작성: NCHOVY 양봉열 (2008년 12월)  
문서에 오류가 있으면 xeraph@nchovy.kr로 알려주시기 바랍니다.  
  
NDIS가 알고 보면 그리 어렵지 않은데도 불구하고 한글 문서가 없어서 고생하는 경우가 많습니다. 지금 MSDN을 보면 아주 상세하게 잘 정리된 것으로 보이지만, 처음에는 어디서부터 어떤 식으로 접근해야 하는지 알기가 어려웠습니다. 몇 가지 자료와 핵심 설명만 있으면 훨씬 수월하게 접근할 수 있으리라 생각합니다.  
  
**기본 NDIS 드라이버 분류**  
  
아래로 갈수록 낮은 계층입니다.

* **프로토콜 드라이버**: TCP/IP나 IPX/SPX 같은 전송 프로토콜 스택을 구현하는 네트워크 드라이버입니다. 상위의 TDI 계층과 맞닿아 있습니다. 응용 프로그램이 보낸 데이터를 패킷으로 만들어서 하위 네트워크 계층으로 내려보내고, 하위 네트워크 계층에서 보내주는 수신 패킷을 받아 올리기도 합니다. 인터미디엇이나 미니포트 드라이버가 하위 계층에 해당됩니다. IDS처럼 패킷 모니터링만 할 용도라면 프로토콜 드라이버를 구현하면 됩니다. 프로토콜 드라이버로 구현하면 NIC의 종류에 상관없이 독립적으로 동작할 수 있다는 장점이 있습니다. 이것은 인터미디엇 드라이버도 마찬가지입니다.
* **인터미디엇 드라이버**: 프로토콜 드라이버와 미니포트 드라이버 사이에 위치하는 드라이버입니다. 중간에 껴있는만큼 위아래 네트워크 드라이버와 모두 통신해야 합니다. 중간에 껴있기 때문에 패킷을 임의로 가공하거나 버릴 수 있습니다. 패킷 필터링할 때 애용됩니다. 여러 개의 NIC를 묶어서 로드밸런싱하는 기능을 구현할 때도 사용되고, 여러가지 용도가 있을 수 있습니다.
* **미니포트 드라이버**: 우리가 일반적으로 보는 미니포트 드라이버는 이더넷 드라이버입니다. 토큰 링 드라이버 같은 것도 있겠지만 볼 일 없겠죠. 미니포트 드라이버는 직접 NIC 하드웨어를 제어합니다. 이더넷 카드이니만큼 원래 TCP나 IP 같은 것은 모르는게 정상입니다. 여기서 볼 수 있는 것은 오로지 데이터링크(MAC)와 물리(PHY) 계층 뿐입니다. 다만 요즘에는 성능 향상을 위해서 TCP 패킷이나 IP 패킷의 체크섬 오프로딩, TCP 세그멘테이션 등을 하드웨어 수준에서 지원하는 경우가 많습니다. PHY도 지원하는 물리 매체에 따라 종류가 나뉘어집니다. 매체가 구리선일 수도 있고 광섬유일 수도 있으니까요.

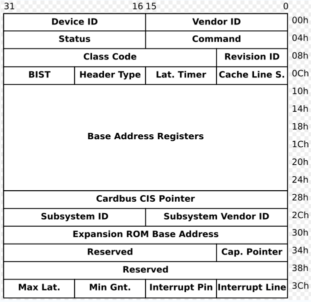
**예제 코드**  
  
일단 예제 코드가 있어야 전체적인 흐름을 볼 수 있습니다. [WDK](http://www.microsoft.com/whdc/devtools/wdk/default.mspx)를 설치하면 NDIS 예제 코드가 같이 설치되는데 이 코드들을 참조하면 됩니다. 다운로드는 MSDN 서브스크립션을 이용하거나 MS 커넥트 웹사이트를 이용하시면 됩니다. 기본값 그대로 설치했다면 C:\WinDDK\6001.18001\src\network\ndis 위치에서 NDIS 예제를 찾아볼 수 있습니다. 각 소스 폴더에 HTM 파일이 하나씩 있는데 예제 코드에 관련된 정보를 비교적 상세하게 기술하고 있습니다.  
  
