口函数DriverEntry。DriverEntry的第一个参数是指针,指向一个刚被初始化的驱动程序对象,该对象代表驱动程序。WDM驱动程序的DriverEntry例程完成对这个对象的初始化并返回。WDM驱动程序的DriverEntry例程的主要工作是把各种函数指针填入驱动程序对象。这些指针为操作系统指明了驱动程序容器中各种子例程的位置。它包括下面这些指针成员:

· DriverUnload 指向驱动程序的清除例程。I/O管理器会在卸载驱动程序前调用该例程。通常WDM驱动程序的DriverEntry例程一般不分配任何资源,所以DriverUnload例程也没有什么清除工作要做。

· DriverExtension->AddDevice 指向驱动程序的AddDevice函数。AddDevice函数是WDM驱动程序的一个特殊函数,PNP管理器为每个设备实例调用该函数。它创建一个设备对象并把它连接到该驱动程序的设备堆栈中。在这个函数里,设备对象被创建,设备需要的中断、DMA等资源被挂接到这个设备对象上,相应地处理函数,如中断服务例程等也被指定。

· DriverStartIo,如果驱动程序使用标准的串行IRP,则必须使用这个函数,使它指向驱动程序的StartIo例程。

· MajorFunction是一个指针数组,它主要处理各种用户的I/O请求IRP。默认情况下,用户的所有I/O请求都会由I/O管理器返回失败。驱动程序为要处理的IRP指定相应的派遣函数。

此外,还有一些重要的例程如用于处理中断操作的中断服务例程,为了提高系统中断处理效率的中断服务推迟调用例程等。

驱动程序的其余部分就是前面所指定的那些例程的实现。

2 为ISA设备分配资源

虽然WDM驱动程序可以简化很多工作,但是给ISA设备编写WDM驱动程序也有困难。这个困难是ISA设备所需资源的分配问题。根据PNP的规范,支持PNP的设备如PCI总线设备有相应的寄存器标识自己和自己可以接受的资源,如中断、端口范围、内存范围等,并且有相应的逻辑支持配置资源。这使得操作系统启动过程中可以动态地规划和调整各个设备的资源而不需要用户的干预。为了使ISA设备具有这种PNP的特性,可以通过PNP管理器和INF文件完成ISA设备的自动资源分配。PNP管理器(PNP MANAGER)依赖INF文件完成

热点专题

- 信心09,冬天来了,春天还会远吗?
- 低功耗技术，是鸡还是蛋?
- 华北计算机系统工程研究所(电子六所)总结表彰暨春节联欢会
- Powerwise高效能解决方案
- 2008Security China中国国际社会公共安全产品博览会
- 视频信号处理技术
- 2008嵌入式技术创新及...
- 2008飞思卡尔技术论坛
- Altera公司SOPC...
- 第十届高交会电子展
- 科技闪耀北京奥运
- ADLINK DAY—2008年量测与自动化技术国际高峰研讨会
- 中国电子学会Xilinx杯开放源码硬件创新大赛
- 赛灵思公司Virtex-5系列FPGA
- 3G知识
- IPTV
- 触摸屏技术
- RoHS

杂志精华

- 基于CC2430的无线传感器...
- 无线传感器网络应用系统综述
- 无线传感器网络在野外测量中的...
- 基于竞争的无线传感器网络
- 用于矿井环境监测的无线传感器...
- 具有自适应通信能力的无线传感...
- 基于传感器网络技术的深孔测径...
- 基于无线传感器网络的家庭安防...
- 基于ATmega128L与C...
- 无线传感器网络中移动节点设备...

PNP管理。INF文件为设备、设备驱动程序、操作系统的安装过程提供信息(包括PNP MANAGER)。为了兼容非PNP设备,INF文件支持几个命令为设备分配资源。这些信息被提供给PNP管理器,并且被PNP管理器用来与系统其它部分协商以完成非PNP设备的自动资源分配。LOGCONFIG为设备制定一个可选的资源设定。还有相关的IRQCONFIG、IOCONFIG、DMACONFIG、MEMCONFIG子命令,它们为设备指定具体的可选资源。语法如下:

```
[]
ConfigPriority=
[DMAConfig=]
[IOConfig=]
[IRQConfig=]
[MemConfig=]
```

