Windows File System Filter Drivers 작성 법 I

written by khealin(jgkim@devguru.co.kr)

여행을 시작하면서...

Windows NT, 2000, XP(이하 Windows라고 말하겠다) O/S상에서 File System Filter Driver를 작성하는 것은 그리 간단한 작업은 아니라는 것은 모두 잘 아는 사실이다. 왜냐하면 어떤 filter driver를 작성하던 간에 filtering하고자 하는 Target Device의 동작 특성을 잘 이해하여야만 제대로 된 filter driver를 작성할 수 있기 때문이다. 이 논지를 이해하기 위해 filter driver에서 IRP_MJ_CREATE를 트랩(trap)하기 위해 complete routine을 등록하고 file system driver로 이 Create 요청을 전달(pass down)한다고 가정하자.
이 때 필터 드라이버의 Create Dispatch 루틴을 <그림-1> 같이 대체로 작성할 것이다.

```
...
IoCopyCurrentIrpStackLocationToNext(Irp);
IoSetCompletionRoutine(
            Irp,
            (PIO_COMPLETION_ROUTINE) CompletionRoutine,
            pdx, TRUE, TRUE, TRUE
            );
return IoCallDriver( NextDeviceObject, Irp);
```

그림-1

그리고 CompletionRoutine은 <그림-2>와 같이 작성할 것이다.

```
// 추가적인 작업을 위해 WorkItem을 사용한다.
ExQueueWorkItem(&workItem);
return STATUS_MORE_PROCESSING_REQUIRED;
```

그림-2

이제 위의 제시된 코드들의 유효성에 대해서 한 번 생각해보자 먼저 DDK에는 크게 언급이 없지만 I/O Manager는 IRP_MJ_CREATE를 synchronous I/O Operation으로 취급을 한다. 따라서 이 사실을 알지 못한다면 다음과 같은

문제점이 존재할 수 있다.
<그림-2>의 루틴에서 STATUS_MORE_PROECESSING_REQUIRED를 리턴함으로서 IoCompIteRequest가 리턴하게 되고 main line의code <그림-1>도 리턴하게 된다. 하지만 이때 Create Diaptch 루틴이 I/O Manager에게 리턴하는 값은 file system driver의 리턴 값이다. 만약 이 리턴 값이 STATUS_PENDING이 아니라면 I/O Manager는 이제 synchronous I/O 인 IRP_MJ_CREATE에 대해서 IRP 메모리 해제 작업에 들어 간다. 그러나 메모리 해제 과정이 필터 드라이버의 work item보다 먼저 수행이 된다면 work item에서 호출하는 IoCompleteRequest는 아마도 Blue Screen Of Death(BSOD)를 야기하게 될 것이다. 따라서 이에 대한 해결책은 크게 두 가지가 될 것 같다.

가. 필터 드라이버에서 STATUS_PENDING을 리턴하던지
나. I/O request를 synchronous로 만들던지.

간단한 예제를 통해 필터 드라이버를 작성할 때 왜 필터링하고자 하는 대상 드라이버와 I/O Manger의 특성 등을 제대로 이해해야 하는지를 살펴보았다.

하지만 이러한 사실들에 대해서 특히 그 대상이 파일 시스템 드라이버인 경우라면 더욱더 우리는 M/S로부터 공식화된 자료를 기대하기 어렵다. 우리가 겨우 접할 수 있는 자료는 Installable File Systems Development Kit에 존재하는 샘플 자료들일 것이다. 긴 겨울이 또 시작된다. 이 겨울이 끝나기 전 기회가 된다면 몇 번에 걸쳐 이제 file system filter driver 작성에 대한 몇 가지 이야기를 하고자 한다. 즐거운 추억이 남는 겨울이 되기를…

여행을 시작하면서... Storage Device Stack

이해를 돕기위해 IFS-Kit에 설명되어 있는 CD-ROM Device와 CDFS(CD-ROM File System)을 예를 들어 설명하기로 한다. 먼저 Windows 부팅 시 CD-ROM device를 위한 <그림-3>과 같은 CD-ROM Storage Device Stack이 생성될 것이다. WDM 드라이버에 대한 간단한 지식이 있다면 <그림-3>을 이해하는 데는 무리가 없을 것이다. 이 시점은 아직 CDFS는 메모리에 로드되지 않았고 단지 CD-ROM device를 위한 Function 드라이버까지 로드된 상태이다. 그리고 CD-ROM에 대한 드라이버 레터와 I/O Manager에 의해서 관리되는 “Device₩CdRom%d” 가 Object name space에 생성되어있

는 상태이다. 그러면 과연 우리가 filtering하고자 하는 CDFS는 언제, 어떤 과정을 통해서 메모리에 로드되는 것인가?