이 중에서 e100bex가 인텔 EtherExpress PRO/100+ 이더넷 PCI 어댑터와 EtherExpress PRO/100B PCI 어댑터를 지원하는 미니포트 드라이버 예제 코드입니다. 해당 이더넷 카드를 구입해서 설치하고 직접 테스트를 해볼 수 있을 것입니다. 그 외에 인터미디엇 드라이버는 passthru 예제를 참고하면 되고, 프로토콜 드라이버는 ndisprot 예제를 참고하시면 됩니다. 이 글에서는 미니포트 드라이버, NDIS 5.0 버전으로 범위를 한정해서 설명합니다.  
  
**NDIS 미니포트 드라이버 구조**  
  
NDIS는 윈도우 네트워크 드라이버 전용으로 만들어진 프레임워크라고 볼 수 있습니다. 그냥 콜백을 잘 끼워맞추기만 하면 됩니다. 콜백의 기능, 전체적인 콜백 호출 흐름, 각 콜백 안에서 호출해야 할 API들을 파악했다면 NDIS의 절반 이상을 이해한 것입니다. 나머지는 예제 코드를 보면서 상세하게 자신이 이해하고 있는 바를 검증하면 됩니다. 이제 드라이버 진입점부터 하나씩 개요를 설명합니다.  
  
**DriverEntry**  
  
[NdisMInitializeWrapper](http://msdn.microsoft.com/en-us/library/ms804169.aspx)를 호출해서 초기화 과정에서 사용할 NDIS 핸들을 얻는 것으로 드라이버 초기화 루틴을 시작합니다. NdisMRegisterMiniport를 호출해서 미니포트 드라이버를 등록하는게 드라이버 초기화 루틴에서 해야 할 일입니다. 미니포트 드라이버를 등록하면서 NDIS wrapper가 호출할 콜백을 등록해줘야 합니다. 콜백 정보는 [NDIS\_MINIPORT\_CHARACTERISTICS](http://msdn.microsoft.com/en-us/library/aa448041.aspx) 구조체에 담습니다. NDIS 버전에 따라 약간씩 지원되는 콜백이 다릅니다. 5.0에서 꼭 필요한 콜백은 아래와 같습니다.  
  
여기서 관례(convention)을 이해하셔야 합니다. 모든 미니포트 콜백 함수는 MiniportXxx의 형태로 문서화되어 있습니다. 모든 NDIS 함수는 NdisXxx의 이름을 가지고 있습니다. 상위 계층에서 NdisXxx 함수를 호출했을 때 그에 대응하는 MiniportXxx 콜백이 호출되는 경우가 있으니 이런 경우 짝을 맞춰서 알아두셔야 합니다.

