네트워크 드라이버 프로그래밍

드라이버 제어 프로그램 만들기

네트워크 드라이버 작성에 이어 이번 호에서는 이를 제어하는 드라이버 제어 프로그램을 만들어 본다. 보통 드라이버는 설치된 후 필요에 따라 적재되어 사용되지만 필터드라이버의 경우 설치 후 원하는 필터 내용에 따라 제어해야 하기 때문에, 제어 루틴이 필요하다.

연 재 순 서
1회 | 2006. 6 | NDIS에 대해서 알아보기
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

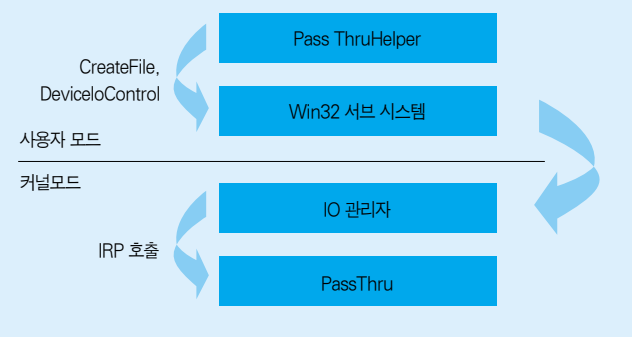
박상민 sangmin.park@gmail.com | 소만사(www.somansa.com)라는 회사에서 웹분석 솔루션의 설계 및 개발을 담당했다. 윈도우 기반 개발, 웹개발, 데이터베이스 프로그래밍 등의 기술을 습득했고, 어떤 프로젝트가 주어지더라도 최상의 결과를 만들어내는 소프트웨어 아키텍트가 되는 것이 꿈이다. 인생을 편하게 사는 것을 목표로 삼고 있고, 내일은 내일의 바람이 분다(tomowind.egloos.com)라는 블로그를 운영하고 있다.

이번 호의 주된 목표는 사용자 프로그램에서 실시간으로 드라이버를 제어할 수 있도록, 드라이버와 사용자 프로그램을 확장하는 것이다. 지난 시간에 다뤘던 PassThru 네트워크 드라이버 소스와 PassThruHelper라는 설치 프로그램을 기반으로 한다. PassThru 드라이버 소스에는 드라이버 제어용 코드를 추가하고, PassThruHelper에는 드라이버를 제어할 수 있는 코드를 추가하도록 한다.

드라이버에 추가하는 코드는 'IMSamp'라는 예제를 기본으로 한다. http://support.microsoft.com/kb/q178322/에서 다운로드할 수 있다. NDIS 4.0 기반의 네트워크 드라이버 소스로 드라이버 제어 코드에 관한 부분을 담고 있다. 소스파일의 구성은 아래와 같다.

main.c - DriverEntry, InitializeNdisWrapper, DoMiniportInit, DoProtocolInit, InitializationCleanup
globals.c - adapter/request 등의 구조체 정의
adapter.c - adapter binding/unbinding 루틴
ndisreq.c - ndis 요청 처리 루틴
recv.c - NIC에 의해 불리는 패킷 수신 루틴
send.c - Transport Layer에 의해 불리는 패킷 전송 루틴
config.c - config information(registry, path)
status.c - status indication
pstub.c - ndis reset completion routine
wdmsup.c - WDM 관련(추가할 드라이버 제어의 핵심코드)

사용자 프로그램에서 드라이버를 호출하는 흐름은 〈그림 1〉과 같다. PassThruHelper가 CreateFile, DeviceIoControl 등의 라이브러리를 사용해 Win32 서브시스템을 호출하면, 윈도우에서는 제어를 커널로 넘긴다. 이때 I/O관리자는 IRP 패킷을 사용해 PassThru 드라이버로 명령을 호출한다.



〈그림 1〉 사용자 프로그램에서 드라이버 호출

예컨대, 드라이버에 제어 인터페이스를 만들고, 사용자 프로그

램에 제어 코드를 추가, 테스트하는 것이 이번 강좌의 기본흐름이다.

