AR6002 NDIS 미니 포트 디바이스 드라이버.

1. Windows CE 네트워크 드라이버 구조.

1.1 OSI 7계층.

어플리케이션 계층

프리젠테이션 계층

세션 계층

전송 계층

네트워크 계층

데이터 링크 계층

물리 계층

어플리케이션 계층

프리젠테이션 계층

세션 계층

전송 계층

네트워크 계층

데이터 링크 계층

물리 계층

물리주소 관리

네트워크 토폴로지

회선사용규칙

오류검출

프레임 전달

흐름 제어

그림 1. OSI 7 계층의 NDIS 드라이버 위치

OSI 7 계층을 보면 데이터 링크 계층이 존재하는데, 이 층을 지원하도록 작성되는 드라이버가 미니포트 드라이버이다. BM170 시스템에서는 AR6002 드라이버가 이에 해당 한다.

1.2 NDIS 모델과 미니포트 드라이버 위치.

…

텔넷

인터넷 익스플로러

소켓

TDI

(Transport Driver Interface)

프로토콜 드라이버

NDIS

미니포트 NIC 드라이버

NDIS

NDIS

AR6002 하드웨어

**NDIS 드라이버**

**Callback 함수로 통신**

**Miniport 드라이버.**

**NIC(Network Interface Card) Layer**

NDIS Callback 함수

AR6002 모듈 제어 코드

**AR6002 H/W 모듈**

모듈 Firmware

**SDIO 통신**

2. Windows Mobile NDIS 드라이버 로딩.

2.1 드라이버 로딩 과정.

**SDIO 디바이스 인식**

**레지스트리 등록**

**NDIS 디바이스 등록**

**DriverEntery() 실행**

**NDIS Initialize() 실행**

**HIF 초기화**

**펌웨어 다운로드**

**HTC 초기화**

**WMI 초기화**

**SDIO Client 등록**

드라이버를 작성하기에 앞서 우선 Windows Mobile이 인식할 수 있게 레지스트리를 준비해야 한다. AR6002 모듈은 SDIO 버스 드라이버를 사용하므로 SDCARD 아래에 작성해야 한다.

**[HKEY\_LOCAL\_MACHINE\Drivers\SDCARD\ClientDrivers\Custom\MANF-0271-CARDID-0108-FUNC-1]**

**"Dll"="ar6k\_ndis\_sdio.dll"**

**"Prefix"="DRG"**

키에서 볼 수 있는 MANF-0271-CARDID-0108-FUNC-1 키는 AR6002 모듈의 PNPID 이다. Windows Mobile에서 SDIO 디바이스가 인식되면 위의 레이스트리 정보를 이용하여 DLL을 로딩한다. DLL이 로딩되면 우선 DRG\_Init() 함수를 호출 하게 된다. 이 함수는 레지스트리의 “Prefix” 정보를 참조하여 DLL 로딩시 불려지게 된다. DGR\_Init() 함수에서는 NdisRegisterAdapter() 함수를 호출하여 MiniportDriverName과 AdapterInstanceName를 이용하여 레지스트리의 HKEY\_LOCAL\_MACHINE\Comm\AR6K\_SD의 키를 찾게 된다. AR6002에서는 MiniportDriverName = “AR6K\_SD”, AdapterInstanceName = “AR6K\_SD1”을 사용한다.

**[HKEY\_LOCAL\_MACHINE\Comm\AR6K\_SD]**

**"DisplayName"="AR6000 WLAN Adapter"**

**"Group"="NDIS"**

**"ImagePath"="ar6k\_ndis\_sdio.dll"**

위 키 정보를 이용하여 NDIS 디바이스가 로딩 된다. 디바이스가 로딩된 후 DriverEntry() 함수가 호출 되는데 이 함수가 실질적인 이 드라이버의 시작 코드 이다. DriverEntery() 함수에서는 NDIS 드라이버에서 사용되어 지는 콜백함수들을 등록한다. 콜백함수에 대해서는 뒤에 설명을 할 거이다. DriverEntry()함수에서 NDIS 콜백함수가 등록이 되면 시스템은 NDIS의 InitializeHandler()를 호출 한다. 이 함수에서 우리는 S/W 및 H/W 초기화를 실행한다. 우선 모듈과 통신을 하기 위해 HIF Layer를 초기화 시키고, 그 다음 모듈 펌웨어를 다운로드 한다. 그리고 나서 HTC 및 WMI Layer 초기화를 실행 한다. 이상의 과정은 모듈의 초기화 과정을 설명한 것이다. 여기서 중요한 점은 모듈이 로딩되기 위해 SDIO 버스 드라이버 등록 및 NDIS 미니포트 드라이버 등록 과정이다.