* InitializeHandler ([MiniportInitialize](http://msdn.microsoft.com/en-us/library/aa932004.aspx)): 네트워크 어댑터가 활성화 되는 경우 이 콜백이 호출됩니다.
* HaltHandler ([MiniportHalt](http://msdn.microsoft.com/en-us/library/aa931103.aspx)): 네트워크 어댑터를 비활성화 하는 경우 이 콜백이 호출됩니다.
* QueryInformationHandler ([MiniportQueryInformation](http://msdn.microsoft.com/en-us/library/aa447849.aspx)): 네트워크 어댑터가 초기화 된 직후부터 네트워크 어댑터 정보를 조회하게 되는데 그 때 이 콜백이 호출됩니다.
* SetInformationHandler ([MiniportSetInformation](http://msdn.microsoft.com/en-us/library/aa931844.aspx)): 네트워크 어댑터의 설정값을 조작해야 되는 경우 이 콜백이 호출됩니다.
* CheckForHangHandler ([MiniportCheckForHang](http://msdn.microsoft.com/en-us/library/ms797251.aspx)): NDIS wrapper는 주기적으로 (기본값 4초) 이 콜백을 호출하면서 어댑터가 정상적으로 동작하고 있는지 점검합니다.
* ResetHandler ([MiniportReset](http://msdn.microsoft.com/en-us/library/ms797254.aspx)): 만약 Hang 점검에서 오류 값을 반환하면, 이 콜백이 호출되면서 어댑터를 리셋하게 됩니다.
* ReturnPacketHandler ([MiniportReturnPacket](http://msdn.microsoft.com/en-us/library/ms797263.aspx)): 상위 계층으로 올려보낸 패킷 버퍼가 이 콜백을 통해 되돌아옵니다.
* SendPacketsHandler ([MiniportSendPackets](http://msdn.microsoft.com/en-us/library/ms797260.aspx)): 상위 계층의 패킷 전송 요청이 이 콜백을 통해 들어옵니다.
* ISRHandler ([MiniportIsr](http://msdn.microsoft.com/en-us/library/ms797244.aspx)): 인터럽트 서비스 루틴입니다. 수신/송신/링크 상태 확인 등 각종 인터럽트를 처리하게 됩니다. 인터럽트 처리는 DIRQL에서 이루어지므로 최대한 빠른 시간 안에 작업을 완료하는 것이 원칙입니다. 따라서 아주 간단한 작업이 아닌 이상, DPC를 이용해서 처리하도록 작업을 큐에 밀어넣습니다. 간단히 매개변수 값만 조작하면 NDIS wrapper가 알아서 DPC를 호출해줍니다.
* HandleInterruptHandler ([MiniportHandleInterrupt](http://msdn.microsoft.com/en-us/library/ms797243.aspx)): 인터럽트 서비스 루틴에서 예약했던 작업을 DISPATCH IRQL 수준에서 처리합니다. DPC는 하드웨어 인터럽트가 아닌 이상 방해받지 않으므로 처리 지연을 걱정하지 않아도 됩니다.

**MiniportInitialize**  
  
위에서 언급했던대로 네트워크 어댑터가 활성화되면 이 콜백이 호출됩니다. 매개변수부터 살펴봅시다.

* [출력] PNDIS\_STATUS OpenErrorStatus: 장치 초기화 성공 여부
* [출력] PUINT SelectedMediumIndex: 선택한 매체 번호 (보통 802.3)
* [입력] PNDIS\_MEDIUM MediumArray: NDIS에서 내려주는 매체 목록
* [입력] UINT MediumArraySize: 매체 목록 길이
* [입력] NDIS\_HANDLE MiniportAdapterHandle: 어댑터 핸들 **(중요)**
* [입력] NDIS\_HANDLE WrapperConfigurationContext: **초기화 과정에서만** 사용되는 핸들

유선 이더넷의 경우 NdisMedium802\_3을 선택하면 됩니다. 그 다음 해야될 일은 디바이스 컨텍스트를 생성하는 일입니다. 커널 메모리 할당 함수를 이용해서 Non-Paged 풀에서 메모리를 할당해서 사용하면 됩니다. 디바이스 컨텍스트에 NDIS 어댑터 핸들을 저장해두어야 합니다. 앞으로 마르고 닳도록 사용하게 됩니다.   
  
이렇게 만들어진 디바이스 컨텍스트를 [NdisMSetAttributesEx](http://msdn.microsoft.com/en-us/library/ms804212.aspx) 함수를 호출해서 설정합니다. 이 함수를 호출해줘야 이후 미니포트 콜백에 디바이스 컨텍스트를 전달할 수 있습니다. 이 함수 호출할 때 Hang 점검 주기도 설정할 수 있습니다. 하지만 이 함수의 가장 중요한 기능은 어댑터 속성 설정입니다. NIC가 DMA 전송을 지원하는 경우 NDIS\_ATTRIBUTE\_BUS\_MASTER 속성을 설정해야 하고, 자체적으로 동기화 처리를 해서 성능을 극대화하려는 경우 NDIS\_ATTRIBUTE\_DESERIALIZE 속성을 설정해야 합니다. 이 2가지가 중요하고 나머지는 공식 문서를 참조하시기 바랍니다.  
  