필자 메모

필자가 이번 연재를 쓰면서 가장 힘들었던 점은 '블루스크린' 현상을 발생시키는 것이었다. 보통 드라이버를 설치하면, 드라이버가 동작을 하다 잘못된 코드에서 죽게 된다. 하지만, 자주 쓰이는 코드에서 버그가 발생하면 컴퓨터를 켜자마자 죽는 상황이 연출되기도 한다. 여기서도 매우 자주 쓰이는 부분의 코드를 다룬다. 그래서 만약 버그가 있을 경우에는, 컴퓨터를 켜고 패킷을 받거나 보내자마자 버그 부분에 돌입할 수 있다. 이런 경우 당황하지 말고 다음과 같이 대처하도록 한다.

1. F8을 눌러 안전모드로 부팅한다.
2. 설치 프로그램을 실행해 드라이버를 내리거나, 컴퓨터에서 해당 드라이버 파일을 전부 찾아 삭제한다.
3. 재부팅한다.

물론 이것이 가장 현명한 방법이 아닐 수도 있으나 필자가 원고를 작성하면서 이런 방법으로 위기를 넘기곤 했다. 정말 '공포'의 블루스크린이라는 것을 다시 한 번 실감했다.

드라이버에 제어 인터페이스 추가

일반적으로 드라이버는 WDM(Windows Driver Model) 인터페이스로 통신을 한다. WDM은 지난 6월호의 연재 첫 시간에 언급한 개념으로 드라이버와 드라이버, 드라이버와 사용자 프로그램간 통신을 도와주는 규격이다. IRP(I/O Request Packet)라는 패킷을 통해 통신하며, WDM 루틴에 해당하는 IRP를 처리해주는 것을 의미한다.

여기서는 드라이버에 제어 코드를 넣는 것이 목적으로 WDM 코드를 찾아 넣고, 필터링 모듈을 추가한다. 우선, IMSamp라는 드라이버 소스에 대해서 알아본다. IMSamp는 PassThru처럼 IM(Intermediate)드라이버다. 이름 그대로 IM(Intermediate)+Samp(Sample)인 것이다. 하지만 NDIS 4.0 규격에 맞춰 제작되어 있고, WDM 코드가 추가되어 있다는 점이 PassThru와 다르다. IMSamp에서 WDM 코드를 찾아내 PassThru로 옮기도록 한다. 또한, 향상된 필터링 기능을 추가한다. 필터링 옵션을 이용해 옵션에 따라서 다른 필터링이 되도록 수정한다.

WDM 모듈 추가

wdmsup.h/c 파일이 핵심이다. 소스를 살펴보면 WDMInitialize이라는 초기화 함수가 있다. 이것은 드라이버의 초기화 함수인 〈리스트 1〉과 같이 DriverEntry에서 호출된다. PassThru에서 DriverEntry는 미니포트 드라이버, 프로토콜 드라이버를 등록하는 일을 한다. 뒷부분에, WDM 초기화 함수를 등록하면 된다.

〈리스트 1〉 WDM 초기화를 해주는 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.
    //
    NdisMInitializeWrapper(&WrapperHandle, DriverObject,
RegistryPath, NULL);
    ...

    //
    // Protocol을 NDIS에 등록시켜준다.
    //
    NdisZeroMemory(&PChars,
sizeof(NDIS_PROTOCOL_CHARACTERISTICS));
    ...
    NdisRegisterProtocol(&Status, &ProtHandle, &PChars,
sizeof(NDIS_PROTOCOL_CHARACTERISTICS));
    ASSERT(Status == NDIS_STATUS_SUCCESS);
    NdisIMAssociateMiniport(DriverHandle, ProtHandle);

    //
    // WDM 초기화 코드를 추가한다.
    //
    Status = WDMInitialize(DriverObject,
&InitShutdownMask);
    if(!NT_SUCCESS( Status ))
    {
        DbgPrint("WDMInitialize Failed!! Status: 0x%x\n",
Status);
        WDMCleanup(InitShutdownMask);
        return (STATUS_UNSUCCESSFUL);
    }

    return(Status);
}
```

그렇다면, 실제 WDM 초기화는 어떻게 발생할까? WDM은 IRP에 따라 해당되는 함수를 호출하는 역할을 하는 인터페이스다. 따라서 WDMInitialize는 〈리스트 2〉처럼 IRP에 따라 MajorFunction을 등록해 주고, IRP에 따른 처리는 〈리스트 3〉과 같이 switch 문에서 분기가 된다.

향상된 필터링 모듈 추가

지난 호에서 설명했던 DebugView에 입력한 포트는 80이다. 이번에는 원하는 포트의 리스트에 입력해본다. 또한, 입력방식도 두 가지로 나눠 보자. 다음과 같은 순서로 진행한다.

● 포트 리스트 정의

포트 리스트를 숫자의 배열로 정의한다. 사용자는 보고자 하는 포트의 리스트를 IRP를 통해 드라이버에 요청하면, 그 정보들이 이 배열에 쌓이게 된다. 다음과 같은 이름으로 global.h에 정의했다.

