WDM Filter Driver

written by Song Jiho(songjiho@devguru.co.kr)

1. 개요

필자가 WDM 관련된 강의를 하면서 자주 접하는 질문들이 있다.

바로 이번에 소개하려고 하는 필터(filter)드라이버에 관한 질문이 그 중 대표적인 내용이다.

드라이버를 작성하는 이유나 사용용도는 하드웨어를 직접 제어해야 하는 필요성 때문일 경우도 있지만 이미 나와 있는 기존 드라이버들의 동작내용을 확인하거나 기존 드라이버를 제어해야 하는 필요성 때문인 경우도 많기 때문인 것 같다.

필터 드라이버는 이럴 경우 가장 적합한 방법이 될 것이다.

필터 드라이버의 실제 사용용도를 먼저 살펴보고 예제를 통해 간단한 필터 드라이버를 작성해 필터 드라이버를 이해해 보자.

필터 드라이버의 용도는 사용자가 사용하기 나름이겠지만 다음과 같은 필요에 의해 사용할 수 있다.

1-1 기존의 드라이버에 대한 I/O를 감시

1-2 기존의 드라이버에 대한 I/O를 수정(통제)

1-3 제공되는 드라이버와 하드웨어간에 문제해결

각각에 대해 조금 더 구체적인 예를 들어보면 사용용도나 필요성에 대해 쉽게 이해가 될 것 이라고 생각된다. 또한 편의상 위의 세가지 정도의 기능을 이번 컬럼 에서는 필터링 이란 단어로 대체해서 컬럼을 진행하겠다.

1-1번과 1-2번 항목의 경우 이번 컬럼의 예제로 간단한 골격이 제공 되겠지만 사내의 자료유출에 대한 보안책으로 저장장치에 대해서 누가 어떤 내용을 복사했는지 기록을 남기거나(log), 또는 인가 받지 않은 사용자가 저장장치에 대해서 접근할 수 없게 하려면 필터 드라이버를 작성해서 탐색기(exploer.exe)와 같은 프로그램에서 플로피와 같은 저장장치로 어떠한 내용을 I/O했는지 감시하거나 인가 받지 않은 사용자의 접근을 제어 하는 방법을 사용할 수 있다.

1-3번 항목의 경우 SCSI (Small Computer System Interface) 장치를 예로 들어 볼수 있다. SCSI하드웨어가 한번에 4K보다 적은양의 데이터를 전송하게 설계되어 있는데 기존의 드라이버에서는 이러한 내용을 감안하지 않고 작성되어 있을 때 필터 드라이버를 중간에 삽입해서 전송되는 데이터가 4K 보가 클 경우 드라이버가 4K단위로 데이터를 나눠서 전송을 하게 수정할 수 있다. 또한 요즘 많이 사용하는 USB MassStorage장치의 경우 OS가 제공하는 드라이