위 매개변수 중에 초기화 과정에서만 사용되는 핸들(WrapperConfigurationContext)이 있는데, 네트워크 어댑터와 관련된 레지스트리를 조회할 때 사용합니다. [NdisOpenConfiguration](http://msdn.microsoft.com/en-us/library/ms804265.aspx), [NdisReadNetworkAddress](http://msdn.microsoft.com/en-us/library/bb648482.aspx), [NdisCloseConfiguration](http://msdn.microsoft.com/en-us/library/bb314364.aspx)이 레지스트리 조회와 관련된 함수들입니다. 원래 NIC는 EEPROM 같은 영구 저장 장치에 고유의 MAC 주소를 저장한 상태로 출하됩니다. 그렇지만 보통 MAC 주소 설정을 변경할 수 있으므로, 레지스트리에 임의의 MAC 주소를 저장해놓고 불러다 쓸 수 있도록 API를 지원하는 것입니다. 일반적으로 MAC 주소가 유효한 경우 (멀티캐스트나 브로드캐스트 주소가 아닌 유효한 주소) 기본값 대신 지정된 MAC 주소를 사용하도록 구현합니다.  
  
다음 작업은 DMA 초기화입니다. [NdisMInitializeScatterGatherDma](http://msdn.microsoft.com/en-us/library/ms804181.aspx) 함수를 이용합니다. 이 때 NIC가 64비트 주소를 지원하는지 알아봐야 합니다. 64비트 주소 지원 여부와 더불어 한 번에 DMA 가능한 최대 크기도 지정해야 합니다. 점보 프레임이나 TCP 세그멘테이션 오프로딩을 사용하지 않는다면 그냥 이더넷 프레임 크기(1514)로 지정하면 됩니다.  
  
이제 NIC의 각종 레지스터와 포트를 메모리에 매핑할 차례입니다. 어느 주소를 매핑할지 알아내야 하는데, PCI처럼 플러그 앤 플레이를 지원하는 경우 PCI 슬롯 정보를 읽어서 주소를 비롯하여 각종 하드웨어 정보를 알아낼 수 있습니다. 아래 PCI 설정 영역 배치도를 보면 더 쉽게 이해할 수 있습니다. MSDN에서는 [PCI\_COMMON\_CONFIG](http://msdn.microsoft.com/en-us/library/aa491637.aspx) 구조체를 참조하시면 됩니다.



Device ID, Vendor ID, Revision ID는 장치를 식별하는데 가장 중요한 정보입니다. 특히 인텔에서 제조한 모든 PCI 장치의 Vendor ID는 8086으로 지정되어 있습니다. 모델과 버전에 따라 미묘하게 버그가 수정되거나, 레지스터의 동작이 변경되는 경우가 있으므로 이를 주의 깊게 살펴보고 반영해야 합니다. Command는 PCI 장치 제어를 담당합니다. Command 비트를 변경하면 관련 기능이 비활성화 됩니다. Base Address Registers 부분에 레지스터나 포트 접근에 필요한 물리 주소가 저장됩니다.  
  
[NdisMQueryAdapterResources](http://msdn.microsoft.com/en-us/library/ms804195.aspx) 함수를 호출해서 이런 하드웨어 리소스 정보들을 가져올 수 있습니다. 리소스 목록을 루프 돌리면서 타입에 따라 (포트, 메모리, 인터럽트) 관련 정보를 저장해놓고 이후 초기화 과정에서 사용합니다. 인터럽트의 경우 인터럽트 레벨이 나오고, 포트나 메모리의 경우 시작 주소와 길이를 얻을 수 있습니다. [NdisMMapIoSpace](http://msdn.microsoft.com/en-us/library/bb314301.aspx) 함수는 보드의 물리 메모리나 레지스터들을 매핑할 때 사용하고, [NdisMRegisterIoPortRange](http://msdn.microsoft.com/en-us/library/bb314324.aspx) 함수는 포트 영역을 매핑할 때 사용합니다. 메모리 영역은 [READ\_REGISTER\_ULONG](http://msdn.microsoft.com/en-us/library/aa490440.aspx) 매크로와 [WRITE\_REGISTER\_ULONG](http://msdn.microsoft.com/en-us/library/aa490208.aspx) 매크로 등(타입별로 존재함)을 이용하여 읽고 쓸 수 있고, 포트 영역은 [READ\_PORT\_ULONG](http://msdn.microsoft.com/en-us/library/aa490156.aspx) 매크로와 [WRITE\_PORT\_ULONG](http://msdn.microsoft.com/en-us/library/aa490180.aspx) 매크로 등을 이용하여 읽고 쓸 수 있습니다. 이 때 주소는 매핑된 주소의 상대적인 위치를 사용합니다. 레지스터나 포트에 관련된 정보는 데이터시트를 참조해야 합니다.  
  
