2000년 한국정보처리학회 춘계 학술발표논문집 제7권 제1호

입출력 포트 제어를 위한 VxD 개발에 관한 연구

박춘근, 오민정, 임성락
호서대학교 컴퓨터공학과
email:srrim@office.hoseo.ac.kr

A Study on Development of VxD for Controlling the I/O Port

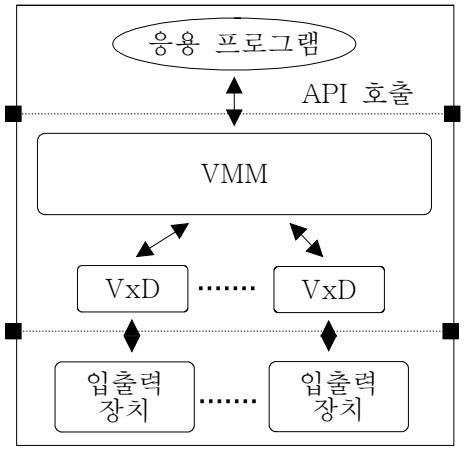
Chun-Geun Park, Min-Jung Oh, Seong-Rak Rim
Dept. of Computer Engineering, Hoseo University

요 약

입출력 장치를 사용하기 위해서는 장치 구동기와 기본적으로 입출력 포트를 제어하는 기능이 요구된다. 본 논문에서는 윈도우즈98 환경에서 이러한 기능을 제공하기 위한 VxD 개발에 관한 기법을 제시하였다. 제시한 기법의 타당성을 검증하기 위하여 입출력 포트 제어용 VxD 프로그램을 개발하고, 실험용 LED 보드를 PIO 포트에 연결하여 VxD의 동작 상태를 확인하였다.

1. 서론

윈도우즈의 겉모습은 GUI 환경이지만 내부의 핵심은 윈도우즈의 커널 부분인 VMM(Virtual Machine Manager)이다. GUI 환경은 VMM을 싸고 있는 서브 시스템 중 USER와 GDI, SHELL에 의해 나타나는 것이며, 실제 운영체제 내부에서 일어나는 일은 대부분 VMM을 중심으로 <그림 1>과 같이 입출력 장치를 제어하는 VxD(Virtual Device Driver)와 연계되어 조화롭게 돌아가고 있는 것이다.



<그림 1> 응용 프로그램과 VxD의 관계

<그림 1>에서 VMM의 역할이 전체 시스템을 가상화 시키는 것이지만 컴퓨터 시스템은 수많은 요소로 구성되어 있으며 컴퓨터마다 여러 가지 조합의 하드웨어로 이루어져 있다. 따라서 VMM이 혼자서 수많은 장치를 가상화할 수는 없으며, VxD 계층 구조의 대표로서의 역할만 하고 있다. VMM은 응용 프로그램으로부터의 요구를 해당 VxD에게 적절하게 보내주는 역할을 하는 것으로 거의 모든 기능은 VxD에서 수행된다. VMM은 전체 관리자이며, 그 밑의 VxD는 VMM의 하부에서 자신이 담당하는 하드웨어의 가상화를 수행함으로써 전체 시스템을 가상화한다.

VxD는 입출력 장치와 운영체제를 연결시켜 주고, 응용 프로그램으로부터 각각의 장치적 특성을 감추고 모든 장치를 동일한 방법으로 접근할 수 있도록 하는 것이다[1][2]. 모든 응용 프로그램들은 운영체제에서 제공하는 API 함수를 호출함으로써 다양한 종류의 입출력 장치를 동일한 방법으로 접근하게 된다. 결국 응용 프로그램은 어떤 입출력 장치에도 직접 접근 할 수 없으며 반드시 해당 입출력 장치의 VxD가 제공하는 인터페이스를 통해 간접적으로 접근할 수 있다. 따라서 새로운 입출력 장치를

개발할 경우, 반드시 그 장치를 제어하기 위한 VxD가 요구된다.