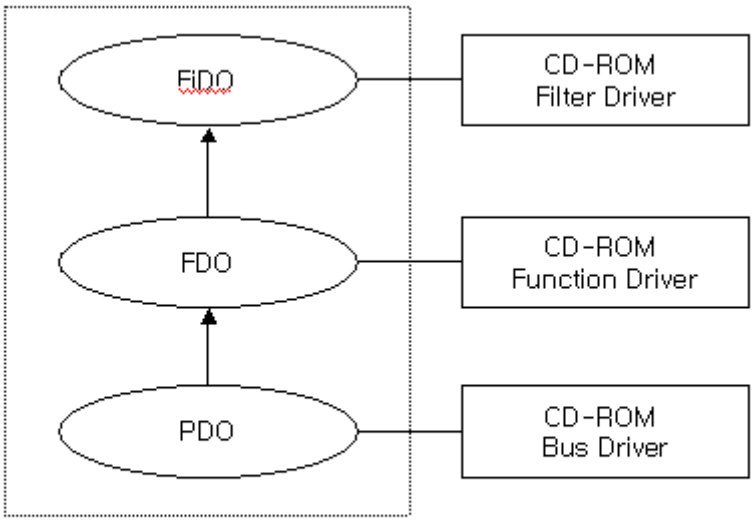

그림-3 CD-ROM Storage Device Stack Before Volume Mount

File System Recognizer

앞에서 제시한 의문점을 해결할 단서는 File System Recognizer이다. 다음의 사항을 한 번 고려해 보자. Boot후 System Shut down까지 한 번도 CD-ROM을 사용하지 않는다면 굳이 사용도 되지 않을 CDFS가 메모리에 로드될 필요가 있겠는가?! 따라서 Windows는 부팅 시 무조건 Full CDFS를 메로리에 로드하는 것이 아니라 파일 시스템 Recognizer로 알려진 mini-File system driver를 먼저 로드 한다. 메모리에 로드될때 mini-FSD들은 I/O Manager로부터 Volume Mount, 로드 파일 시스템 같은 요청을 받기위해 자신의 Device Object를 만들고
IoRegisterFileSystem(&DeviceObject) 함수를 호출하여 I/O Manager에게 파일 시스템 드라이버로 등록을 한다.
이러한 과정이 끝난 후 최초 CD-ROM 디바이스가 Access될 때 mini-FSD는 I/O Manager로부터
IRP_MJ_FILE_SYSTEM_CONTROL(IRP_MN_MOUNT_VOLUME) 요청을 받는다. 그리고 이 요청을 성공적으로 처리하게 되면 다시 I/O Manager로부터 IRP_MJ_FILE_SYSTEM_CONTROL(IRP_MN_LOAD_FILE_SYSTEM)요청을 받

게 된다. 이때 mini-FSD는 ZwLoadDriver()함수를 사용하여 Full CDFS를 메모리에 로드하게 되는 것이다. 위에서 설명한 내용들을 다음의 순서로 정리해볼 수 있겠다.

먼저, 부팅 시 CD-ROM은 드라이버 레터 "E"로 할당되었다고 가정하고 디바이스 이름은 "₩Device₩CdRom0"라고 가정을 하자.

