带缓存磁盘技术PCI卡在Windows NT下的应用

姜戡 谢长生 (华中理工大学计算机学院外存储系统国家实验室 430074)

摘要：本文研究了带缓存磁盘技术的PCI卡在Windows NT环境下驱动程序的结构和原理，并详细阐述了PCI空间访问的机制和方法，尤其是在核心设备驱动程序层对PCI设备操作的关键技术进行了深入的研究。

关键词：计算机存储系统 缓存磁盘 PCI配置空间 PCI设备驱动程序

1 缓存磁盘技术的结构和原理

1.1 缓存磁盘技术的原理

缓存磁盘技术(Disk Caching Disk,DCD)是美国Rhode Island大学电子计算机工程系高性能计算实验室杨庆教授等提出的一种用于优化小写请求I/O性能的磁盘存储体系结构[1]。它的基本思想是使用一个cache-disk(高速缓存盘)作为一个RAM cache的扩展来记录文件的改变，当系统空闲的时候再把cache-disk上的数据迁移到真正的数据盘上去。小写和随机写最初是缓存在一个小的RAM buffer中，当cache-disk空闲时，RAM buffer中的所有数据都通过一次数据传输写入到处于RAM buffer和data-disk之间的cache-disk中，从而使得RAM buffer很快清空以接收另外的系统I/O请求，这样两级的cache对主机来说是相当于一个大的RAM，当data-disk空闲时，就会进行cache-disk到data-disk的数据的迁移过程[1]。我们与美国Rhode Island大学合作，将上述缓存磁盘技术的思想在Windows NT环境下实现。已经完成的系统表明，根据负载的情况不同，磁盘缓存技术可将小写的请求的响应时间缩短3-80倍。

1.2 磁盘缓存技术的总体设计

在具体的实现过程中，我们采用了一种DCD加速卡方案，其原理的硬件部分是一个特殊的有NVRAM的PCI卡，而软件部分是利用NT下的驱动程序模块来实现的，也就是说我们把DCD算法中的数据结构放在PCI加速卡上，卡上有一块NVRAM，能做到掉电后数据不丢失。而整个DCD的机制用驱动程序来完成。

如图1所示，在处理器总线和PCI总线之间就是DCD的工作区，在这个层次我们对磁盘的请求进行截获，截获的基础上，通过在驱动程序中加入DCD算法达到实现缓存磁盘技术的最终目的。

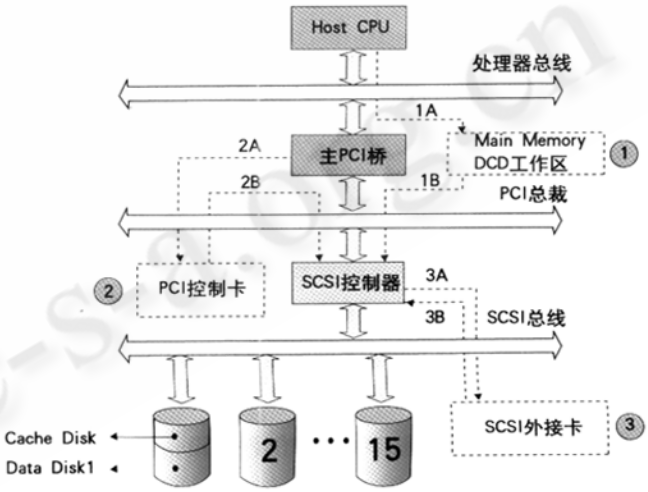

图 1 带缓存磁盘技术的PCI加速卡实现总体硬件方案

2 NT环境下的驱动程序的结构及实现

2.1 NT的系统结构

从组成结构上说，Windows NT主要由硬件抽象层、内核层、执行层三个主要的模块组成[2]。

硬件抽象层(HAL)是一薄层软件，用CPU之外的硬件抽象模型代表系统的其余部分；内核层提供中断和异常处理、线程调度和同步、多处理机同步、定时控制等功能；

Executive 层的工作是实现与操作系统相关联的多种基本功能。

2.2 NT 驱动程序的层次结构分析

在 Windows NT 下，驱动程序有三个层次：设备驱动程序、文件驱动程序和中间层驱动程序。

在本文中我们主要设计的是设备驱动程序，设备驱动程序是管理实际数据传输和控制特定类型物理设备操作的驱动程序。

2.3 NT 驱动程序的构成

驱动程序实际上也是由一些功能例程构成，通过在这些例程中加入缓存磁盘技术的算法，我们可以达到实现缓存磁盘技术的目的。I/O 系统负责驱动设备驱动程序的执行。设备驱动程序包括一组用以处理不同阶段的I/O请求的例程，如图 2 所示：