2.2 NDIS 콜백함수

■ NDIS 주요 콜백함수

- InitializeHandler

- QueryInformationHandler

- SetInformationHandler

- SendPacketsHandler

- ReturnPacketHandler

- HaltHandler

- ResetHandler

위 콜백함수들은 NDIS의 DriverEntery()함수에서 등록이 되어 지는 함수 들이다. DriverEntery() 함수에서는 NDIS Wrapper 핸들러를 구한 뒤, 자신의 미니포트 콜백 함수들을 NdisMRegisterMiniport()함수를 이용하여 NDIS에 등록한다.

■ **InitializeHandler**

DriverEntery()가 호출되고 나서 네트워크 어댑터가 활성화되면 이 콜백 함수가 호출된다.

장치 타입 설정

NdisMSetAttributesEx()

호출

CAR6k::Initialize() 호출

CAR6k 클래스 생성

이 함수에서는 SupportedMedium 매개 변수에 장치 타입을 알려 주고 CAR6K 클래스를 생성한다. 여기서 알아둘 사항은 AR6002 Windows Mobile 모듈의 제어는 CAR6K 클래스에 의해 제어 된다. 디바이스 제어와 관련된 일들은 CAR6K 클래스에 구현되어 있다. 그리고 나서 NDISMSetAttributesEx()를 호출 하는데, 이 함수는 CAR6K 클래스의 포인터를 MiniportAdapterContext에 등록한다. 이 부분이 중요한데, NDIS 콜백 함수들이 불려 질 때 이 Context정보를 이용하여 디바이스에 접근하기 때문이다.

**VOID NdisMSetAttributesEx(**

**NDIS\_HANDLE MiniportAdapterHandle,**

**NDIS\_HANDLE MiniportAdapterContext,**

**UINT CheckForHangTimeInSeconds,**

**ULONG AttributeFlags,**

**NDIS\_INTERFACE\_TYPE AdapterType**

**);**

NdisMSetAttributesEx() 함수의 AttributeFlags 변수와 AdapterType 변수에 값을 넣어 준다. AR6002 모듈은 Flag 값으로 **NDIS\_ATTRIBUTE\_DESERIALIZE | NDIS\_ATTRIBUTE\_NO\_HALT\_ON\_SUSPEND** 을 사용하고 AdapterType 값으로 NdisInterfaceInternal 을 사용한다. 값들의 내용을 알고 싶으면 MSDN을 참고 하기 바란다.

그 다음으로 Initialize 함수가 하는 일이 CAR6K::Initialize()를 호출 하는 일이다. 여기서는 AR6002 모듈의 S/W와 H/W를 초기화 하는 역할을 한다.

**이밴트 초기화**

- Target Ready Event

- WMI Ready Event

- Rx Pending Event

- Rx Work Item Exit Event

RX/TX Buffer 메모리 할당

Wait Target Ready Event

HIF Init() 호출

Set Wifi 레지스트리

우선 모듈 제어에 필요한 각종 이벤트를 초기화 시킨다.

**- Target Ready Event.**

HIF Init()이 실행 되고 SDIO 디바이스 등록 및 probe 함수가 정상적으로 이루어졌을 때 이 이벤트를 발생 시킨다.

**- WMI Ready Event**

AR6002 모듈에 펌웨어를 다운로드 하고 BMI\_DONE 메시지를 전송하면 AR6002는 WMI\_READY\_EVENT 메시지를 보내준다. 이 메시지가 뜻 하는 것은 모듈의 초기화 과정이 모두 정상적으로 이루어 졌으며, 모듈이 명령을 받을 준비가 됐다는 것을 의미한다.

