네트워크 드라이버 프로그래밍

네트워크 필터 드라이버 작성

지난 호에서는 네트워크 드라이버 작성에 필요한 이론들을 살펴봤다. 이번 시간부터는 네트워크 드라이버를 직접 개발하는 것을 목적으로 삼아 직접 설치와 테스트 등을 실시하고, 이를 통해 네트워크 드라이버의 실제 동작을 이해하는 데 주안점을 둔다. 지금부터 소개되는 내용을 충분히 이해한다면 윈도우의 드라이버 계층과 네트워크 계층에 대한 이해 폭이 자연스레 넓어질 것이다.

연 재 순 서
1회 | 2006. 6 | NDIS의 이해와 활용법
2회 | 2006. 7 | 네트워크 필터 드라이버 작성
3회 | 2006. 8 | 드라이버 제어 프로그램 만들기

연 재 가 이 드
운영체제 | 윈도우 2000/XP
개발도구 | 비주얼 스튜디오 6.0, DDK
기초지식 | C++ 또는 MFC 프로그래밍
응용분야 | 윈도우 드라이버 프로그래밍 또는 커널 모드 프로그래밍

박상민 sangmin.park@gmail.com | 소만사(www.somansa.com)에서 웹 분석 솔루션 설계 및 개발을 담당했다. 윈도우 기반 개발, 웹 개발, 데이터베이스 프로그래밍 등의 기술을 습득해왔고, 어떤 프로젝트가 주어지더라도 최상의 결과를 만들어내는 소프트웨어 아키텍트가 되는 것이 꿈이다. 인생을 편하게 사는 것을 목표로 삼고, '내일은 내일의 바람이 분다(tomowind.egloos.com)' 라는 블로그를 운영하고 있다.

이번 시간부터는 네트워크 드라이버를 직접 만들어본다. C:\NTDDK\src\network에 있는 DDK 샘플인 PassThru와 NetCfg를 바탕으로 코드를 작성해본다. 기본적으로는 다음과 같은 핵심 파일들이 존재한다.

- PassThru

PassThru.c : DriverEntry를 포함한다. IM 드라이버로서 Protocol, Miniport를 등록
PassThru.h : PassThru.h의 헤더
Miniport.c : Miniport에 관련된 함수들
Protocol.c : Protocol에 관련된 함수들
Precomp.h : Precompiled 헤더
Sources : Build Utility에서 사용하는 소스 파일 리스트

- NetCfg

implinc.cpp : include 파일들을 가지고 있다.
main.cpp : 메인 함수를 가지고 있다. exe 파일의 진입점
pch.h : Precompiled 헤더
snetcfg.cpp : 드라이버 등록에 관한 함수들 집합
snetcfg.h : snetcfg.h의 헤더

여기서는 두 가지 결과물 파일을 만들 예정이다. PassThru 예제에 필터링 코드를 추가한 네트워크 드라이버 파일을 작성하고, NetCfg를 기반으로 한 PassThruHelper 프로젝트를 만들어 드라이버의 등록이 가능하도록 구현한다.

드라이버 필터링 코드 삽입

먼저 가장 핵심이라 할 수 있는 필터링 코드를 삽입하는 부분이다. 우선 PassThru에 대해 알아보고 필터링 코드를 삽입해보도록 하자.

InterMediate 드라이버의 구조

앞에서 이미 설명한 바와 같이 Intermediate 드라이버는 〈그림 1〉과 같이 중간 단계의 드라이버로 위와 아래의 드라이버 영역에 대해 다소 특수하고 투명한 역할을 담당한다. 다시 말해 상위 단계의 트랜스포트 드라이버에게는 미니포트 드라이버인 것처럼 보이고, 하위 단계의 NIC 드라이버에게는 프로토콜 드라이버인 것처럼 보이는 것이다. 참고로 말해 드라이버의 진입점인 DriverEntry에서는 미니포트 드라이버, 프로토콜 드라이버를 등록해주는 역할을 수행한다.

패킷을 받는 경우에는 NIC 드라이버와 Intermediate 드라이버의 프로토콜 부분을 각각 거친 후 트랜스 포트 드라이버를 지

나게 된다. 그리고 패킷을 보내는 경우에는 트랜스포트 드라이버에서 Intermediate 드라이버의 미니포트 부분을 지나 NIC 드라이버로 가게 된다. 따라서 우리가 패킷 필터링을 하고 싶다면, Intermediate 드라이버의 프로토콜 부분에 있는 받는 함수(PtReceive)와 미니포트 부분의 보내는 함수(MpSend)에 조작을 가하면 된다.

