WDM을 이용한 가상 디바이스 드라이버 구현

정재기°, 이상욱, 김일곤
경북대학교 컴퓨터 과학과
newday@tgssm.co.kr°, leesw@cs.knu.ac.kr, ikkim@knu.ac.kr

The Implementation of Virtual Device Driver Kit using Windows Device Model

Jae Ki Jung°, Sang Wook Lee, Il Kon Kim
Dept. of Computer Sicence, Kyungpook National University, Korea

요 약

본 논문은 windows 상에서 디바이스 드라이버 구현을 위하여 타겟 디바이스를 사용하기 이전에 가상의 범용 디바이스 드라이버 개발도구를 이용하여 시뮬레이션 함으로써 최종 타겟 디바이스 개발의 효율성을 증대하고 개발 기간의 단축 및 비용 절감 하는데 목표를 둔다. 일반 PC에서 COM 포트를 이용하는 시리얼 통신으로 테스트 킷을 구현하여 드라이버를 개발하고 테스트할 수 있으며, 나아가 드라이버 연구에 있어서 실제 타겟 디바이스 없이 S/W 만으로도 올바른 드라이버를 개발할 수 있으며, 드라이버 동작과 내부 메커니즘을 비주얼하게 확인하여 초보 드라이버 개발자들에게도 도움을 주는데 목적이 있다. 이에 본 연구에서 새로운 개발 방향을 제시 하고 실험을 하였다.

1. 서 론

디바이스 드라이버 개발은 일반 응용 프로그램에 비하여 개발이 난해하고, 디바이스 드라이버에 대한 내부지식을 가진 개발자도 부족할 뿐만 아니라, 그 자료 면에서도 거의 전무한 상태이다. 최근 디바이스 드라이버 개발은 고부가가치 수익모델로 각광 받고 있으나 일반 개발자들은 어떻게 접근해야 할지에 대한 어려움을 가지고 있으며, 더욱이 디바이스 드라이버를 개발하기 위해서 고가의 부속장치가 필요하므로 접근이 쉽지가 않다. 따라서 보다 많은 개발자들에게 다양한 요구사항을 만족시기 위해서는 그것을 대체할 수 있는 개발용 도구가 필요하다. 이를 해결하기 위해서 실제 하드웨어를 구입하지 않고도 디바이스 드라이버의 동작을 미리 검사 할 수 있는 가상 COM 포트를 이용한 드라이버 개발 도구의 필요성은 절대적이라 할 수 있다. 본 논문에서는 디바이스를 가상으로 구현함으로써 보다 많은 개발자들이 효율적으로 디바이스 드라이버 개발을 할 수 있도록 도움을 주고자 한다.

2. 관련 연구

2.1 디바이스 드라이버

도스시대에는 일반 응용 프로그램이 직접 개개의 H/W를 직접 제어를 하였으나 windows로 운영체제가 변천하면서 운영 체제와 응용 프로그램 사이에서 디바이스 드라이버가 그 역할을 대신하기 시작하였다. 이것은 장치 제어기 또는 구동 드라이버라고도 하며 하드웨어와 운영체제 응용프로그램의 연결 고리가 되는 프로그램으로 하드웨어 구성 요소가 운영체제 아래서 제대로 작동하는데 꼭 필요하다. 대표적으로 크게 windows 디바이스 드라이버와 Linux 디바이스 드라이버가 있는데, Linux 디바이스 드라이버는 Linux O/S가 오픈 소스이므로 운영체제를 참조하여 작성이 용이하나, windows O/S는 마이크로소프트에 의해 그 구현 내용이 숨겨져 있기 때문에, 마이크로소프트사의 windows DDK(Driver Development Kits)를 이용하여 라이브러리 형태의 함수로 개발이 가능하기 때문에, windows 디바이스 드라이버 개발이 더 어렵다고 할 수 있다. 현재 MS Windows XP embedded는 삼성전자, KT와 기술 협력 제휴를 맺고 있으며, 생활 가전과 통신 서비스 등에 windows 디바이스 드라이버 개발의 수요가 급증 할 것으로 예상 하고 있다.

2.2 윈도우즈 디바이스 모델 (WDM)

WDM은 다중의 플랫폼(Windows 9X/2000/XP)에 걸친 실행 파일 수준에서의 호환성이 뛰어나며, Windows NT 4.0 I/O 서브시스템을 기반으로 하고 있으며, 확장성을 가진 설계로, 새로운 디바이스, 프로토콜 또는 버스들의 지원이 가능하며, 최신의 디바이스 관리 기술을 채택하여 클래스/포트 설계로 드라이버의 개발 속도를 향상 시킨다. 대부분의 디바이스들에 대해 클래스 드라이버를 제공하고, 특정 프로토콜, 물리적 디바이스 또는 버스들을 지원하기 위해, 포트 드라이버를 통하여 클래스 드라이버의 기능을 확장 할 수 있다.