**- RX Pending Event**

모듈로부터 데이터가 들어 왔을 때 발생하는 이벤트로 시스템에 Wireless 데이터가 들어왔다는 것을 알리기 위해 사용된다.

**- Rx Work Item Exit Event**

Rx Pending Event 와 함께 사용되는 Event 이다.

이벤트 초기화가 완료되면, 사용할 버퍼들을 메모리에 할당하고 초기화를 시킨다. 버퍼는 전송 버퍼와 수신 버퍼 두 개를 사용하며, 수신 버퍼의 경우 NdisAllocatePacketPool() 함수를 이용하여 만들어야 한다. 수신된 데이터를 NDIS Layer에서 가져가기 위해 사용하는 방법이다. 버퍼 초기화가 완료되면 AR6002 모듈을 사용 할 수 있게 HIFInit()을 호출 한다. 이 함수는 AR6002 모듈과 통신이 가능하게 SDIO 드라이버에 모듈 디바이스를 추가 하고 HCT Layer 및 WMI Layer 초기화 까지 진행 한다. 이 부분에 대해서는 다음 장에 자세히 설명되어 있다. HIFInit() 함수 까지 정상 완료 되면 이제 WLAN 모듈을 사용할 수 있는 단계까지 완료 되었으며 이를 Windows Mobile 시스템에 알려 주기 위해 “SYSTEM\\STATE\\HARDWARE” 레지스트리에 “Wifi” 레지스터를 추가 하여 Wifi가 장착되었다는 것을 시스템에 알려 준다. 여기 까지가 NDIS의 **InitializeHandler** 콜백 함수의 역할 이다.

■ QueryInformationHandler

장착되어 있는 네트워크 WLAN의 속성을 확인하기 위해 이 콜백 함수가 사용된다. AR6002 모듈의 경우 장치 타입으로 NdisMedium802\_3을 입력했으므로 802\_3에 관련된 OID만 내려온다. 함수는 Switch 구문으로 작성되어져 있으며 각 항목에 맞게 적절한 처리를 해줘야 한다. 각 항목별 처리는 뒤에서 자세히 다룰 것이다.

■ SetInformationHandler

QueryInformationHandler 함수가 WLAN의 속성을 확인하는 것이라면 이 콜백 함수는 WLAN의 속성을 변경하는 역할을 한다. 따라서 이 함수 역시 Switch 구문으로 작성되어 져야 하며 각 항목별로 적절한 처리를 해줘야 한다.

■ SendPacketsHandler

프로토콜 드라이버로부터 다른 곳으로 내보낼 데이터 패킷이 전달 되는 곳이다. 전달 되는 패킷은 NDIS\_PACKET 형태를 가지는 구조체이므로, 이 패킷을 열어서 실제 보낼 데이터를 가져오는 알고리즘을 추가 해야 한다.

■ ReturnPacketHandler

WLAN 모듈로부터 수신된 데이터를 가져가기 위한 콜백 함수이다. AR6002 모듈로부터 들어온 데이터가 RX 버퍼에 있으면 이 콜백 함수를 통해 NDIS Layer로 전달 되어 진다.

**3. AR6002 모듈 Packet 송수신.**

**3.1 Packet 구조.**

**Payload & Trailer**

**HTC Frame Header**

**Control Byte**

(2Byte)

**Length**

(2Byte)

**Flags**

(1Byte)

**Endpoint ID**

(1Byte)

Packet의 구조는 HTC Frame Header 와 Payload & Trailer 로 구성이 되어있다. HTC Frame은 Endpoint ID, 패킷 크기 등의 정보를 가지고 있고, 실질적은 데이터는 payload & Trailer에서 가지고 있다.

|  |  |
| --- | --- |
| **Field** | **Description** |
| Endpoint ID | Endpoint(0~255), 메시지 송/수신시 사용됨 |
| Flags | 모듈이 상태 표시. |
| Length | Payload 길이 |
| Control Byte | Flags에 따른 부가 정보. |