〈그림1〉 Intermediate 드라이버의 구조

필터 작성

〈리스트 1〉의 MpSend 구현을 통해 패킷 전송 시에 필터링이 가능하도록 해보자. 이 함수를 살펴보면 그 주요 역할은 인자로 받는 패킷으로 NdisPacketSend를 호출하는 것임을 알 수 있다. 따라서 NdisPacketSend 호출 전에 패킷 조사를 실시해 조건에 맞으면 필터링이 가능하도록 구현한다. 실제 필터링은 조건에 맞을 경우 리턴하도록 만들면 되고, 따라서 우리는 이 부분에 PrintSendPacketInfo라는 함수를 호출해 패킷 출력만 하도록 구성한다.

〈리스트 1〉 패킷 전송시에 거치는 MpSend 함수

```
NDIS_STATUS
MPSend(
        IN      NDIS_HANDLE
MiniportAdapterContext,
        IN      PNDIS_PACKET    Packet,
        IN      UINT            Flags
        )
{
        PADAPT          pAdapt =
(PADAPT)MiniportAdapterContext;
        NDIS_STATUS             Status;
        PNDIS_PACKET    MyPacket;
        PRSVD           Rsvd;
        PVOID           MediaSpecificInfo = NULL;
        ULONG           MediaSpecificInfoSize = 0;

        //
        //  According to our LBFO design, all sends will
be performed on the secondary miniport
        //      However, the must be completed on the
primary's miniport handle
        //

        ASSERT (pAdapt->pSecondaryAdapt);
        pAdapt = pAdapt->pSecondaryAdapt;

        if (IsIMDeviceStateOn (pAdapt) == FALSE)
        {
                return NDIS_STATUS_FAILURE;
        }

        NdisAllocatePacket(&Status,
                           &MyPacket,
                           pAdapt-
>SendPacketPoolHandle);
```

필자 메모

필자는 지난 4월까지 한 소프트웨어 개발 회사에 근무했고, 현재는 재충전(?)의 시간을 가지는 중이다. 주위에 회사를 다니다가 그만둔 개발자들은 흔히 한 달만 지나면 프로그래밍에 대한 감이 떨어진다고 말한다. 이어 몇 달 후 다시 입사해 프로그래밍을 시작하면 프로그래밍 지식이 쉽사리 떠오르지 않아 막막함을 느끼곤 한다고 호소한다. 필자 역시 최근 이와 비슷한 경험(?)에 시달리고 있다. 이번 연재에 쓰일 간단한 예제를 만드는 데도 몇 차례나 블루 스크린을 발생시키면서 시행착오를 겪었기 때문이다.

앤드류 헌트의 『실용주의 프로그래머』라는 책을 보면, 자신의 지식 포트폴리오를 관리하라고 적혀 있다. 아마도 평소에 자신의 실력을 기르는데 힘쓰라는 의미일 것이다. 이를 위해 앤드류 헌트는 해마다 하나의 새로운 프로그래밍 언어를 배우고, 분기마다 기술 서적 하나씩을 꼭 읽으라고 강조한다. 이에 영향을 받은 탓일까? 필자 역시 프로그래밍의 감이 떨어지는 것을 막고, 아울러 지식 포트폴리오와 경험을 확대하기 위해 이처럼 마소를 통한 기고에도 아낌없이 시간을 투자하고 있다. 필자의 지식을 공유함으로써 독자들은 새로운 지식을 얻게 되고, 마찬가지로 필자 역시 원고 작성 과정에서 갖가지 자료들을 새롭게 접하면서 지식을 키워가고 있는 셈이다.

