NDIS 协议驱动开发

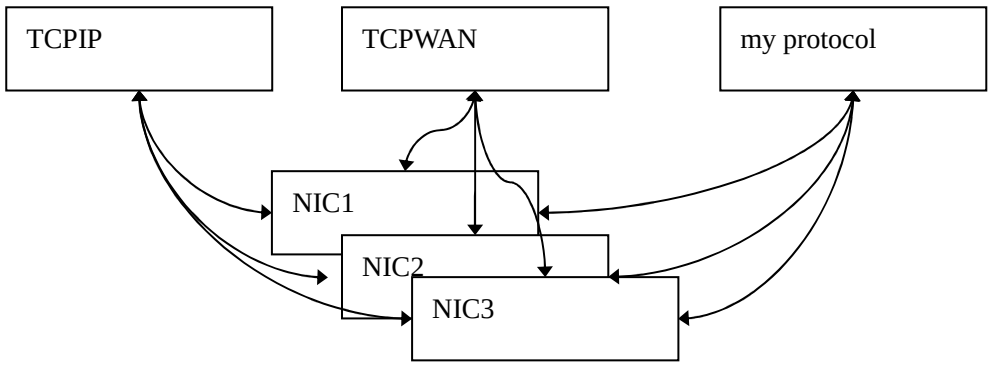
By boywhp

Email:boywhp@126.com

一、Windows 网络 NDIS 接口模型

协议驱动直接建立同下层微端口数据接口通信，可以直接处理底层网卡数据。下层的中间层驱动主要提供一个数据过滤功能，对上层协议驱动提供微端口接口，对下层 NIC 微端口驱动提供协议驱动接口。下层的微端口驱动是网卡厂家提供的网卡驱动（基于 NDIS 框架，非标准 Windows 驱动），具体的细节参考 NDIS 框架资料。

开发虚拟网卡以及防火墙推荐 NDIS 中间层，只是收发数据推荐在协议层驱动，下面是协议层驱动数据收发模型。



其中的 NIC1、NIC2、NIC3 就是你机器上的网卡，可以是微端口驱动也可能是中间层虚拟的一个 NIC，通常就是你在 IPConfig /ALL 下看到的网络适配器。一个协议驱动可以和几个网卡建立绑定。具体和哪个网卡绑定决定于协议驱动的需要。我们的协议驱动和底层的 NIC 直接交互，通常系统已经自带了 TCPIP、TCPWAN 等协议驱动，我们的驱动和这些是平行的关系。一个 NIC 接受到数据后会查询其所有绑定的协议驱动链，依次调用协议驱动注册的收包函数。因此使用协议驱动来做防火墙并不合适，因为你并不能阻止 TCPIP 的收发数据，但是嗅探数据是可以的，实际上 winpcap 就是基于标准的 NDIS 协议驱动写的。

另外还有一点就是尽管上层的 socket 接口以及应用如何变化，协议层看到的只是一个个的 NDIS Pakcet 收发请求，每个 NDIS Packet 对应一个网络数据包。具体的数据包格式参考 TcpIp 详解。数据包的组包、校验以及重传等功能在上层处理。协议层驱动只是把收的网络数据包缓冲起来等待上层读取，并将上层的数据包写入到下层的网卡，因此写协议驱动真是很轻松的事情。

二、协议驱动编程

在没有熟悉协议驱动编程前可以很多人会很茫然，不知道如何下手真正开发一个协议驱动，这里我强烈推荐 DDK2000 里面的 packet 实例，使用 Source Insight 来看（强烈不推荐 VC6），我就是这样入门的，当然我悟性也许比你低那么一点点。另外参考文章就不要找了，找也找不到的，尤其是中文的，我这个就是目前最好的文档了，自己看 DDK 会有点收获的，另外