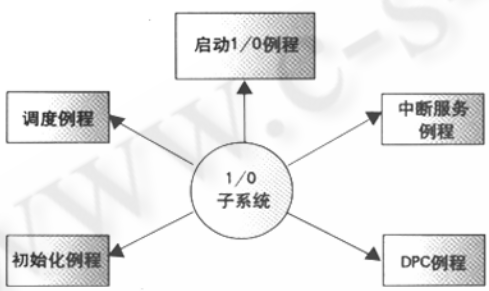

图 2　主要的设备驱动程序例程

(1) 初始化例程：当 I/O 管理器把驱动程序加载到操作系统中时，它执行驱动程序的初始化例程。这个例程将创建系统对象，I/O管理器利用这些系统对象去识别和访问程序。

(2) Dispatch例程：Dispatch例程是设备驱动程序提供的主要函数。主要有打开、关闭、读取、写入以及设备、文件系统或网络支持的任何其他功能。

(3) 启动I/O例程：驱动程序可以使用启动I/O例程来初始化去设备之间的数据传输。

(4) 系统关闭通知例程：这个例程允许驱动程序在系统关闭时做清理工作。

2.4 过滤器驱动程序的概念

过滤器驱动程序是我们实现过程的重要的环节，它是一种特殊类型的中间驱动程序，它们位于某个其他驱动程序之上，截获从上层发给低层驱动程序的 Device 对象的请求 [2]。

过滤器驱动程序与其他分层驱动程序的主要差别在于它们创建的 Device 对象没有系统命名。过滤器驱动程序通过把这些没有命名的 Device 对象挂接到低层驱动程序创建的 Device 对象来工作。

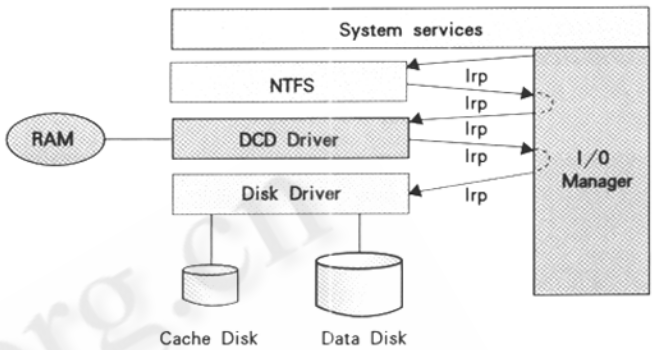

图 3　缓存磁盘技术过滤器驱动层次结构

如图3所示，在我们的带缓存磁盘技术的PCI卡的软件实现中，整个软件模块实际上就是一个过滤器驱动程序模块，这个模块位于文件系统驱动层与磁盘设备驱动层之间。图中看出，缓存磁盘驱动程序截获来自文件系统的IRP，对其进行算法处理，最终数据还会传送到原始的数据磁盘上。

3　NT 环境下的 PCI 设备驱动概念及应用

3.1 PCI 配置空间

(1) PCI总线配置空间组成。PCI规范指出总线上的每个功能部件必须有它自己的256字节的存储区存放配置数据，这个区域称为 PCI 功能部件的配置空间 [3]。

这个空间被分成一个预定义区域和一个依赖设备的区域，不同的设备只要把每一个区域中的相关寄存器进行设置即可。软件必须使用 I/O，通过进行内存的存取去访问设备配置空间。软件要处理以位进行编码的域，在读的时候，使用合适的屏蔽去抽取所定义的位，在写的时候，则要保证被保留位上的值不被改变。