```
extern UINT     MASO_PORT_LIST[MASO_MAX_PORT_LIST];
```

〈리스트 2〉 WDMInitialize : IRP에 따라 함수를 달리 등록해준다. 물론, 여기서는 전부 IMIoctl이라는 함수를 호출하며 그 함수 내부에서 분기되어 제각각 처리된다.

```
NTSTATUS
WDMInitialize(
    PDRIVER_OBJECT DriverObject,
    PULONG InitShutdownMask
    )
{
    NTSTATUS Status;
    UINT FuncIndex;

    //
    // IRP에 따른 함수를 등록해준다.
    //

    DriverObject->FastIoDispatch = NULL;

    for (FuncIndex = 0; FuncIndex <=
IRP_MJ_MAXIMUM_FUNCTION; FuncIndex++) {
        DriverObject->MajorFunction[FuncIndex] = IMIoctl;
    }
    ....

    return Status;
}
```

〈리스트 3〉 IMIoctl : IRP에 따른 처리를 해준다.

```
STATIC NTSTATUS
IMIoctl(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
{
    //
    // IRP에 따라 다른 처리를 해준다.
    //
    switch (irpStack->MajorFunction) {
    case IRP_MJ_CREATE:
        ImDbgOut( DBG_TRACE, DBG_IO, ("IRP Create\n" ));
        break;

    case IRP_MJ_CLOSE:
        ImDbgOut( DBG_TRACE, DBG_IO, ("IRP Close\n" ));
        break;

    case IRP_MJ_CLEANUP:
        ImDbgOut( DBG_TRACE, DBG_IO, ("IRP Cleanup\n" ));
        break;

    case IRP_MJ_SHUTDOWN:
        ImDbgOut( DBG_TRACE, DBG_IO, ("IRP Shutdown\n" ));
        break;

    case IRP_MJ_DEVICE_CONTROL:

        //
        // 우리는 이 부분에서 원하는 코드를 넣는다.
        //

        ioControlCode = irpStack-
>Parameters.DeviceIoControl.IoControlCode;
        switch (ioControlCode) {

       // This is where you would add your IOCTL handlers

        default:
            ImDbgOut( DBG_INFO, DBG_IO,
                    ("unknown IRP_MJ_DEVICE_CONTROL\n =
%X\n",ioControlCode));
            Status = STATUS_INVALID_PARAMETER;
            break;

        }
        break;

    default:
        ImDbgOut(DBG_INFO, DBG_IO,
                ("unknown IRP major function = %08X\n",
irpStack->MajorFunction));

        Status = STATUS_UNSUCCESSFUL;
        break;
    }

    return Status;
}
```

● 프린트 타입 정의

DebugView에 보이는 프린트 타입도 두 가지로 정의하였다. 기존에는 80포트에 대해 아래와 같이 입력했다.

srcIP:127.0.0.1 dscIP:211.111.111.111 srcPort:3030, dscPort:80

이것을 두 가지로 나눠 입력되도록 변경했다. 프린트 타입은 global.h에 아래와 같이 정의해 두었다.

```
#define    MASO_IO_PRINT_TYPE_1    10
#define    MASO_IO_PRINT_TYPE_2    20
extern UINT MAXO_PRINT_TYPE;
```

1번일 경우에는 예전처럼 IP 포트를 입력하도록 하고, 2번일 때에는 포트만 입력되도록 했다. 물론, 이 예제는 IRP를 이용해 드라이버를 제어할 수 있다는 것을 목표로 하기 때문에 간단하게 IP 포트만 입력되도록 하였으나 더욱 더 많은 정보를 출력할 수도 있다.

- 프린트 타입 1일 경우의 예 : srcIP:127.0.0.1 dscIP:211.111.111.111 srcPort:3030, dscPort:80
- 프린트 타입 2일 경우의 예 : srcPort:3030, dscPort:80

● 프린트 루틴 수정

패킷을 조사해 DebugView에 출력해주는 PrintSendPacketInfo를 수정하도록 한다. 위 예제와 같이 정의돼 있다는 가정 하에 〈리스트 4〉와 같이 수정한다.

〈리스트 4〉 PrintSendPacketInf● : 패킷을 조사해 사용자 프로그램이 정한 기준에 따라 출력해주는 함수