**3.2 Packet 전송.**

AR6002 모듈에서 패킷 전송은 단순 Read/Write 가 있는 것이 아니라, Queue-Buffer와 Flow-control을 통해 이루어 진다. 우선 전송할 데이터를 Buffer Q에 넣고 나서 모듈이 데이터를 받아 들일 수 있는지를 확인 한다. 이 때 사용되는 것이 Credit이다. Credit는 모듈이 현재 받을 수 있는 buffer의 개 수를 나타낸다. 패킷을 전송할 때 Credit 개수를 확인하여 buffer가 여유가 있다면, 그 때 SDIO Write 함수를 이용하여 데이터를 모듈로 전송 한다. 만약 Credit 개수가 부족하다면 시스템은 Credit 개수가 확보될 때 까지 Write 함수를 잠시 대기 시킨다. Credit 개수의 업데이트는 Credit 개수가 필요 Credit 보다 작을 때, 모듈로 Credit 업데이트를 요청한다. 이러한 과정은 모듈과 Host간의 Flow-Control 을 하기 위한 동작이다. 모듈이 Host로부터 받은 데이터를 처리하고 나서 새로운 데이터를 받아들이기 위해 Credit를 이용하여 Flow-Control을 하게 된다.

Buffer 1

Buffer 2

Buffer 3

Buffer 4

**HTCSendPkt**

**HTCTrySend**

**AR6002**

**Check Credit**

**SDIO Write**

**Updata Credit**

**Check Credit**

**Count**

**EpTxComplete**

**Ar6k\_tx\_complete**

패킷 전송은 HTCSednPkt() 함수부터 시작 된다. Control 데이터 또는 일반 데이터 모두 최종적으로 HTCSendPkt() 함수를 호출하여 데이터 전송을 시작 한다. HTCSendPkt() 함수는 HTC Packet 구조체에 전송 완료 콜백 함수를 등록하고 나서 HTCTrySend() 함수를 호출 한다. HTCTrySend() 함수에서는 우선 전송 패킷을 버퍼에 등록한다. 전송 버퍼는 HTC\_PACKET 구조체의 Linked-List로 되어 있다. 패킷을 전송 버퍼에 등록 한 후, Credit 개수를 확인하여 사용 가능한 Credit이 있다면 데이터를 전송한다. 만약 Credit 개수가 부족하다면, 버퍼에 등록만 하고 빠져 나온다.

전송되지 않고 버퍼에 등록된 패킷들은 Update Credit 이밴트에 의해 재 전송 된다. AR6002 모듈로부터 Updata Credit 이밴트가 발생하면 HTCProcessCreditRpt() 함수에서 Credit 개수를 Update 하고, 버퍼에 패킷이 남아 있는지를 검사하여 만약 패킷이 남아 있다면 HTCTrySend() 함수를 호출하여 남아 있는 패킷을 전송한다. 이러한 구조로 데이터 전송이 이루어지며 Flow-Control이 이루어 진다.

**3.3 Packet 수신.**

AR6002의 데이터 수신은 두 가지 방법으로 이루어 진다. 하나는 Polling 방식이고 다른 하나는 인터럽트를 이용하는 방식이다. Polling 방식은 데이터를 보내고 곧 바로 수신 받아야 할 데이터가 있을 때 사용되는데, HTCConnectService() 함수에서 이를 찾아 볼 수 있다. HTCConnectServic()는 모듈에 Connect Service를 요청한 후 곧 바로 응답을 받아야 하기에 이 방식을 사용한다. 그러나 polling 방식의 경우 모듈의 응답 대기 시간이 필요 하므로 시스템에 부하를 주는 방식이다. 꼭 필요한 경우가 아니라면 사용되지 않는다, 인터럽트 방식은 SDIO의 인터럽트를 이용하는 방식인데, 모듈에서 Host로 보낼 데이터가 있으면 SDIO 인터럽트를 발생시켜 Host에 이를 알리는 방식이다. 일반적인 Control 데이터 및 일반 데이터가 이 방식을 이용한다.