3. WDM 의 Architecture

그림1은 Windows 2000에서의 WDM의 구조를 나타낸다[1].

- **User mode:** 응용 프로그램은 Kernel32.DLL에 구현되어 있는 API를 통해 Kernel mode 루틴과 인터페이스한다.
- **Kernel mode:** 드라이버는 HAL 함수를 호출하여 하드웨어에 접근한다.
- **IRP(I/O Request Packet):** 드라이버와 혹은 드라이버간에 통신하기 위해 사용되는 데이터 구조

• I/O Manager: 디바이스 드라이버에게 IRP를 전달한다.

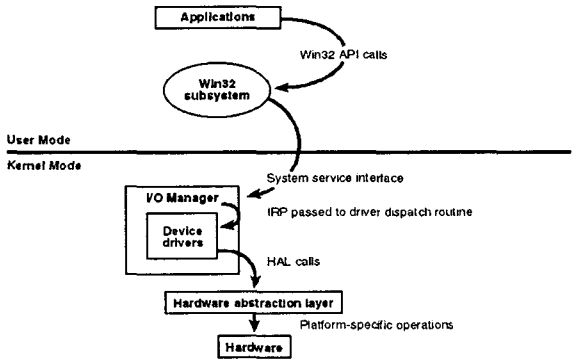

그림1: Windows 2000 WDM 구조

• HAL(Hardware Abstraction Layer): 플랫폼 독립적으로 드라이버가 하드웨어에 접근할 수 있게 한다.

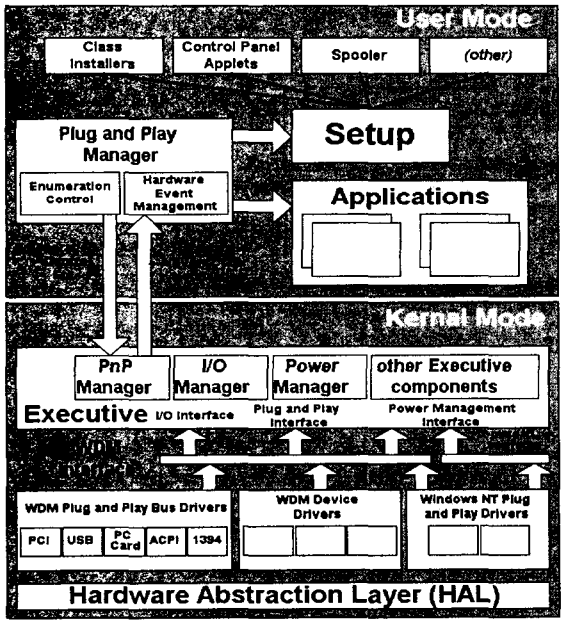

그림2: Windows 2000의 새로운 기능

Windows 2000이전의 MS 운영체제에서는 VxD(Virtual x Device), KMD(Kernel Mode Driver)까지만 지원 하였으나, 그림 2에서와 같이 Windows 2000이상의 운영체제부터 본격적으로 WDM을 지원 하고 있다[1]. 새로운 기능으로써 PnP(Plug-and-Play)를 지원하여 프로그램적인 리소스 할당과 동적으로 적절한 디바이스 드라이버 할당을 제공하며, Power Management의 지원으로 효율적인 전원 관리를 할 수 있으며, 이 밖에도 WDM 드라이버 클래스를 제공하고 있다. 다음 절에서는 WDM의 핵심 드라이버와 주요 기능에 대한 언급을 하고자 한다.

3.1 WDM의 주요 드라이버

• Bus Driver(PDO): Bus 계산, PnP 및 Power Management IRP에 반응하고, 필요시 버스의 다중 엑세스에 관여함.

• Function Driver(FDO) : 디바이스를 위한 메인 드라이버이며, 기조 I/O를 위한 기초 연산 관리, 디바이스 지정 전원 요구량 관리.

• Filter Driver(FIDO) : 값을 추가 하거나 디바이스의 상태를 수정하는 선택적 드라이버 이며, 드라이버 스택내의 Function 드라이버 상위 또는 하위에 존재[2].

