

3장. 윈도우즈 95의 시스템 구조

(Windows 95 System Architecture)

윈도우즈 95는 인텔칩을 기반으로 한 중급수준의 개인용 컴퓨터에서 16비트와 32비트 응용 프로그램을 실행할 수 있는 32비트 보호모드 운영 시스템이다. 윈도우즈 95는 윈도우즈 기반 응용 프로그램과 MS-DOS 기반 응용 프로그램에게 가상 메모리(4GB 까지, 해당 컴퓨터의 물리 메모리와 스왑공간 영역에 따라 다르다)와 선점적 멀티태스킹을 지원한다. 윈도우즈 3.1과 같이, 윈도우즈 95는 인텔 프로세서 칩을 기반으로 한 PC에서만 운영된다. 그러나 이전 버전의 윈도우즈와는 달리 윈도우즈 95는 80386이상의 프로세서를 필요로 하며 “386 향상모드”라 불리는 모드만을 사용한다. 윈도우즈 95는 응용 프로그램에게 기본적으로 윈도우즈 NT와 같은 환경을 제공하지만, 다른 프로그램의 우연한 실수나 의도적인 오동작으로부터 프로그램과 데이터를 분리시켜 주는 보안환경을 제공하지 않는다.

가상 메모리 (virtual memory)

하드디스크에 임시적으로 데이터를 저장함으로써 물리적으로 사용 가능한 것보다 더 많은 메모리를 응용 프로그램에게 제공하는 운영 시스템 기술.

특권 수준(privilege level)

프로그램이 실행하는 수준으로서, 프로그램에 의해 어떤 데이터가 액세스 될 수 있는지, 메모리의 어떤 코드가 실행될 수 있는지, 어떤 기계어 명령이 실행될 수 있는지를 결정한다. 인텔 프로세서에서는 0순위 권한(ring zero)이 제일 높고, 3순위 권한(ring three)이 제일 낮은 수준이다.

메모리 모델(memory model)

프로그램이 코드와 데이터의 어드레스를 지정하는데 사용되는 방법이다. 플랫 메모리 모델에서는 단일 세그먼트 내에서 프로세서의 어드레스 영역에 있는 모든 가상 메모리를 액세스 할 수 있다. 16비트 응용 프로그램은 종종 **라지** 모델을 사용하는데, 이것은 사용 가능한 가상 메모리를 64KB 크기의 세그먼트로 나눈다.

윈도우즈 95의 운영 시스템 구성 요소들은 인텔 프로세서에서 가장 신뢰도가 높은 특권 수준인 0순위 권한에서 실행하며 32비트 플랫 메모리 모델을 사용한다. 응용 프로그램들은 가장 신뢰도가 낮은 특권 수준인 3순위 권한에서 실행한다. 응용 프로그램은 역시 32비트 플랫 모델을 사용할 수 있으며, 전통적으로 16비트 프로그램에서 사용하던 메모리 모델(라지, 미디엄, 콤팩트, 스몰)도 사용할 수 있다.

윈도우즈 기반 응용 프로그램의 개발 프로그래머는 윈도우즈 운영 환경의 기초가 되는 세 개의 모듈(KERNEL, USER, GDI)을 배워야 한다. 응용 프로그램이 이 모듈들과 관련되어 있는 한 이 말이 사실이지만, 이것을 가지고는 운영체제를 전혀 설명할 수 없다. 사실 윈도우즈 KERNEL, USER, GDI 모듈들은 카드게임(Solitaire game)과 신뢰도가 같은 3순위 권한 프로그램이다. 윈도우즈 95에 있는 **진짜** 운영체제는 가상 머신 관리자(Virtual Machine Manager; VMM)에 있다. (이전 버전이 윈도우즈 3.x 버전에서도 그랬다)

이 책의 많은 부분이 VMM 그 자체와 시스템 프로그래머에게 의미 있는 VMM의 API들을 자세히 설명하고 있다. 이 장에서는 윈도우즈 95의 구조를 설명하기 위해 VMM을 개략적으로 설명한다. **가상 머신(virtual machine)**의 개념은 내부적으로 이것이 윈도우즈 95 실행의 중심이라는 것이다. “시스템” 가상 머신(System virtual machine; System VM)은 모든 윈도우즈 기반 응용 프로그램을 가지고 있으며, 다른 가상 머신은 MS-DOS 기반 응용 프로그램을 가지고 있다. 시스템 VM은 16비트와 32비트 윈도우즈 기반 응용 프로그램을 실행시킬 뿐만 아니라, 모든 것을 제어하는 선점적 멀티태스킹을 스케줄러(preemptive multitasking scheduler)하에서 프로세서를 공유하고 있는 여러 개의 32비트 실행 쓰레드를 가지고 있다. MS-DOS 가상 머신들은 호환성에 있어 중요하지만, MS-DOS 프로그램들은 이전 버전 윈도우즈에서 해왔던 것보다 그 입지가 많이 줄어들었다.