```
void PrintSendPacketInfo(PNDIS_PACKET Packet)
{
    // 패킷을 복사해온다.
    NdisQueryPacket(Packet, NULL, NULL, &FirstBuffer,
&TotalPacketLength);

    // Ethernet 헤더를 읽어온다.
    ...

    // IP 헤더를 읽어온다.
    ...

    // TCP 헤더를 읽어온다.
    ...

    // 필터링 옵션에 따라서 출력해준다.
    DesPort=ntohs(pTcpHeader->th_dport);

    for(i=0; i<MASO_MAX_PORT_LIST; i++)
    {
        if( MASO_PORT_LIST[i] == DesPort )
        {
            if( MAXO_PRINT_TYPE == 1 )
            {
                DbgPrint("srcIP:%s dscIP:%s srcPort:%d,
dscPort:%d",
                    SrcIP, DesIP,
                    ntohs(pTcpHeader->th_sport),
                    ntohs(pTcpHeader->th_dport));
            }
            else
            {
                DbgPrint("srcPort:%d, dscPort:%d",
                    ntohs(pTcpHeader->th_sport),
                    ntohs(pTcpHeader->th_dport));
            }
            return;
        }
    }

    return Status;
}
```

● IRP 제어 함수 추가

WDM 인터페이스를 추가하고, 필터링에 필요한 구조체를 정의했으니, 실제 사용자 프로그램과 주고받는 인터페이스를 정의해보자. 우선 4가지의 옵션이 필요하다. 포트 리스트를 삭제하는 옵션, 포트 리스트를 추가하는 옵션, 프린트 타입을 1로 바꾸는 옵션, 그리고 프린트 타입을 2로 변경하는 옵션을 정의해야 한다. 다음과 같이 global.h에 IRP 타입을 정의해둔다.

```
#define MASO_IO_PORT_CLEAR      0
#define    MASO_IO_PORT_ADD     1

#define    MASO_IO_PRINT_TYPE_1    10
#define    MASO_IO_PRINT_TYPE_2    20
```

그리고 이 옵션을 담을 IRP 구조체를 아래와 같이 간단한 struct로 정의한다.

```
typedef struct MASO_IOCTL {
    unsigned char        type;
    unsigned char        port;
} MASO_IOCTL_T;
```

이제 여기에 해당하는 IRP에 대한 코드를 작성할 차례다. 사용자 프로그램에서 WDM를 사용해 명령 할 때에는 CreateFile과 DeviceIoControl 이라는 윈도우 함수를 사용해 호출한다. 그러면 드라이버는 IRP_MJ_DEVICE_CONTROL 이라는 IRP 명령을 받아 처리하게 된다. 여기서는 특히 그 중에서도 buffered I/o라는 방식을 사용해(별도의 설명은 없으므로 디바이스 드라이버 관련 서적 참고) 앞서 정의한 구조체를 넘겨주도록 한다. 그러면 〈리스트 5〉와 같이 IRP 명령에서 앞서 정의한 타입에 따라 값을 채워주면 된다.

〈리스트 5〉 IMIoctl : IRP 처리에서 로직을 추가

```
STATIC NTSTATUS
IMIoctl(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
{
    //
    // IRP에 따라 다른 처리를 해준다.
    //
    switch (irpStack->MajorFunction) {
        // 다른 irp 처리
        ...

    case IRP_MJ_DEVICE_CONTROL:

        //
        // 여기서 원하는 코드를 넣어주도록 한다.
        //

        ioControlCode = irpStack-
>Parameters.DeviceIoControl.IoControlCode;
        switch (ioControlCode) {

       // This is where you would add your IOCTL handlers

        case IOCTL_SIOCTL_METHOD_BUFFERED :        //
buffered i/o.

