文章编号 1004-924X(2001)01-0089-04

PCI总线数据采集卡Windows NT驱动程序的设计

袁晓兵, 张 新, 王 沛, 王国辉
(中国科学院长春光学精密机械与物理研究所, 吉林 长春 130021)

摘要: Windows NT操作系统不允许直接访问硬件, 给图像的实时采集、存储、显示等处理工作带来了很大困难。对在核心态下采用编制虚拟设备驱动程序的方法进行探讨, 重点讨论了如何在Windows NT下实现数据采集卡的中断和DMA过程并给出了相应例程。对于图像处理工作所要求的实时性及同步性在软件方面提出了一种解决方案。

关 键 词: Windows NT; 核心态; 中断; DMA; 设备驱动程序
中图分类号: TP391.72 **文献标识码**: A

1 引 言

在遥感图像处理过程中, 需要支持多任务及快速、可靠的系统来完成繁重的图像采集、存储、显示等功能, 因此, 建立一个快速、稳定、安全的处理平台是很重要的工作, 这一点在航空机载处理系统中尤其突出。在数据采集、存储、显示过程中, 不可避免地涉及到操作系统的底层操作, 不论采用何种操作系统, 都应将图像处理系统建立在操作系统的特权级上, 这样不仅能取得最高的优先级, 避免应用级程序的干扰, 更重要的是, 实现对系统资源的完全访问, 提高整个系统的效率。为了建立这样一个图像处理平台, 首先应对操作系统及相关的设备驱动程序有一定研究。

为了使PCI总线数据采集卡正常工作, 需要在软件中对系统资源(如中断、DMA、物理内存等)进行访问, 但Windows为保证系统的安全性、稳定性、可移植性, 对应用程序访问物理资源加以限制, 只有编制设备驱动程序才能实现对PCI总线设备的完全访问。设备驱动程序不是单独存在的, 是操作系统内核的一部分, 因此, 需要对操作系统有一定了解。基于稳定性和可靠性的原因, 我们采用Windows NT作为操作平台。

2 Windows NT体系结构简介

Windows NT的系统结构决定了Windows NT下访问设备的特殊性。Windows NT是建立在MACH和VAX思想上的一种客户/服务器模型操作系统, 由一个特权执行体以及一系列被称为保护子系统的非特权服务器组成。整个操作系统被分为用户态模式和核心态模式。

所谓核心态(Kernel Mode)概念, 就是指一种核心层的结构, 它管理系统的大部分基本功能, 能完全的、没有保护的对系统进行访问。一个驱动程序或一个线程运行在核心态可以对系统内存、硬件进行存取。它本身是一种最接近于计算机操作系统级的软件处理, 可以直接驱动硬件或接口。

Windows NT系统提供一个真32位抢先式可重入的虚拟内存操作系统, 能够在多种硬件体系结构和平台上运行, 能够在对称多处理系统上运行并具有良好的可伸缩性。

在Windows NT中, 核心态模式又被称为Windows NT执行体(Executive), 包括了系统服务和硬件抽象层(HAL), 它们运行于CPU的特权层Ring0(在驱动程序内部, 在不同的部分还分为不同的权限层)。系统服务包含了一个操作系统所有应有的服务: 文件系统、进程控制、内存管理、设备管理等。Windows NT中执行体包括一些独特的部分, 如对象管理器、I/O管理器、配置管理器、执行体支持、本地进程调用、安全监视器等, 所有这些都建立在HAL之上且完全独立。其中I/O管理器实现对设备的管理, 包含了文件系统、中间介质和设备驱动。通过HAL, Windows NT可以防止内核和NT执行体的其它部分受硬件平台不同的影响。另外Windows NT还将网络管理器加入了核心态模式。

收稿日期: 2000-07-11; **修订日期**: 2000-09-20

3 Windows NT 中的设备驱动程序

设备驱动程序是Windows系统的基本成分，所有的设备驱动程序一起构成了操作系统的基本框架。Windows NT 驱动程序是基于下列目标设计的：可移植性、硬件和软件可配置性、总是抢占和可中断的、在多处理平台上的安全性、基于对象、可重用的 I/O 请求包、支持异步传输等。