3.1 가상 머신 (Virtual Machines)

가상 머신은 윈도우즈 3.0에서 만들어 졌는데, 동시에 여러 개의 MS-DOS 응용 프로그램과 여러 개의 윈도우즈 응용 프로그램을 지원하기 위해 만들어졌다. 윈도우즈 프로그래머는 가상 머신이 멀티태스킹 환경 안에 들어 있다는 것을 알고 있으며, 그래서 일정 기간이 지난 후에 다른 응용 프로그램이 실행될 수 있도록 제어권을 반납한다. 윈도우즈 기반 프로그램들은 입력 디바이스를 사용함으로써 이러한 동작이 잘 이루어지고 있는데, 그것은 어떤 윈도우즈 핵심 요소(KERNEL, USER, GDI)들이 디바이스 드라이버와 응용 프로그램간의 매개체로 동작하고 있기 때문이다. 한편, MS-DOS 기반 응용 프로그램들은 이러한 동작이 이루어지지 않는 것으로 잘 알려져 있는데, 그것은 이러한 프로그램들이 키보드, 마우스, 디스플레이, 프로세서, 심지어 사용자까지도 자기만이 소유하고 있다고 생각하기 때문이다. 여기서 발생하는 문제점들을 열거해 본다면, MS-DOS는 그 자체적으로 여러 개의 응용 프로그램을 지원하지 않는 단일 쓰레드, 리얼모드, 재진입 불가능 코드 등이다.

실제로 한 개밖에 없는 프로세서와 리소스들을 몇 개의 응용 프로그램들이 공유하기 위해서 윈도우즈 95는 **가상 머신(virtual machine, VM)**을 사용한다. 가상 머신은 컴퓨터가 응용 프로그램에게 작용하는 방법과 똑같은 방식으로 작용한다. (이 작업은 주로 소프트웨어적으로 이루어진다) 신기하게도, API 선언으로부터 PC 구조를 볼 수 있는데, "API"의 일부 요소들은 하드웨어 I/O 시스템과 인터럽트를 바탕으로 한 BIOS나 MS-DOS 인터페이스 등을 포함하고 있다. 윈도우즈 95는 종종 회귀한 하드웨어를 다중으로 사용하기 위해 전통적인 "API" 대신 자체적인 소프트웨어로 바꾸기도 한다. 하지만 윈도우즈 95는 이러한 API들을 표준 인터페이스와 똑같이 표현함으로써 하위 호환을 가능하게 했다. 그래서 이전 버전에 맞도록 작성된 윈도우즈 응용 프로그램뿐만 아니라 대부분의 MS-DOS 응용 프로그램도 잘 실행된다.

3.1.1 가상 머신이라 무엇인가 (What Is a Virtual Machine)

1960년 IBM은 VM/370이라고 하는 운영 시스템을 개발했다. VM/370은 기계는 하나밖에 없는 상황에서 여러 개의 가상 머신을 시뮬레이트 해 줌으로써 전혀 다른 응용 프로그램들에게 선점적 멀티태스킹을 지원했다. VM/370 세션에서, 사용자는 원격 터미널 앞에 앉아 있지만 하면 되었는데, 이 원격 터미널은 콘트롤 프로그램 명령을 이용하여 실제 기계에 있는 IPL(Initail Program Load)의 푸시 버튼의 동작을 시뮬레이트 해주는 기능을 수행했다. 이 시뮬레이션은 너무나 완벽해서 나중에는 시스템 프로그래머들이 새로운 버전을 디버깅하기 위한 VM/370 자체의 복사본도 실행시킬 수 있었다.

IBM이 VM/370에서 풀었던 문제와 마이크로소프트가 윈도우즈 3.0에서 풀어야 하는 문제는 매우 비슷해 보인다. 이 문제란 응용 프로그램에게 기계에는 단지 혼자만 존재하고 있다고 착각하도록 해 주기 위한 것이었다. 그리고 가상 머신이란 개념은 다시 한번 이 해답을 제공해 주었다. 가상 머신에서 실행되고 있는 응용 프로그램이나 운영 시스템은 실제의 키보드나 마우스로부터 입력을 받으며 실제의 모니터로 출력하는 것이라고 믿을 수 있다. 제한적이기는 하지만 존재하는 프로세서와 모든 메모리를 자신만이 소유하고 있다고 하는 것까지 믿을 수 있다. 하드웨어와 소프트웨어의 **가상화(virtualize)**가 바로 이러한 신비스러운 것에 대한 열쇠다.