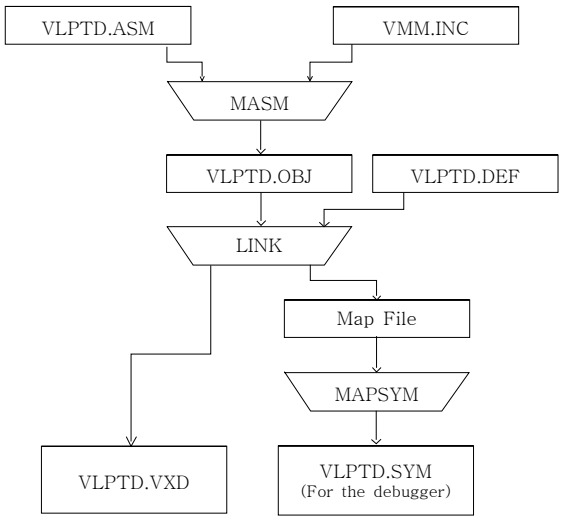
2. VxD의 구조

윈도우즈98에서 VxD는 주기억장치에 적재되는 시기에 따라 정적 VxD와 동적 VxD로 구분한다. 정적 VxD는 윈도우즈98이 시작할 때 적재되고, 동적 VxD는 응용 프로그램이나 다른 VxD에 의해 호출될 때 적재된다.

동적 VxD는 DEF 파일에 "DYNAMIC" 키워드를 추가해야하며 정적 VxD는 처리하지 않는 두 개의 시스템 컨트롤 메시지(Sys_Dynamic_Device_Init, Sys_Dynamic_Device_Exit)를 처리해야 한다. VMM은 동적으로 적재되는 VxD를 초기화하는데 Sys_Dynamic_Device_Init 메시지를 보내고, 해제하기 전에 Sys_Dynamic_Device_Exit 메시지를 보낸다. 두 개의 메시지 외에는 모두 동일한 시스템 컨트롤 메시지를 받는다.

2.1 구성 요소

VxD는 세그먼트의 속성을 선언하는 DEF 파일, INC 헤더 파일, 실제 VxD의 코드가 들어있는 소스 파일로 구성된다. 이러한 파일들은 DDK에서 제공하는 프로그램을 이용하여 컴파일 및 링크시킬 수 있다. <그림 2>는 “VLPTD.VXD”를 생성하기 위한 구성 요소와 이들을 컴파일 및 링크시키는 절차를 나타내고 있다.



<그림 2> VxD 생성절차

VxD는 한 개 이상의 어셈블리 모듈이나 헤더를 합쳐야 하며 DDK에서 제공되는 툴을 사용하여 작성한다.

DEF 파일은 VxD가 사용할 수 있는 세그먼트 속성을 나타내고 있다. VxD에 있는 코드와 데이터의 대부분은 LCODE, PCODE, PDATA 세그먼트에 있다. LCODE 세그먼트는 항상 메모리에 존재해야 하는 코드와 데이터를 가지고 있고, PCODE와 PDATA는 각각 윈도우즈의 필요에 따라 메모리에 저장하거나 삭제될 수 있는 코드와 데이터를 가지고 있다.

INC 파일은 어셈블리에서 쓰이는 데이터를 선언하는 헤더 파일로써 기본적으로 DDK에서 지원해주는 것을 포함하여 사용자가 직접 선언하는 것으로 이루어진다.

2.2 응용 프로그램과 VxD 인터페이스

VxD를 개발한 후 정상적인 동작을 검증하기 위해서는 반드시 해당 VxD의 서비스를 받아서 사용자에게 결과를 보여주는 응용 프로그램이 필요하다.

응용 프로그램에서 특정 VxD의 서비스를 받기 위해서는 가장 먼저 CreateFile() 함수로 VxD를 열고, DeviceIoControl() 함수를 이용하여 명령을 VxD로 전송한다. VxD의 서비스가 불필요하면 CloseHandle() 함수를 이용하여 VxD를 닫는다.

응용 프로그램에서 VxD를 열기 위한 CreateFile() 형식은 “\\.\VxDName”이며, 여기서 VxDName은 VxD의 모듈 이름이나 VxD의 파일명, 또는 VxD를 나타내는 레지스트리 값 중에 하나이다. CreateFile() 함수에는 몇 개의 매개 변수가 있는데 VxD를 열 때는 두 가지 매개 변수를 이용한다. 그것은 lpName과 fdwAttrAndFlags이고, 매개변수의 값은 0, FILE_FLAG_DELETE_ON_CLOSE, FILE_FLAG_OVERLAPPED이다.