1. User로부터 최초로 CD-ROM에 존재하는 "E:₩Directory1₩foo"를 Open하고자 하는 요청이 발생한다.
2. Object Manager에 의해 드라이브 레터 "E:"는 "₩Device₩CdRom0"로 변환이 되고 이 object의 이름은 I/O Manager에의해 관리가 되므로 I/O Manager에게 이 Open요청을 보낸다. 이 Open 요청을 받은 I/O Manager는 CD-ROM 디바이스 스택의 Physical Device Object(PDO)와 연결되어 있는 Volume Parameter Block(VPB)의 Flags에서 VPB_MOUNTED 필드가 set되어 있는지를 확인한다. 이 VPB_MOUNTED 가 의미하는 것은 이미 해당 device에 volume이 마운트 되었는지 아닌지를 나타낸다. 아마도 이 시점에서는 당연히 set되어 있지 않을 것이다. 왜냐하면 이 Open요청이 최초의 CD-ROM device에 대한 access 요청이고 아직 CDFS도 메모리에 로드되어 있지 않았기때문이다. 이미 Volume이 마운트되어져 있었다면 I/O Manager는 Full-CDFS로 이 Open 요청을 보냈을 것이다.
3. 자, 계속해서 Volume이 마운트가 되어 있지 않는 경우를 보자. 이 경우 I/O Manager는 이미 로드되어 있는 모든 Full File System Drivers과 mini-File System Drivers(file system recognizers)에게 IRP_MJ_FILE_SYSTEM_CONTROL(IRP_MN_MOUNT_VOLUME)요청을 누군가가 성공적으로 처리하여 STATUS_FS_DRIVER_REQUIRED를 리턴할 때 까지 보낸다.
4. 위의 3번 과정에서 CDFS를 위한 mini-CDFS(recognizer)도 역시 마운트 요청을 받게 되는 것이다. 이때 mini-FSD는 CD-ROM disk드라이버에게 Volume 마운트에 필요한 정보를 구하여 이 정보의 유효성을 판단한 다음 유효하다고 판단이 되어지면 STATUS_FS_DRIVER_REQUIRED를 그렇지 않다면 STATUS_UNRECOGINZED_VOLUME을 리턴한다.
5. I/O Manager는 위의 4번 과정에서 mini-FSD가 STATUS_FS_DRIVER_REQUIRED를 리턴했다면 그 mini-FSD에게 IRP_MJ_FILE_SYSTEM_CONTROL(IRP_MN_LOAD_FILE_SYSTEM)

요청을 다시 보낸다.

6. LOAD_FILE_SYSTEM 요청을 받은 mini-FSD는 이제 ZwLoadDriver() 함수를 호출하여 full CDFS를 메모리에 로드하게 된다.
7. 6번 과정에서 mini-FSD가 full CDFS로 정상적으로 로드하면 I/O Manager에게 STATUS_SUCCESS를 리턴한다. 이때 I/O Manager는 다시 한 번 더 모든 로드되어 있는 FSD들에게 마운트 요청을 보낸다.

CDFS Driver Stack

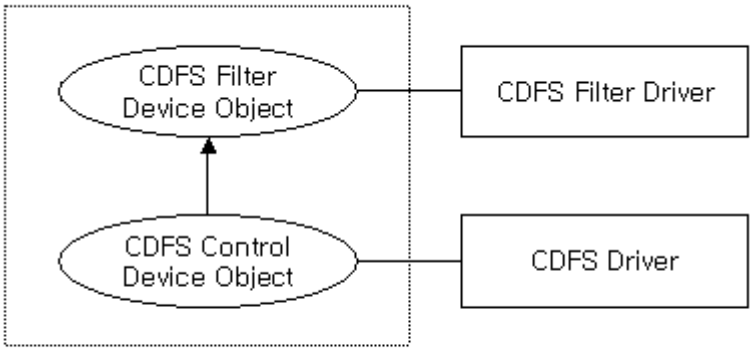

그림-4 CDFS Before Volume Mount

위의 <그림-4>는 CDFS가 mini-FSD의 요청으로 메모리에 로드된 직후의 모습을 나타내고 있다. 이때 CDFS에 의해 만들어진 디바이스 object를 CDFS control device object라고 하는데 특별한 의미가 있는 것은 아니고 단지 CDFS가 만드는 Volume device object와 구별하기 위한 것이다. 사실 이 control device object는 CDFS 자체를 open하여 CDFS로 요청들은 보내기 위해 만들어진 것이다. 따라서 CDFS는 CDFS Control device Object를 통해서 Volume 마운트 요청을 받게 된다. 그리고 <그림-4>는 또한 CDFS가 로드될 때 CDFS filter드라이버가 존재하여 같이 로드되어 있는 상황을 묘사하고있다. 따라서 CDFS의 filter 드라이버가 CDFS의 Volume 마운트 요청을 Trap하고자 한다면 CDFS의 Control Device Object에 Attach 되어야 함을 알 수 있다.