독자 여러분들도 자신의 지식 포트폴리오를 보다 능동적으로 관리할 방법을 틈틈이 고민해주었으면 하는 바람이다.

```
        if (Status == NDIS_STATUS_SUCCESS)
        {
                PNDIS_PACKET_EXTENSION      Old, New;
                Rsvd = (PRSVD)(MyPacket-
>ProtocolReserved);
                Rsvd->OriginalPkt = Packet;
                MyPacket->Private.Flags = Flags;
                MyPacket->Private.Head = Packet-
>Private.Head;
                MyPacket->Private.Tail = Packet-
>Private.Tail;
                NdisSetPacketFlags(MyPacket,
NDIS_FLAGS_DONT_LOOPBACK);

                // Copy the OOB Offset from the original
packet to the new
                // packet.

NdisMoveMemory(NDIS_OOB_DATA_FROM_PACKET(MyPacket),

NDIS_OOB_DATA_FROM_PACKET(Packet),

sizeof(NDIS_PACKET_OOB_DATA));
                // Copy the per packet info into the new
packet
                // This includes ClassificationHandle,
etc.
                // Make sure other stuff is not copied
!!!
                //
                NdisIMCopySendPerPacketInfo(MyPacket,
Packet);
                // Copy the Media specific information

NDIS_GET_PACKET_MEDIA_SPECIFIC_INFO(Packet,

&MediaSpecificInfo,

&MediaSpecificInfoSize);
                if (MediaSpecificInfo ||
MediaSpecificInfoSize)
                {

NDIS_SET_PACKET_MEDIA_SPECIFIC_INFO(MyPacket,

MediaSpecificInfo, MediaSpecificInfoSize);
                }

                // 이곳에 패킷 분석 정보나 필터링 정보를 넣는다.
                PrintSendPacketInfo(Packet);
                NdisSend(&Status,
                          pAdapt->BindingHandle,
                          MyPacket);
                if (Status != NDIS_STATUS_PENDING)
                {

NdisIMCopySendCompletePerPacketInfo (Packet, MyPacket);
                          NdisFreePacket(MyPacket);
                }
        }
        else
        {
        }
        return(Status);
}
```

특정 포트의 패킷 잡아내기

지금부터는 특정 포트의 패킷을 잡아내는 루틴을 만들어 본다. 이를 위해서는 먼저 패킷에 대한 이해가 선행되어야 한다. 이더넷(ethernet) 패킷 헤더는 〈그림 2〉와 같은 모습이다. 14 바이트의 이더넷 헤더와 20 바이트의 IP 헤더, 20 바이트의 TCP 헤더로 각각 구성되어 있다. 물론 IP 헤더와 TCP 헤더에 4 바이트씩의 옵션이 추가로 붙을 수 있지만, 그렇게 구현된 경우는 실제로 찾아보기 어렵다. 일반적으로 IP 헤더에서 IP를 얻어내고, TCP 헤더에서 포트를 얻는다.