由于Windows NT 运行于核心模式，Windows NT 驱动程序只能使用系统提供的 Rt1Xxx 运行库，文件系统的驱动还可以使用 FsRtlXxx 运行库。Windows NT 核心态不支持浮点运算。

至于可配置性，Windows NT 配置管理器提供了一个数据库，叫做注册库，它包含了硬件、外围设备和给定机器的驱动信息。Windows NT 驱动程序可以使用注册库信息获得硬件配置信息，也可以通过 HAL，HAL 响应系统中所有硬件层，包括Windows NT 驱动程序。HAL 隐藏了平台硬件的特殊性，如 Cache、I/O 总线、中断等，Windows NT 设备驱动只要调用 HAL 提供的例程，就可以获得一些硬件信息。

3.1 抢占与中断

中断停止处理器当前任务的执行，并强制它运行某个中断处理代码。处理器对中断划分优先级，使得低优先级中断可以被高优先级中断所中断，这样就保证了重要任务的进行。

Windows NT 的抢先多任务不仅表现在用户级上，在设备驱动级也是抢占和可中断的。在系统中每一个线程都被赋予优先属性，大部分的优先属性都是可变的，除非是实时优先，实时优先只有自己放弃控制。无论优先属性如何，当硬件或某种软件中断产生时，系统中的一些线程将被抢占。

为使一些内核态代码运行在更高的 IRQL 上，内核将硬件和软件中断划分优先次序。这样，就使驱动程序拥有比系统其它所有线程更高的优先权成为可能。在一段内核代码执行时，其特殊的 IRQL 决定它的硬件优先权。而内核态代码经常是可中断的，一个更高 IRQL 值的中断能在任一时刻发生，因此导致被系统分派更高 IRQL 的核心态代码在处理器上立即运行。换句话说，当一段代码在一个给定的 IRQL 值上运行时，内核在微处理器上屏蔽所有低级别或相等中断级的中断。处理器中断级的抽象视图如下表所示，其中最上面的中断优先级最低。

Table 1 Interrupt levels

	IRQL	Description
No Interrupt	PASSIVE-LEVEL	Execute thread
Software Interrupt	APC-LEVEL	Execute special Kernel APC Page fault
	DISPATCH-LEVEL	Dispatch (Execute DPC)
	DPC-LEVEL	Deferred procedure call
Hardware Interrupt	PROFILE-LEVEL	Configuration file timer
	CLOCK2-LEVEL	Interval-timer execution
	SYNCH-LEVEL	Synchronize execution
	IPI-LEVEL	Multiprocessor machine
	POWER-LEVEL	Power failure notification

中断的目的在于使操作系统资源分配的抢占功能能达到总体性能的最大平均值。任一驱动程序的 ISR 能被中断，有利于高 IRQL 例程的运行。也就是说，任一线程能被一个更高优先权的线程抢占。虽然一些线程有实时优先权属性，但这些线程也能被 Still-higher 优先权的线程抢占。因此，Windows NT 结构不提供一个固有的实时系统。

3.2 DMA 驱动程序

DMA 操作是真正高效的数据传输模式，一方面DMA 传输大大提高了数据吞吐率，另一方面使 CPU 在数据传输操作中的参与最小，即减少了 CPU 的占用率。然而，在实际应用中，编写 DMA 设备的驱动程序比编写 I/O 硬件的驱动程序更复杂。

在数据采集的应用中，我们采用的是基于公共缓存区的 DMA 驱动程序，这种类型的 DMA 使采集、存储、显示使用公共缓存区成为可能，同时减少了系统开销。在基于公共缓存区的总线主控设备的 DMA 驱动程序中，设备使用属于该设备一部分的 DMA 控制器与一个连续的非分页池缓存区传输数据。其主要过程如下：

(1) DeviceEntry 例程

DeviceEntry 例程执行下列步骤建立公共缓存区:

· 调用 HalGetAdapter, 找到设备的一个 Adapter Object。

· 调用 HalAllocateCommonBuffer, 得到一块连续的非分页内存, 驱动程序和设备都能访问。

· 在 Device Extension 中存放公共缓存区的虚地址。

· 向设备寄存器中写入公共缓存区的物理地址和大小。

(2) Start I/O 例程

当向设备发送命令时, Start I/O 例程执行以下步骤:

· 使用在 Device Extension 中存放的公共缓存区的虚地址, 在公共缓存区中建立一个命令结构。

· 如果 DeviceEntry 例程对 HalAllocateCommonBuffer 的 CacheEnabled 参数指定为 True, 则 Start I/O 例程需要调用 KeFlushIoBuffers, 强制数据从 CPU 的 Cache 转移到物理内存。

· 最后, Start I/O 例程在设备控制寄存器中置 1, 通知设备有一条命令在等待。设备响应通知位, 并开始处理命令。

(3) 中断服务例程

当设备完成设在公共缓存区的命令或完成一次 DMA 操作时, 产生一个中断, 这时转入中断服务程序。

(4) Unload 例程

公共缓存区总线主控设备 DMA 驱动程序卸载时, 首先需要阻止设备再次使用缓存区。一旦设备停下来, Uload 例程调用 HalFreeCommonBuffer 释放与缓存区关联的内存。

4　编制 PCI 总线数据采集卡的 Windows NT 设备驱动程序

一个典型的驱动程序的组成包括: 初始化、创建和删除设备、处理 Win32 打开和关闭文件的请求、处理 Win32 I/O 请求、串行化对设备的访问、访问硬件、调用其它驱动程序、取消 I/O 请求、超时 I/O 请求、处理热插拔设备加入或删除的情况、处理电源管理请求、处理 WMI(Windows Management Instrumentation) 和 NT 事件等。

驱动程序不必支持以上所有的功能, 严格地说, 只有"初始化"模块是必需的, 其余的模块都是可选的, 但大部分驱动程序都有例程来处理 I/O 请求。

PCI 总线数据采集卡的 Windows NT 设备驱动程序编制主要过程如下:

(1) DriverEntry 部分　当启动相应服务时, 系统创建 Driver Object, 调用 DriverEntry。在 DriverEntry 中进行的工作有:

· 调用 HalGetBusData 遍历系统所有总线和槽位, 寻找实际设备, 若找不到则返回设备不存在信息, 否则保存总线、槽位和 PCI 配置数据在 PCI_COMMON_CONFIG 结构中。

· 调用 HalAssignSlotResources 通告在指定总线和槽位上的资源, 所有资源信息保存在 CM_RESOURCE_LIST 结构中。

· 若正确则调用 HalGetInterruptVector 映射中断向量。

· 在 IRP 中定义调用例程位置。

· 调用 IoCreateDevice 创建设备对象, 清理 Device Object 的扩展结构, 调用 IoCreateSymbolicLink 使 Device Object Win32 可见, 要先调用 RtlInitUnicodeString 将字符 Unicode 化。

· 将设备资源保存到设备对象扩展结构中。从 CM_RESOURCE_LIST 结构获得 I/O、内存以及中断信息。并将其保存到设备对象扩展结构中。

· 调用 IoInitializeDpcRequest 注册 DPC 例程。

· 调用 IoConnectInterrupt 将中断向量和设备例程挂接。

· 调用 IoCreateSynchronizationEvent 创建同步信息。要先调用 RtlInitUnicodeString 将字符 Unicode 化。

· 使用 pDeviceObject->Flags |= DO_DIRECT_IO 语句, 将设备设置为直接 I/O。

· 调用 HalTranslateBusAddress 映射设备 I/O 和内存。I/O 类型为 BIO。内存使用直接 I/O, 调用 MmMapIoSpace 将物理地址映射到非分页系统空间。

· 分配核心态非分页内存, 作为核心态内存链。

(2) DPC 部分　调用 KeSetEvent 设置内存链空事件, 由于事件是传递给用户态的, 按照规则要求 KeSetEvent 必须工作在小于等于 DISPATCH_LEVEL 层上, 中断工作在 DIRQL, 线程调度器和延迟过程调用执行在 DISPATCH_LEVEL, 异步过程调用执行在 APC_LEVEL, 一般的线程工作在 PASSIVE_LEVEL。