지금 우리가 보고 있는 것은 최초 CD-ROM을 접근하는 요청을 처리하는 일련의 과정이며 이 과정에서 CDFS가 막 메모리에 로드된 것이다. 아직 최초

요청된 “E:₩Directory1₩foo”에 대한 open요청은 완료된 상태가 아니고 진행 중임을 잊지 말자!!!

Mounted CDFS Volume

위에서 설명하였듯이 <그림-4>에서 CDFS에 의해 생성된 control device object를 통해 I/O Manager는 메모리에 이제 막 로드된 CDFS에게 Volume 마운트 요청을 보낸다. 만약 CDFS에 대한 filter 드라이버가 존재한다면 이때 이 filter 드라이버 역시 이 Volume 마운트 요청을 보게 될 것이다.
자, 이제부터는 CDFS가 Volume 마운트 요청을 어떻게 처리하는가를 살펴보기로 하자.

1. CDFS는 IOCTL 요청을 CD-ROM 디바이스 드라이버에게 보내 필요한 정보를 얻은 다음 mounted volume을 위한 device object를 <그림-5>와 같이 만든다.

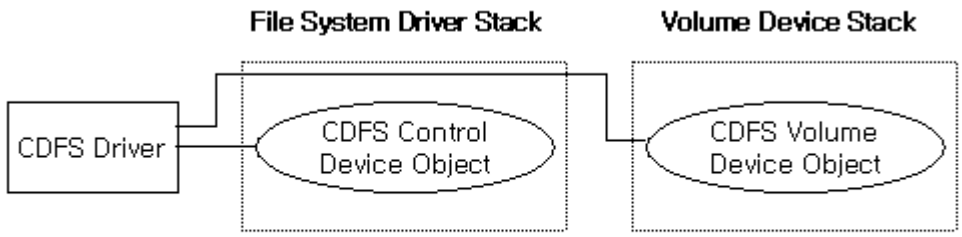

그림-5 Volume Device Stack

2. 이제 CDFS는 CD-ROM Physical device object의 Volume Parameter Block(VPB)를 이용하여 <그림-6>과 같이 Storage Device Stack과의 연결 포인터를 만들게 된다.
3. 이 마운트 요청을 처리하면서 CDFS는 Volume management를 위한 Volume Control Block(VCB)와 Root Directory를 위한 File Control Block(FCB)같은 구조체를 만들고 적절하게 초기화 하는 작업을 한다. 이러한 구체적인 작업은 기회가 된다면 다음 기회에 좀 더 구체적으로 언급을 하기로 하자.
4. CDFS가 이렇게 Volume 마운트 요청을 정상적으로 처리하여

STATUS_SUCCESS를 리턴하면 I/O Manager는 이제 CDFS에 의해 만들어진 volume device object를 통해 최초 "E:₩Directory1₩foo"대한 IRP_MJ_CREATE 요청을 CDFS로 보낸다.

여기까지 오면 CD-ROM 디바이스를 위한 Storage Device Stack과 CDFS에 의해 생성된 Volume Device Stack간에 연결 고리가 생겼고 정상적으로 CD-ROM에 존재하는 data를 접근할 수 있는 모든 준비가 되었다. 자, 이제부터 살펴볼 것은 File System Filter 드라이버는 어떤 과정을 통해 메모리에 로드되고 CDFS에 어떻게 Attach되는가를 살펴보아야 할 것이다.