我假定你对内核编程有一个基本了解，入门请参考楚狂人的文件过滤驱动文档。

（一）初始化 DriverEntry

Windows NDIS 内部维护了一个协议数据结构的双向链表，我们当然得把我们的协议驱动挂上去先，同时该数据结构包含了一系列函数指针需要我们指定，代码在 packet.c 里。

```
//填写 NDIS_PROTOCOL_CHARACTERISTICS 数据结构
NdisZeroMemory(&protocolChar,sizeof(NDIS_PROTOCOL_CHARACTERISTICS));

protocolChar.MajorNdisVersion            = 5;
protocolChar.MinorNdisVersion            = 0;
protocolChar.Name                        = protoName;
protocolChar.OpenAdapterCompleteHandler  = PacketOpenAdapterComplete;
protocolChar.CloseAdapterCompleteHandler = PacketCloseAdapterComplete;
protocolChar.SendCompleteHandler         = PacketSendComplete;
protocolChar.TransferDataCompleteHandler = PacketTransferDataComplete;
protocolChar.ResetCompleteHandler        = PacketResetComplete;
protocolChar.RequestCompleteHandler      = PacketRequestComplete;
protocolChar.ReceiveHandler              = PacketReceiveIndicate;
protocolChar.ReceiveCompleteHandler      = PacketReceiveComplete;
protocolChar.StatusHandler               = PacketStatus;
protocolChar.StatusCompleteHandler       = PacketStatusComplete;
protocolChar.BindAdapterHandler          = PacketBindAdapter;
protocolChar.UnbindAdapterHandler        = PacketUnbindAdapter;
protocolChar.UnloadHandler               = NULL;
protocolChar.ReceivePacketHandler        = PacketReceivePacket;
protocolChar.PnPEventHandler             = PacketPNPHandler;
```

我下面加载次序依次讲解各个函数，首先是 **NdisRegisterProtocol**

```
NdisRegisterProtocol(
    &status,
    &Globals.NdisProtocolHandle,
    &protocolChar,
    sizeof(NDIS_PROTOCOL_CHARACTERISTICS));
```

如果调用成功会返回一个协议 NDIS50_PROTOCOL_BLOCK 的数据指针(微软称为 NdisProtocolHandle)，实际上他是一个注册协议驱动单链表首指针。通过该指针即可遍历所以系统安装的协议，实现一些邪恶的用途，鉴于其邪恶性，故不给出代码，微软也不推荐该做法。当然你还要规规矩矩的填写 DriverObject->MajorFunction 等回调函数，好在 Packet 代码帮我们做了。

正常情况下,如果你使用了标准的官方做法(使用标准的 inf 安装文件，并添加协议驱动到网卡)，系统会在你注册协议驱动后，调用 BindAdapterHandler，也就是你的 PacketBindAdapter 函数，但是很多不喜欢驱动签名，且不喜欢 inf 安装并邪恶的人，发明了直接在 DriverEntry 里面使用 NdisOpenAdpater 直接打开网卡的方法，DDK 对此的原文描述如下：

Comments

A protocol driver calls NdisOpenAdapter from its *ProtocolBindAdapter* function. NDIS no longer supports calling NdisOpenAdapter from the **DriverEntry** function, which was an option available to legacy (V3.0) protocols. NDIS no longer supports V3.0 protocols. NDIS fails any attempt to call NdisOpenAdapter outside the context of *ProtocolBindAdapter*.

下面公布我测试的结果：windows2000 可以正常打开，但是好像不怎么稳定，有时能够正常截获数据有时不行，XP 和 Windows 2003 都不怎么好用。

（二）绑定网卡 PacketBindAdapter

```
PacketBindAdapter(
    OUT PNDIS_STATUS                Status,
    IN  NDIS_HANDLE                 BindContext,
    IN  PNDIS_STRING                DeviceName,
    IN  PVOID                       SystemSpecific1,
```

```
    IN    PVOID                         SystemSpecific2
)
```

由于有多块网卡，所以一般情况下在此为每个网卡创建一个 Device，初始化一些数据并打开下层适配器，官方做法如下：

首先为每个网卡创建一个设备，在该设备扩展 DeviceExtension 中指明一个 OPEN_INSTANCE 自定义数据结构，该数据结构标识了一个我们协议驱动打开的下层网卡所需数据，当然你也可以不记录。

然后就是打开下层适配器了，用的参数主要就是一个 DeviceName，好像是对应注册表里面一项网卡的唯一标识吧。唯一要说明的就是 ProtocolBindingContext：

```
VOID NdisOpenAdapter( OUT PNDIS_STATUS    Status,
    OUT PNDIS_STATUS    OpenErrorStatus,
    OUT PNDIS_HANDLE    NdisBindingHandle,
    OUT PUINT    SelectedMediumIndex,
    IN PNDIS_MEDIUM    MediumArray,
    IN UINT    MediumArraySize,
    IN NDIS_HANDLE    NdisProtocolHandle,
    IN NDIS_HANDLE    ProtocolBindingContext,
    IN PNDIS_STRING    AdapterName,
    IN UINT    OpenOptions,
    IN PSTRING    AddressingInformation    OPTIONAL);
```