(3) 线程调度器 Dispatch　将 IRP 要求的服务例程指向各个函数, 其中 IRP_MJ_DEVICE_CONTROL 包含用户自定义的 I/O 操作。

(4) DriverObject->DriverStartIo 例程　Start I/O 是在当调用 IoStartPacket 时启动的, 在这里是在发 IRP_MJ_WRITE 指令时调用

IoStartPacket 启动 Start I/O 的。基于安全的考虑,在 IRP-MJ-WRITE 中首先将 IRP 挂起,然后启动设备 IO。在 Start I/O 中,由于是直接 IO,首先调用 MmGetSystemAddressForMdl,将用户态传递的数据地址空间(IRP 中的 MDL)转换成系统空间,然后从 IRP 的本地堆栈中取出用户态传递的数据长度。

由于使用中断的原因,必须保证同步向核心态送数据和 ISR。在这里调用 KeSynchronizeExecution 完成此项工作。KeSynchronizeExecution 指向同步函数。最后继续下一个 IRP 的传递(IoStartNextPacket),并且告诉 IO 管理器完成此 IRP 操作(IoCompleteRequest)。

(5) 同步函数　将用户层的数据转移到核心态数据链储存起来,并且检查此数据链是否有空的部分,若有就向用户态发送送数据事件。

(6) ISR　当中断到来时,首先检查中断,然后将核心态数据发送到数据总线,即物理设备地址空间,并通知 DPC 向用户态发送送数据事件,最后清理中断源。

(7) 卸出例程　处理清扫工作,关闭事件句柄,解除中断联系,释放核心态内存,释放资源,端口和内存映射,删除 Win32 符号联接,删除设备等。

在具体编制过程中加入要实现的功能,如在自定义的初始化设备例程中设置采集卡的中断信息和一些初始化工作、设置寄存器、清内存链,清事件等。如果使用 DMA 传输,在 DeviceEntry 例程、Start I/O 例程和 Unload 例程中加入相应代码,同时设置采集卡的寄存器,并在 ISR 中或 ISR 后加入图像数据存储的代码,并将缓存区地址及消息传递给显示程序,在最后完成清中断、清缓存等工作。

5 结　论

本文主要论述了 Windows NT 的体系结构及如何在 Windows NT 下编制设备驱动程序。当然,在实际编制过程中要考虑的情况复杂得多,不仅要解决设备驱动程序与用户态应用程序的通信、共享问题,还要考虑操作过程中的容错事件。

限于篇幅,本文并没有更多地讨论 DMA 传输中的代码构成。在编制过程中我们注意到,Windows NT 并不允许执行设备到设备的直接 DMA 操作,然而这正是 PCI 总线设备的一个重要功能,要实现设备到设备的直接 DMA 传输,只能等待 64 位的 Windows 版本了。

参考文献:

[1] Microsoft Corp. Windows 2000 DDk documentation[CP]. Microsoft Corp., 1999, 7.
[2] Baker, A. The Windows NT Device Driver Book: A Guide for Programmers[CP]. Prentice Inc, 1997.
[3] Matt Pietrek. Windows 95 system programming secrets[CP]. IDG Books Worldwide, 1995.
[4] 刘立峰, 等. Windows 95 环境下的混合编程技术[J]. 光学 精密工程, 2000, 8(4): 402-405.

Device driver programming in Windows NT for PCI data acquisition card

YUAN Xiao-bing, ZHANG Xin, WANG Pei, WANG Guo-hui

(*Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Changchun* 130021, *China*)

Abstract: The Microsoft Windows operating system insulates the application programmer from dealing with hardware directly. So it is difficult for image processing on real time, such as acquisition, storage and display. This paper studied Kernel mode of Windows NT, and discussed the routine of interruption and DMA, then a device driver program was designed to solve this problem. The method of VDD has a wide application prospect and practical value in real time image processing.

Key words: Windows NT; Kernel mode; interruption; DMA; virtual device driver

作者简介: 袁晓兵(1969-),男,吉林省长春市人,1991 年 7 月毕业于浙江大学信息与电子工程学系,1997 年 3 月于中国科学院长春光机所获得硕士学位,目前在长春光机所攻读博士学位。所从事的工作和研究方向为计算机图像处理。