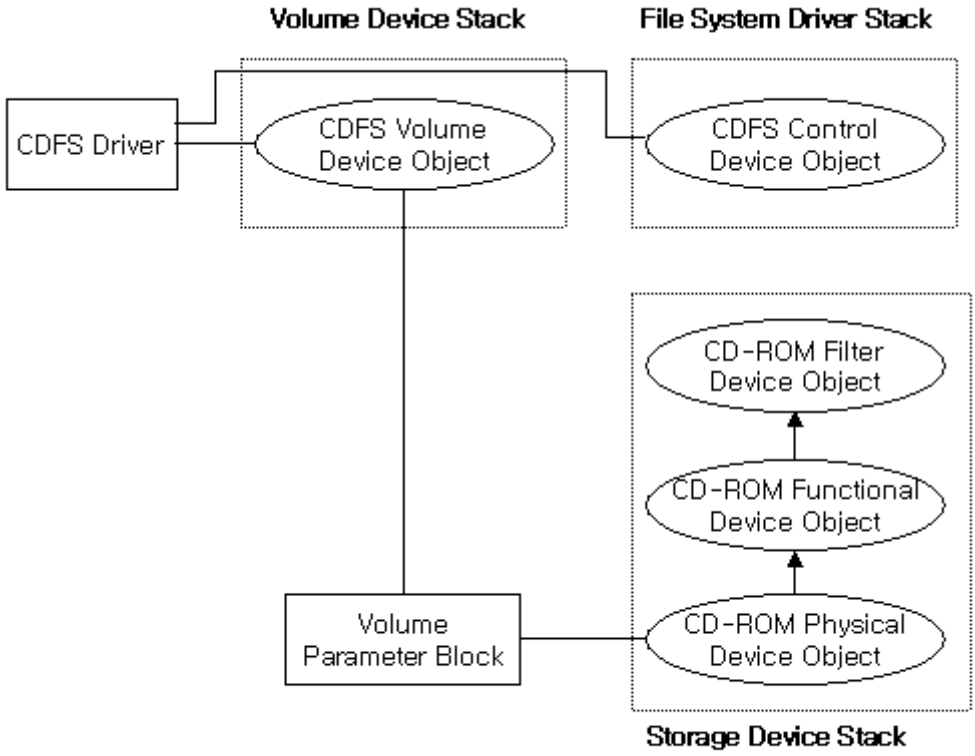

그림-6 Mounted CDFS Volume

File System Filter Driver는 어디에...

이제는 마지막으로 File System Filter Driver는 어디에 위치하며 어떤 과정을 통해 CDFS에 Attach되는가를 살펴보기로 하자.
먼저 우리의 CDFS filter driver가 CDFS보다 먼저 메모리에 로드되어 있다고 가정하자. 이렇게 디자인 하는 것은 CDFS가 메모리에 로드되는 시점을 잡

아서 CDFS가 로드될 때 CDFS의 control device object에 attach하여야만 CDFS로 오는 Volume Mount요청을 Trap할 수 있기때문이다.
이 작업을 수행하기 위해 우리 필터 드라이버는 메모리에 로드될 때 IoRegisterFsRegistrationChange(이 함수는 Win2K이상의 O/S에서 사용가능하다)함수를 이용하여 콜백 함수를 등록하여 어떤 File System Driver가 로드되더라도 로드가 될 때 마다 Notify를 받도록 한다. 현재 우리 filter 드라이버가 막 로드된 시점은 <그림-7>에서와 같이 Filter Driver가 메모리에 로드될 때 생성한 Filter Control Device Object만이 존재한다. 왜냐하면 아직 CDFS가 메모리에 로드되기 전에 우리의 필터 드라이버가 먼저 메모리에 로드되기 때문이다. <그림-7>의 Filter driver Control Device Object는 Filter Driver가 만드는 여러 다른 device object와 구별을 위한 것이다. 그리고 Filter Driver를 응용 프로그램을 통해 제어를 할 필요가 있다면 이 control device object를 사용하게 될 것이다.