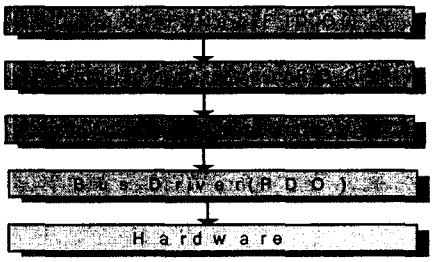

그림 3: 범용 WDM Driver Stack

3.2 WDM의 실행 가능 처리

• DriverEntry Point : 커널은 보통 IRP를 보내서 드라이버의 코드를 실행한다. 드라이버는 초기화 엔트리 포인트 즉, DriverEntry를 반드시 호출해야 하는 루틴을 1개 가지고 있다. 커널은 드라이버가 로드 되면 DriverEntry 루틴을 호출한다.

• Dispatch 루틴 : 드라이버의 DriverEntry 루틴은 IRP를 처리하기 위한 일련의 콜백을 준비해야 한다. 또 필요한 경우 Unload, AddDevice 및 StartIo 루틴들의 콜백도 설정한다.

• Device의 작성 : 디바이스 개발자가 DriverEntry 루틴에서 작성하거나, PnP 관리자로부터 통지가 있을 때 작성해야 한다. 그리고 드라이버가 언로드 되거나, PnP 관리자로부터 디바이스가 제거 되었다는 통지가 왔을 때 디바이스를 삭제 한다.

• 하드웨어 자원의 할당 : 하위 레벨의 드라이버는 어느 하드웨어 자원이 자신에게 할당되어 있는지를 알 필요가 있다. 가장 일반적인 하드웨어 자원은 I/O 포트, 메모리주소, 인터럽트 및 DMA 라인이다.

• 다른 드라이버의 호출 : WDM 드라이버는 다른 드라이버와 통신 하는데 많은 시간을 소비한다. PnP 디바이스 객체의 스택에 존재하며, IRP를 스택의 하위에 있는 다음 디바이스에 전달하는 것이 보통이다. 다른 경우는 드라이버의 주 처리 스택의 하위에 있는 다음 디바이스를 호출함으로써 수행된다.

• 하드웨어 액세스의 직렬화 : 하드웨어의 액세스하는 모든 디바이스는 어떠한 기능을 사용해서 드라이버의 다른 부분이 하드웨어에 동시에 엑세스 하지 않도록 해야 한다. 이러한 여러 가지 충돌을 예방하기 위하여 임계 영역 루틴을 사용하거나, Strarlo 루틴을 사용하여 IRP 처리를 직렬화 한다.

• **하드웨어와의 통신** : I/O포트나 메모리의 주소를 갖게 되면, 하드웨어 레지스터 등에 읽기와 쓰기를 하는 것이 용이하다. 이때 여러 가지 발생 할 수 있는 문제점들은 시스템 스레드를 이용하거나, 인터럽트 서비스 루틴을 이용하여 해결 가능하다.
• **전원 관리** : 디바이스가 전원 소비를 제어할 수 있는 경우, Windows98과 Windows2000의 드라이버는 전원 관리를 지원해야 한다. 전원 관리를 통하여 휴대용 컴퓨터에서는 배터리 전원을 절약 할 수 있고, 데스크톱 시스템에서는 에너지 소비와 하드웨어의 소모를 억제할 수 있다. 드라이버는 Power IRP를 처리하는 것으로 전원관리를 지원한다. 대다수의 드라이버는 단지 디바이스 스택의 하위에 Power IRP를 전달 전달할 뿐이다.
• **Windows Management Instrumentation** : 가능 하다면 드라이버가 WDM의 WMI(Windows Management Instrumentation)확장 기능을 제공할 수 있어야 한다. 이 기능은 진단과 성능에 대한 정보를 기술자와 관리자에게 보고 한다.
• **NT 이벤트 보고** : NT와 Windows 2000에서는 시스템 이벤트 로그를 이용할 수 있다. 이것은 드라이버의 문제를 보고하는 전통적인 방법이다. 드라이버가 생성하는 에러 로그 엔트리에는 이벤트 번호와 경우에 따라서는 약간의 문자열과 데이터가 포함되어 있다. 이벤트 번호와 드라이버 자원에 포함되는 메시지 문자열을 결합한다.
• **시스템 스레드** : 시스템 스레드를 사용하면, 일부 작업을 백그라운드에서 실행 할 수 있다. 시스템 스레드는 특별히 저속인 디바이스와 통신하거나, 우선순위가 낮은 데이터의 뒤처리를 실행할 수 있다[3].

4. 가상 드라이버의 동작

4.1 실험 환경

Windows 2000 OS에서 Visual C++ 6.0과 Win2000 DDK를 이용하여 구현하였으며, 2대의 데스크톱에서 COM 포트를 이용한 시리얼 통신을 사용 하였다.

