Windows NT 环境下网卡驱动程序的编制

朱志良　陈栩　高晓兴　马沂夫　（东北大学计算中心　110006）

摘要：本文介绍了目前网卡的组成结构、基本工作原理，及其在 WINDOWS NT 环境下驱动程序的设计与实现。

关键词：IRP　NDIS　分页内存　非分页内存　DMA 通道　缓冲环

网卡作为微机的一种外部设备，在网络数据的传输中起着十分关键的作用，而基于网卡的编程在网络记费、安全性过滤、网络监控等系统的设计与实现方面有着十分重要的应用，因此近年来对网卡编程操作的需求明显增加。但是，WINDOWS NT 对底层硬件的调度和管理方式与其他操作系统明显不同，目前对 NT 环境下的网卡编程设计比较少，也限制了 NT 环境下网络系统的开发。本文在介绍一种网卡的基本工作原理的基础上重点介绍了如何在 WINDOWS NT 环境下实现对网卡的编程。

一、网卡基本工作原理

目前网卡种类呈现多样化，但市场上的多数产品符合 NE2000 兼容网卡标准。NE2000 网卡的基本结构体现了 OSI 物理层的全部内容及数据链路层的一部分内容，实现了体系结构中的物理层和介质访问控制（MAC）子层的功能（IEEE802.3 标准）。微机对网卡的控制是通过 I/O 端口进行的，与网卡的数据交换为程控方式。数据缓冲区为 64KB，发送/接收缓冲区都由程序定制在该区中，且可编程设定其大小与首末地址。物理地址在网络上是唯一的，它固化在 EPROM 中。

NE2000 兼容网卡主要由三个模块组成－收发器、编码/解码器、控制器。收发器模块用作传输介质（如同轴电缆及双绞线等）的驱动及收发。编码/解码器模块提供网卡传输所需的曼彻斯特编码/解码功能，同时也是收发器与控制器之间的接口。控制器模块是整个网卡的核心，它控制着网卡的大部分操作。接收期间，串行数据转换为字节数据写入 FIFO，接着由本地 DMA 操作从 FIFO 读取数据写入网卡上的缓存中，然后系统再调用远地 DMA 操作将网卡缓存的数据调入主存，提供给应用程序。与接收相反，发送期间由系统调用远地 DMA 操作将主存的数据调入网卡的缓存中，接着由本地 DMA 操作从网卡上的缓存中读取数据写入 FIFO 队列，交由串行化电路发送到传输介质上。在主存和传输介质之间的数据传输过程中，网卡上的缓冲存储器在系统（主存）和外设（网卡）之间的数据通信中起桥梁的作用，而 FIFO 队列则联系了串行数据和字节数据，两者的作用相似，但作用的级别不同，网卡上的缓冲存储器联系的是设备级，FIFO 队列联系的是芯片级（如图 1）。

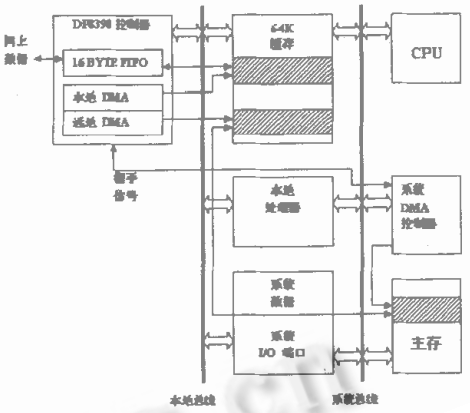

图 1　数据传输原理

对 NE2000 兼容网卡的操作是通过改变寄存器组的值来实现的。寄存器组实际上是指控制器内的寄存器组，共有 16 个寄存器，所有的寄存器都是 8 位的，所以 NE2000 兼容网卡能够正常地运行于 8 位、16 位、32 位处理器系统。寄存器的管理采用了复用技术，首先，同一寄存器在收、发时功能不同；其次，通过命令寄存器，把这 16 个寄存器映射为 4 页，划分了常用操作和不常用的操作，避免为了访问常用的寄存器而不得不执行两个时钟周期。第零页为 DP8390 操作过程中常用的寄存器，第一页中的寄存器用于存储物理地址以及多目地址，第二页中的寄存器用于网卡上缓冲存储器的存取控制，第三页用于测试。

二、网卡驱动程序的设计与实现

在 WINDOWS NT 环境中，程序运行在两种模式下——用户模式、核心模式。用户模式(即 NT 执行体)以 win32API 的接口调用形式与用户进行交互，如果涉及到底层操作 win32API，则将命令传向 NT 内核，在核心模式下执行。核心模式下运行的都是内核代码或硬件设备操控代码，用户的应用程序无法直接运行在核心模式下，网卡设备驱动程序通过微软的 DDK 开发包调试、运行在核心模式下。WINDOWS NT 网络功能是以网络组件和分层的驱动程序实现的，这样可以分离网络协议和数据，以支持广泛的网络协议和硬件。