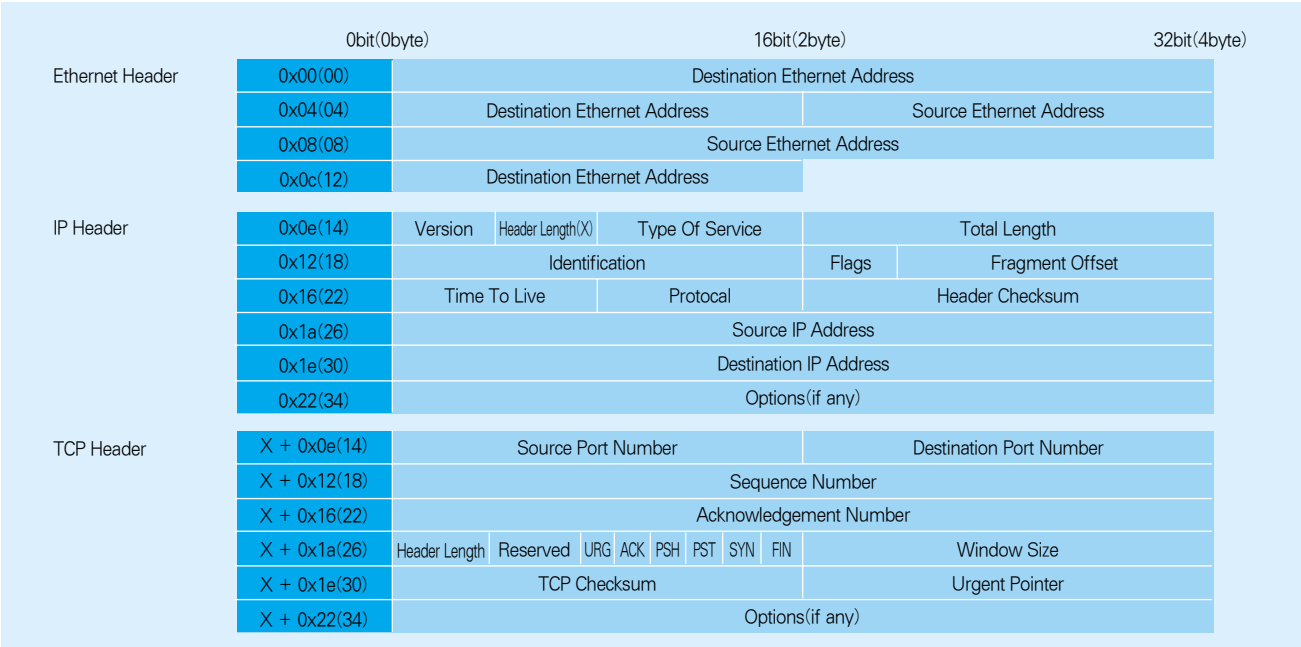
PrintSendPacketInf● 함수

PrintSendPacketInfo는 〈리스트 2〉처럼 패킷을 받아 이더넷 헤더분석, IP 헤더분석, TCP 헤더 분석을 실시하고 80 포트일 경우에만 출력을 해주도록 한다. 물론 그 뒤의 데이터들에 대해서도 처리는 가능하다. 예를 들어 HTTP 헤더에 대해서도 추가적인 처리가 가능한데, IP나 TCP 헤더를 얻어 온 것처럼 HTTP 헤더를 처리할 수 있다.

이미 설명했다시피 이 함수에서 특정 값을 반환하고 MpSend의 함수 호출부에서 리턴을 반환하면 패킷이 필터링 되는 것이다. 또한 올라오는 패킷에 대해 필터링하고 싶다면 PtReceive 함수에 동일하게 적용하면 된다.

〈리스트 2〉 PrintSendPacket 함수

```
void PrintSendPacketInfo(PNDIS_PACKET Packet)
{
        PNDIS_BUFFER        FirstBuffer;
        PVOID               pData;
        UINT                TotalPacketLength;
        UINT                DataLength;
        UINT                PacketLength;
        ether_t             *pEtherHeader;
        ip_t                *pIpHeader;
        tcp_t               *pTcpHeader;
        UCHAR               SrcIP[15];
        UCHAR               DesIP[15];
```



	0bit(0byte)	16bit(2byte)	32bit(4byte)
Ethernet Header	0x00(00)	Destination Ethernet Address	
	0x04(04)	Destination Ethernet Address	Source Ethernet Address
	0x08(08)	Source Ethernet Address	
	0x0c(12)	Destination Ethernet Address	
IP Header	0x0e(14)	Version / Header Length(X) / Type Of Service	Total Length
	0x12(18)	Identification	Flags / Fragment Offset
	0x16(22)	Time To Live / Protocal	Header Checksum
	0x1a(26)	Source IP Address	
	0x1e(30)	Destination IP Address	
	0x22(34)	Options(if any)	
TCP Header	X + 0x0e(14)	Source Port Number	Destination Port Number
	X + 0x12(18)	Sequence Number	
	X + 0x16(22)	Acknowledgement Number	
	X + 0x1a(26)	Header Length / Reserved / URG / ACK / PSH / PST / SYN / FIN	Window Size
	X + 0x1e(30)	TCP Checksum	Urgent Pointer
	X + 0x22(34)	Options(if any)	

