네트워크 드라이버 프로그래밍
NDIS의 이해와 활용법

드라이버 프로그래밍에 대해 많은 개발자들이 어려움을 호소한다. 그중에서도 네트워크 드라이버 프로그래밍은 자료부족으로 더 높은 진입장벽을 느끼고 있다. 이번 호에서는 네트워크 드라이버 프로그래밍에 대한 기본사항에 대해 알아보자. 드라이버 프로그래밍을 하기 위한 기본 지식과 개발에 필요한 프로그램들을 설치해보자. 더불어 예제 프로그램에 대해서도 살펴보는 시간을 가져본다.

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

아직까지 국내에서 드라이버 개발은 널리 알려진 분야가 아니다. 커널을 다루고 디버깅을 해야하는 힘든 분야라는 인식이 강한 것 같다. 우선, 커널에 대한 기본적인 지식이 있어야 하는 것이 첫 번째 장벽이다. 기존에 사용하던 IDE(통합개발환경, 이하 IDE) 대신 프로그램을 손수 작성해 DDK 커맨드 창에서 빌드 하는 것이 두 번째 장벽이다. 세 번째 장벽은 설치 후 디버깅 할 때 WinDBG라는 프로그램을 사용해 심볼과 소스를 설정해준 후 값을 살펴봐야 하는 어려움이 있다. 기존의 IDE를 사용하던 개발자에겐 굉장히 불편한 방식이 아닐 수 없다. 개발시 〈화면 1〉같은 공포(?)의 블루스크린이 활성화 되면 거의 패닉 상태에 빠지게 된다. 이런 장벽들 때문에 아직도 개발자들이 쉽게 드라이버 프로그래밍에 접근하지 못하고 있다. 최근들어 국내 개발자가 집필한 윈도우 디바이스 드라이버 책이 나왔을 정도로 업계가 발전하고 있지만 쉽게 접근하지 못하는 것은 예전과 마찬가지다.

드라이버 개발 중에서도 특히 네트워크 드라이버 개발은 미개척분야이다. 네트워크 드라이버 개발은 일반 윈도우 드라이버와 구조부터 다르기 때문에 처음부터 다른 방향으로 접근해야 한다. 첫 단계인 커널과 드라이버에 대한 기본 지식을 배울 때부터 다르게 접근해야 한다. 문제는 그럼에도 불구하고 국내에 변변한 네트워크 드라이버 관련 사이트나 책이 한권도 없다는 점이다. 그렇다보니, 국내 개발자가 네트워크 드라이버 프로그래밍을 할 경우 윈도우 드라이버에 대한 공통적인 내용을 배우고, ndis.com이나 DDK와 관련된 영문서를 힘겹게 읽으며 네트워크 드라이버 개발에 접근하게 된다. 이번 연재에서 필자가 알고 있는 지식을 바탕으로 네트워크 드라이버에 대한 접근을 용이하게 해보고자 한다.

〈화면 1〉 드라이버 개발의 진입장벽 (공포의 블루스크린)

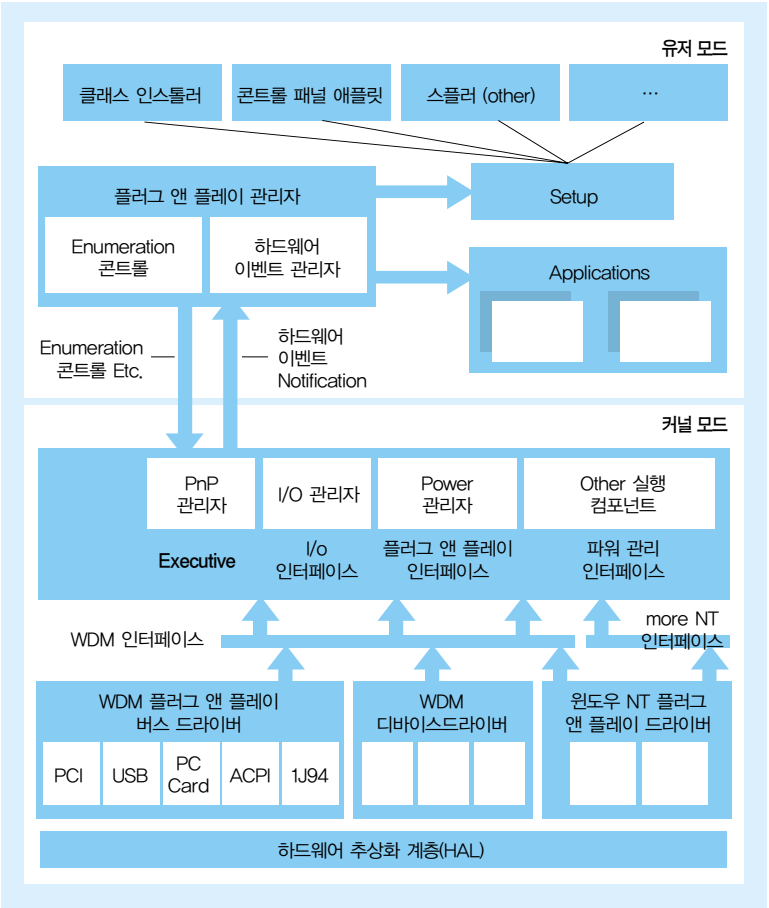
박상민 sangmin.park@gmail.com | 소만사(www.somansa.com)에서 웹분석 솔루션 설계 및 개발을 담당했다. 윈도우 기반 개발, 웹 개발, 데이터베이스 프로그래밍 등의 개발 기술을 경험해 왔고, 어떤 프로젝트가 주어지더라도 최상의 결과를 만들어내는 소프트웨어 아키텍트가 되는 것이 꿈이다. 인생을 편하게 사는 것을 목표로 삼고, 내일은 내일의 바람이 분다(tomowind.egloos.com)라는 블로그를 운영하고 있다.