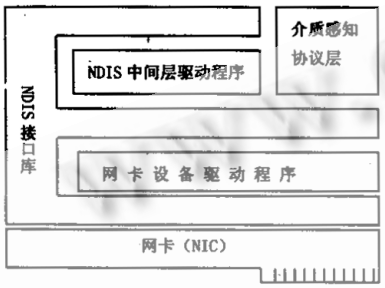

图 2　NDIS 接口库

网卡设备驱动程序位于分层驱动程序的最底层，直接管理网卡硬件。网卡设备驱动程序向上层驱动程序(中间层驱动程序)提供一个统一的接口，允许上层驱动程序接受和发送数据包，启动和停止网卡工作，查询和设置网卡的中断状态、接收状态等信息；网卡设备驱动程序对上层、对下层的操作都是通过 NDIS 接口库实现的，事实上，网卡设备驱动程序是嵌到 NDIS 中的(如图 2)。NDIS 接口库提供一套完整的函数库，所以编写的驱动程序只需同 NDIS 库打交道，而不涉及操作系统。网卡设备驱动程序分为两种：小端口(miniports)及全驱动程序(full NIC drivers)。其中小端口只处理硬件的特定操作，编写的代码少，调试容易，执行效率高，更易为人采用，是今后发展的方向(微软从 NT4.0 以后开始支持小端口)。全驱动程序要考虑一个硬件设备的全部事情，代码庞大，难于调试和控制掌握，效率也较小端口低，但全驱动程序通用性强。

在设计初始，一定要考虑网卡都有哪些操作，即使编写一个小端口驱动程序也要有以下派发例程——初始化、重置、中断处理、接收、传输、停止等。由于篇幅所限，这里只介绍典型三部分——入口、初始化、接收。接收、传输、远地传送操作十分类似，只是对寄存器操作不同，所以传输、远地传送操作只介绍原理。

1. 入口

整个驱动程序的入口点是 DriverEntry()，它建立 Driver 对象，联接派发例程和 Driver 对象。

在核心模式下，对底层驱动程序的 I/O 调用采用包的形式，也就是 IRP(I/O 请求包)。当一个 I/O 请求产生时，I/O 管理器从非分页内存中分配一个 IRP，将其传递给底层驱动程序的派发例程(Dispatch)，派发例程取出 IRP 中的参数，设置设备操作。操作完成后，要根据成功与否设置 IRP 中的完成状态，提交给上层驱动程序。在进行 IRP 操作时，要注意中断请求级别(IRPL)，一般运行在 DISPATCH-LEVEL 及 DIRQLs 级；否则，可能出现 xxxx 地址不能读(写)的错误，而导致应用程序崩溃。

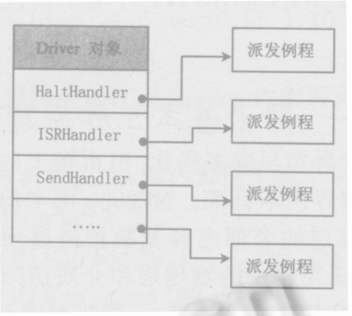

图 3　Driver 对象

WINDOWS NT 中把设备和驱动程序都视为对象。I/O 管理器在加载一个驱动程序时，建立一个 Driver 对象，当有 I/O 请求特定设备时，I/O 管理器将 IRP 传到关联的 Driver 对象，再由 Driver 对象控制设备对象完成操作(如图 3)。设备对象保存设备的状态和执行参数以及 IRP 的实际执行，I/O 管理器通过设备对象的一个参数指针找到相应的 Driver 对象中的操作例程，为了解决同时多个 IRP 请求，它还维护一个 IRP 队列。与此同时，驱动程序也需要分配存储空间，有两种空间可供分配——分页内存及非分页内存。分页内存位于虚存中，它的分配只能以页为单位，只能供 DISPATCH-LEVEL-IRQL 以下级别运行的驱动程序使用；非分页内存位于物理内存中，核心堆也属于非分页内存。所以，非分页内存要比分页内存宝贵得多。