〈그림 2〉 이더넷, IP, TCP 패킷 헤더

```
        // 패킷을 복사해온다.
        NdisQueryPacket(Packet, NULL, NULL, &FirstBuffer,
&TotalPacketLength);

        // 이더넷 헤더를 읽어온다.
        NdisQueryBuffer(
                FirstBuffer,
                &pData,
                &DataLength
                );
        PacketLength = DataLength;
        pEtherHeader = (ether_t*)(pData);
        /*
        DbgPrint("Src Addr:%.2X%.2X%.2X%.2X%.2X%.2X   Des
Addr:%.2X%.2X%.2X%.2X%.2X%.2X \
                        d_type:%.4X",
                pEtherHeader->src_addr[0], pEtherHeader-
>src_addr[1], pEtherHeader->src_addr[2],
                pEtherHeader->src_addr[3], pEtherHeader-
>src_addr[4], pEtherHeader->src_addr[5],
                pEtherHeader->dest_addr[0], pEtherHeader-
>dest_addr[1], pEtherHeader->dest_addr[2],
                pEtherHeader->dest_addr[3], pEtherHeader-
>dest_addr[4], pEtherHeader->dest_addr[5],
                ntohs(pEtherHeader->d_type));
        */

        // IP 헤더를 읽어온다.
        if(     pEtherHeader->d_type != htons(0x0800))
                return;

        if(PacketLength >= sizeof(ether_t)+sizeof(ip_t))
        {
pIpHeader=(ip_t*)(pEtherHeader+sizeof(ether_t));
        }
        else
        {
                NdisGetNextBuffer(
                        FirstBuffer,
                        &FirstBuffer
                        );
                if(FirstBuffer != NULL)
                {
                        NdisQueryBuffer(
                                FirstBuffer,
                                &pData,
                                &DataLength
                                );
                        pIpHeader=(ip_t*)(pData);
                        PacketLength+=DataLength;
                }
                else return;
        }
        MakeIPNum2Str(ntohl(pIpHeader->ip_src), SrcIP);
        MakeIPNum2Str(ntohl(pIpHeader->ip_dst), DesIP);
        // DbgPrint("srcIP:%s desIP:%s", SrcIP, DesIP);

        // TCP 헤더를 읽어온다.
        if(pIpHeader->ip_p!=IPPROTO_TCP)
                return;

        if(PacketLength >=
sizeof(ether_t)+sizeof(ip_t)+sizeof(tcp_t))
        {
```

```
pTcpHeader=(tcp_t*)(pIpHeader+sizeof(ip_t));
        }
        else
        {
                NdisGetNextBuffer(
                        FirstBuffer,
                        &FirstBuffer
                        );
                if(FirstBuffer != NULL)
                {
                        NdisQueryBuffer(
                                FirstBuffer,
                                &pData,
                                &DataLength
                                );
                        pTcpHeader=(ip_t*)(pData);
                        PacketLength+=DataLength;
                }
                else return;
        }
        if(pTcpHeader==NULL)
                return;

        // 조건에 맞는 것에 대해 찍어준다.
        if( ntohs(pTcpHeader->th_dport) == 80 )
        {
                DbgPrint("srcIP:%s dscIP:%s srcPort:%d,
dscPort:%d",
                        SrcIP, DesIP,
                        ntohs(pTcpHeader->th_sport),
                        ntohs(pTcpHeader->th_dport));
        }
}
```

드라이버 컴파일

드라이버는 제작과 컴파일이 어려워 이를 돕기 위해 몇 가지 수단이 등장했다. Makefile 역시 그 대표적인 수단 가운데 하나다. 여기서는 Makefile을 이용한 컴파일에 대해 살펴보고, 이와 함께 VC로 포팅해 컴파일 하는 방법도 소개한다. IDE를 사용한 개발이 효율성이 좋은 만큼 포팅을 통한 방법이 개발을 훨씬 더 편리하게 해줄 것이다.

Makefile을 이용한 컴파일

Makefile을 이용해 컴파일 할 때는 소스 파일과 sources, makefile이 한 폴더에 존재해야 한다. makefile은 〈리스트 3〉과 같고 항상 동일하다. DDK에 있는 makefile.def를 자동으로 포함시키는 역할을 수행하는데, 우리가 직접 수정하지는 않는다.

〈리스트 3〉 드라이버 제작시 이용하는 makefile

```
#
# DO NOT EDIT THIS FILE!!!  Edit .\sources. if you want to
add a new source
# file to this component.  This file merely indirects to
the real make file
# that is shared by all the components of NT
#
!INCLUDE $(NTMAKEENV)\makefile.def
```

sources 파일에는 사용할 컴파일 옵션들을 전부 명시하게 된다. 〈리스트 4〉처럼 컴파일 옵션, 사용하는 소스 파일들, 목적 파일 이름 등을 서술한다. 소스 파일을 추가시켜주거나, Include에 덧붙일 것이 있을 때 sources에서 직접 포함시키면 된다. 〈리스트 4〉에서는 Include 부분을 DDK의 inc 폴더에 맞게끔 각자 수정해 사용한다.