네트워크 드라이버 이론

네트워크 드라이버를 만들기 위한 기본 이론들을 살펴보도록 하자. 우선, 기본적으로 드라이버 프로그래밍이 커널 모드 프로그래밍이기 때문에 커널모드와 유저모드에 대해서 간단하게 살펴보자. 더불어 네트워크 드라이버 개발 스펙인 NDIS(Network Driver Interface Specification, 이하 NDIS)에 대해서 살펴보도록 하자.

유저모드 vs 커널모드

〈그림 1〉에서 보는 것과 같이 윈도우는 크게 유저모드와 커널 모드로 나뉜다. 유저모드는 일반적으로 우리가 사용하는 모드다. 윈도우가 응용프로그램에 API를 제공하기 때문에, 응용프로그램이 파일접근이나 네트워킹 같은 하드웨어 관련 조작을 가능하게 한다. 커널 모드는 〈그림 1〉과 같이 시스템이 하드웨어에 접근하는데 사용한다. 〈그림 1〉서 보듯이 WDM Interface라는 것도 이 영역에 있다. 윈도우의 드라이버도 이 영역에 존재한다는 뜻이며, NDIS도 이 영역에 존재하고 있다.



〈그림 1〉 유저 모드 vs 커널모드

WDM vs NDIS

앞서 언급했듯이 네트워크 드라이버 개발은 일반 드라이버 개발과는 조금 다르다. 여기서는 기본 윈도우 드라이버와 네트워크 드라이버가 어떤 차이를 가지고 있는지 살펴보자. 사실, 윈도우 드라이버의 종류는 여러 가지가 있다. WDM(Windows Driver Model, 이하 WDM)드라이버, 비디오 드라이버, 프린터 드라이버, 멀티미디어 드라이버, 네트워크 드라이버 등이 있다. 이중 일반적인 드라이버를 만들때 사용하는 WDM 드라이버와 네트워크 드라이버를 만들기 위한 스펙인 NDIS에 대해서 비교해 보자.

● WDM

기존의 드라이버는 WDM이라는 모델을 사용한다. WDM을 사용하면 다음과 같은 특징을 얻을 수 있다.

- 플러그 앤 플레이 지원
- 전원관리
- 자동 설정(autoconfiguration)
- 핫 플러그 인 기능(Hot Plugability)

위와 같은 특징들을 지원 받을 수 있고, 쉽고 빠른 드라이버 개발이 가능하기 때문에 WDM을 사용한다. WDM이 사용되는 구조는 〈그림 2〉와 같다. 〈그림 2〉처럼 윈도우 응용프로그램들은 Win32 API를 통해 Win32 하부 시스템과 통신 한다. 그러면, Win32 하부시스템은 시스템 콜을 통해 I/O Manager에 접근이 가능하고, I/O Manager는 IRP(I/O Request Packet)라는 패킷 구조체를 통해 디바이스 드라이버와 통신이 가능하다. 여기서 주의할 점은 IRP라는 구조체가 드라이버 사이의 통신 및 제어에 사용된다는 점이다. 이것이 WDM을 나타내는 가장 큰 특징이다.

여기서 잠시 코드를 살펴보자. 〈리스트 1〉처럼 모든 드라이버는 DriverEntry라는 함수를 진입점으로 삼는다. 즉, 윈도우 프로그래밍에서 WinMain이 진입점인 것처럼 C 프로그래밍에서는 main함수가 진입점이다. WDM드라이버에서는 진입점에서 IRP들에 대한 등록을 해줘야 한다. 〈리스트 2〉를 살펴보자. WDM 드라이버에서는 진입점에서 IRP에 해당하는 처리 함수를 등록시켜 주도록 되어 있다.