이후 Rx/Tx 디스크립터 링과 패킷 버퍼 할당에 필요한 DMA용 공유 메모리를 할당합니다. 공유 메모리 할당은 [NdisMAllocateSharedMemory](http://msdn.microsoft.com/en-us/library/bb314309.aspx) 함수를 사용합니다. 이 함수를 호출하면 지정한 크기의 공유 메모리를 할당하고 가상 주소와 DMA 물리 주소 쌍을 반환합니다.   
  
**디스크립터와 버퍼, 그리고 DMA**  
  
이 시점에서 인텔 8257x에서 일어나는 DMA에 대해 알아보도록 합시다. 기본 원리는 동일하기 때문에 다른 NIC들도 비슷한 동작을 할 것입니다. 반드시 디스크립터와 버퍼 개념을 이해하고 넘어가야 합니다. 디스크립터는 버퍼에 대한 부가적인 정보를 담은 메타데이터입니다. 디스크립터는 버퍼가 위치한 물리 주소와 길이, 송신/수신 완료 여부를 비롯하여 체크썸 오프로드 등 다양한 제어 비트를 포함하고 있습니다. 버퍼에는 실제 패킷 데이터를 기록합니다.   
  
수신 과정을 예로 들어보겠습니다. 드라이버가 디스크립터에 버퍼의 물리 주소와 길이를 집어넣고 수신 제어 레지스터를 조작합니다. 하드웨어는 디스크립터를 보고 수신한 패킷 데이터를 지정한 버퍼에 DMA로 전송합니다. DMA 전송이 완료되면 디스크립터에 완료 비트와 수신한 패킷 길이를 설정하고, 체크썸 오프로딩이 설정된 경우 IP/TCP/UDP 체크썸을 계산하여 결과를 디스크립터에 기록합니다. 이 외에도 VLAN 태그나 CRC 오류, 심볼 오류, 시퀀스 오류, 정렬 오류 등 다양한 오류를 디스크립터에 기록합니다. 기가비트 급 이상의 NIC라면 설정에 따라 RSS(Receive-side Scaling)에 필요한 해시 값도 계산하여 디스크립터에 기록합니다. 디스크립터와 버퍼를 대상으로 한 DMA 전송이 끝나면 인터럽트가 걸립니다.  
  
이렇게 디스크립터와 버퍼 모두 DMA 전송을 사용합니다. 일반적으로 DMA 공유 메모리 블럭을 크게 받아놓고 쪼개어서 씁니다만, 부하가 적은 상황에서는 메모리 낭비가 되기 때문에 동적으로 공유 메모리를 할당하고 해제하도록 구현하는 경우도 있습니다. 이에 관해서는 [NdisMAllocateSharedMemoryAsyncEx](http://msdn.microsoft.com/en-us/library/ms804228.aspx) 함수를 찾아보시기 바랍니다.  
  
**다시 MiniportInitialize**  
  
무슨 초기화만 이렇게 복잡하냐 하겠지만 사실 하드웨어 종속적인 부분의 설명은 하나도 없기 때문에 이 정도 분량인 것이고, 실제로는 하드웨어 초기화에 필요한 각종 레지스터 제어 코드 부분이 상당하기 때문에, 리셋에 성공하면 드라이버 개발의 절반 이상이 끝났다고 봐도 될 것입니다. 인텔 8257x의 경우 매뉴얼에서 레지스터 설명 부분만 200쪽이 넘으니까요. 사소한 LED 제어부터 시작해서 하드웨어 전원부터 올려야 하고, 멀티캐스트 수신 테이블, VLAN 설정, 자동 링크 속도 협상 등 많은 작업이 필요합니다.  
  