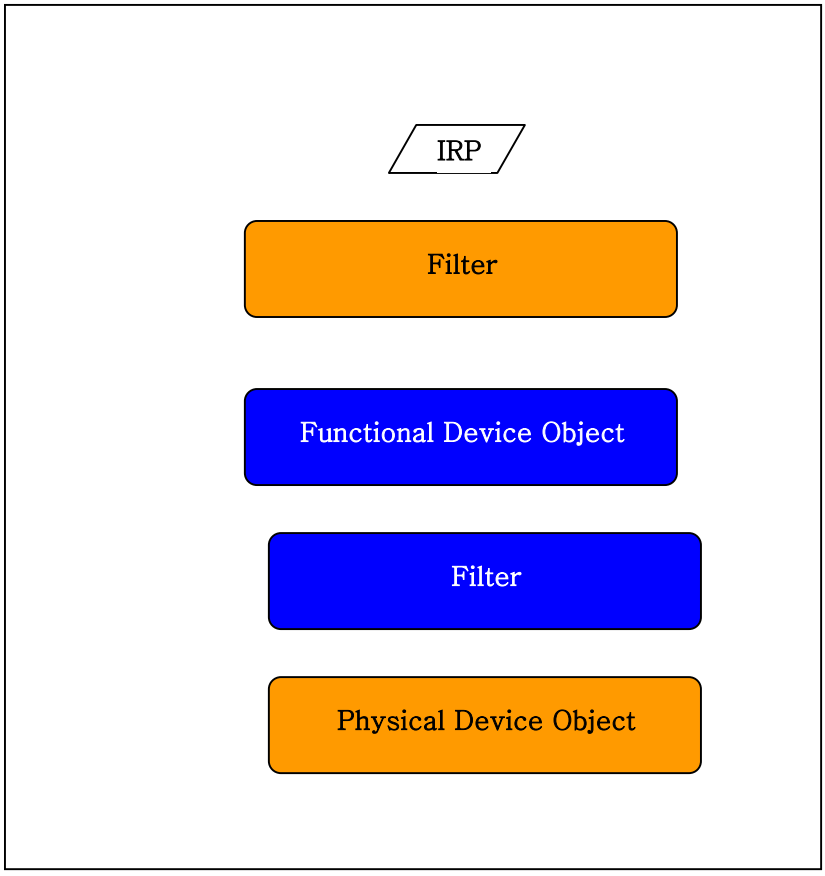
버를 사용하므로 정해진 방법대로밖에 I/O를 할 수 없으나 여기에 필터 드라이버를 삽입해서 다른 저장공간에 나만의 데이터를 저장할 수도 있다.

이밖에도 필터 드라이버를 잘 응용하면 여러 가지 다양한 분야로 응용이 가능할 것이다. 그럼 부디 이번 컬럼을 통해서 필터 드라이버에 대한 개념을 알아보고 작성 방법에 관한 기본 지식들을 습득하기를 바란다.

2. WDM Filter Driver의 구조

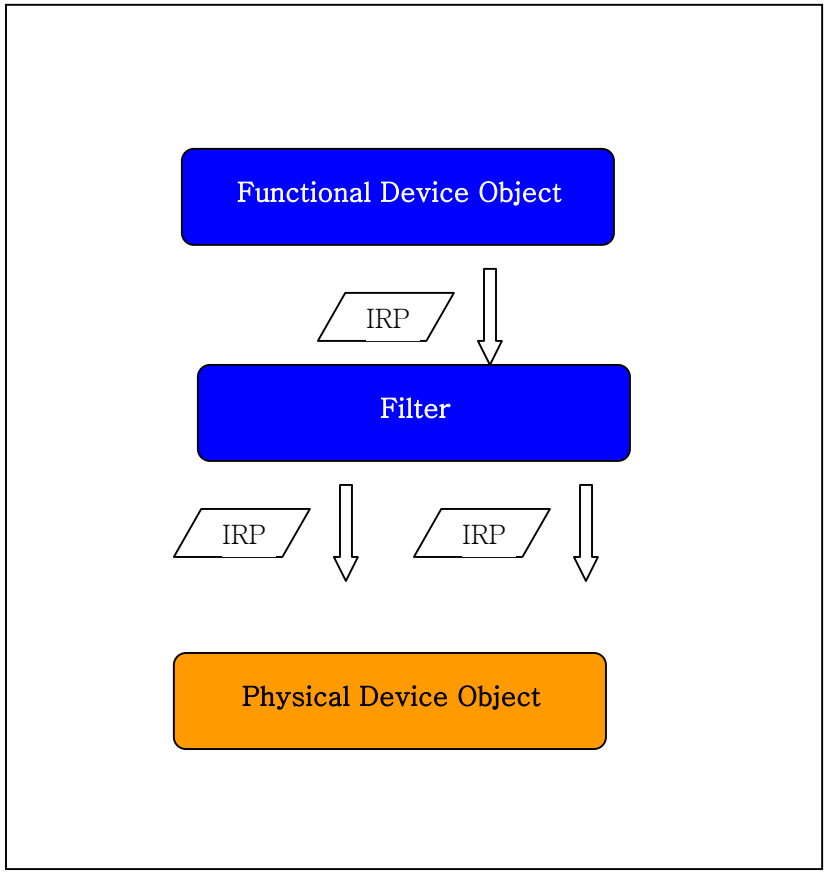
2-1 구조

WDM 드라이버는 계층화된(layered) 드라이버 구조를 가지고 있다. 그렇기 때문에 Filter Driver도 이러한 WDM의 계층 속에 포함되게 작성하는 방법이 일반적이다. 개념적으로 filter driver는 다음의 그림 2-1과 같은 모양을 가지고 있다.