该参数会在以后的收发数据里面经常出现，且是用户自定义的，只能在打开适配器的时候指定，以后在数据操作时候就可以通过该参数来得知是那个网卡来的数据，通常情况下我们需要将该参数指定为上述 OPEN_INSTANCE 数据结构

（三）打开网卡完成 PacketOpenAdapterComplete

这里没有什么好处理的，packet 代码就是简单是设置了一个事件，以便通知打开操作完成。你也可以在此时完成设置网卡过滤模式，这一点很重要，否则你不会接受到什么数据。但是在设置网卡过滤模式上要注意其操

作是异步完成的，如下代码是会 BSOD。

```
NDIS_STATUS      status;
ULONG            Mode = NDIS_PACKET_TYPE_DIRECTED;
INTERNAL_REQUEST  Request ;
if(Request)
{
    Request.Irp = NULL;
    Request.Request.RequestType = NdisRequestSetInformation;
    Request.Request.DATA.SET_INFORMATION.Oid = OID_GEN_CURRENT_PACKET_FILTER;
    Request.Request.DATA.SET_INFORMATION.InformationBuffer = &Mode;
    Request.Request.DATA.SET_INFORMATION.InformationBufferLength = sizeof(ULONG);
    NdisRequest( &status, open->AdapterHandle, &Request.Request);
}
```

原因就是因为 Request 是一个局部变量，而 NdisRequest 是异步操作的，不会等待操作完成才返回。因此导致 NdisRequest 传递的指针会提前被释放。最后 **PacketRequestComplete** 得到的是一个实际不存在的指针，导致 BSOD。解决的办法很简单，就是申请一块全局内存。并且记得在 PacketRequestComplete 中记得释放就行了。

```
NDIS_STATUS              status;
ULONG                    Mode = NDIS_PACKET_TYPE_DIRECTED;
PINTERNAL_REQUEST        pRequest = NULL;
//注意 NdisRequest 是异步操作，不能使用局部变量！
//pRequest 申请的内存会在 RequestComplete 中释放
pRequest = ExAllocatePool(NonPagedPool, sizeof(INTERNAL_REQUEST));
if(pRequest)
{
    pRequest->Irp = NULL;
    pRequest->Request.RequestType = NdisRequestSetInformation;
    pRequest->Request.DATA.SET_INFORMATION.Oid = OID_GEN_CURRENT_PACKET_FILTER;
    pRequest->Request.DATA.SET_INFORMATION.InformationBuffer = &Mode;
    pRequest->Request.DATA.SET_INFORMATION.InformationBufferLength = sizeof(ULONG);
    NdisRequest( &status, open->AdapterHandle, &pRequest->Request);
}
```

PacketRequestComplete 函数在 NDIS_PROTOCOL_CHARACTERISTICS 指定，你可以参考 Packet.c 的代码。

（四）接受到网络数据包

主要有两个个函数：ReceiveHandler、ReceivePacketHandler

都是在 NDIS_PROTOCOL_CHARACTERISTICS 时候指定，这两个回调函数有一点点不同，通常教新的网卡会支持后者，因为效率更加高一点，前者一般在教老的 windows2000 下常见一些，参数也复杂一点。但是很不幸的是，你要同时支持这两种数据接收函数，我们先从简单的 ReceivePacket 入手。

```
PacketReceivePacket(
    IN      NDIS_HANDLE             ProtocolBindingContext,
    IN      PNDIS_PACKET            Packet
    )
```

第一个参数就是我们 NdisOpenAdapter 时候时候指定的自定义数据(一个 POPEN_INSTANCE 数据结构)，第二个参数就是我们要接受的 NDIS_Packet 了，里面含有我们需要的网络数据包。现在的问题是我们要怎么把这个包收下来，一般情况下这个包是临时的，你别指望把这个指针记录下来，等过完年再收这个包。我们必须自己创建一块内存区把包给缓冲起来，这一点是很重要的，DDK 的 Packet 收包比较彪悍。如果你不怕丢包就用他的代码，否则老老实实的自己管理收包。