            // 버퍼에서 값을 읽어온다.
            inputBuf  = Irp->AssociatedIrp.SystemBuffer;
            outputBuf = Irp->AssociatedIrp.SystemBuffer;

            if(inputBufferLength != 0)
            {
                //
                // 사용자 프로그램에서 패킷 하나를 보낸다.
                // 패킷을 받아 원하는 작업을 한다.
                //
                usrMsg = (MASO_IOCTL_T*)inputBuf;
                switch(usrMsg->type)
                {
                case MASO_IO_PORT_CLEAR :
                    //
                    // 포트에 있는 값을 모두 없애는 기능이다.
                    // 포트 배열의 인덱스를 0으로 지정해 해결한다.
                    //

                    DbgPrint("PASSTHRU:
MASO_IO_PORT_CLEAR");
                    MASO_PORT_CNT=0;
                    break;

                case MASO_IO_PORT_ADD:
                    //
                    // 포트에 값을 넣어주도록 한다.
                    // 포트 배열의 인덱스를 하나 올려준다.
                    //

                    DbgPrint("PASSTHRU: MASO_IO_PORT_ADD");
                    if( MASO_PORT_CNT<MASO_MAX_PORT_LIST )
                    {

MASO_PORT_LIST[MASO_PORT_CNT]=usrMsg->port;
                        MASO_PORT_CNT++;
                        DbgPrint("PASSTHRU: PORT %d ADDED",
usrMsg->port);
                    }
                    else
                    {
                        DbgPrint("PASSTHRU: PORT CNT
EXCEEDED %d", MASO_MAX_PORT_LIST);
                    }
                    break;

                case MASO_IO_PRINT_TYPE_1:
                    //
                    // 프린트 타입을 1로 맞춰준다.
                    // IP만 입력되도록 하는 옵션이다.
                    //

                    DbgPrint("PASSTHRU: CHANGE PRINT TYPE TO
1 (show IP)");
                    MAXO_PRINT_TYPE=1;
                    break;

                case MASO_IO_PRINT_TYPE_2:
                    //
                    // 프린트 타입을 2로 맞춰준다.
                    // 포트만 입력되도록 하는 옵션이다.
                    //

                    DbgPrint("PASSTHRU: CHANGE PRINT TYPE TO
2 (show IP, PORT)");
                    MAXO_PRINT_TYPE=2;
                    break;

                default:
                    break;
```

```
                }
            }
            break;
        default:
            ImDbgOut( DBG_INFO, DBG_IO,
                    ("unknown IRP_MJ_DEVICE_CONTROL\n =
%X\n",ioControlCode));
            Status = STATUS_INVALID_PARAMETER;
            break;
        }
        break;
```

```
    default:
        ImDbgOut(DBG_INFO, DBG_IO,
                ("unknown IRP major function = %08X\n",
irpStack->MajorFunction));

        Status = STATUS_UNSUCCESSFUL;
        break;
    }

    return Status;
}
```

드라이버 제어 사용자 프로그램 작성

이제 지난 호에서 사용했던 PassThruHelper를 업그레이드할 차례다. 〈화면 1〉과 같이 UI 변경으로 필터링 부분을 추가한다.

〈화면 1〉
PassThruHelper

UI 설명

〈화면 1〉처럼 새로 만든 사용자 프로그램은 기존의 프로그램에 비해 버튼이 많아졌다. 각각의 버튼은 다음과 같은 역할을 한다.

+ **Insert** : 리스트에 새로운 포트 숫자를 추가한다. 〈그림 1〉처럼 80, 8080, 88이 있으면 그에 해당하는 포트에 대한 필터링을 한다는 뜻이다.
+ **remove** : 리스트에서 필터링을 하고 싶지 않은 포트를 선택하고, remove 버튼을 누르면 해당 포트가 리스트에서 삭제된다.
+ **Update PortList** : 실제로 드라이버와 통신을 한다. Insert, Remove 버튼으로 작성한 포트 리스트를 드라이버에 알려준다.
+ **Print 1** : 드라이버와 통신하는 버튼. 1번 방식(ip 보여주기)로 프린트한다.
+ **Print 2** : 드라이버와 통신하는 버튼. 2번 방식(포트 보여주기)로 프린트한다.

사용자 프로그램에서 드라이버와 통신

드라이버와 통신할 때는 CreateFile과 DeviceIoControl이라는 윈도우 함수를 사용하게 된다. CreateFile은 파일을 만들 때 사용하는 윈도우 라이브러리로 드라이버 접근은 파일 접근과 동일한 방식으로 하면 된다. 다만, 인자는 조금 다르게 넣어줘야 한다.