〈리스트 1〉 DriverEntry함수(모든 드라이버의 진입점이다)

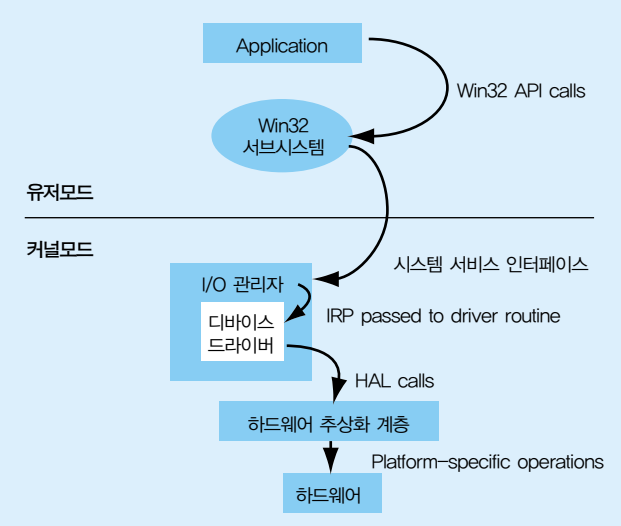
```
NTSTATUS
DriverEntry(
        IN  PDRIVER_OBJECT DriverObject,
        IN  PUNICODE_STRING         RegistryPath
        )
{
        NDIS_STATUS status=STATUS_SUCCESS;
        return status;
}
```

〈리스트 2〉 WDM드라이버의 DriverEntry함수
(IRP처리를 위한 진입점을 가지고 있다)

```
NTSTATUS
DriverEntry(
        IN  PDRIVER_OBJECT DriverObject,
        IN  PUNICODE_STRING         RegistryPath
        )
{
        NTSTATUS Status;
        UINT FuncIndex;

        // 진입점 초기화
        DriverObject->FastIoDispatch = NULL;

        //
        // IRP에 해당하는 Major 함수 등록
        // IMIoctl이라는 것은 함수포인터로서
        // IRP들에 대한 함수를 포워딩 해주는 역할을 한다.
        // 즉, IRP_MJ_XXX에 대해 다른 함수를 포워딩 한다.
        //
        for (FuncIndex = 0; FuncIndex <
IRP_MJ_SET_SECURITY; FuncIndex++) {
                DriverObject->MajorFunction[FuncIndex] =
IMIoctl;
        }

        // 드라이버에 디바이스 객체를 만든다.
        Status = IoCreateDevice(DriverObject, 0,
&IMDriverName,
           FILE_DEVICE_NETWORK, 0, FALSE,
&IMDeviceObject);
        ...
        return status;
}
```

마지막으로 WDM으로 작성 가능한 드라이버의 종류는 다음과 같다.

- 버스 드라이버 : IEEE 1394, USB
- Human Interface Devices : 키보드, 마우스, 게임 포트
- 배터리 디바이스 : 배터리, UPS
- Still 이미지 디바이스 : 디지털 카메라, 스캐너
- Streaming 디바이스 : 오디오, DVD, 비디오 캡쳐



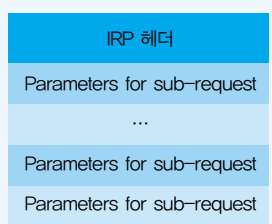
〈그림 2〉 WDM이 유저모드와 커널모드에서 호출되는 구조

● NDIS

NDIS는 Network Driver Interface Specification의 약자다. 3Com과 마이크로소프트가 1989년에 공동으로 만들었다. 초창

IRP (I/O Request Packet)

IRP (I/O Request Packet)는 WDM에서 드라이버간의 통신이나 제어 시 사용하는 패킷 구조체다. C 프로그램 관점에서 하나의 struct로 이루어진 구조체이다. 〈그림 3〉과 같은 구조로 되어있다.



〈그림 3〉 IRP 구조체

〈그림 3〉처럼 헤더와 하부요청으로 되어 있다. IRP 헤더 영역은 전체 I/O 요구에 대한 다양한 정보 부분을 지니고 있다. 이 헤더의 일부분은 드라이버에서 직접 접근할 수도 있는 반면, 다른 부분은 I/O관리자에 대해서 배타적인 성질을 갖는다. 예를 들어, 드라이버에서는 헤더의 IoStatus영역을 읽어 그 IRP가 어떤 상태인지 알아볼 수 있다.

하부요청의 주된 목적은 I/O요청의 함수 코드와 파라미터를 담는 것이다. 여기에 MajorFunction에 사용될 IRP_MJ_XXX를 보고 IRP가 수행할 함수를 찾는 것이다. 만약, 우리가 IRP를 조정한다고 하면 IRP_MJ_XXX와 함수를 만들어 드라이버를 제어 가능 하도록 만들 수 있는 것이다.

더 자세한 내용은 http://www.osronline.com/ddkx/kmarch/k112_3z5e.htm 에서 확인 할 수 있다.

기 개발 목적은 네트워크 카드 (NIC:Network Interface Card) 벤더에 상관없이 드라이버의 윗 단계에서 동일한 인터페이스를 통해 접근 가능하도록 고안되었다.

NDIS의 가장 큰 특징은 WDM에서 사용하는 IRP를 사용하지 않는다는 것이다. NDIS_PACKET이라는 구조체를 만들어 드라이버 간에 통신하도록 만들어졌다. 나중에 언급하겠지만, TDI라는 NDIS 윗 레벨에서 NDIS 라이브러리에 접근 할 때 NdisAllocatePacket 등의 함수를 사용해 NDIS 패킷을 만들어 통신이 가능하다.