하드웨어 리셋이 끝났다면 이제 [NdisMRegisterInterrupt](http://msdn.microsoft.com/en-us/library/ms804191.aspx) 함수를 호출해서 인터럽트를 등록하는 것으로 초기화 작업이 끝납니다. 인터럽트 등록할 때 사용하는 인수는 이전에 [NdisMQueryAdapterResources](http://msdn.microsoft.com/en-us/library/ms804195.aspx) 함수를 이용하여 얻어놨던 인터럽트 레벨입니다. 하드웨어 제어를 통해 인터럽트를 활성화 해놨다면 등록하자마자 인터럽트가 걸리는 것을 확인할 수 있습니다.  
  
**MiniportQueryInformation**  
  
네트워크 어댑터 초기화가 끝난 직후부터 어댑터 정보 조회가 들어옵니다.

* [입력] MiniportAdapterContext
* [입력] OID
* [입력] 버퍼
* [입력] 버퍼 길이
* [출력] 출력 길이
* [출력] 필요한 바이트 수

아까 [NdisMSetAttributesEx](http://msdn.microsoft.com/en-us/library/ms804212.aspx) 함수로 설정했던 NDIS 핸들이 첫번째 매개변수로 들어옵니다. 두번째 매개변수는 OID 값으로 조회할 정보의 유형을 가리킵니다. 세번째와 네번째 매개변수는 버퍼 포인터와 길이입니다. 다섯번째는 버퍼에 몇 바이트나 썼는지 알려주는 역할이고, 여섯번째는 버퍼 길이가 모자란 경우 실제 어느 정도 길이의 버퍼가 필요한지 알려주는 역할을 합니다.  
  
이 함수에서는 NDIS에서 미리 정의한 OID 값에 따라 처리를 해줘야 합니다. 여기서는 몇 가지 OID만 설명할 것이고, 나머지는 MSDN을 찾아보시면 됩니다.

* [OID\_GEN\_SUPPORTED\_LIST](http://msdn.microsoft.com/en-us/library/bb648533.aspx): NDIS가 막무가내로 드라이버에게 OID 요청을 날리기는 곤란하겠죠. 물론 처리할 수 없는 OID가 들어오면 NDIS\_STATUS\_NOT\_SUPPORTED 상태 값을 반환하면 되지만, 실제로 어떤 OID 값을 드라이버에서 지원하는지 알 수 있도록 지원하는 OID 목록을 조회하는 기능을 하는 OID를 만들어두었습니다. 드라이버 개발자는 NDIS\_OID의 배열을 [NdisMoveMemory](http://msdn.microsoft.com/en-us/library/bb314399.aspx)를 이용하여 버퍼에 복사하면 됩니다. 만약 넘어온 버퍼의 길이가 복사하기에 충분하지 않다면, NDIS\_BUFFER\_TOO\_SHORT를 반환하면 됩니다. 그러면 마지막 매개변수 값을 보고 다시 버퍼를 할당해서 요청할테니까요.
* [OID\_GEN\_MEDIA\_IN\_USE](http://msdn.microsoft.com/en-us/library/bb648532.aspx): 현재 어떤 물리 매체를 사용하고 있는지 조회합니다. 802.3을 사용한다면 NdisMedium802\_3을 버퍼에 복사합니다.
* [OID\_GEN\_MAC\_OPTIONS](http://msdn.microsoft.com/en-us/library/bb648520.aspx): 하드웨어가 지원하는 기능을 조회합니다. 가령 하드웨어 수준에서 루프백을 지원하지 않는다면 NDIS가 소프트웨어적으로 루프백을 에뮬레이트하게 되어있습니다. 어떤 옵션이 있는지에 대해서는 MSDN을 봅시다.
* [OID\_802\_3\_CURRENT\_ADDRESS](http://msdn.microsoft.com/en-us/library/bb314157.aspx): 현재 MAC 주소 값을 조회합니다. 일반적으로 NIC의 EEPROM이나 플래시 등 영구저장매체에 6바이트짜리 MAC 주소가 저장되어 있습니다. MiniportInitialize 콜백이 호출될 때 미리 하드웨어를 제어해서 읽어놨다가 여기서 반환해주면 됩니다. 하드웨어가 MAC 주소 변경을 지원하고 레지스트리에 사용자가 지정한 MAC 주소가 저장되어 있다면 그 값으로 반환합니다. 물론 하드웨어에는 사용자가 지정한 MAC 주소를 써넣어야겠지요.
* [OID\_802\_3\_MAXIMUM\_LIST\_SIZE](http://msdn.microsoft.com/en-us/library/bb314152.aspx): NIC가 지원하는 멀티캐스트 주소의 최대 수가 얼마인지 조회합니다.
* [OID\_GEN\_MEDIA\_CONNECT\_STATUS](http://msdn.microsoft.com/en-us/library/bb648515.aspx): NIC의 물리 매체 연결 상태를 조회합니다. 쉽게 말하면 랜선 연결됐나 확인하는 기능입니다. 하드웨어를 제어하여 현재 연결 상태를 조회한 다음, 연결되어 있다면 NdisMediaStateConnected를, 연결되어있지 않다면 NdisMediaStateDisconnected를 반환하면 됩니다.
* [OID\_GEN\_LINK\_SPEED](http://msdn.microsoft.com/en-us/library/bb648521.aspx): 현재 링크 연결 속도를 조회합니다. 약간 특이한 것은 단위가 100bps라는 것입니다. 따라서 1Gbps라면 10000000, 100Mbps라면 1000000, 10Mbps라면 100000으로 써야 합니다.