```
HANDLE hDevice = CreateFile("\\\\.\\PassThru",
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL,
    NULL);
```

위와 같이, 파일 경로를 넣어주는 자리에 드라이버 이름을 넣어주면 된다. 이제 받은 핸들을 가지고 DeviceIoControl로 값을 넘겨준다.

〈리스트 6〉 DeviceIoControl : 윈도우 라이브러리인 DeviceIoControl을 사용하기 쉽게 감싸준 함수. 드라이버를 호출하는 역할을 한다.

```
bool CPassThruHelperDlg::DeviceIOControl(HANDLE hDevice,
MASO_IOCTL_T iBuf)
{
    ULONG bytesReturned;
    bool status;

    memset(m_sInBuf, 0, sizeof(m_sInBuf));
    memcpy(m_sInBuf, (unsigned char*)&iBuf, sizeof(iBuf));

    status = DeviceIoControl(hDevice,
        (DWORD) IOCTL_SIOCTL_METHOD_BUFFERED,
        &m_sInBuf,
        sizeof(MASO_IOCTL_T),
        &m_sOutBuf,
        sizeof(m_sOutBuf),
        &bytesReturned,
        NULL);

    return status;
}
```

앞에서 정의한 MASO_IOCTL_T 구조체에 값을 넣고 그 구조체를 〈리스트 6〉과 같이 DeviceIoContorl의 인자로 넣어주면 드라이버를 호출한다.

〈리스트 7〉 OnButtonWdmPort : 버튼을 누를 경우 호출되는 함수. 포트 리스트를 드라이버에 넘겨준다.

```
void CPassThruHelperDlg::OnButtonWdmPort()
{
    HANDLE    hDevice;
    CString strTmp;

    //
    // 일반적인 파일을 만들듯 CreateFile을 통해 접근한다.
    // 첫 인자로 파일의 경로를 넣어주는 것이 아니라
    // 드라이버 이름을 넣어주도록 한다.
    //
    hDevice    =CreateFile("\\\\.\\PassThru",
            GENERIC_READ | GENERIC_WRITE,
            0,
            NULL,
            CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL,
            NULL);

    if (hDevice == INVALID_HANDLE_VALUE) {
        strTmp.Format("Failed to obtain file handle to
device(PassThru)\
                        with Win32 error code: %d",
GetLastError() );
        MessageBox(strTmp);
        return;
    }

    //
    // IRP 구조체를 만들어 채워준다.
    // Type에서 Port의 리스트를 없애주도록 한다.
    //
    MASO_IOCTL_T    iBuf;
    iBuf.type = MASO_IO_PORT_CLEAR;

    bool status;
    status = DeviceIOControl(hDevice, iBuf);

    //
    // IRP 구조체를 만들어서 채워준다.
    // Type에서 Port의 리스트를 추가해준다.
    //
    iBuf.type = MASO_IO_PORT_ADD;

    for(int i=0; i<m_lstPort.GetCount(); i++)
    {
        m_lstPort.GetText(i, strTmp);
        iBuf.port=atoi(strTmp);

        status = DeviceIOControl(hDevice, iBuf);
    }

    CloseHandle(hDevice);
}
```

〈리스트 7〉은 포트 리스트를 넘겨주는 함수다. CreateFile을 사용해 드라이버에 접근을 만들고, MASO_IO_PORT_CLEAR를 명령해 기존에 드라이버가 가지고 있는 포트 목록을 삭제한다. 그리고 MASO_IO_PORT_ADD 명령과 포트 번호를 넣은 MASO_IOCTL_T 구조체를 만들어 드라이버로 포트 목록을 넘겨주도록 한다.

〈리스트 8〉 OnButtonWdmPrint1 : 프린트 옵션을 바꿔주는 함수

```
void CPassThruHelperDlg::OnButtonWdmPrint1()
{
    HANDLE    hDevice;
    CString strTmp;