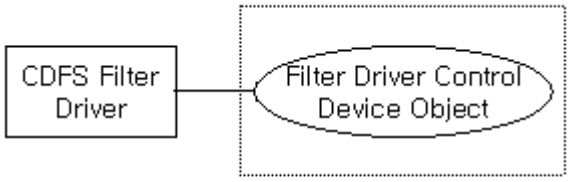

그림-7 Filter Driver Stack

이제 CDFS가 앞에서 살펴본 여러 과정들을 통해 최초 Open 요청을 만족시키기 위해 Load된다면 우리의 filter 드라이버에서
IoRegisterFsRegistrationChange 함수에의해 등록된 콜백 함수가 호출될 것이다. 이 콜백 함수의 인자로서 <그림-6>의 CDFS control device object가 넘어온다. 이때 우리 Filter 드라이버는 <그림-8>과 같이 새로운 device object를 하나 더 만들어 file system control device object에 attach 시킨다. 이렇게 하면 조만간 CDFS로 올 Volume 마운트 요청을 Trap할 수 있게 된다. 왜냐하면 Volume 마운트 요청은 앞에서 설명한 바와 같이 CDFS의 control device object가 있는 file system driver stack을 통해서 오기때문이다.

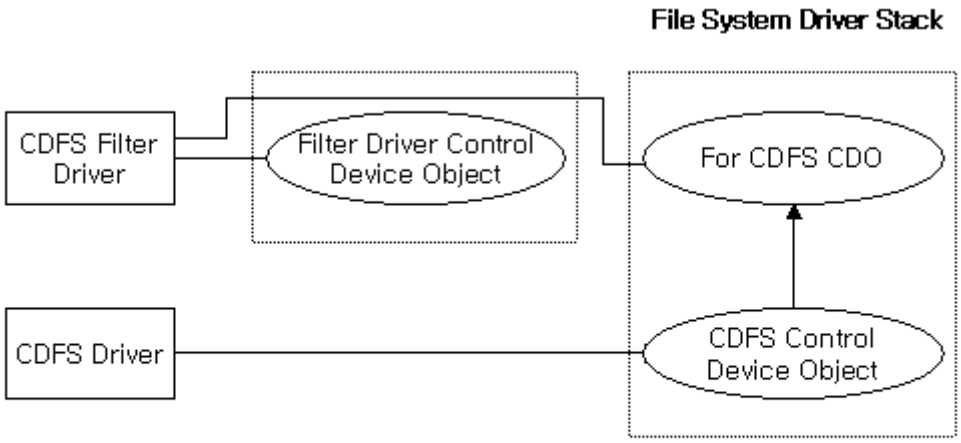

그림-8 Filter Driver & File System Driver Stack Before Volume Mount

Volume Device Stack

앞의 <그림-6>에서 보다시피 Volume device stack은 CDFS에 의해 Volume 마운트 요청이 정상적으로 수행되어야만 CDFS에 의해서 생성된다. 이제 우리의 CDFS filter 드라이버는 앞서 설명된 <그림-8>에서 File System Driver Stack을 통해 오는 Volume 마운트 요청을 Trap할 수 있다. 우리의 CDFS filter 드라이버가 Volume 마운트 요청을 받았을 시점에는 아직 CDFS가 이 사실을 알 수는 없다. 왜냐하면 우리가 이 Volume 마운트 요청을 CDFS에게 전달 할 것이기 때문이다. 따라서 우리가 이 Volume 마운트 요청을 받은 시점에서는 CDFS에 의해 아직 처리되기 전이므로 CDFS에 의해 생성되어야 할 volume device object도 존재하지 않는다. 따라서 우리는 volume device object를 filtering하기 위해서는 이 Volume 요청을 CDFS에게 먼저 처리하도록 전달한 다음 CDFS가 Volume 마운트 요청을 완료하여 Volume device object를 생성하면 비로소 이때 우리 필터 드라이버도 새로운 device object를 생성하여 CDFS가 생성한 Volume device object에 Attach 시켜야만 한다. 이 작업을 위해서 Volume 요청을 CDFS에게 전달하기 전에 complete Routine을 등록하고 completion routine에서 Attach 작업을 구현한다. 그리고 필요하다면 이 completion routine에서 WorkItem을 이용하여 구현을 하도록 한다. 이 과정이 끝나면 우리는 <그림-9>를 보게 된다. 이제 CDFS에 filter 드라이버가 존재하는 CD-ROM에 대한 최초 Open 요청은 이제 <그림-9>에서 CDFS Volume Device stack으로 오게되는 것이다.(bingo!!!)