VxD에게 특정 서비스를 요청하고자 할 때에는 DeviceIoControl() 함수를 이용하는데 VxD 식별에 이용되는 장치 핸들과 제어코드, 입출력 매개변수

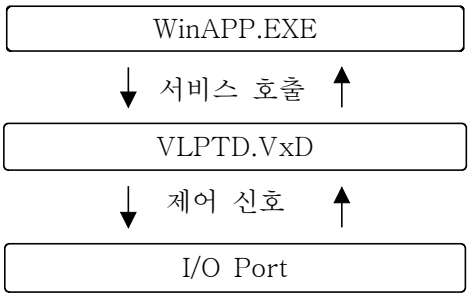
등을 기술한다. 이 함수는 일종의 범용 API로 VxD 개발자가 제공하는 다양한 형식의 API를 모두 포용할 수 있다

현재 열려 있는 VxD를 닫기 위해서는 응용 프로그램에서 CloseHandle() 함수를 이용한다. 그러나 VxD를 닫는다고 해서 바로 메모리에서 제거되는 것은 아니다. 시스템은 VxD에 대한 참조계수를 유지(이 값이 0일 때만 메모리에서 제거)하여 응용 프로그램이 VxD를 사용하지 않을 때만 실제로 닫는다.

정적 VxD는 참조계수 없이 운영체제가 처음에 부팅 될 때 실행을 시작하여 운영체제가 종료될 때까지 메모리에 적재되어 있게 된다.

3. VxD 프로그램 설계 및 구현

VxD는 입출력 장치와 운영체제를 연결시켜 응용 프로그램에서 입출력 장치를 사용할 수 있게 하는데, 응용 프로그램은 입출력 장치에 직접 접근 할 수 없으며 반드시 해당 입출력 장치의 VxD가 제공하는 인터페이스를 통해 서비스를 요청해서 간접적으로 하드웨어에 접근할 수 있다. 따라서 <그림 3>과 같이 입출력 포트를 제어할 VLPTD.VxD와 VxD의 기능을 사용할 응용 프로그램 WinAPP.EXE를 개발 설계하고, 제시한 VxD의 정상적인 동작을 확인하기 위해서 입출력 포트에 LED 보드를 연결하여 결과를 확인하도록 한다.



<그림 3> 전체 프로그램의 구성도

3.1 VxD 프로그램 개발

3.1.1 DEF 파일

DEF 파일은 VxD를 생성시키기 위해 함께 컴파일해야 한다. 각 세그먼트의 속성은 VxD의 세그먼트 정의 파일(DEF)에 선언되어있는데 DEF 파일에 오브젝트 코드에 있는 세그먼트의 속성을 올바로 기술해야 하며 형식은 <그림 4>와 같다.

```
VXD VLPTD.VXD DYNAMIC
; 작성되는 VxD 이름과 동적 VxD임을 선언

DESCRIPTION 'LPT Control VxD for Microsoft Windows'

SEGMENTS
; 세그먼트 선언자들간의 인터페이스 유지를 위해 공통된 것을 사용
_LPTEXT CLASS 'LCODE' PRELOAD NONDISCARDABLE
                    ⋮
EXPORTS
        VLPTD_DDB @1 ; VxD 이름을 DDB에 익스포트
```

<그림 4> VLPTD.DEF

DDB(Device Description Block)를 <그림 4>와 같이 익스포트 해야 하는데 그 이유는 VMM이 모듈에서 제일 처음 익스포트 된 것을 DDB로 인식하고 기록된 내용에 의해 VxD의 이름을 알아내기 때문이다.

3.1.2 INC 파일

INC 파일에 <그림 5>와 같이 VLPTD.VxD에서 사용할 고유한 ID와 VxD에서 제어할 포트의 주소를 정의한다.

```
VLPTD_Dev_ID equ 7fe5h ; 사용할 ID 선언
; ID는 마이크로 소프트에 등록해서 사용되지 않은 것을 써야 충돌이 없다. 일단은 충돌하지 않는 번호로 정한다.

BASEADDR equ 378h ; 제어할 포트의 주소
```