물론, 그 외에도 NDIS와 WDM의 다른 점은 많다. 둘 다 마찬가지로 DriverEntry라는 초기 진입점을 갖지만 WDM은 진입점에서 IRP 처리 루틴을 많이 다루는 반면, NDIS는 그렇지 않다. 〈리스트 3〉을 살펴보자. NDIS중에서도 Intermediate 드라이버의 DriverEntry 예제이다. WDM 드라이버에서 보았던 IRP에 따른 함수 등록은 보이지 않는다. 대신 Miniport, Protocol 드라이버의 등록이 보인다. 즉, Intermediate 드라이버는 중간에서

필자 메모

드라이버 프로그래밍에 대해서는 아무리 쉽게 써도 초보자가 감을 잡기에는 어려운 것이 사실이다. 사실, 필자도 처음 개발시 어려움이 많았다. NDIS를 처음 접했던 것은 학교 프로젝트로 IP를 자동으로 할당해주는 프로그램을 개발할 때였다. 디버깅 방법을 몰라 WinDBG라는 것을 설치하고, 시리얼 케이블을 꽂아서 PC 두 대를 연결해 밤새 삽질을 하다 결국 컴퓨터를 연결 못해 디버깅을 못한 경험이 있다. 그래서, 파란화면이 나올 때마다 소스를 끔끔히 찾아보며 고치는 프로그래밍을 해야 했다.

독자들에게 그런 시행착오를 조금이나마 줄일 방법을 가르쳐 주고자 한다. 모든 업계의 일이 다 비슷하겠지만, 기술이 어려운 개발일수록 고수에게 직접 강의를 듣는 것만큼 좋은 것이 없다. 참고자료에 있는 KOSR이나 DriverOnLine 사이트를 참고하라. 그곳에서 열리는 세미나에 참석해 강사들이 직접 개발하고 디버깅하는 모습을 보면 드라이버 개발에 금방 익숙해 질 수 있을 것이다.

〈리스트 3〉 NDIS Intermediate Driver의 DriverEntry 함수

```
NTSTATUS
DriverEntry(
        IN   PDRIVER_OBJECT         DriverObject,
        IN   PUNICODE_STRING        RegistryPath
        )
{
        // 각종 구조체를 선언해준다.
        NDIS_STATUS                         Status;
        NDIS_PROTOCOL_CHARACTERISTICS       PChars;
        ...

        //
        // Miniport를 NDIS에 등록시켜준다.
        // Protocol 드라이버가 보게 되는 Miniport이다.
        // 따라서, 이곳에서 나가는 패킷들에 대한 조작을 해 줄 수 있다.
        //
        NdisMInitializeWrapper(&WrapperHandle,
DriverObject, RegistryPath, NULL);
        NdisZeroMemory(&MChars,
sizeof(NDIS_MINIPORT_CHARACTERISTICS));

        // NDIS 버전은 5.0이다.
        MChars.MajorNdisVersion = 5;
        MChars.MinorNdisVersion = 0;
        ...

        // Miniport 드라이버가 보게 될 Send 함수를 등록해준다.
        MChars.SendHandler = MPSend;

        Status =
NdisIMRegisterLayeredMiniport(WrapperHandle,
            &MChars, sizeof(MChars), &DriverHandle);
        ASSERT(Status == NDIS_STATUS_SUCCESS);

        NdisMRegisterUnloadHandler(WrapperHandle,
PtUnload);

        //
        // Protocol을 NDIS에 등록시켜준다.
        // Miniport 드라이버가 보게 되는 Protocol이다.
        // 따라서, 이곳에서 들어오는 패킷들에 대한 조작을
            해 줄 수 있다.
        //
        NdisZeroMemory(&PChars,
sizeof(NDIS_PROTOCOL_CHARACTERISTICS));

        // NDIS 버전은 5.0이다.
        PChars.MajorNdisVersion = 5;
        PChars.MinorNdisVersion = 0;

        ...

        // Protocol 드라이버가 보게 될 Receive 함수
        PChars.ReceiveHandler = PtReceive;

        NdisRegisterProtocol(&Status, &ProtHandle,
&PChars,
            sizeof(NDIS_PROTOCOL_CHARACTERISTICS));
        ASSERT(Status == NDIS_STATUS_SUCCESS);

        NdisIMAssociateMiniport(DriverHandle, ProtHandle);
        return status;
}
```

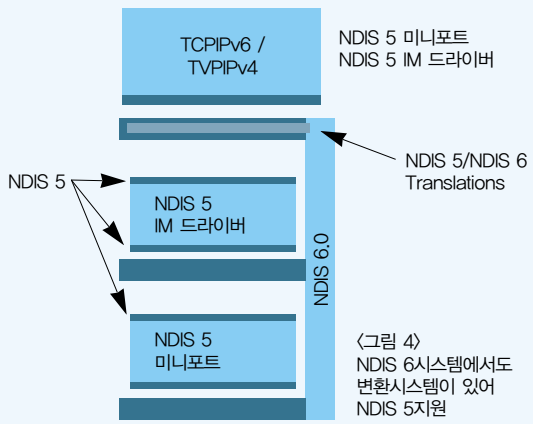
작용을 하기 때문에 위아래 레벨의 드라이버에게 투명하게 보이도록 하는 함수를 등록시켜주는 작용을 해주는 것이다.

여기서 사용하게 될 NDIS버전은 5.0이다. 〈리스트 3〉에서 NdisMajorVersion, NdisMinorVersion을 등록하는 것을 보면 5.0이라는 것을 알 수가 있다.