入口操作主要分为三部分：

(1) 因为网卡驱动程序是包在 NDIS 中，所以要调用 NdisMInitializeWrapper()申请 Wrapper 句柄。

(2)建立派发例程和 Driver 对象的联系。

(3) 调用 NdisMRegisterMiniport()注册 Miniport 对象，使之生效。

2.初始化

由于网卡需要同主存交换数据，所以应先介绍一下网卡同主存数据的交换方式。NT 是一个 32 位系统，每个进程有 4G 虚拟地址空间，用户缓冲区位于虚拟地址空间的低端 2G，而驱动程序属于系统，位于虚拟地址空间的顶端 2G，这就引起一个数据通信的问题。NT 的 I/O 管理器提供两种方法解决，第一种方法是在一个 I/O 操作开始时，从非分页内存中分配一个缓冲区，利用这个缓冲区连接用户缓冲区与驱动程序，可以看出这种方法效率低，而且当进行大批量数据传输时需要较多的非分页内存，所以不适合用于大批量频繁数据交换的设备。另一种方法是把用户缓冲区的内存页直接映射到驱动程序中，这样可直接操作，但要注意应对使用的内存进行锁定。由此可见，第二种方法比较适合网卡。

网卡初始化时首先调用 NdisAllocateMemory()从非分页内存中分配并初始化一个 Adapter 对象，接着用 NdisReadConfiguration()查询卡类型、总线号、总线类型、IO 基地址和中断号，并把查询结果和可直接访问的用户内存映射地址填入 Adapter 对象。调用 NdisMSetAttributes()、NdisMRegisterIoPortRange()和 NdisMRegisterInterrupt()设置 Adapter 对象，通过 NdisRawWritePortUchar(地址，数值)来对寄存器操作，并进行如下初始化：

(1) 给命令寄存器赋初值(21H)，使用寄存器页的第 0 页(CR)；

(2) 初始化数据配置寄存器(DCR)和接收配置寄存器(RCR)；

(3) 对远地字节计数寄存器清空(RBCR0，RBCR1)；

(4) 设置网卡的环回模式为 1 或 2(对应的传输配置寄存器等于 02H 或 04H)；

(5) 初始化接收缓冲环：初始化边界指针寄存器(BNDRY)、页面起始寄存器(PSTART)、页面终止寄存器(PSTOP)；

(6) 设置中断状态寄存器为 0FFH(ISR)；

(7) 初始化中断掩码寄存器(IMR)；

(8) 给命令寄存器赋值(61H)，使用寄存器页的第 1 页(CR)；

(9) 初始化物理地址寄存器组(PAR0 - PAR5)、多目地址寄存器组(MAR0 - MAR5)和当前页寄存器；

(10)给命令寄存器赋值(22H)，启动 START 模式；

(11)初始化传输配置寄存器。

现在，网卡就可以接收和发送了。

3.接收操作

这里所说的接收操作是指通过本地 DMA 接收通道把数据由传输介质读入卡上的缓冲存储器。本地 DMA 接收通道对数据的存放使用缓冲环结构，缓冲环由一系列连续的长度固定为 256 字节的缓冲页构成，用来存放接收到的数据包，一旦 DMA 地址达到页面终止地址就被复位成页面起始地址。接收缓冲环在缓冲存储器中的位置是通过对页面起始地址寄存器和页面终止地址寄存器进行编程来设置的。两个静态寄存器和两个工作寄存器控制着缓冲环的操作，这些寄存器是页面起始寄存器、页面终止寄存器以及当前页面寄存器和边界指针寄存器。当前页面寄存器指向缓冲环中可存放以太包的第一个缓冲页。边界指针寄存器指向环中尚未被主机读取的第一个包(如图 4)。

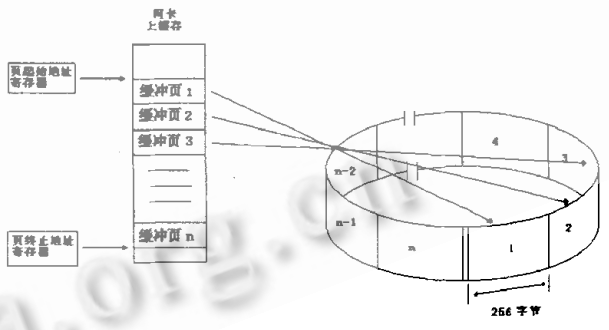

图 4　接收缓冲环

接收过程中，若本地 DMA 地址达到这个边界，该次接收将被中止。数据包被主机读取后，边界指针寄存器中的指针向前移动，这等于释放了被该数据包占有的缓冲页。形象地讲，当前页面寄存器的值相当于写指针，边界指针寄存器中的值相当于读指针。初始化时，页面起始寄存器的内容应装入到当前页面寄存器和边界指针寄存器中。当一个包到来时，开始从当前页面寄存器所指向的位置存放该包。在存放该包的第一个缓冲页中留有 4 个字节的空间以便存放该包的接收状态信息。若包的长度用完了第一个缓冲器页，DMA 执行向前链接，用下一个缓冲页来存放该包的余下部分。链接时不允许跳跃