我的方案是维护一个 NdisRecvPackets 队列，每接受到一个数据包就是进入队列，用户端负责读取数据，并清队列中的包。思路比较简单直接 show 代码。

```
UINT                    bytesTransfered = 0;
UINT                    size;
PNDIS_PACKET            myPacket = NULL;
PVOID                   virtualAddress = NULL;

virtualAddress = SafeCopyNdisPacket(
                        &myPacket,
                        Packet,
```

```
                                (POPEN_INSTANCE)ProtocolBindingContext,
                                &size
                                );
if (virtualAddress)
{
    POPEN_INSTANCE open = (POPEN_INSTANCE)ProtocolBindingContext;
    ExInterlockedInsertTailList(
                &open->RcvPackets,
                &RESERVED(Packet)->ListElement,
                &open->RcvPacksLock
                );
}
```

其中 SafeCopyNdisPacket 是我写的一个复制 NDIS 包的函数。他创建一个 NdisPacket 并 Copy 指定的 Packet 数据到创建好的数据包。下面简单的讲解一个 NdisPacket 的创建过程：

1、首先你得通过 **NdisAllocatePacketPool** 创建一个包缓冲池，通常在 OpenAdapterComplete 中已经创建好了包缓冲池。

2、通过 **NdisAllocatePacket** 向包缓冲池中申请一个 NdisPacket。

3、通过 **NdisAllocateMemory** 向系统申请一块指定大小的内存地址空间。

4、通过 **NdisAllocateBuffer** 创建一个缓冲描述符映射到上述内存地址。

5、通过 **NdisChainBufferAtFront**，将该缓冲描述符添加到 NdisPacket 链的首部。

另外得到一个 NdisPacket 后，通过 **NdisQueryBuffer** 以及 **NdisGetNextBuffer** 即可得到所有 Ndis 包中的数据，具体细节见代码，就是简单的照样画老虎了，纯体力活。

```
PVOID
SafeCopyNdisPacket(
    OUT     PNDIS_PACKET* dst_packet,
    IN  PNDIS_PACKET  src_packet,
    IN  POPEN_INSTANCE  open,
    OUT     UINT*               pSize
    )
/*
```

```
 *   创建一个线性内存 NDISPacket,并复制 src_packet 的数据
 */
{
    UINT                    bytesTransfered = 0;
    UINT                    packetLength;
    PNDIS_PACKET            myPacket;
    NDIS_STATUS             status;
    PVOID                   virtualAddress = NULL;

    NdisQueryPacket(src_packet, NULL, NULL, NULL, &packetLength);

    virtualAddress = BuildNdisPacket(&myPacket, open, packetLength);
    if (virtualAddress == NULL)
    {
        DebugPrint(("BuildNdisPacket Faild!\n"));
        return NULL;
    }
    //
    // Following block of code locks the destination packet
    // MDLs in a safe manner. This is a temporary workaround
    // for NdisCopyFromPacketToPacket that currently doesn't use
    // safe functions to lock pages of MDL. This is required to
    // prevent system from bugchecking under low memory resources.
    //
    //备注：后来的 DDK 版本直接提供了 NdisCopyFromPacketToPacketSafe 实现
    {
        PVOID               virtualAddress;
        PNDIS_BUFFER        firstBuffer, nextBuffer;
        ULONG               totalLength;

        NdisQueryPacket(src_packet, NULL, NULL, &firstBuffer, &totalLength);
        while( firstBuffer != NULL)
        {
            NdisQueryBufferSafe(firstBuffer, &virtualAddress,
                &totalLength, NormalPagePriority );
            if(!virtualAddress) {
                //
                // System is running low on memory resources.
                // So fail the read.
                //
                status = STATUS_INSUFFICIENT_RESOURCES;
                goto CleanExit;
            }
            NdisGetNextBuffer(firstBuffer,    &nextBuffer);
```

```
                    firstBuffer = nextBuffer;
                }
        }

        NdisCopyFromPacketToPacket(
                myPacket,
                0,
                packetLength,
                src_packet,
                0,
                &bytesTransfered
                );

        if (bytesTransfered > 0)
        {
                *pSize = bytesTransfered;
                *dst_packet = myPacket;
                return virtualAddress;
        }

CleanExit:
        FreeNdisPacket(myPacket);
        return NULL;
}

PVOID
BuildNdisPacket(
        IN    OUT PNDIS_PACKET *pPacket,
        IN    POPEN_INSTANCE    open,
        IN    ULONG size)
{
        PVOID                   virtualAddress;
        PNDIS_PACKET            myPacket;
        PNDIS_BUFFER            buffer;
        NDIS_PHYSICAL_ADDRESS highestAcceptableAddress = {-1, -1};
        NDIS_STATUS status = NDIS_STATUS_SUCCESS;

        NdisAllocatePacket(
                &status,
                &myPacket,
                open->PacketPool
                );
        if (status != NDIS_STATUS_SUCCESS)
        {
```

```
            DebugPrint(("BuildPacket - No free packets\n"));
            return NULL;
        }

        status = NdisAllocateMemory(
            &virtualAddress,
            size,
            0,
            highestAcceptableAddress
            );
        if (status != NDIS_STATUS_SUCCESS)
        {
            DebugPrint(("BuildPacket - AllocateMemory Failed\n"));
            NdisFreePacket(*pPacket);
            return NULL;
        }

        NdisAllocateBuffer(
            &status,
            &buffer,
            open->BufferPool,
            virtualAddress,
            size
            );
        if (status != NDIS_STATUS_SUCCESS)
        {
            DebugPrint(("BuildPacket - AllocateBuffer Failed\n"));
            NdisFreeMemory(virtualAddress, size, 0);
            NdisFreePacket(myPacket);
            return NULL;
        }
        NdisChainBufferAtFront(myPacket, buffer);
        *pPacket = myPacket;

        return virtualAddress;
}
```