其中CONFIGPRIOITY命令指定这个LOGCONFIG项的优先级。其余子命令表示设备选用的硬件资源。下面是一个例子:

```
[CX2590.Install]
……;其它命令
LogConfig=CX2590_DMA;指定配置项的名称
[CX2590_DMA]                          ;配置项的名称
ConfigPriority=NORMAL                 ;配置的优先级
IOConfig=4@300-3ff%3ff(3ff: : )       ;指定IO范围
IRQConfig=4, 5, 9, 10, 11             ;指定可选的中断
DMAConfig=0, 1, 2, 3                  ;指定可选的DMA
```

在INF文件中加入LOGCONFIG命令可以解决ISA总线设备资源的自动分配问题,就可以为ISA总线设备编写WDM驱动程序。

3 一个ISA设备WDM驱动程序的实现

使用Numega公司的开发工具DriverStudio 2.01可以大大地简化驱动程序的开发过程。DriverStudio开发通用内核模式的开发包是DriverWorks,DriverWorks是一个面向对象的工具包。它封装了比较底层的繁复操作,提供给开发者一个简洁的界面。同时,DriverWorks和Visual C++有很好的接口:DriverWorks专门为Visual C++提供了一个专门开发WDM驱动程序的WIZARD。在WIZARD指导下,开发者可以很容易地生成一个驱动程序的框架。开发者要作的工作就是根据特定硬件编写相应的代码。

WIZARD生成的框架不仅包括设定设备的资源如中断、IO端口范围、内存范围;还包括自动生成驱动程序与应用程序的接口命令及处理这些命令的处理函数,生成标准驱动程序的标准驱动程序例程;包括DriverEntry驱动程序入口例程、UnLoad卸载例程、Dispach标准派发器例程、StartIO标准I/O传输例程、中断处理例程等。对于WDM驱动程序,包括生成驱动程序接口(WDM专有),设备的符号连接,I/O传输方式以及WDM电源管理和处理PNP请求的IRP处理子程序。WIZARD完成后的代码主要包括两类,一类是驱动对象,它构成了一个驱动程序的框架;另一类是驱动程序的设备对象。驱动程序对象负责创建并管理驱动程序设备对象。设备对象负责管理硬件的各种资源如I/O、IRQ、DMA。它还负责处理各种IRP的例程。设备对象是系统所有I/O操作的基石,一个驱动程序必须有一个或一个以上的设备对象才真正有意义。驱动对象和设备对象的关系很象WINDOWS系统中进程和线程的关系:进程是可执行代码(就是线程)的框架。如果说驱动对象是WINDOWS中的进程,那么设备对象就像进程中的线程,它们完成实际的I/O操作。设备对象使用保护成员变量来管理设备的资源,使用成员函数来处理各种IRP请求,并由这些函数完成实际的I/O操作。

笔者开发的高速旋转机械监控与故障诊断系统采用的是北京大恒公司的具有FIFO(队列形式实现的缓存)的8路数据采集卡AC1810。它的特点是由硬件自动完成采样操作:当FIFO半满的时候,系统产生中断通知用户取走数据;同时,硬件继续自动采样FIFO的另一半。驱动程序的主要工作包括设计I/O读操作和中断服务操作。下面是WIZARD生成的驱动对象和设备对象的定义(只取主要部分):

```
// 驱动对象
class AC_1810 :  public KDriver
{
SAFE_DESTRUCTORS
public:
virtual NTSTATUS
DriverEntry(PUNICODE_STRING
RegistryPath);
// 驱动程序的入口函数
virtual NTSTATUS
AddDevice(PDEVICE_OBJECT Pdo);
// 完成设备对象和驱动对象的连接
……
};
// 设备对象
class AC_1810Device :  public KPnpDevice
{
// Constructors
public:
AC_1810Device(PDEVICE_OBJECT Pdo,
ULONG Unit);  // 构造函数完成资源配置
……
public:
BOOLEAN Isr_Irq(void);    // IRQ中断服务例程
virtual NTSTATUS Create(KIrp I);
// 标准IRP处理函数
```

```
virtual NTSTATUS Close(KIrp I);
// COMMENT_ONLY
virtual NTSTATUS Read(KIrp I);
// COMMENT_ONLY
virtual VOID StartIo(KIrp I);
// 开始I/O传输
VOID CancelQueuedIrp(KIrp I);
// 判断传输IRP是否被取消
VOID Invalidate(void);
// 释放资源例程
virtual NTSTATUS DefaultPnp(KIrp I);
// 处理默认的PNP操作
virtual NTSTATUS DefaultPower(KIrp I);
// 管理电源
void SerialRead(KIrp I);
// 完成实际的I/O操作
NTSTATUS IOCTL_SETUP_Handler(KIrp I);
//采样参数设定
NTSTATUS IOCTL_REW_Handler(KIrp I);
// 开始/停止采样
……
protected:
KIoRange                    m_IoPortRange0;
// 管理I/O资源
KInterrupt                   m_Irq;
// 管理中断资源
……
};
```