(2) 预定义区域的结构。预定义区域有 64 个字节，前 16 个字节对任意类型的设备都相同，剩下的 48 个字节则根据设备所提供的基本功能的不同而具有不同的设置 [4]。其结构如图 4 所示。

基于PCI总线的设备，都必须在此预定义区域提供制造商 ID，设备 ID，命令和状态。其他寄存器的设置则可以根据设备的功能进行选择(如作为保留寄存器)。

31			0	
设备ID		制造商ID		00h
状 态		命 令		04h
类 码			版本ID	08h
BIST	头部类型	等待计时器	Catch线	0ch
基地址寄存器				10h
保 留				28h
扩展ROM基地址寄存器				30h
保 留				34h
Mar lat	Min Gnt	中断Pin	中断Line	3ch

图 4 配置空间预定义区域结构

3.2 NT下对缓存磁盘加速卡配置空间的访问

3.2.1 对PCI设备的资源访问的方法概述

对于PCI设备的访问是调用函数HalGetBusData，从HalGetBusData的返回值来看，如果其返回值是2，说明在指定的SlotNumber没有相应的Device；如果返回值是0，则表示PCI总线指定的BusNumber并不存在。一旦HalGetBusData找到了相应的Device，通过BusNumber和SlotNumber，设备的硬件信息就被确定了，设备的硬件资源并还需要函数HalAssignSloNumber获得。

在一个端口或一个内存地址被驱动程序访问时，它将被转换成逻辑地址空间，这种转换是通过函数HalTranslateBusAddress来完成的。

然后我们必须把这些地址映射成核心态的虚拟地址空间，这可以用系统函数MmMapIoSpace来完成[2]。

3.2.2 缓存磁盘加速卡中PCI配置空间访问的具体实现

(1)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。否则保存总线，槽位和PCI配置数据在PCI_COMMON_CONFIG结构中。

从0到255，对每一个BusNumber遍历系统所有总线和槽位。

```
for (BusNumber=0;BusNumber<=255;BusNumber++)
    {
    for (i=0;i<256;i++)
     { SlotNum=i;
    Result=HalGetBusData(
        PCIConfiguration,
        BusNumber,// 总线号
        SlotNum,
        &PciCfgBuffer,// 保存 pci 配置数据的结构体
        Length
        );
         if(PciCfgBuffer.DeviceID==5922 &&
PciCfgBuffer.VendorID==0xle08)
    KdPrint(("Find the PCI device! /n"));// 找到 pci 设备
    }
    }
```

(2) 调用HalAssignSlotResources宣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。

```
    NTStatus=HalAssignSlotResources(
        RegistryPath,
        NULL,
        DriverOBject,
        NULL, // 设备对象
        PCIBus, // 总线类型
            BusNumber,// 从 HalGetBusData 获得的
BusNumber
        SlotNum,// 从 HalGetBusData 获得的 SlotNum
&pResourceList // 资源信息结构
        );
```

(3) 调用HalTranslateBusAddress映射设备I/O和内存。I/O使用Buffer I/O。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。

首先取出在pDevEx中的由HalGetBusData函数得到的PCI_COMMON_CONFIG结构中的PCIINFO.u.type0.BaseAddresses即RAM的地址空间。

```
        MemoryAddress.LowPart=pDevEx-
>PCIINFO.u.type0.BaseAddresses
[FRONTEND_MEMORY_INDEX] & 0xFFFFFFF0;
        // 然后调用 HalTranslateBusAddress 转换
MemoryAddress
    ntStatus=HalTranslateBusAddress(
            PCIBus,// 指名为 PCI 总线
            pDevEx->BusNumber,//PCI 总线号
            MemoryAddress,// 要转换的 PCI 总线地址
            &inIoSpace,
            &physicalAddressBase); // 转换后的地址
    pDevEx->MemoryBase// 将物理地址映射到非分页系
统空间
```