여기까지 제대로 구현했다면 장치관리자에서 정상적으로 활성화가 됩니다. MiniportInitialize만 구현하고 MiniportQueryInformation 콜백을 구현하지 않은 상태에서는 장치가 정상적으로 동작하지 않음을 알리는 아이콘이 표시될 것입니다.  
  
**MiniportHalt**  
  
어댑터 활성화를 구현했으니 이제 비활성화를 구현해야 합니다. 장치관리자나 네트워크 연결 관리에서 비활성화를 누르면 MiniportHalt 콜백이 호출됩니다. 제일 먼저 하드웨어를 제어하여 인터럽트부터 비활성화 합니다. 그 다음 하드웨어를 제어하여 송/수신 기능을 정지시킵니다. 전원까지 다 내리면 하드웨어가 완전히 멈춘 상태가 되고, 이제 MiniportInitialize 콜백에서 했던 자원 할당을 반대 순서로 해제하면 됩니다. [NdisMDeregisterInterrupt](http://msdn.microsoft.com/en-us/library/ms804203.aspx) 함수를 호출해서 인터럽트 연결을 제거하고, 송/수신 디스크립터 링과 패킷 버퍼에 사용된 공유 메모리를 [NdisMFreeSharedMemory](http://msdn.microsoft.com/en-us/library/bb314303.aspx) 함수를 호출하여 해제합니다. 하드웨어를 제어하는데 사용했던 레지스터나 포트 매핑도 제거합니다. 각각 [NdisMUnmapIoSpace](http://msdn.microsoft.com/en-us/library/bb314300.aspx) 함수와 [NdisMDeregisterIoPortRange](http://msdn.microsoft.com/en-us/library/bb314315.aspx) 함수를 사용합니다. 나머지 스핀락 등 자잘한 자원도 해제해주면 비활성화 작업이 완료됩니다.  
  
**MiniportIsr**  
  
인터럽트 서비스 루틴입니다. MiniportInitialize 콜백에서 초기화를 완료하고 하드웨어가 송/수신 작업을 시작하도록 해놨다면 그 때부터 각종 인터럽트가 걸립니다. 인터럽트 발생은 패킷 송신이나 수신이 완료되었을 때, 링크 연결 상태가 변경되었을 때, 하드웨어 버퍼가 오버런 나고 있을 때 등 다양한 경우가 있습니다. 인터럽트 종류에 따라 아주 빠르게 즉시 처리 가능한 것(단순 상태 변경 등)은 이 콜백에서 바로 처리하면 되고, 그렇지 않은 경우(송/수신 처리 등)라면 DPC 큐에 넣고 나중에 처리하도록 합니다.