在实际操作过程中,采用中断读数的方法。在驱动程序中设置了两个缓冲区,一个前台缓冲区,一个后台缓冲区(用作后备缓冲区),系统总是先使用前台缓冲区。当前台缓冲区已满而仍然未被读出,系统触发中断时,此时使用后台缓冲区。读数的方法很简单,系统先读前台缓冲区的数据,只有当前台缓冲区未满而后台缓冲区满的时候才读后台缓冲区。这样可以保证采样数据序列的时间顺序。

具体实现如下:

```
// ISR例程,完成数据从FIFO中读出
BOOLEAN AC_1810Device：：Isr_Irq(void)
{
// 是否触发中断?未触发则返回,判断两个缓冲区情况,都满则返回
……
// 前台缓冲区未满,使用前台缓冲区
if(m_pBuffer->numused < MAX_READ_BUF )   {
              for( int i = 0; i < BLOCK_SIZE;  i++){
                     m_pBuffer->buff[m_pBuffer->numused+i]
                            =READ_FIFO;
              }
m_pBuffer->numused += BLOCK_SIZE;
}
else
// 完成其它情况的判断
……
return TRUE;  // 中断成功返回
}
```

当应用程序使用标准Win32 API对设备进行读操作的时候,I/O管理器通知驱动程序并触发对Read函数的调用。对于通常的串行设备,在Read函数的最后排队IRP请求,此时系统就可以触发StartIO例程,并且保证这个过程是串行处理的。

```
// 读例程,处理IRP_MJ_READ
NTSTATUS AC_1810Device：：Read(KIrp I)
{
// 检查输入的合法性
       if ( I.ReadSize()< BLOCK_SIZE ) {
              // 不合法返回错误代码
              I.Information() = 0;
              return I.PnpComplete(this,
              STATUS_INVALID_PARAMETER);
}
```

```
// 读0字节,永远成功
if (I.ReadSize() == 0){
            I.Information() = 0;
            return
            I.PnpComplete(this, STATUS_SUCCESS);
      }
// 排队这个IRP触发StartIO,完成数据传输
return QueueIrp(I, LinkTo(CancelQueuedIrp));
}
// StartIo例程,完成I/O操作
VOID AC_1810Device::StartIo(KIrp I)
{
// 检测这个IRP是否被取消
   if (!I.TestAndSetCancelRoutine(LinkTo(CancelQueuedIrp), NULL, CurrentIrp()) )
         {
         return; // 取消则返回
         }
         switch (I.MajorFunction()){
         case IRP_MJ_READ:
         //这个函数按逻辑完成读操作
               SerialRead(I);
               break;
               ……
         // 开始处理下一个IRP
         PnpNextIrp(I);
         break;
}
}
```

ISA卡的WDM驱动程序的主体部分就完成了。它具有PNP功能,可以方便地安装卸载。在中断到来的时候,系统使用两个缓冲区完成数据的传输,可以避免数据丢失。

以上驱动程序是使用Numega公司的DriverStudio2.01版,结合Visual C++ 6.0,Microsoft Windows 2000 sp1 DDK开发调试通过,并且成功地应用到实验室开发的高速旋转机械实时状态监控与故障诊断系统中,该系统界面友好高度可靠。同时由于加入了PNP的支持,使得该系统的硬件安装卸载非常简便。这种技术可取代传统的以DOS为核心的工业用监控与故障诊断系统,具有广阔的应用前景。

参考文献

1 Microsoft. Microsoft Winodows 98 DDK Document,1998

2 Microsoft. Microsoft Winodows 2000(build 1295) sp1 DDK Document, 2001

3 Walter Oney,Forrest Foltz. Programming the Microsoft Windows Driver Model[M] Microsoft Press,1999

4 Art Baker, 科欣翻译组.Windows NT设备驱动程序设计指南[M]. 北京:机械工业出版社,1997

5 Numega. Numega DriverStudio 2.01 DriverWorks Help,2001

地址：北京海淀区清华东路25号电子六所大厦
联系电话：82306084 / 82306085 传真：62311179 京ICP备05053646号
推荐分辨率1024*768 IE6.0版本