=MmMapIoSpace(physicalAddressBase,// 要映射的物理地址空间

MemoryLength,// 要映射的长度

FALSE)

pDevEx->MemoryWasMapped=TRUE;// 此参数表明地址已经映射

}

4 缓存磁盘技术的性能测试与分析

对于NT环境下缓存磁盘技术的测试思想就是对加载了缓存磁盘和未加载缓存磁盘程序的硬盘进行频繁的事务性处理，从而得到该硬盘的响应时间，并计算相应的数据传输率等各方面参数，得出缓存磁盘技术算法在各种情况下的性能情况。

我们主要用了Iometer这种测试工具，它是INTEL公司专门开发的用于测试I/O性能的测试程序。它的测试参数比较全面，能非常全面的反映服务器的I/O性能。

测试的参数如下所示：

(1) 测试硬件平台：

CPU:Cerleron366,内存：64MB,硬盘型号：MAXTOR 6.4G

(2) 测试程序：Iometer(Intel co.,1td)

4.3 测试设置

并发进程数据：4个，访问模式：异步，写：100%，随机：100%，传输延迟：1000毫秒，突发长度：500个

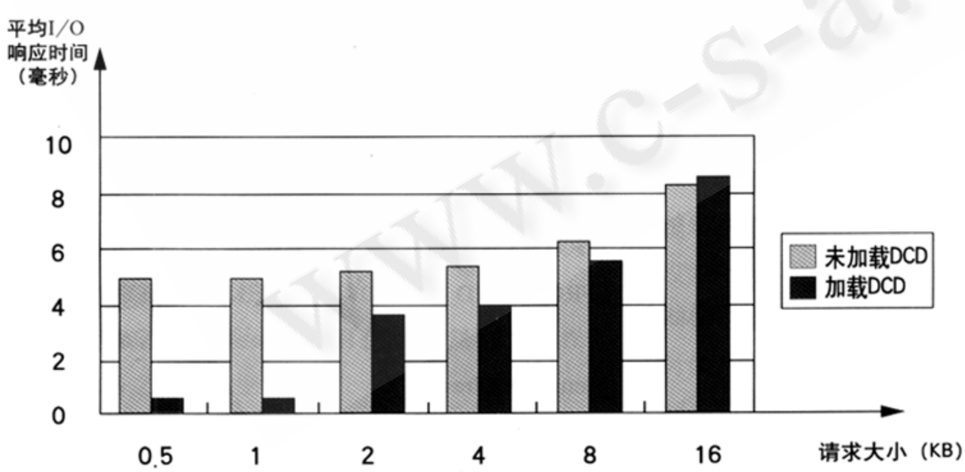

图 5 10METER 测试的平均 I/O 响应时间示意图

从图5测试的结果可以知道，采用的DCD技术后，小写请求的平均响应时间有很大的提高，对于0.5K的小写请求，有十几倍的提高，是因为小写的请求被Cache盘吸纳，而随着请求的增大，响应时间逐渐变长，例如16K的请求的提高的倍数就明显不大了，这是因为要Cache盘吸纳不了，就开始倒盘的缘故。

5 结论

目前计算机存储系统对于小写I/O请求的效率亟待提高。特别是在工程/办公环境中，小写I/O请求是一种突发式的请求模式，而常规磁盘存储方式又由于磁盘的机械特性以及一般文件系统的管理方式等原因，导致小写I/O性能的效率很低。

为了解决这类问题，美国Rhode Island大学的杨庆教授提出了一种改善突发小写请求环境下I/O性能的技术——缓存磁盘技术(Disk Caching Disk)与杨庆教授合作，我们主要对NT环境下的驱动程序进行了研究，把缓存磁盘的实现过程加入到驱动程序中。最后的测试结果证明，通过对I/O请求的截获和加入算法，对I/O请求的平均相应时间有了较大程度的改善，在峰值情况下有数量级的提高。■

参考文献

1 Tycho Nightingale, Y. Hu and Q Yang, The Design ana Implementation of a DCD Device Driver for Unix, 1999 USENIX Technical Conference, June 6—11, 1999, Monterey, CA, June, 1999.

2 Windows NT 4.0 DDK Document, Microsoft 1997

3 李贵山，戚得虎，PCI局部总线开发者指南，西安电子科技大学出版社，1997.

4 曾繁泰，冯保初，PCI总线与多媒体计算机，电子工业出版社，1998.