缓冲器，包总是存放在连续的缓冲页中。在下一个缓冲页可以被链接之前，应该先进行两个比较，首先把下一个缓冲页的地址与页面终止寄存器比较，若相等，则把 DMA 地址设成缓冲环中第一个缓冲页的地址，该地址编程在页面起始寄存器中。第二个比较是测试下一个缓冲页的地址是否与边界指针寄存器的内容相等，若相等说明缓冲环已满则中止接收。边界指针寄存器可以用来保护接收缓冲环中尚未被读取的区域不被覆盖。若缓冲器地址与边界指针和页面终止地址都不相等，则进行与下一个缓冲页的链接。

包接收操作结束时要决定是接受还是拒绝该包，它或转去执行存放缓冲页头的操作，或去进行恢复被该包占用的缓冲页。若接受该包，则把 DMA 地址设为存放该包的第一个缓冲页的地址(当前页面寄存器指向的地址)，然后存放接收状态、指向下一个包存放地址的指针以及接收到的字节数。最后一个缓冲页中未用完的字节是不能被利用去分配给下一个数据包，下一个包从下一个空的缓冲页开始存放，接着当前页面寄存器的内容被置成缓冲环中下一个可用的地址。若被拒绝，则把 DMA 地址恢复为存放该包的第一个缓冲页的地址，以释放由该包占有的所有缓冲页。

因为网卡读到数据包时是以中断的方式通知系统，所以接收操作是在 ISR(中断服务例程)中实现的，设置相应寄存器的值，再调用 DPC(推迟过程调用)函数具体实现硬件设备的操作。在这个过程中首先要判断中断类型，如果是接收中断的话，将 FIFO 数据传入缓冲区，同时修改页指针及接收状态。具体实现是通过 NdisQueryPacket 查询接收包的信息，调用 NdisRawWritePortUchar 和 NdisRawReadPortUchar 函数设置、读取寄存器的值来控制网卡的接收。

4.传输操作

这里所说的传输操作是指通过本地 DMA 通道把数据由卡上的缓冲存储器发送到传输介质上。发送操作由三个寄存器控制——传输页面起始寄存器、传输字节计数寄存器 0 和传输字节计数寄存器 1。传输操作与接收操作在缓冲区的组织上有所不同。接收操作的缓冲区为逻辑环，能接收一定数量的包，而且为了操纵逻辑环，用了许多寄存器。传输操作的缓冲区只能容纳一个要传输的数据包，所以操作简单，也只用到三个寄存器。

传输操作初始化：给传输页面起始寄存器、传输字节计数寄存器 0 和传输字节计数寄存器 1 赋值，使传输状态寄存器复位，设置命令寄存器的传输位为'1'来启动传输。

启动传输操作后，把三个寄存器指向的数据包送入 FIFO，前导同步码以及 CRC 校验码由网卡的电路自动计算添入数据包。数据包在缓冲区中的格式应如图 5 所示。由 IEEE802.3 规程，对包的长度有要求，发送数据不应少于 46 字节，否则应加入填充字节，加入和删除填充字节由程序员负责。

图 5 数据格式

5.远地 DMA 操作

远地 DMA 通道用来为主存和网卡上的缓冲区进行数据传输。远程写用来把数据从主存传送到缓冲存储器，远程写 DMA 操作从 I/O 端口读取数据并从远程起始地址开始顺序地把数据写入网卡上的缓冲存储器。远程读用来把数据从网卡上的缓冲存储器传送到主存，远程读 DMA 操作从远程起始地址开始顺序地从网卡上的缓冲存储器中读取数据送入 I/O 端口。远程 DMA 操作每传送一个字节，DMA 地址加 1，远程字节计数器减 1。当远程字节计数器减至 0 时，DMA 操作结束(参见图 3)。当执行远程读成功后，数据包已从网卡上的缓冲存储器读入主存，这时，通过修改边界指针来把数据包所占用的缓冲页释放以供接收使用。

通过以上设计，作者实现了在 WINDOWS NT 环境下对网卡的编程，该程序已应用在东北大学计算中心的网络管理系统中。

注：以上所使用的函数和结构都不是 WIN32API，而是微软 DDK 软件包的函数和结构。

参考文献

[1] "DP83901A SNIC Serial Network Interface Controller", National Semiconductor, Novermber 1995

[2] Microsoft DDK Documentations for NT4.0 1997

[3] "The Little Device Driver Writer", Microsoft Development Network, Technology Group, Ruediger R. Asche

(来稿时间：1999 年 3 月)