NDIS 버전

NDIS 버전에 대해 좀 더 자세히 알아보자. 윈도우 2000에서 사용되는 버전은 NDIS 5.0이고, 윈도우 XP, 윈도우 2003에서 사용되는 NDIS 버전은 5.1이다. 그리고 롱혼에서 사용되는 버전은 6.0이다.

최신 버전인 NDIS 6.0에 대해서 알아보자. 마이크로소프트 자료에 의하면 성능이 향상되었고, 확장성증가, 보안강화, 테스트 보강, 견고함이 특징이라고 한다. 실제 변화된 모습을 보면, IPv4와 IPv6를 전부 지원한다고 한다. 또한, NDIS 5.0, NDIS 6.0에 대한 변환계층이 있어 동시에 지원하며, NDIS 6.0도 차기 버전에서 구동 되도록 확장 가능한 설계를 한다고 한다.

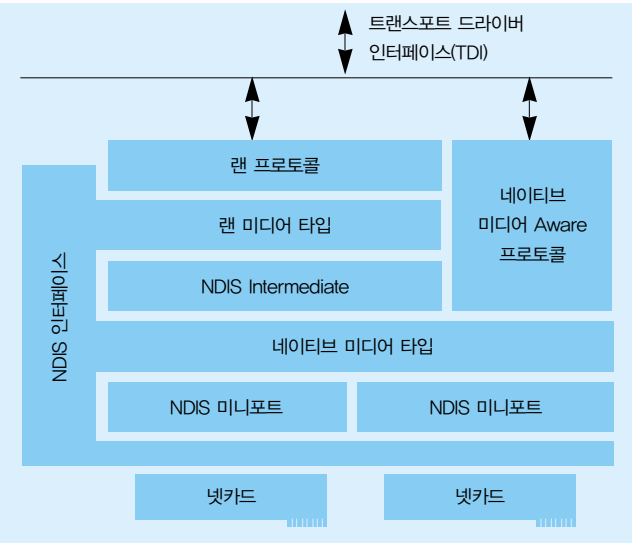


〈그림 4〉
NDIS 6시스템에서도
변환시스템이 있어
NDIS 5지원

관심이 있는 독자는 마이크로소프트 사이트(http://www.microsoft.com/whdc/device/network/ndis/default.mspx)에서 자료를 더 찾아볼 수 있다.

NDIS의 구조

〈그림 5〉에서 네트워크 드라이버의 레벨을 잘 보여주고 있다. 가장 아래에 NIC가 꽂혀있다. 그리고 크게 세 가지의 NDIS 드라이버가 있음을 알 수 있다. NIC 카드에서부터 Miniport 드라이버, Intermediate 드라이버, Transport 드라이버가 있다. 이 세 가지 드라이버가 NDIS 드라이버이며 윈도우에서는 NDIS.sys를 통해 이 드라이버들이 사용할 라이브러리를 제공해 준다. NDIS 드라이버는 윗 단계 응용프로그램과 TDI (Transport Driver Interface)라는 인터페이스를 사용해 통신이 가능하다(윈도우에서는 TDI.sys라는 파일을 통해 이 일을 담당한다). 이제 각각의 드라이버의 특징을 살펴보자.



〈그림 5〉 NDIS 드라이버의 계층구조

● Miniport 드라이버

Miniport 드라이버는 하드웨어와 상위 계층 간의 통신을 담당하는 드라이버다. 하드웨어 의존적인 드라이버이기 때문에, 대부분 NIC 제조업체에서 제공한다. 별도로 이쪽 부분의 드라이버를 다룰 일은 거의 없다고 보면 된다. 기능은 NIC을 통한 데이터 송/수신이 가능하도록 한다.

● Intermediate 드라이버

Intermediate 드라이버는 Miniport와 Protocol의 중간에 놓여 둘을 연결하는 역할을 한다. 따라서 Miniport 드라이버에게는 자신이 Protocol 드라이버인 것처럼 보이게 하고, Protocol 드라이버에게는 Miniport 드라이버인 것처럼 보이게 하면서, 자신의 로직을 수행할 수 있도록 한다. 이 부분에서 할 수 있는 역할은 필터링, 모니터링, 그리고 이기종 하드웨어에 대한 Multiplexing, Demultiplexing 등을 할 수 있다. 일반적으로 NDIS 드라이버를 만든다고 하면 보통 Intermediate, Protocol 드라이버를 만든다고 보면 된다. 이 부분의 드라이버를 사용하는 예제로 뒤에서 설명할 Network Monitor 등이 있다.

● Protocol 드라이버

Protocol 드라이버는 NDIS의 윗 단계에서 하드웨어 벤더에 종속적이지 않은 역할을 한다. TCP/IP, NETBEUI와 같은 프로토콜의 인터페이스를 정의하는 사양이라고 보면 된다. Protocol 드라이버도 Intermediate 드라이버처럼 패킷 필터링, 미러링 등에 자주 사용된다. 이 부분의 드라이버 예제는 상용인 PCAUSA

드라이버(http://www.pcausa.com)를 들 수가 있다. 응용프로그램에게 API를 노출시켜 패킷 모니터링을 할 수 있도록 하는 인터페이스를 Protocol 드라이버 레벨에 구현한 RawEther라는 제품이 있다.