Interrupt

**HTCIssueRecv**

**AR6002**

**dsrHandler()**

**ProcessPendingIRQs**

**HTCRecvMessagePending**

**Handler**

Read Data

**Ar6k\_rx 호출**

Read Data 완료 후

위 그림은 AR6002의 패킷 수신 과정을 그린 것이다. 최종 적으로 호출 되는 것은 ar6k\_rx() 함수 이며 이 함수에서 NDIS Recevie Buffer 함수를 호출하여 수신된 패킷을 처리 한다. 수신 과정을 살펴 보면 우선 SDIO Interrupter가 발생하면, dsrHandler()가 호출 된다. 이 함수에서 다시 ProcessPendingIRQ()함수를 호출 한다. 각 함수별로 세세한 처리과정이 있지만 여기서는 전체적인 흐름만 살펴 보겠다. ProcessPendingIRQ()함수에서는 Interrupt의 종류를 확인하고 데이터 수신 인터럽트이면, HTCRecvMessagePendingHandler() 함수를 호출 한다. 데이터 수신 인터럽트가 아니면 각 인터럽트에 따른 처리를 해준다. HTCRecvMessagePendingHandler() 함수에서는 실질적은 데이터 Read가 발생한다. HTCIssuRecv() 함수를 호출 하여 수신되어져야 할 데이터를 읽어 들인다. 그리고 나서 ar6k\_rx() 함수를 호출 한다. Ar6k\_rx() 함수에서는 수신되어진 데이터가 control 데이터인지, 일반 데이터인지를 구분하여 각 상황에 맞게 처리 한다.

**4. AR6002 모듈 펌웨어 다운로드 및 초기화.**

**4.1 모듈 초기화 과정.**

CAR6K 클래스에서

HIFInit()호출

SDIORegisterFunction()

실행하여 SDIO 디바이스 등록

SDIO의 pProbe 호출

hifDeviceInserted()가 호출됨

hifDeviceInserted()에서

SDIO 디바이스 초기화.

SDIO 디바이스 초기화 후

AR6000\_avail\_ev 호출

펌웨어 다운로드 실행

펌웨어 다운로드 완료 후

AR6000\_Init() 호출

BMI\_DONE 메시지 전송

HTCCreat() 실행

WMI\_Init() 실행

모듈의 초기화는 CAR6K::Initialize()에서 HIFInit()함수를 호출함으로써 시작된다. HIFInit()함수에서는 OSDRV\_CALLBACKS 구조체에 디바이스 콜백 함수들을 등록하고 SD\_FUNCTION 구조체 변수에 맴버 함수를 등록하고 나서, SDIORegisterFunction()을 호출 하여 SDIO 디바이스 등록을 한다. SD\_FUNCTION 구조체의 경우 SDIO 드라이버에 맞추어져 있어 특별한 것이 없지만 OSDRV\_CALLBACKS 구조체의 경우 HIF Layer에서 사용 되어지는 콜백 함수들이다.

SDIORegisterFunction()함수에서는 HIFInit()에서 등록된 hifDeviceInserted()함수를 호출 하는 것으로 역할을 다한다. SDIO 디바이스 드라이버 등록과정에서 Probe 함수를 호출 하는 것이다.

hifDeviceInserted()함수는 SDIO 드라이버에 디바이스를 등록하고 초기화 하는 과정을 담고 있다. SDIO 디바이스 전송 타입 및 클럭등의 속성을 설정한다. 이 부분은 Windows Mobile의 SDIO 디바이스 부분을 참고 하기 바란다. 속성 설정이 끝이 나면 CAR6K::Initialize()에서 등록된 deviceInsertedHandler 콜백 함수를 호출 하는데 여기서는 ar6000\_avail\_ev()함수에 연결이 되어 있다.

Ar6000\_avail\_ev()함수가 불려지기 전까지는 모듈의 H/W 통신 부분인 SDIO 디바이스 드라이버 등록에 관련된 부분이고, 이 함수부터는 AR6002 모듈을 제어하기 위한 모듈 초기화가 들어 있다. AR6002 드라이버의 가장 중요한 부분이라 할 수 있다.

4. AP 리스트 스캔.