[그림 2-1]Filter Driver의 Stack 구조

기본적인 구조(파란색 부분)인 FDO와 PDO의 구조에서 필터링 하기 원하는 곳에 필터 드라이버가 위치하게 되고 필터 드라이버는 FDO와 PDO를 거쳐가는 모든 데이터를 보거나 변경할 수 있게 된다. 상기 1절의 3)항에서 설명한 내용은 그림 2-2 와 같은 모습을 하게 된다.



Device Node

[그림 2-2] Filter Driver 예제

2-2 필터드라이버의 종류

WDM 필터 드라이버는 크게 Class Filter, Device Filter, Bus Filter의 세가지 종류로 구분할 수 있고 각각은 또한 설치되는 위치에 따라 Lower Filter와 Upper Filter 두 가지로 나눠 질 수 있다. 또한 WDM 필터 드라이버들은 레지스트리에 자신의 종류와 위치를 기록함으로서 원하는 시점에 원하는 드라이버 계층에 포함(Load)되어 질 수 있다.

우선 종류에 따라 이들 각각을 간략히 살펴보면 다음과 같은 특징이 있다.

1) Class Filter

주어진 Class에 모든 드라이버들이 Load 될때 함께 Load되어서 모든 Class의 드라이버들

에 대한 필터링을 할 수 있다. 이 위치에 설치된 필터 드라이버는 단지 Class의 영향을 받을 뿐이고 하드웨어 ID나 제조사 심지어는 Bus Type과도 무관하게 설치되기 원하는 Class의 다른 드라이버가 Load될때 함께 Load 되어 진다.

예를 들어 우리가 Mouse Class에 대한 Class 필터 드라이버로 우리가 제작한 MyMouse.sys 라는 드라이버를 등록한다면 USB Mouse 드라이버가 Load될 때 MyMouse.sys가 load 될것이고 Serial Mouse가 설치될 때도 MyMouse.sys가 load되어 결국 모든 Mouse에 대한 필터링이 가능해진다.

2) Device Filter

Class filter와는 다르게 특정 device node에만 설치되는 필터 드라이버이다. 예를 들어 VID가 1234이고 PID가 5678인 USB 장치에 대한 필터링이 필요할 경우 이 장치에 대한 Device Filter를 설치해서 다른 장치에 대한 I/O는 받지 않고 단지 원하는 장치에 대해서만 필터링이 가능하게 할 수 있다.

3) Bus Filter

말 그대로 특정 BUS Driver에 대해서 필터링이 가능한 드라이버이다. 위의 2)번 항목 에서는 USB 장치중 특정 장치만을 필터링 할 수 있지만 BUS Filter Driver를 USB BUS Filter Driver로 설치하면 모든 USB 장치에 대한 내용들을 필터링 할 수 있다.

이러한 세가지 종류의 필터 드라이버는 각각마다 설치되는 위치에 따라 또다시 Upper Filter와 Lower Filter로 구분되어 진다. 2)번 항목의 Device Filter에서 필터링하려는 드라이버보다 먼저 I/O 요청을 받아서 필터링하기 원하면 Upper Filter로 설치하게 되고 필터링 하려는 드라이버에서 처리가 끝난 I/O요청에 대해서 필터링 하려면 LowerFilter로 설치하게 된다.(단 Bus Filter Driver는 Upper Filter Driver만 존재한다.)

1) LowerFilter

LowerFilter driver는 PDO(Physical Driver Object)로 전달되는 I/O를 가로채서 살펴보거나 수정하는 용도로 사용한다. 1-3의 기능을 수행하기에 적당하다.