<그림 5> VLPTD.INC

3.1.3 ASM 파일

VxD 소스 파일의 기본 틀은 간단하다. <그림 6>과 같이 크게 VxD의 DDB를 서술하는 곳과 메시지를 처리하는 제어 프로시저로 나뉜다. VMM은 VxD에 대한 정보를 Declare_Virtual_Device 매크로를 이용해 작성한 DDB를 통해 알게되며 모든 VxD는 시스템 컨트롤 메시지에 응답하기 위한 제어 프로시저가 필요하다.

```
                                        :
                                        :
Declare_Virtual_Device VLPTD, 1, 0, VLPTD_Control, VLPTD_De
v_ID , ₩ UNDEFINED_INIT_ORDER
; Declare_Virtual_Device는 VxD를 설명하는 부분, VxD DDB를 만
드는 부분.  .DEF 파일에 DDB라는 라벨을 붙인 심벌을 익스포트.
                                        :
                                        :
VLPTD_IOCTL_Table LABEL DWORD ; 테이블을 작성
                                        :
                                        :
dd offset32 VLPTD_IOCTL_Set_NUMBER ; 포트로 출력
dd offset32 VLPTD_IOCTL_Set_INPUT ; 포트로부터 입력
                                        :
                                        :
BeginProc VLPTD_Dyna_Init ; 동적 VxD 초기화 메시지 처리
        mov al, 00000000b ; VxD 로딩시 00000000 출력
        mov dx, BASEADDR ; 포트의 주소 저장
        out dx, al ; 지정된 주소로 al 값 출력
        clc
        ret
EndProc VLPTD_Dyna_Init

BeginProc VLPTD_Dyna_Exit ; 동적 VxD 종료 메시지 처리
        mov al, 11111111b ; VxD 종료시 11111111 출력
        mov dx, BASEADDR
        out dx, al
        clc
        ret
EndProc VLPTD_Dyna_Exit
                                        :
                                        :
BeginProc VLPTD_IOCTL_Set_NUMBER ; 포트로 데이터 출력
        cmp [esi+cbInBuffer], 4 ; 입력되는 버퍼의 포인터
        mov edx, [esi+lpvInBuffer] ; 입력 버퍼에서 eax로 복사
        mov eax, dword ptr[edx]
        mov dx, BASEADDR
        out dx, eax
        clc
        ret
EndProc VLPTD_IOCTL_Set_NUMBER

BeginProc VLPTD_IOCTL_Set_INPUT ; 포트에서 데이터 입력
        mov dx, BASEADDR ; 포트의 주소 저장
        in eax, dx ; 포트에서 데이터를 읽들임.
        mov edx, [esi+lpvOutBuffer] ; 값을 출력 버퍼에 복사
        mov ebx, edx
        mov [ebx], eax
        mov edx, [esi+lpcbBytesReturned] ; 데이터의 크기
        mov ebx,edx
        mov [ebx], 4
        clc
        ret
EndProc VLPTD_IOCTL_Set_INPUT
                                        :
                                        :
```

<그림 6> VLPTD.ASM

3.2 응용 프로그램

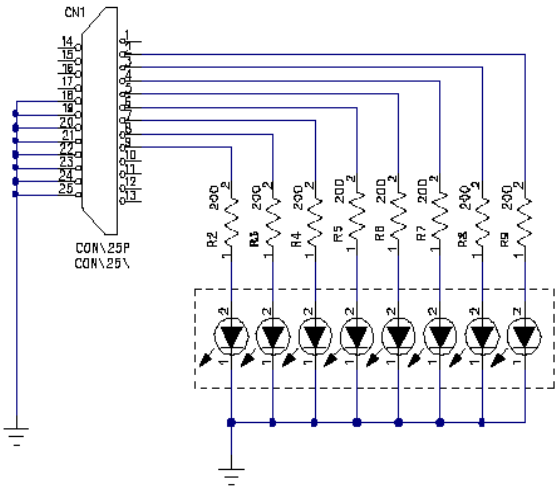
WinApp.EXE는 VxD의 서비스를 받아서 사용자에게 보여주는 윈도우 응용 프로그램으로 기본적인 틀은 <그림 7>과 같이 VxD의 서비스를 받기 위해 CreateFile()를 사용해서 VxD를 열고, DeviceIoControl()를 이용하여 지정된 명령을 VxD로 전송하고, CloseHandle()를 사용하여 VxD를 닫는 형식을 취한다.

```
                                        :
                                        :
m_hVxD = ::CreateFile("WWWW.WWVLPTD.vxd", 0, 0, NULL, 0, FILE_
FLAG_DELETE_ON_CLOSE, NULL); // VxD를 연다
if(m_hVxD==INVALID_HANDLE_VALUE)
         MessageBox("VLptD.VxD를 열지 못했습니다.");
         // VxD를 열지 못했을때 에러 메시지
return TRUE;
}
                                        :
                                        :
void CWinappDlg::OnButtonSetNumber() // VxD 서비스 호출
{
        UpdateData(TRUE);
        byte InBuf[1];
        InBuf[0] = m_nOutput;
        ::DeviceIoControl(m_hVxD, SET_OUTPUT,
                InBuf, sizeof(InBuf),
                NULL, 0,
                NULL,
                NULL);
        UpdateData(FALSE);
}

void CWinappDlg::OnButton2() // VxD 서비스 호출
{
        UpdateData(TRUE);
        byte InBuf;
        ::DeviceIoControl(m_hVxD, SET_INPUT,
                NULL, 0,
                &InBuf, sizeof(InBuf),
                NULL,
                NULL);
        m_nInput = InBuf;
        UpdateData(FALSE);
}

void CWinappDlg::OnDestroy() // 응용 프로그램 종료시 VxD 닫기
{
        CDialog::OnDestroy();
        if(m_hVxD != INVALID_HANDLE_VALUE)
                ::CloseHandle(m_hVxD); // VxD 닫기
}
                                        :
                                        :
```