3.1.2 가상 하드웨어 (Virtual Hardware)

앞에서도 지적했듯이, 가상 하드웨어(역자 주 : 실제로는 소프트웨어 프로그램)는 이 소프트웨어가 할 수 있는 한 실제의 하드웨어가 동작하는 반응과 똑 같은 방법으로 동작한다. 예를 들어, MS-DOS 기반 응용 프로그램이 키보드로부터의 입력을 읽어야 하는 경우를 가정해 볼 수 있다. 응용 프로그램 코드는 `fgets`나 `_kbhit` 같은 자체적인 런타임 라이브러리를 사용할 것이고, 런타임 라이브러리는 결국 MS-DOS를 호출하기 위해 인터럽트 21h를 발생시키거나 직접 BIOS를 호출하기 위해 인터럽트 16h를 발생시킬 것이다. MS-DOS는 인터럽트 16h를 통해 BIOS를 통해 대화할 수 있는 키보드 드라이버를 가지고 있다. BIOS는 키보드나 인터럽트 콘트롤러 칩과 대화하기 위해 I/O 포트 동작(IN이나 OUT 명령)을 수행하고, 사용자가 그때그때 입력한 키에 따라 키보드 콘트롤러에서 발생한 하드웨어 인터럽트를 처리한다. 그림 3-1은 응용 프로그램과 시스템 호출간의 상호작용에 대한 전형적인 계층구조를 보여주고 있다.

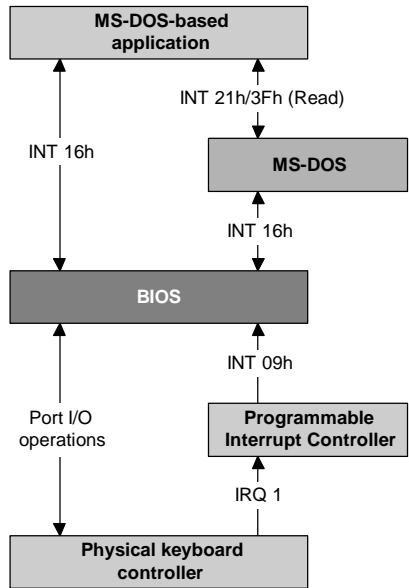


그림 3-1. MS-DOS 기반 응용 프로그램이 키보드로부터 입력을 읽는 방법

이전 버전의 윈도우는 MS-DOS하의 리얼모드에서 실행했으며, (씩 잘 실행되지는 못했다) 필자가 설명한 것과 똑 같은 방법으로 키보드를 사용했다. 그러나 윈도우가 보호모드로 바뀌었을 때, 곧 혼란이 발생했다. 이 혼란이란 보호모드 프로그램은 직접적으로 MS-DOS INT 21h 핸들러를 호출할 수 없다는 것뿐만 아니라, 리얼모드의 BIOS가 키보드 인터럽트를 처리할 수 없다는 것이다. 마이크로소프트는 BIOS나 MS-DOS에 조금도 의존하지 않고 키보드 핸들링의 세세한 것까지 모드 보호모드에서 처리하는 완전히 새로운 보호모드 운영 시스템을 만드는 큰 길을 택해야 하거나, 아니면 보호모드 코드가 리얼모드로 전달되고 혹은 그 반대로 되게 하는 방법을 만드는 중간 정도의 길을 택해야 했다. 사실, 마이크로소프트는 동시에 두 가지 모두를 선택했다. 큰길을 선택한 것은 OS/2 버전 2와 윈도우 NT에서 마무리되었고, 중간 길을 선택한 것은 윈도우 95에서 마무리 (어쨌든 현재로서는 그렇다) 되었고 몇 차례 배포된 윈도우 3.x에서도 이러한 마무리 판이 포함되어 있다. (작은 길을 선택하는 것은 전혀 아무 것도 할 것이 없었고, PC의 수 메가바이트의 메모리는 MS-DOS와 16비트와 리얼모드 운영 시스템에 의해서 강요되는 640KB의 장벽 앞에 무용지물이었다.)