〈리스트 4〉 PassThru에서 사용되는 s●urces 파일

```
TARGETNAME=PASSTHRU
TARGETPATH=obj
TARGETTYPE=DRIVER
TARGETLIBS=$(DDK_LIB_PATH)\ndis.lib
C_DEFINES=$(C_DEFINES) -DNDIS40 -DNDIS_MINIPORT_DRIVER -
DNDIS40_MINIPORT

INCLUDES=..\..\inc
SOURCES= passthru.c    \
         protocol.c    \
         miniport.c    \
         passthru.rc

MSC_WARNING_LEVEL=/W3 /WX
PRECOMPILED_INCLUDE=precomp.h
PRECOMPILED_PCH=precomp.pch
PRECOMPILED_OBJ=precomp.obj
```

이제 준비를 마쳤으니 실제 컴파일을 시작해본다. 시작 -〉 프로그램 -〉 Development Kits -〉 Windows 2000 DDK -〉 Checked Build Environment를 클릭하면 콘솔 창이 나타난다. 이 곳의 free build/checked build는 디버깅 정보 포함 유무에 따라 구분되는 것인데, 여기서는 checked build에서 빌드하도록 한다. build라는 유틸리티를 써서 컴파일이 이뤄지는데, 다음과 같은 옵션을 활용할 수 있다.

- Build 컴파일 옵션

[-c] : delete all object file

[-e] : generate build.log, build.wrn & build.err files

[-Z] : no dependency checking or scanning source files

일반적으로 컴파일이 이뤄질 때는 build -cZ, build -ceZ 등과 같이 옵션이 적용된다. 〈화면 1〉처럼 나타나면 컴파일이 성공한 것이다.

〈화면 1〉 build가 성공한 모습

VC로 포팅

컴파일의 용이성 뿐 아니라 코드 작성의 용이성을 위해서도 VC로 포팅하는 편이 바람직하다. PassThru.dsp라는 프로젝트 파일을 만들고 sources에 있던 내용을 기술한다. 여기서 중요한 것은 Includes 디렉토리와 컴파일 될 소스 파일에 대한 설정이다. 모두 설정하고 나면 〈리스트 5〉와 같이 구성된다.

〈리스트5〉 VC로 포팅된 PassThru.dsp

```
CFG=PassThru - Win32 Debug

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "PassThru - Win32 Release"

... Release 빌드에서 컴파일 옵션이 정해진다.
# ADD BASE CPP /nologo /W3 /GX /O2 /D "NDEBUG" /D DBG=0
/YX /FD /c
# ADD CPP /nologo /Gz /W3 /Gi /O2 /I "C:\NTDDK\INC" /I
"C:\NTDDK\INC\DDK" /D "NDEBUG" /D DBG=0 /D "_X86_" /D
_WIN32_WINNT=0x500 /D "NDIS50_MINIPORT" /D
"NDIS_MINIPORT_DRIVER" /D "NDIS50" /D "!dbgprint" /FR /YX
/FD /Gs -GF /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /win32
...
!ELSEIF  "$(CFG)" == "PassThru - Win32 Debug"

... Debug 빌드에서 컴파일 옵션이 정해진다.
# ADD BASE CPP /nologo /W3 /Gm /GX /ZI /Od /D DBG=1 /YX
/FD /GZ /c
# ADD CPP /nologo /Gz /W3 /Gm /Gi /Zi /Od /I
"C:\NTDDK\INC" /I "C:\NTDDK\INC\DDK" /D DBG=1 /D "_X86_"
/D _WIN32_WINNT=0x500 /D "NDIS50_MINIPORT" /D
"NDIS_MINIPORT_DRIVER" /D "NDIS50" /FR /YX /FD /Gs -GF /c
# ADD BASE MTL /nologo /D "_DEBUG" /mktyplib203 /win32
...

!ENDIF

# Begin Target

# Name "PassThru - Win32 Release"
# Name "PassThru - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"
# Begin Source File

... 여기서부터는 사용되는 소스파일들을 적어준다.
SOURCE=.\miniport.c
# End Source File
# Begin Source File

SOURCE=.\passthru.c
# End Source File
# Begin Source File

SOURCE=.\protocol.c
# End Source File
# End Group

# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl"
# Begin Source File

SOURCE=.\passthru.h
# End Source File
# Begin Source File

SOURCE=.\precomp.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter
"ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe"
# Begin Source File

SOURCE=.\passthru.rc
# End Source File
# End Group
# End Target
# End Project
```

이렇게 만들고 나면 makefile, sources가 필요 없어진다. 보통 VC 개발을 하듯 dsp를 열어 코드를 작성하고, F7을 눌러 컴파일 한다.