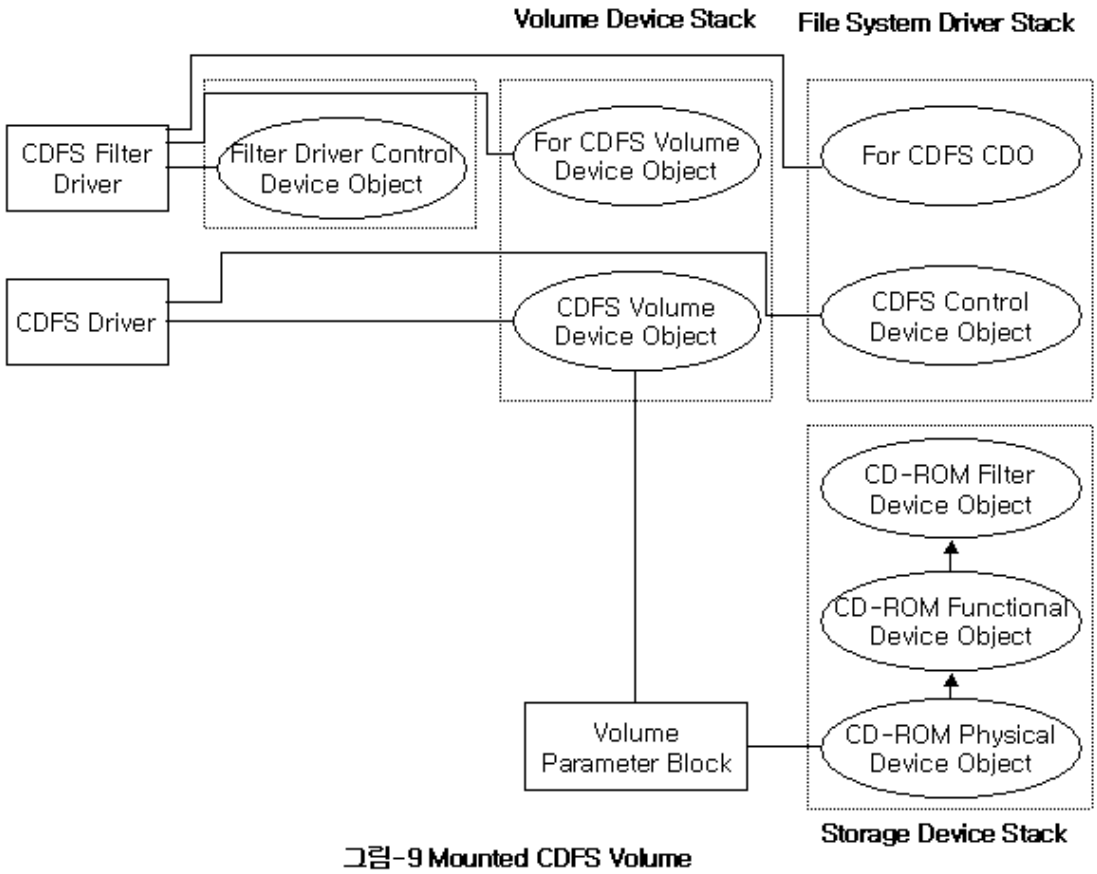

그림-9 Mounted CDFS Volume

겨울의 길목을 긴 여정의 중간 기착지로 삼으며.

이상으로 우리는 Windows O/S상에서 CDFS를 예제로 사용하여 File System Filter Driver가 어떤 과정을 통해 CDFS에 Attach 되는가를 알아보았다.
사실 File System Filter Driver를 만들기 위해서는 지금까지 설명한 이외에도 Cache Manager, Memory Manager, Fast I/O Mechanism 등 알아야 할 사항들이 무척 많다. 이 겨울이 끝나기 전에 다시 한 번 만나서 다음 번에는 NTFS filter드라이버에 대한 실제 code를 보면서 구체적으로 이번에 설명한 내용을 살펴보도록 하자.
즐거운 추억을 안고 다시 만나기를…