네트워크 드라이버 관련 툴들

여기서는 NDIS에 관련된 툴들을 알아본다. 다음 호부터 사용하는 개발도구들을 알아보고, NDIS와 관련된 도구도 살펴보자. 물론, 비주얼 스튜디오는 기본적으로 설치해야 한다.

DDK

드라이버를 개발하기 위해서는 어떤 라이브러리를 설치해야 할까? DDK(Driver Development Kit)라고 하는 마이크로소프트에서 제공하는 라이브러리를 사용해야 한다. 드라이버는 윈도우 버전마다 다르기 때문에, 개발하는 드라이버에 따라서 DDK 버전도 달리 설치해야 한다. 여기서는 윈도우 2000 드라이버를 사용하기 때문에 윈도우 2000용 DDK를 설치하겠다. DDK는 마이크로소프트 사이트(www.microsoft.com/whdc/devtools/ddk/orderddkcd.mspx)에서 CD를 신청해 받은 후 설치하도록 한다. 기본적으로 C:\NTDDK라는 곳에 DDK 라이브러리가 설치 된다. 드라이버 프로그래밍 환경을 위해 이 곳에 include file, link 등을 걸어주면 된다.

DDK를 설치하면 각종 예제 드라이버 프로그램이 생성된다. 일정기간 드라이버 개발을 하다보면 책을 한권 독파하고 스스로 작업하는 단계에 이르게 된다. 그 때 이 예제 드라이버 프로그램을 많이 활용하게 된다.

WinDBG

드라이버 개발시 가장 많이 사용하는 디버거는 크게 두 가지가 있다. WinDBG라는 마이크로소프트에서 제공하는 커널 디버거가 있고, 누메가라는 회사에서 제공하는 상용 디버거인 SoftICE가 있다. 드라이버 디버깅을 위해서는 둘 중 하나의 프로그램은 익숙하게 다룰 수 있어야 한다. 그래야, 블루 스크린의 공포에서도 프로그래밍이 가능하다. 기존엔 상업용 디버거인 SoftICE가 WinDBG보다 강력한 기능을 지원했기 때문에, 개발자들이 두 개의 디버거로 나뉘어 사용하는 경향을 보였지만, 요즘은 WinDBG의 기능도 SoftICE 못지않게 강력해져 WinDBG를 더 많이 사용하는 추세다. 여기서는 마이크로소프트에서 무료로 제공해주는 WinDBG를 소개한다.

WinDBG는 DDK 설치시 자동으로 설치된다. 하지만, 마이크로소프트 사이트(http://www.microsoft.com/whdc/ddk/debugging)에서 최신버전을 다운로드 받도록 하자. WinDBG는 커널 디버깅과 애플리케이션 디버깅이 모두 가능하며, 덤프파일 분석도 가능하다. 우선, 처음에 기본 셋팅을 먼저 해야 한다. 여기서는 커널에 대한 디버깅을 하므로, 커널 소스에 대한 심볼파일(Symbol File)이 필요하다. File -> SymbolFile Path에 들어가 심볼 파일 경로를 설정해 준다. 다음과 같이 윈도우 심볼을 받아서 컴퓨터에 저장하도록 한다.

```
SRV*C:\OsSymbol\WebSymbol*http://msdl.microsoft.com/download/symbols
```

위와 같이 웹에서 받은 심볼을 사용해 디버깅이 가능하다. 처음 다운로드시 시간이 오래 걸리지만, 설치 후엔 변경된 내용만 받기 때문에 시간이 많이 걸리지 않는다.

굳이 드라이버 개발이 아니어도 관련 명령어 몇 개를 사용하면 윈도우 커널에 대한 이해의 폭을 넓힐 수 있다. !process라는 명령어를 사용해 프로세스에 대한 정보를 보거나 !thread라는 명령어를 사용해서 스레드에 대한 정보를 볼 수가 있다. 기타 관련 명령어들은 WinDBG 도움말을 참조해서 보도록 한다.

DebugView

앞에서 대표적인 커널 디버거 두 가지를 언급했다. 하지만, 실제로 이번 연재에서 가장 많이 쓸 프로그램은 DebugView이다. DebugView는 sysinternals라는 회사에서 제공하는 프리웨어다. 홈페이지(http://www.sysinternals.com/Utilities/DebugView.html)에서 다운로드 받을 수 있다. 내 컴퓨터에서 출력되고 있는 디버그 문자열들을 보여주는 기능을 지녔다. 이 기능으로 인해 응용프로그램 레벨에서 찍어주는 디버그 문자열이나 커널모드 레벨에서 찍어주는 디버그 문자열을 모두 볼 수 있다. DebugView를 통해서 내 컴퓨터에 들어오는 패킷들을 보기로 하자.

〈화면 2〉 DebugView에서 디버깅 문자열들을 보는 모습

Network Monitor

이번 드라이버 개발을 위해서 사용되는 프로그램은 아니다. 하

지만, NDIS Intermediate 드라이버의 예제를 볼 수 있는 프로그램으로써, NDIS 계층을 이해하는데 도움을 준다. 또한, 네트워크 패킷을 실제로 봄으로써 네트워크에 대한 이해도를 높여줄 수 있는 프로그램이기도 하다.