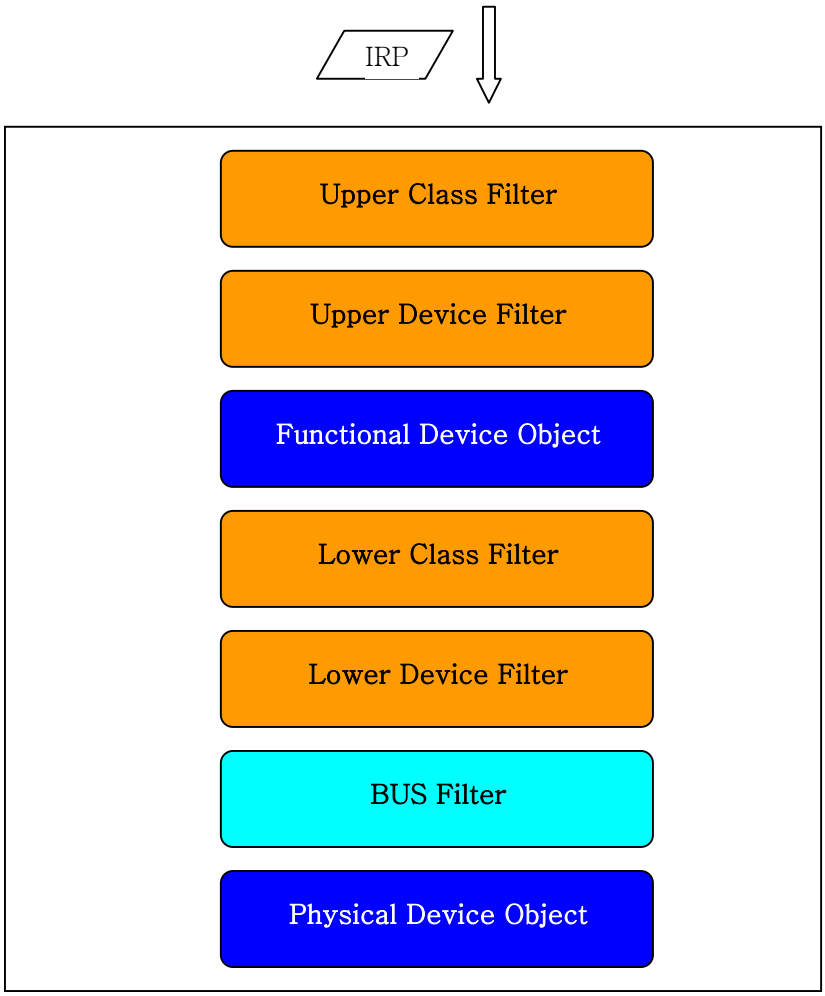
2) UpperFilter

FDO(Function Device Object)로 전달되는 I/O를 가로채서 살펴보거나 수정하는 용도로 사용한다. FDO로 전달되는 I/O요청을 수정하거나 새로운 I/O요청으로 변경하는데 사용되기에 적당하다.

이러한 드라이버들은 Class Filter, Device Filter, Bus Filter의 순서로 설치되게 된다.

이상의 내용을 그림으로 정리해 보면 다음과 같다. 드라이버들의 위치와 순서를 잘 생각해 보면서 다음 “그림2-3”을 살펴보기 바란다.

s



[그림 2-3] Device Stack 구조

3. 작성법

3-1 기본 수칙

빌드 하는 방법은 일반적인WDM드라이버 빌드 방법과 동일하다.

코드 상에서는 다음과 같은 점들을 주의 해야 한다.

주의 1) PnP와 Power 관련된 루틴들은 조심해서 다뤄야 한다.

주의 2) 작성되는 드라이버가 올라가는 Device Stack를 정확히 알고 있어야 하며 characteristics는 수정하면 안 된다.

주의 3) 수정 하려고 하는 기능이외의 다른 IRP들은 건드리면 안 된다.

주의 4) 모든 dispatch routine들에 대해서 처리해주어야 한다.
주의 5) 정의 되지 않았거나 처리하지 않는 모든 IRP에 대해서는 반듯이 밑으로 전달(PassDown) 해 주어야 한다.