<그림 7> WinApp.CPP

3.3 구현 및 실험

제시한 VxD 프로그램의 타당성을 검증하기 위해 <그림 8>과 같이 LED 보드를 설계한다. 제작한 응용 프로그램에서 VLPTD.VxD의 서비스를 호출하여 PIO 포트를 제어하고 설계한 LED 보드를 통해 결과를 확인한다.



<그림 8> 포트 제어 실험에 사용한 LED 보드

<그림8>에 LED가 PIO 포트의 2번 핀에서부터 9번 핀까지 연결되어 있는데 이들은 PIO 포트의 기본 어드레스인 0x378(PC에서 LPT1을 사용할 때)에 연결 되어있다. 0x378번지에 각 비트의 값에 '1'을 보내면 해당 LED에 불이 켜지고 '0'을 보내면 불이 꺼진다.

실험에서는 VxD에 프로그램 한바와 같이 VxD의 초기화 메시지 처리시 모든 LED가 소등되었으며 종료 메시지 처리시에는 모든 LED가 점등되었다. 데이터 출력 동작도 정상적이었으며 PIO 포트를 양방향 모드로 설정하고, 출력한 데이터를 다시 입력으로 받아들여 데이터의 입력 동작이 정상적으로 이루어짐을 확인하였다.

4. 결론 및 향후 연구

본 논문에서는 DDK를 이용한 전형적인 VxD 개발 기법을 알아보고, 간편하게 입출력 포트를 제어할 수 있는 포트 제어용 VxD를 개발하였다. 개발된 VxD를 이용하면 간단한 조작으로 여러 입출력 포트를 제어할 수 있으며 포트 제어 실험에서 무리 없이 정상 동작함을 확인하였다. 이로 인해 최근 활용도가 떨어지고 있는 입출력 포트를 이용하여 보안 관련 장비나 가전제품 제어에 활용할 수 있는 기반을 마련하였다.

향후 연구 과제로는 범용으로 사용할 수 있는 VxD를 개발하고 입출력 포트 제어에 필요한 부가적인 기능들을 추가, 보완하여 장비 제어에 활용할 수 있도록 하기 위한 추가적인 연구가 요망된다.

참고문헌

[1] Walter Oney, Forrest Foltz, "Programming the Microsoft Windows Driver Model", Microsoft Press, 1999
[2] Karen Hazzah, "Writing Windows VxDs and Device Drivers", R & D Technical Books, 1997
[3] 황광일, "Visual C++ Professional Programming Bible", 영진출판사, 1999
[4] 류성렬, "C 언어에 의한 디바이스 드라이버 작성 방법", 세화, 1995
[5] 김성환, "C++로 배우는 PC하드웨어", 피씨북, 1998
[6] 류기주외, "C로 미는 PC자동제어 응용 1", OHM사 1998
[7] 김민주, "마이크로프로세서와 주변장치의 응용", 생능, 1994