Network Monitor는 네트워크 패킷을 잡아 보여주는 프로그램이다. 일반적으로 네트워크에 지나다니는 패킷은 볼 수가 없고, 응용프로그램에서 최종적으로 패킷이 정리가 되어 사용자가 알아볼 수 있는 사용자 정보만 볼 수가 있다. 예를 들어, 인터넷 익스플로러에서 클릭 한다고 하면 많은 패킷이 날아다니며 정보를 주고받게 된다. 여기서 보는 것은 페이지 하나일 뿐이다. 하지만, Network Monitor와 같은 프로그램을 사용하면, 내 컴퓨터의 랜카드에 물려있는 모든 패킷을 전부 모니터링 할 수 있다.

Network Monitor는 마이크로소프트에서 제공하는 프로그램이다. 설치는 마이크로소프트의 SMS패키지를 설치하면 볼 수 있다. 또한, 제어판 -> 프로그램 추가/제거 -> 윈도우 구성요소 추가/제거에서 설치할 수도 있다. 관리 및 모니터링 도구의 '자세히' 버튼을 눌러서 네트워크 모니터를 설치하도록 한다(윈도우 XP Home에서는 Network Monitor 설치를 지원하지 않는다).

설치 후 바탕화면의 '내 네트워크 환경'에서 오른쪽 버튼을 클릭해 속성을 연다. 그리고 사용하고 있는 네트워크 카드(일반적으로 '로컬 연결 영역'일 것이다)를 선택해 오른쪽 버튼을 눌러 속성을 연다. 그러면 〈화면 3〉처럼 네트워크 카드의 등록정보에 네트워크 모니터 드라이버가 설치된 모습을 볼 수 있을 것이다. 독자들도 Intermediate 드라이버를 작성해 설치하면 〈화면 3〉처럼 드라이버를 등록 할 수 있다.

〈화면 3〉 Network Monitor의 Intermediate Driver가 설치된 모습

그러면, 이제 실제로 패킷을 잡아보도록 하자. Network Monitor를 활성화 시켜 메뉴에서 ▶ 버튼을 눌러 패킷 수집을 시작한다. 눌러놓고 인터넷 익스플로러를 통해 웹서핑을 한참 하고 ■ 옆에 있는 ■와 안경이 있는 버튼을 눌러 수집한 패킷을 보도록 하자. 그러면 〈화면 4〉에서 처럼 잡힌 패킷들을 볼 수 있다. 필자는 네트워크 수집을 하도록 하고 인터넷 익스플로러를 사용해 웹서핑을 했기 때문에 TCP, HTTP등의 패킷이 수집된 걸 볼 수 있다.

〈화면 4〉 Network Monitor에서 패킷을 잡은 모습

네트워크 패킷 수집 모듈은 Network Monitor만 있는 것이 아니다. Ethereal, Network Sniffer 등의 여러 가지 상용/무료 제품이 있다. 각각의 제품이 패킷을 끌어올리는 레벨도 다르다. Network Monitor는 Intermediate 드라이버 레벨에서 패킷을 수집하고, Protocol 드라이버, Trasnport 드라이버 레벨에서 TCP 패킷만 잡아 올리는 제품도 있으며, 브라우저에 내장되어 응용프로그램에 도착하는 패킷들만 수집하는 제품도 있다. 예를 들어, 파이어폭스 유저는 livehttpheaders(http://livehttpheaders.mozdev.org/)라는 확장기능을 사용해 파이어폭스가 주고받는 패킷들을 볼 수가 있다. 이런 여러 가지 제품들을 사용하면 네트워크에 대한 이해와 윈도우 네트워크 레벨에 대한 이해가 더 커질 것이다.

TDI Mon

이것 또한 sysinternal에서 제공하는 무료 소프트웨어다. 홈페이지(http://www.sysinternals.com/Utilities/TdiMon.html)에서 다운로드 할 수 있다. Transport 레벨(TDI)에서 TCP, UDP 패킷을 수집해 보여주는 프로그램이다. 이는 패킷이 사용되는 프로그램과 패킷의 프로토콜을 모니터링 할 수 있도록 해준다.

File Mon

sysinternal 홈페이지(http://www.sysinternals.com/Utilities/Filemon.html)에서 다운로드 받을 수 있다. 실시간으로 파일시스템이 주고받는 액션들을 볼 수 있도록 해주는 툴이다. 옵션에 가보면 Network, MailSlot 등의 옵션을 정해줄 수 있어 네트워크를 사용하는 프로그램에 대한 액션을 볼 수가 있다. 또한, 옵션에서 Advanced Output을 주면 실제 요청되는 IRP가

어떤 식으로 수행되는지 볼 수 있다.