Inf 파일의 설정

드라이버 파일을 만들고 나면 해당하는 inf 파일을 작성해줘야 한다. inf 파일에는 드라이버에 대한 버전과 설치 위치 등 몇가지 정보가 포함된다. PassThru에는 netsf.inf, netsf_m.inf라는 두 가지의 inf 파일이 존재한다. 파일들을 기본적으로 가져다 쓸 수 있지만, 드라이버가 표시되는 이름만은 바꿔주도록 한다. 〈리스트 6〉에는 netsf.inf에 기술된 내용들이 제시되어 있다.

〈리스트6〉 netsf.inf

```
[Version]
Signature  = "$Windows NT$"
Class      = NetService
ClassGUID  = {4D36E974-E325-11CE-BFC1-08002BE10318}
Provider   = %Msft%
DriverVer  = 06/24/1999,5.00.2071.1
...

[SourceDisksFiles]
sfilter.dll=1
passthru.sys=1
netsf_m.inf=1
...

; 스트링을 Sample Filter에서 Maso Filter로 바꿔준다.
[Strings]
Msft = "Microsoft"
DiskDescription = "Maso Filter Disk"

SFilter_Desc = "Maso Filter"
SFilter_HELP = "Maso Filter"
```

설치 프로그램 작성

NetCfg 예제를 바탕으로 이번에는 윈도우 프로그램을 작성해 본다. 우선 예제 프로그램의 구조부터 파악해보자.

NetCfg의 구조

NetCfg는 작성된 네트워크 드라이버를 설치하도록 해주는 콘솔 exe 프로그램이다. main.cpp에서 main 함수를 살펴보자. 실행 인자에 따라서 주어진 이름의 드라이버를 설치하거나 삭제할 수 있다. 우리는 그 가운데 설치 및 삭제 부분인 HrInstallNetComponent와 HrUninstallNetComponent를 가져다 쓰도록 한다. 함수 원형과 구현은 전부 snetcfg.h/cpp에 존재한다.

여기서 HrInstallNetComponent에 대해 짚고 넘어가 보자. 〈리스트 7〉은 그 소스 코드이다. 이를 살펴보면 네트워크 드라이버의 세 가지 타입인 프로토콜, 서비스, 클라이언트 타입에 대해 설치하도록 되어있음을 알 수 있다. 우선 inf 경로를 가져오고 INetCfg 인터페이스를 써서 inf에 따른 sys 드라이버를 설치하도록 되어 있다.

〈리스트 7〉 네트워크 드라이버 설치 함수 HrInstallNetComponent

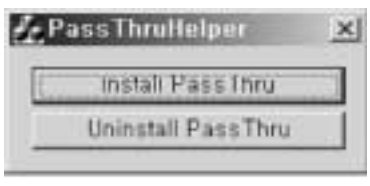
```
HRESULT HrInstallNetComponent(IN PCWSTR szComponentId,
                              IN enum NetClass nc,
                              IN PCWSTR szInfFullPath)
{
    HRESULT hr=S_OK;
    INetCfg* pnc;
    // cannot install net adapters this way. they have to
be
    // enumerated/detected and installed by PnP

    if ((nc == NC_NetProtocol) ||
        (nc == NC_NetService) ||
        (nc == NC_NetClient))
    {
        ShowMessage(L"Trying to install '%s'...",
szComponentId);

        // if full path to INF has been specified, the INF
        // needs to be copied using Setup API to ensure
that any other files
        // that the primary INF copies will be correctly
found by Setup API
        //
        if (szInfFullPath && wcslen(szInfFullPath))
        {
            WCHAR szInfNameAfterCopy[MAX_PATH+1];
            if (SetupCopyOEMInf(
                    (LPTSTR)szInfFullPath,
                    NULL,        // other files are in the
                                 // same dir. as primary
INF
                    SPOST_PATH,  // first param. contains
path to INF
                    0,           // default copy style
                    (LPTSTR)szInfNameAfterCopy, //
receives the name of the INF
                                 // after it is copied to
%windir%\inf
                    MAX_PATH,    // max buf. size for the
above
                    NULL,        // receives required size
if non-null
                    NULL))       // optionally retrieves
filename
                                 // component of
szInfNameAfterCopy
            {
                ShowMessage(L"...%s was copied to %s",
                            szInfFullPath,
                            szInfNameAfterCopy);
            }
            else
            {
```

```
                DWORD dwError = GetLastError();
                hr = HRESULT_FROM_WIN32(dwError);
            }
        }

        if (S_OK == hr)
        {
            // get INetCfg interface
            hr = HrGetINetCfg(TRUE, &pnc);

            if (SUCCEEDED(hr))
            {
                // install szComponentId : INetCfg 인터페이스
를 사용해 설치함
                hr = HrInstallNetComponent(pnc,
szComponentId,

c_aguidClass[nc]);
                if (SUCCEEDED(hr))
                {
                    // Apply the changes
                    hr = pnc->Apply();
                }

                // release INetCfg
                (void) HrReleaseINetCfg(TRUE, pnc);
            }
        }
        // show success/failure message
        ShowHrMessage(hr);
    }

    return hr;
}
```