下面我们看 ReceiveHandler 的处理

```
NDIS_STATUS
PacketReceiveIndicate (
        IN NDIS_HANDLE ProtocolBindingContext,
        IN NDIS_HANDLE MacReceiveContext,
        IN PVOID            HeaderBuffer,
```

```
    IN UINT             HeaderBufferSize,
    IN PVOID            LookAheadBuffer,
    IN UINT             LookaheadBufferSize,
    IN UINT             PacketSize
    )
```

第一个参数不解释了，第二个参数也不解释了，因为我不清楚，用不上。HeaderBuffer 指向一个以太帧，格式自己看 TcpIp 详解，HeaderBufferSize 一般就是 14 了，至少我没有见过其他大小的。LookAHeadBuffer 指向剩下的数据，LookAHeadBufferSize 是其数据大小，PacketSize 是后面实际的数据大小。

一般情况下 PacketSize <=LookAHeadBufferSize 这是最理想的情况，说明我们的数据包完整的存在于 LookAHeadBuffer 中，否则就麻烦一点，但是对于我们优秀的程序员来说也不过是小菜了，好在我见过的绝大多数情况都是正常情况。

```
PVOID                   virtualAddress;
NDIS_STATUS             status = NDIS_STATUS_SUCCESS;
UINT                    bytesTransfered = 0;
ULONG                   bufferLength = HeaderBufferSize + PacketSize;
PVOID                   packetVA = NULL;
if (HeaderBufferSize > ETHERNET_HEADER_LENGTH)    return NULL;
virtualAddress = BuildNdisPacket(pPacket, open, bufferLength);
if (!virtualAddress)    return NULL;
if (PacketSize <= LookaheadBufferSize)
{
    NdisMoveMappedMemory(
            (PUCHAR)virtualAddress,
            HeaderBuffer,
            ETHERNET_HEADER_LENGTH
            );
    NdisMoveMappedMemory(
            (PUCHAR)virtualAddress + ETHERNET_HEADER_LENGTH,
            LookAheadBuffer,
            PacketSize
            );
        return virtualAddress;
```

```
}
Else //处理 BT 情况
```

BT 情况下需要我们自己调用 **NdisTransferData** 通知下层网卡将数据传输上来，到

```
VOID
  NdisTransferData(
    OUT PNDIS_STATUS    Status,
    IN NDIS_HANDLE    NdisBindingHandle,
    IN NDIS_HANDLE    MacReceiveContext,
    IN UINT    ByteOffset,
    IN UINT    BytesToTransfer,
    IN OUT PNDIS_PACKET    Packet,
    OUT PUINT    BytesTransferred
    );
```