3-2 DriverEntry routine

3-1의 주의점들을 감안하면 (특별히 주의4)항목) DriverEntry루틴은 "표3-1"과 같은 모습을 가지게 된다.

```
ULONG i;
UNICODE_STRING uniNtNameString, uniWin32NameString;
NTSTATUS status;
PDEVICE_OBJECT filterDevObj = NULL;

PAGED_CODE();

UNREFERENCED_PARAMETER(RegistryPath);

DBGOUT(("DriverEntry"));

/*
 *  Route all IRPs on device objects created by this driver
 *  to our IRP dispatch routine.
 */
for (i = 0; i <= IRP_MJ_MAXIMUM_FUNCTION; i++){
    DriverObject->MajorFunction[i] = VA_Dispatch;
}

DriverObject->DriverExtension->AddDevice = VA_AddDevice;
DriverObject->DriverUnload = VA_DriverUnload;

return STATUS_SUCCESS;
```

[표 3-1] DriverEntry

3-3 AddDevice Routine

AddDevice 루틴은 흔히 TopOfStackObject로 불리는 DeviceObject를 얻어와서 보관해야 한다. 종종 드라이버소스 코드들 중에 하위 드라이버 IoCallDriver함수를 호출할 때 처음인자로 사용하는 DeviceObject로 BUS Driver가 제공하는 PDO(Physical Device

Object)를 사용하는 코드들이 있다. 물론 이 코드들은 로직상 특별한일이 없으면 동작에 무리가없는 경우가 대부분이지만 특별히 필터 드라이버에서는 PDO로 직접 IoCallDriver()함수를 호출해서는 안된다.

그리고 AddDevice 루틴 에서는 하위드라이버의 Device object flag들을 그대로 사용해야 하므로 flag들(DO_POWER_PAGABLE,DO_BUFFERED_IO,DO_DIRECT_IO 등등)과 관련된 코드들이 일반적인 WDM 드라이버와 달라진다.

코드를 살펴보면 다음 표3-2와 같다. 중요한 부분은 코드상에 주석으로 설명을 대신하도록 한다.

```
filterDevObj->Flags |=(devExt->topDevObj->Flags & (DO_BUFFERED_IO | DO_DIRECT_IO));
filterDevObj->Flags|=(devExt->topDevObj->Flags&(DO_POWER_INRUSH | DO_POWER_PAGABLE));
```

[표 3-4] IRP 처리

모든 IRP들은 IoAttachDevieToDeviceStack()함수가 리턴 해준 DeviceObject쪽으로 IoCallDriver()함수를 사용해서 전달되어야 하며 우리는 우리가 관심을 가지는 IRP에 대해서만 변경하거나 처리 방법을 바꾸어야 한다.

다음 표 3- 에서 플로피로 Read/Write되는 값들에 대해서 몇 byte만큼 Read/Write했는지 디버깅 메시지를 출력하는 코드를 소개한다. 전체 코드는 www.devguru.co.kr에서 자료실에 있는 플로피 필터 드라이버 소스를 참고해서 확인해 보기 바란다.

4. 설치 및 실행 방법

우선 제공되는 예제는 편의를 위해 windows 2000 DDK의 srcWwdmWgeneralWfilter에 있는 예제를 기본골격으로 사용하였고 대부분의 주석이나 코드에 수정을 가하지 않았음을 밝힌다. 이 예제를 참고하면서 전체 코드를 살펴보기 바란다. 또한 이 코드는 windows2000 advenced server 에서 작성되었고 다른 OS와의 호환성은 검토되지 않았다.

4-1 설치 및 실행 방법

필터 드라이버도 WDM으로 작성되었으므로 WDM드라이버를 설치하듯이 확장자가 inf인 설치 파일을 사용해서 드라이버를 설치할 수 있다.