PassThruHelper

PassThruHelper는 우리가 만든 PassThru에 대해 조작을 수행하는 UI 프로그램이다. 이번 호에서는 설치 기능만 추가한 간단한 UI를 만들지만, 다음에는 드라이버와 통신을 주고 받는 기능까지 구현할 예정이다.

HrInstallNetComponent와 HrUninstallNetComponent를 호출하는 간단한 프로그램을 만들어보면 아마도 〈화면 2〉처럼 아주 간단한 프로그램이 될 것이다. VC6를 띄워서 새 다이얼로그(Dialog) 프로그램을 만들고, snetcfg.h/cpp 파일을 추가한다. Include Path에 C:\NTDDK\INC,C:\NTDDK\INC\DDK를 추가하고, Link Path에 C:\NTDDK\LIBFRE\I386,C:\NTDDK\LIB\I386\FREE를 추가한다. 컴파일 인자로는 _WIN32_WINNT=0x500을 넣는다.



〈화면 2〉
PassThruHelper의 모습

설치

설치는 우리가 만들었던 NetCfg 프로그램을 써서 이뤄지거나, 직접 네트워크 카드에 설치함으로써 수행된다. 여기서는 두 가지 방법 모두를 설명한다.

네트워크 등록정보에서 직접 설치

'제어판'의 '네트워크 연결'을 선택한다. 설치하고자 하는 네트워크 카드를 클릭하고, 마우스 오른쪽 버튼을 눌러 '등록정보'를 클릭한다. '일반' 탭에서 '설치'를 누른다. 〈화면 3〉처럼 네트워크 구성 요소 선택 창이 뜨면 '서비스'를 선택하고 '추가'를 클릭한다. 네트워크 클라이언트 선택 창이 뜨면 '디스크 있음'을 눌러 netsf.inf가 있는 위치를 설정한다. 여기서 주의할 점은 netsf.inf, netsf_m.inf, PassThru.SYS가 같은 위치에 존재해야 한다는 사실이다. 한편 '서명되지 않은 드라이버'라는 경고가 나타나지만 이를 무시하고 '예'를 선택해 드라이버 설치를 시작한다. 그러면 〈화면 4〉처럼 드라이버 설치가 완료된 것을 볼 수 있다.

〈화면 3〉 네트워크 등록정보에서의 직접 설치

〈화면 4〉 드라이버 설치 완료

테스트는 드라이버를 설치한 후 네트워크 활동을 하면서 곧바로 수행할 수 있다. DbgPrint를 사용해 특정 조건에 대해 프린트하도록 만들었으므로 DebugView 프로그램으로 조건에 맞게 값이 나오는지를 확인할 수 있다. 웹 서핑을 할 때마다 해당하는 IP와 포트 등이 찍히면 성공한 것이다.

참고 자료

1. 『Windows 2000 디바이스 드라이버』 - 아트 베이커, 제리 로자노 공저
2. 『Microsoft Windows Internals』 - Mark E. Russinovich, David A. Solomon
3. NDIS.com (www.ndis.com)
4. The Windows Driver Developer's Digest (http://www.wd-3.com/)
5. KOSR (http://www.kosr.org/)
6. DriverOnLine (http://www.driveronline.org/)