개발 할 내용

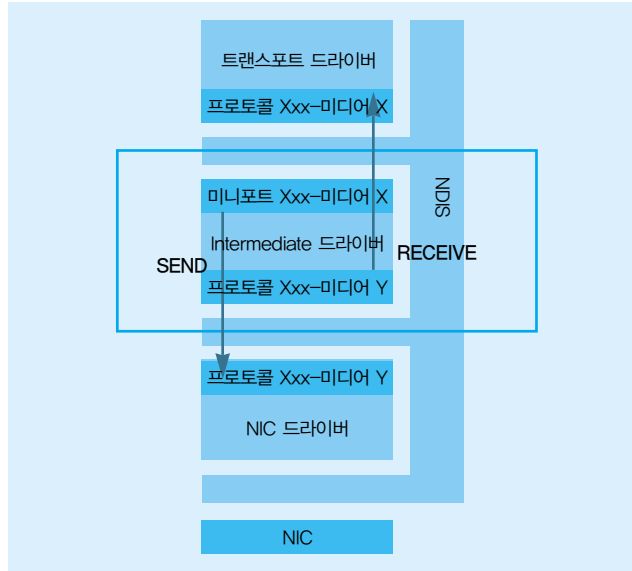
필자는 개인적으로 NDIS Intermediate 드라이버, Protocol 드라이버를 전부 사용해 보았다. 우선, 앞에서 언급한 PCAUSA라는 회사의 모니터링 드라이버를 사용한 프로그래밍을 담당했었다. 패킷 모니터링 드라이버를 사용해서 패킷을 끌어올려 분석하고 대처하는 보안 프로그램을 작성했었다. 당시 문제는 패킷을 모니터링 할 때 모든 패킷을 다 분석하지 않고 필터링 해야 하는 상황에 있었다. PCAUSA 드라이버는 응용프로그램에 API를 제공해주기 때문에, 응용프로그램에서 패킷을 끌어올리고 나서 필터링 하는 게 일반적인 방법이었다. 하지만, 네트워크 트래픽이 많을 경우 PCAUSA API를 사용해 패킷을 끌어올리기까지 CPU와 메모리 부하가 상당했다. 따라서 Intermediate 드라이버를 직접 개발해 적용시켰다. 원리는 Intermediate 드라이버 레벨에서 패킷을 필터링해 Protocol 레벨까지 패킷이 올라가지 않게 하는 것이었다. 이로 인해 줄어드는 부하는 상당했다. 필자의 프로그램이 깔린 한 고객사에서는 필터링 부하 감소로 인한 CPU 사용량이 90%->20%로 줄어든 적도 있었다(이런 이유로 드라이버 개발을 하게 되는 것이다).

그래서 이번 연재에서는 NDIS Intermediate 드라이버를 작성하고 제어하는 프로그램을 만드는 것을 목적으로 한다.

네트워크 필터링 드라이버

다음 연재에서 하게 될 내용이다. <그림 4>가 그 구조를 잘 나타내주고 있다. Imtermediate 드라이버는 Miniport 드라이버로부터 패킷을 받아 Protocol 드라이버에 전해주고, Protocol 드라이버로부터 패킷을 받아 Miniport 드라이버로 전송해 준다. 그 부분들은 Receive, Send라는 표시가 되어 있고, 드라이버 내에 함수가 해당하는 함수들이 있다. 여기서는 Receive 함수를 조작해 패킷을 필터링 하는 것을 목적으로 한다.

코드를 간단히 살펴보기로 하자. <리스트 3>에서 NDIS Intermediate 드라이버의 DriverEntry 함수를 살펴봤다. 이 함수에서 MpSend와 PtReceive 두 가지 함수에 주목해야 한다. MpSend는 Protocol 드라이버가 Miniport 드라이버에게 패킷을 전송할 때에 가로채는 함수이다. 그리고 PtReceive는 Miniport 드라이버가 Protocol 드라이버에게 받은 패킷을 올려줄때에 가로채는 함수이다. DriverEntry에서는 이 부분을 등록하는 부분만 있지만, 다음 호에서 이 함수들을 직접 조작해서 필터링 드라이버를 만들 것이다.



<그림 6> 사용하게 될 Intermediate 드라이버의 구조

드라이버 제어 프로그램

보통 드라이버는 한번 설치되면 그대로 기능을 다하도록 되어 있다. 하지만, 필터링 드라이버의 경우는 때때로 필터링 옵션을 드라이버에 넘겨줄 경우가 있다. 따라서 드라이버와 통신하는 응용프로그램도 하나 제작할 것이다.

앞에서 말했듯이 WDM과 NDIS는 구조가 다르다. WDM에서는 드라이버를 제어하거나, 드라이버간 통신에 IRP를 사용했고, NDIS에서는 IRP를 사용하지 않았다. 기본적으로 제공되지 않지만 IRP를 다루는 부분을 추가해 응용프로그램과 IRP통신을 통한 필터링 제어 문자열을 주고받는 역할을 하도록 구현해 보자.

사용할 예제 프로그램들

드라이버 개발은 구조를 익히고 예제 프로그램들을 잘 사용하는 것부터 시작된다. 여기서 만들 프로그램도 예제 프로그램의 수정을 통해 구조를 익히는 것이 목적임으로 예제 프로그램을 잘 분석해야 한다. 사용할 예제 프로그램들은 아래와 같다.

254쪽에서 계속

정리 | 문경수 objectfinder@imaso.co.kr

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/)