请注意其中的 ByteOffset，我以为可以指定 Copy 的首地址，先创建一个数据包，首先把以太帧填充到前 14 个字节，然后指定 BytesOffset 为 14 调用 **NdisTransferData**，天真的认为 Ndis 复制后面的数据到 14 以后的 Packet 缓冲中。但是得到的结果是函数失败：-）没办法了，只好在拷贝的时候先把以太帧复制到包的最后面，然后调用，当然在 PacketTransferDataComplete 记得要将以太头移到前面来，虽然方法猥琐了点，但是凑合着用吧。

```
            NdisMoveMappedMemory(
                (PUCHAR)virtualAddress + bufferLength - ETHERNET_HEADER_LENGTH,
                HeaderBuffer,
                ETHERNET_HEADER_LENGTH
                );
            //调用 NdisTransferData 将剩下的数据传上来
            IoIncrement(open);
            NdisTransferData(
                &status,
                open->AdapterHandle,
                MacReceiveContext,
                0,
                PacketSize,
                *pPacket,
                &bytesTransfered
                );
```

```
        KdPrint(("NdisTransferData STATUS %08x\n", status));
        if (status == NDIS_STATUS_PENDING)
        {
            return NDIS_STATUS_SUCCESS;
        }

        PacketTransferDataComplete(
            open,
            *pPacket,
            status,
            bytesTransfered
            );
        return NDIS_STATUS_SUCCESS;
```

（五）完成数据传输 PacketTransferDataComplete

该函数同样是在注册协议的时候指定，这里我们只需将数据包简单的重组一下就可以了，先创建一个 NdisPacket，然后将数据 Copy 过来即可，记得先拷贝后面 14 字节的以太头，然后再拷贝后面的实际数据，并释放 IO 计数器，原型如下。

```
PacketTransferDataComplete (
    IN NDIS_HANDLE      ProtocolBindingContext,
    IN PNDIS_PACKET     pPacket,
    IN NDIS_STATUS      Status,
    IN UINT             BytesTransfered
    )
```

注意其中的 pPacket 就是我们调用 **NdisTransferData** 指定的数据包指针，用完后一定要记得释放！另外关于 IO 计数器我还要讲几句。Packet 里面进行一个异步数据操作时候一般会先调用一下 **IoIncrement** 函数，此函数简单的维护一个计数器，你可以把他理解为 COM 里面的引用计数器。每发出一个异步 IO 调用时候，Packet 将计数器加一，在数据操作完成后再将计数器减一。感觉很无趣吧：-）哈哈！这样做主要是为了在驱动释放资源的时候，它会等待计数器清空，否则你将有可能在卸载驱动时候 BSOD，因为你把一些资源 Free 掉后，Ndis 可能出现一个完成事件，结果调用你的完成函数

就容易出现意外，总之，安全起见，还是遵守这个原则吧，毕竟人家微软程序员也这么干。

（六）应用程序读数据 PacketRead 操作

```
NTSTATUS
PacketRead(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
```

注意 DeviceObject 扩展数据指向的是一个 POPEN_INSTATNCE 数据指针，Irp 里面有用户程序的缓冲区大小以及指针。我们把 NDIS 接受数据包队列里面的数据包复制到 Irp 里面的缓冲区，并释放队列。

```
remainSize = irpSp->Parameters.Read.Length;
pBuffer = MmGetSystemAddressForMdlSafe(Irp->MdlAddress, NormalPagePriority);
do {
    packListEntry = ExInterlockedRemoveHeadList(
                    &open->RcvPackets,
                    &open->RcvPacksLock
                    );
    if (packListEntry != NULL)
    {
        PNDIS_BUFFER    firstBuffer;
        PVOID           pPacketVAddress;
        UINT            packSize;
        //首先得到当前 NDIS_PACKET 指针
        reserved = CONTAINING_RECORD(packListEntry, PACKET_RESERVED, ListElement);
        pPacket = CONTAINING_RECORD(reserved, NDIS_PACKET, ProtocolReserved);
        //注意：
        //1、收包的时候一个 NdisPacket 只创建一个 Buffer
        //2、用户缓冲至少可以完整收一个 NdisPacket 包
        NdisQueryPacket(pPacket, NULL, NULL, &firstBuffer, NULL);
        NdisQueryBuffer(firstBuffer, &pPacketVAddress, &packSize);
        if (packSize <= remainSize)
        {
            NdisMoveMemory(
                    (PUCHAR)pBuffer,
                    (PUCHAR)pPacketVAddress,
                    packSize
                    );
```

```
                (PUCHAR)pBuffer += packSize;
                remainSize -= packSize;
                totalSize += packSize;
                FreeNdisPacket(pPacket);
            }
            else
            {
                //重新加入链表
                ExInterlockedInsertHeadList(
                    &open->RcvPackets,
                    &RESERVED(pPacket)->ListElement,
                    &open->RcvPacksLock
                    );
                    break;
            }
        }
} while(packListEntry && remainSize > 0);
```