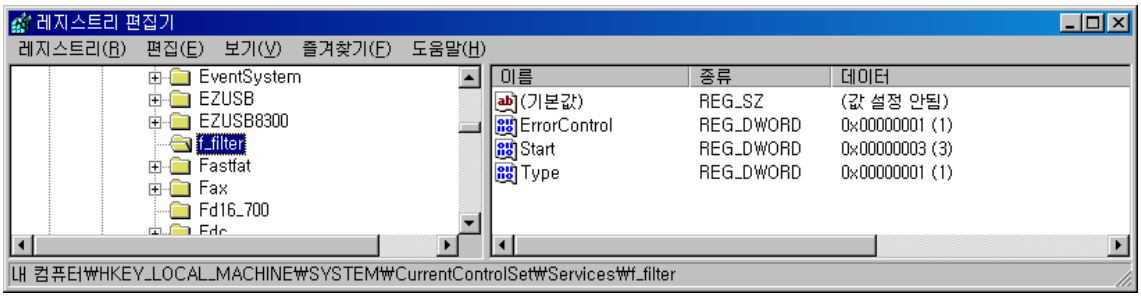
물론 SetupDiGetClassDevs() 함수나 SetupDiEnumDeviceInfo(), SeupDiGetDeviceRegistryProperty(), SetupDiSetDeviceRegistryProperty()함수등을 사용해서 프로그램적으로 설치할 수도 있다. 이 방법은 독자들에게 과제로 남겨 두고 이번 컬럼 에서는 수동으로 드라이버를 설치 하는 방법에 관해서만 다루도록 하겠다. 관심이 있는 독자는 DDK의 srcWgeneralWtoasterWfilter나 srcWwdmWgeneralWfilter 에 있는 소스를 참조하기 바란다.

또한 filter driver와 어플리케이션과의 통신에 관한 MS의 권고사항은 KB Article에 있는 Article ID 262305 인 문서(이 문서의 현재 링크는 http://support.microsoft.com/default.aspx?scid=kb;EN-US;Q262305 로 되어 있다.)

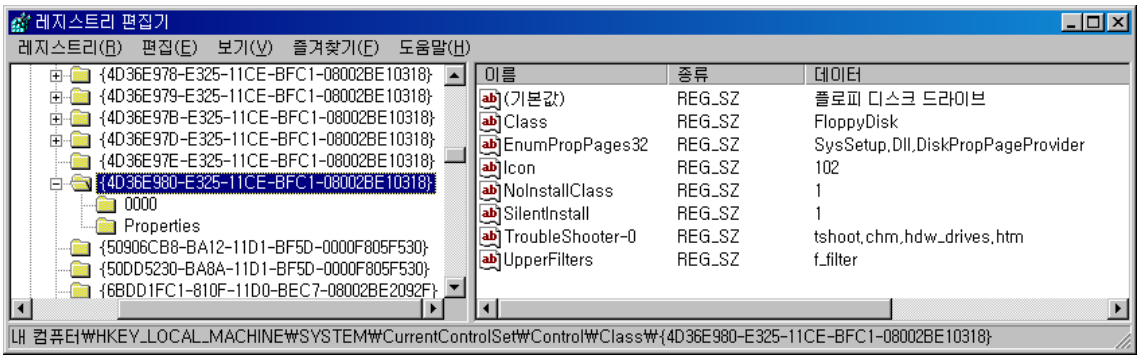
를 참조해서 DevGuru에서 제공되는 샘플과 비교해 보기 바란다.

테스트 방법은 첨부되는 Testapp.exe를 실행한 후 Install 버튼을 누른다. 이때 필터 드라이버에 관한 서비스키와 f_filter.sys 라는 파일을 복사하는 작업이 이루어진다. 또한 f_filter.sys파일은 Testapp.exe와 같은 곳에 있어야 한다.

설치가 잘 되었다면 레지스트를 확인해 보자. regedit.exe를 실행해서 레지스트리를 살펴보면 다름 [그림 4-1]과 [그림 4-2]와 같은 내용을 확인할 수 있다.



[그림 4-1] Service Key



[그림 4-2] Class Key

Install 후 메시지에서 나오는 것과 같이 재 부팅 후 다시 testapp.exe를 실행하고 Do Not Access 버튼을 클릭한 후 플로피에 접근해 보면 정상적으로 사용할 수 없음을 확인할 수 있다. 다시 Access OK버튼을 클릭하고 플로피 디스크에 접근해 보면 정상적으로 플로피 디스크를 사용할 수 있다.