4.2 주요 구성 요소

본 실험에서 가상 디바이스 드라이버를 두 부분 Virtual Device Driver Application (호스트측 컴퓨터에서 테스트할 수 있는 환경)과 Virtual Device Driver Kit (타겟측 컴퓨터 테스트 킷) 으로 나누었다.
• **Virtual Device Driver Application** : 가장 중요한 역할을 담당하며, 테스트 할 수 있는 환경인 사용자 응용 프로그램과 이 응용 프로그램을 구동하게 하는 핵심이 되는 WDM Driver 가 있다. 사용자 응용 프로그램은 단지 해당 드라이버를 생성하고 Read나 Write로 데이터를 주고받아 화면에 반영하는 역할만을 담당하게 된다. 그리고 본 실험에서 구현한 WDM 드라이버 의 계층을 살펴보면 사용자 응용 프로그램 -> WDM 드라이버 -> 시리얼 드라이버 -> Hal 을 통해 타겟 드라이버로 데이터가 전송 된다. 사용자 응용 프로그램에서 CreateFile 을 요청하면 WDM 드라이버는 일반 WDM 구동 방식과 마찬가지로 DRIVERENTRY 루틴으로 점프하여 해당 IRP인 IRP_MJ_CREATE와 매핑이 되며 드라이버의 전역적인 상태만을 초기화 하고 드라이버의 디스패치 테이블을 초기화한다. 그 다음 AddDevice 루틴으로 점프한다. 이 루틴에서 해당 디바이스 오브젝트가 생성이 되며 디바이스 오브젝트의 전원 관리 플래그 설정과 디바이스를 관리하는데 사용되는 모든 구조체나 소스들을 초기화 한다. 그리고 IoAttachDeviceToDeviceStack 함수를 사용하여 디바이스 오브젝트를 DevNode의 최상단에 위치시킨다. 이런 일련의 과정은 모두 OS 내의 IOManager가 관리해 주게 되어있다. 이런 초기화 루틴을 거친 뒤 사용자 응용 프로그램에서 WriteFile과 ReadFile을 호출하여 해당 타겟과의 I/O와 인터럽트를 수행하게 되는 것이다. 이런 수행의 내부 구조는 사용자 응용 프로그램에서 WriteFile을 호출할 경우 IOManager는 IRP_MJ_WRITE가 발생되었음을 인지 한 뒤 Dispatch 루틴으로 건너뛰어 WDM 드라이버에서 이런 정보를 타겟으로 보내게 구성을 한다. WDM 드라이버 관점에서 이런 IRP가 들어오면 데이터를 타겟으로 전송을 하기 위해 COMFileObject를 구해야 한다. 이렇게 ComFileObject의 핸들을 얻은 뒤 해당 시리얼 드라이버에게 데이터를 전송하기 위해 다시 IRP를 생성을 해야 된다. 이렇게 IRP를 생성하여 하위단의 시리얼 드라이버에게 최종적으로 IOCALLDRIVER() 함수를 통해 알리게 되면 대기하고 있던 시리얼 드라이버는 해당 IRP를 받음과 동시에 타겟 컴퓨터에 해당 데이터를 전송하게 된다.
• **Virtual Device Driver Kit** : 호스트로부터 시리얼로 들어온 해당 데이터를 이 프로그램에서는 하나의 스레드를 생성하여 항상 대기 한 뒤 해당 시리얼 메시지가 들어오면 시리얼 드라이버에서 이것을 감지한 뒤 프로그램에 반영을 하게 된다.

5. 결 론

일반적으로 디바이스 드라이버를 작성 시에 H/W가 뒷받침 되어야 한다. 하지만 H/W 없이 간단히 데스크톱 2대만을 이용하여, 필요로 하는 드라이버 루틴을 실행시키고 디버그 할 수 있는 환경을 만드는 것이 본 논문의 가장 큰 목표이다. S/W 적으로는 한계가 있지만 기본적인 WDM 루틴을 구현하였다. 이를 바탕으로 USB나 IEEE1394 등과 같은 다양한 인터페이스를 지원하는 디바이스 드라이버 킷의 개발을 향후 연구 과제로 제시한다.

참고 문헌

[1] Microsoft URL : http://www.microsoft.com
[2] Chris Cant, Writing Windows WDM Device Drivers, CMP books
[3] Walter Oney, Microsoft Programming the Windows Driver Model, Microsoft Press
[4] Art Baker, Jerry Lozano, The Windows 2000 Device Driver Book(2nd Edition), Prentice Hall
[5] Bryan Woodruff, Writing Windows Virtual Device Drivers, Addison-Wesley
[6]Microsoft Corpation, Microsoft Windows 2000 Driver Development Kit, Microsoft Press