注意：这里处理数据如果缓冲区不足一个 NdisPacket 的大小，我重新将包放入队列，代码很简单，别问我为什么，因为彪悍的代码不需要注释。

（七）数据包发送 PacketWrite

```
NTSTATUS
PacketWrite(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
```

太简单了，但是有始有终吧，先创建一个 NdisPacket，然后将 Irp 数据复制过去，呵呵！其实不是啦，可以直接用 **NdisChainBufferAtFront** 把 Irp 的 MdlAddress 链接到 Ndispacket 上去，记得处理 SendComplete（该函数也是在注册协议驱动的时候指定）就可以了。不过好在 Packet 代码貌似处理很好，我基本上不用改了，自己看看吧，无非就是设置 Irp 完成，释放 NdisPacket，IO 计数器减一，下面我直接 show 微软的代码。

PacketWrite 代码

```
NdisAllocatePacket(
            &Status,
            &pPacket,
```

```
            open->PacketPool
            );
        if (Status != NDIS_STATUS_SUCCESS) {
            Irp->IoStatus.Status = STATUS_INSUFFICIENT_RESOURCES;
            IoCompleteRequest (Irp, IO_NO_INCREMENT);
            IoDecrement(open);
            return STATUS_INSUFFICIENT_RESOURCES;
        }
        RESERVED(pPacket)->Irp=Irp;
        NdisChainBufferAtFront(pPacket,Irp->MdlAddress);
        IoMarkIrpPending(Irp);
        NdisSend(
            &Status,
            open->AdapterHandle,
            pPacket);

        if (Status != NDIS_STATUS_PENDING) {
            PacketSendComplete(
                open,
                pPacket,
                Status
                );
        }
```

PacketSendComplete 代码

```
irp=RESERVED(pPacket)->Irp;
if (irp == NULL)
{
    //该 Ndis Packet 是转发的包
    FreeNdisPacket(pPacket);
    IoDecrement((POPEN_INSTANCE)ProtocolBindingContext);
    return;
}
irpSp = IoGetCurrentIrpStackLocation(irp);

NdisFreePacket(pPacket);

if(Status == NDIS_STATUS_SUCCESS) {
    irp->IoStatus.Information = irpSp->Parameters.Write.Length;
    irp->IoStatus.Status = STATUS_SUCCESS;
} else {
    irp->IoStatus.Information = 0;
    irp->IoStatus.Status = STATUS_UNSUCCESSFUL;
}
IoCompleteRequest(irp, IO_NO_INCREMENT);
```

```
IoDecrement((POPEN_INSTANCE)ProtocolBindingContext);
```

（八）结束语

很高兴你还能认真的看到这里，希望能够对你有点帮助，我之所以花一天的时间专门来写一个开发文档，就是因为我自己在开发的时候走了太多的弯路了，中文资料实在是少了点，尤其是涉及到一些细节更是如此。另外我没有附完整的代码，是因为我的代码不是正统的，有点邪恶：-）DDK 里面的 Packet 代码才是，而且自己动手才能体会到其中的乐趣，如果你不能从中享受到乐趣，那还是不要搞开发。另外，希望看到我 Email:boywhp@126.com 的 MM 不要太兴奋，因为这个 Email 估计有 6 年之久了，昔日的 boy 早已是人老珠黄了，要发 manwhp@126.com :-)这个也是我的邮箱，希望你能接受。

By boywhp 08/03/26 广州