그림 4-2를 보면 우리의 f_filter.sys 드라이버는 FloppyDisk드라이버의 상위에 설치되는 UpperFilter드라이버로 동작한다는 것을 알 수 있다.

Do Not Access 버튼과 Access OK버튼을 클릭할 때 에는 어플리케이션에서는 DeviceIoControl()함수를 사용해서 f_filter.sys드라이버쪽으로 단순한 컨트롤코드만을

넘겨주고 이를 받은 드라이버에서는 다음 표 4-1과 같이 플래그를 설정해서 다음번 Read 요청에 관해서 아랫단 드라이버로 전달할지 UNSUCCESS로 완료할지를 결정한다.

플로피드라이버의 경우 많은 부분을 IRP_MJ_DEVICE_CONTROL을 통해서 처리한다. 제공되는 샘플은 IRP_MJ_DEVICE_CONTROL에 관해서 처리하지 않기 때문에 Do Not Access 버튼을 클릭한 후에도 폴더 목록등은 여전히 보일 것이다. 하지만 파일들을 실행한다거나 editor등에서 파일을 열어 볼때는 파일을 읽을 수 없다는 메세지를 보게 될 것이다. 주요 처리코드는 표 4-1과 같다.

```
switch (majorFunc){

    case IRP_MJ_PNP:
        status = VA_PnP(devExt, Irp);
        passIrpDown = FALSE;
        break;

    case IRP_MJ_POWER:
        status = VA_Power(devExt, Irp);
        passIrpDown = FALSE;
        break;

    case IRP_MJ_READ:
        if(devExt->bAccess == FALSE){
            status = Irp->IoStatus.Status = STATUS_UNSUCCESSFUL ;
            IoCompleteRequest(Irp, IO_NO_INCREMENT);
            passIrpDown = FALSE;
        break;
        }
    case IRP_MJ_DEVICE_CONTROL:
    {
        ioControlCode = irpSp->Parameters.DeviceIoControl.IoControlCode;

        if(ioControlCode == IOCTL_ACCESS_OK){
            devExt->bAccess = TRUE;
            status = Irp->IoStatus.Status = STATUS_SUCCESS;
            IoCompleteRequest(Irp, IO_NO_INCREMENT);
            passIrpDown = FALSE;
            break;
```

```
        }
        else if(ioControlCode == IOCTL_DO_NOT_ACCESS){
            devExt->bAccess = FALSE;
            status = Irp->IoStatus.Status = STATUS_SUCCESS;
            IoCompleteRequest(Irp, IO_NO_INCREMENT);
            passIrpDown = FALSE;
            break;
        }
        break;
    }


    case IRP_MJ_CREATE:
    case IRP_MJ_CLOSE:
    case IRP_MJ_SYSTEM_CONTROL:
    case IRP_MJ_INTERNAL_DEVICE_CONTROL:

                - 생략-
```

[표 4-1]

한번씩 실행해 보고 필요한 부분을 수정하거나 디버깅해보기 바란다. 이를 조금 더 잘 다듬고 어플리케이션을 확장한다면 간단한 플로피입출력통제 시스템이 될 것이다.

5. 결론

위에서 설명한 방법을 통해 WDM 필터 드라이버를 작성 할 수 있다. 물론 여기에 소개된 내용들 보다는 수정해야 할 부분들이 더 많이 있겠지만 기본 골격과 개념에는 크게 변화가 없으리라 생각한다.

이번 컬럼에서 제시한 방법 외에도 조금 고전적인 방법으로 함수포인터를 직접 수정해서 필터링을 하는 방법도 있다. 필자 개인적인 생각으로는 굳이 그러한 작업이 필요한때가 아니라면 MS에서 제시하는 표준방법(이번 컬럼에서 소개한 방법)으로 필터 드라이버를 작성하는 것이 바람직할 것이다.

부디 이번 컬럼이 필터드라이버의 개념을 설정하는데 도움이 되기를 바라며 또한 여러분들의 드라이버를 디자인할 때 한번쯤 필터 드라이버로 작성이 가능한지 고민해 볼 수 있는 계기가 되기를 희망한다.