* [출력] PBOOLEAN InterruptRecognized
* [출력] PBOOLEAN QueueMiniportHandleInterrupt
* [입력] NDIS\_HANDLE MiniportAdapterContext

DPC 큐에 넣으려면 간단히 출력 매개변수 조작만 하면 됩니다. QueueMiniportHandleInterrupt 변수 값을 TRUE로 하면, NDIS가 DPC를 예약해주고 나중에 DISPATCH\_LEVEL에서 MiniportHandleInterrupt 콜백을 호출합니다.   
  
InterruptRecognized는 자신의 인터럽트인 경우에만 TRUE로 바꿔야 합니다. 만약 인터럽트를 다른 장치와 공유하고 있는 상황에서 다른 장치의 인터럽트를 빼앗아버리면 올바른 동작을 보장할 수 없게 됩니다. 반대로, 자신의 인터럽트인데 그냥 흘려버리게 되면, 다른 장치는 상황과 문맥에 맞지 않는 이상한 인터럽트([spurious interrupt](http://en.wikipedia.org/wiki/Interrupt))를 받게 될 수 있습니다. 인터럽트를 공유하지 않더라도 인터럽트 발생을 Edge 기준으로 하지 않고 Level 기준으로 하는 일반적인 경우, 처리하지 않은 인터럽트가 계속 걸리게 되고 이는 성능 저하로 이어질 수 있습니다.  
  
인터럽트 서비스 루틴 처리를 시작할 때 인터럽트가 중복해서 발생하지 않도록 인터럽트를 비활성화하고, 인터럽트 처리를 바로 완료할 수 있는 경우에는 인터럽트를 활성화합니다. 만약 여기서 바로 처리를 끝낼 수 없고 MiniportHandleInterrupt 콜백에서 지연 처리를 해야한다면 그 때 가서 인터럽트를 활성화합니다.  
  
**MiniportHandleInterrupt**  
  
MiniportIsr 콜백 이후 DISPATCH LEVEL에서 실질적인 인터럽트 처리를 수행합니다. 수신 디스크립터 링을 순회하면서 수신 처리가 완료되었는지 상태를 확인하고, DMA 전송이 완료된 패킷 버퍼는 상위 스택으로 올려보내고 빈 패킷 버퍼로 교체합니다. 완료 처리가 끝났으면 디스크립터 링 포인터를 처리한만큼 이동시켜 줍니다. 송신 처리는 반대로 돌아갑니다. 전송할 패킷 버퍼 포인터를 넣고, 전송이 완료된 빈 패킷을 상위 스택으로 되돌려 보냅니다. 마지막에 인터럽트를 활성화하고 끝납니다. 어댑터를 비활성화하기 전까지 동일한 과정으로 인터럽트 처리가 계속 이어집니다.  
  
**마무리**  
  
이 정도가 NIC를 구동하는데 필요한 가장 핵심적인 부분이라고 할 수 있고, 나머지 콜백들([MiniportCheckForHang](http://msdn.microsoft.com/en-us/library/ms797251.aspx), [MiniportReset](http://msdn.microsoft.com/en-us/library/ms797254.aspx), [MiniportSetInformation](http://msdn.microsoft.com/en-us/library/ms797238.aspx))은 MSDN을 참고하시기 바랍니다. 이 정도만 알고 시작해도 훨씬 수월하게 느껴질 겁니다. 사실 NDIS 버전에 따라 함수들도 조금씩 바뀌기 때문에 구 버전으로 달달 외워봐야 별 의미가 없습니다. 기본 원리와 흐름만 알고 달려들면 됩니다. 특히 NDIS 6 (비스타부터 지원)은 [Receive-side Scaling (RSS)](http://msdn.microsoft.com/en-us/library/ms795609.aspx)를 지원하는데, 기가비트 이상을 제대로 지원하려면 반드시 알아두셔야 합니다.