    //
    // 일반적인 파일을 만들듯 CreateFile을 통해 접근한다.
    // 첫 인자로 파일의 경로를 넣어주는 것이 아니라
    // 드라이버 이름을 넣어주도록 한다.
    //
    hDevice    =CreateFile("\\\\.\\PassThru",
            GENERIC_READ | GENERIC_WRITE,
            0,
            NULL,
            CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL,
            NULL);

    if (hDevice == INVALID_HANDLE_VALUE) {
        strTmp.Format("Failed to obtain file handle to
device(PassThru)\
                        with Win32 error code: %d",
GetLastError() );
        MessageBox(strTmp);
        return;
    }

    //
    // IRP 구조체를 만들어 채워준다.
    // Type에서 Print를 사용하도록 해준다.
    //
    MASO_IOCTL_T    iBuf;
    iBuf.type = MASO_IO_PRINT_TYPE_1;

    //
    // DeviceIOControl을 불러준다.
    // 실제로 IRP 구조체를 넘겨주게 된다.
    //
    bool status;
    status = DeviceIOControl(hDevice, iBuf);
```

```
    CloseHandle(hDevice);
}
```

〈리스트 8〉은 프린트 타입을 바꿔주는 함수다. 〈리스트 7〉과 기본적으로 동일한 구조를 가지고 있다. MASO_IOCTL_T 구조체에서 인자를 프린트 설정에 관한 것으로 넣어주면 된다.

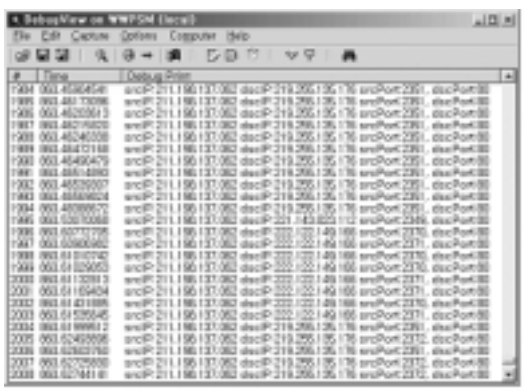
설치&테스트

지난 시간에는 드라이버를 설치한 후 재부팅하면 DebugView를 통해 바로 결과를 확인할 수 있었다. 이번에도 새로운 프로그램을 사용해 여러 가지 테스트를 해보도록 한다. 기본적으로 다음과 같은 순서로 한다.

1. PassThruHelper의 Uninstall 버튼을 클릭하여 기존 드라이버 삭제
2. 재부팅
3. PassThruHelper의 Install 버튼을 눌러 새로운 드라이버 설치(이때 드라이버에 해당하는 inf가 드라이버와 같은 폴더에 존재해야 함)
4. 재부팅
5. PassThruHelper의 새로 만든 기능 버튼을 클릭하며 DebugView로 테스트

포트 테스트

80, 8080, 88포트를 찾도록 테스트 해본다. 설치 & 테스트 순서의 5번에서 UI의 리스트에 80, 8080, 88을 넣고, PassThru Helper에서 UpdatePortList 버튼을 누른다. 그리고 DebugView를 실행한 후 인터넷 사이트를 서핑하면 〈화면 2〉와 같이 IP가 나타나는 것을 확인할 수 있을 것이다.



〈화면 2〉 ip, 포트가 잡히는 화면

프린트 테스트

PassThruHelper에서 Print1 버튼과 Print2 버튼을 눌러가며 웹서핑을 해본다. 그리고 DebugView를 확인하면 다른 로그가 입력되는 것을 발견할 수 있다. 실시간으로 드라이버가 사용자 프로그램에 의해 제어가 되는 좋은 예다.

이로써 3회에 걸쳐 네트워크 드라이버 프로그래밍에 대해 알아보았다. 연재를 시작하면서 사용자가 다가가기 쉬운 드라이버를 만들어 보자는 것이었는데, 기획 의도대로 많은 도움이 되었으면 하는 바람이다. 윈도우2000 환경에서 DDK만 있다면 지금까지 배워보았던 내용을 컴파일하고 테스트해 볼 수 있을 것이다.

이번 연재에서는 기본 이론을 시작으로 예제 소스구조, 컴파일, 제어까지 조금씩 모두 다뤄보았다. 드라이버 개발시 여기서 소개된 내용을 기반으로 하여 참고 자료를 활용한다면 개발에 많은 도움이 될 것으로 생각된다.

이달의 디스켓 : NDIS.zip

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/)