보호모드 윈도우 구성요소들이 리얼모드 MS-DOS나 BIOS 요소들과 상호 동작하도록 해주는 메커니즘을 설명하기 위해서는 소프트웨어 가상화(software virtualization)이란 말을 사용할 수 있다. 소프트웨어 가상화는 기본적으로 운영체제의 도움을 필요로 하는데, 이것은 보호모드와 리얼모드 사이의 경계를 넘으려고 시도하는 호출을 가로채고, 여기서 인자로 사용된 레지스터를 적당히 조정한 후 프로세서의 운영모드를 바꾸어야 하기 때문이다. **가상 디바이스 드라이버(virtual device driver; VxD)**에 의해 호출되는 어떤 소프트웨어 구성요소는 보호모드 인터럽트 호출을 리얼모드를 위한 **인터럽트 벡터 테이블(Interrupt Vector Table)**을 통한 호출(연속 호출)로 바꾸어 준다. 이 변환 프로세스의 일부분으로, VxD는 보호모드의 익스텐디드 메모리로 지정된 파라미터를 리얼모드 운영체제에 의해 액세스하는데 적당한 리얼모드 파라미터로 생성한다. 그러나 리얼모드 운영체제는 리얼모드라고 하기보다는 가상 8086(virtual 8086; V86)모드에서 실행된다. 그리고 나서 VxD는 리얼모드 호출의 결과를 익스텐디드 메모리에 있는 보호모드 호출자로 되돌려 준다. 이 변환 역시 보호모드에서 실행되는 가상 버전의 MS-DOS와 BIOS를 만드는 것이다.

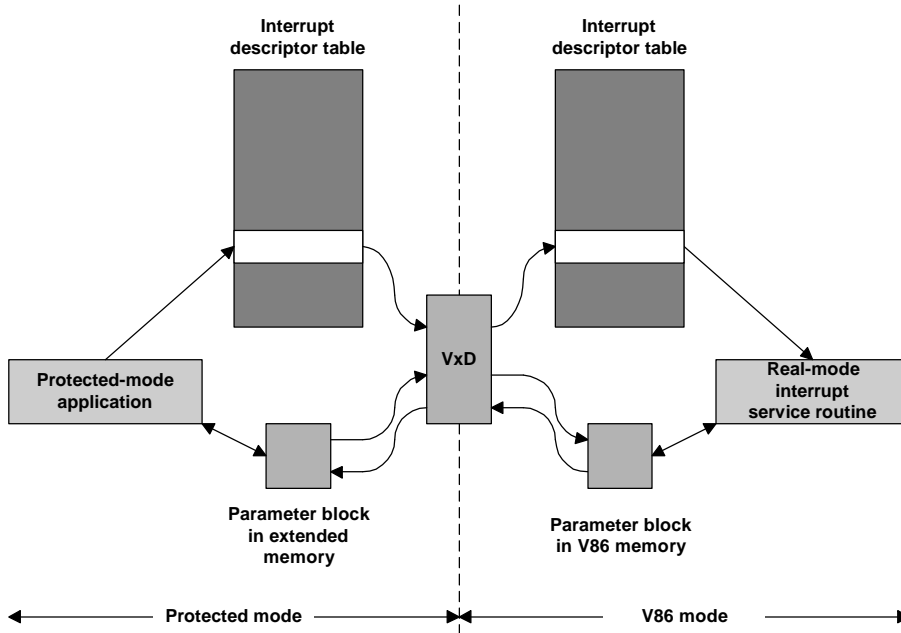


그림 3-2. MS-DOS와 BIOS의 소프트웨어 가상화

하드웨어 가상화는 소프트웨어의 가상화보다 한층 더 나아간다. 가상 하드웨어는 하드웨어 IRQ 라인에서 나타난다. 또한 가상 하드웨어는 IN이나 OUT 명령에 대한 응답, 특정하게 맵핑된 메모리의 위치를 바꾸는 것, 등에서 나타난다. 그러나 실제적으로는 겉으로 드러나는 것과는 매우 다를 것이다. 다음의 그림 3-3은 서로 다른 두 개의 MS-DOS 기반 응용 프로그램이 동시에 가상 키보드를 어떻게 액세스하는가에 대한 실례를 보여주고 있다. 중요한 것은, 이러한 프로그램들이 그림 3-1에서 보여준 간단한 MS-DOS 프로그램과 완전히 똑같은 기계어를 사용한다는 것이다. 지금까지 이러한 하드웨어 가상화에 필요한 몇 가지 가상 디바이스 드라이버를 계속 조금씩 언급해 왔다. 이제 가상화 된 키보드와 가상화 된 인터럽트 컨트롤러를 각각 VKD(가상 키보드를 짧게 나타낸 말)와 VPICD(가상 PIC 디바이스를 짧게 나타낸 말)라 부른다. 마우스에 대한 가상 디바이스 드라이버는 VMD 등이 될 것이다.

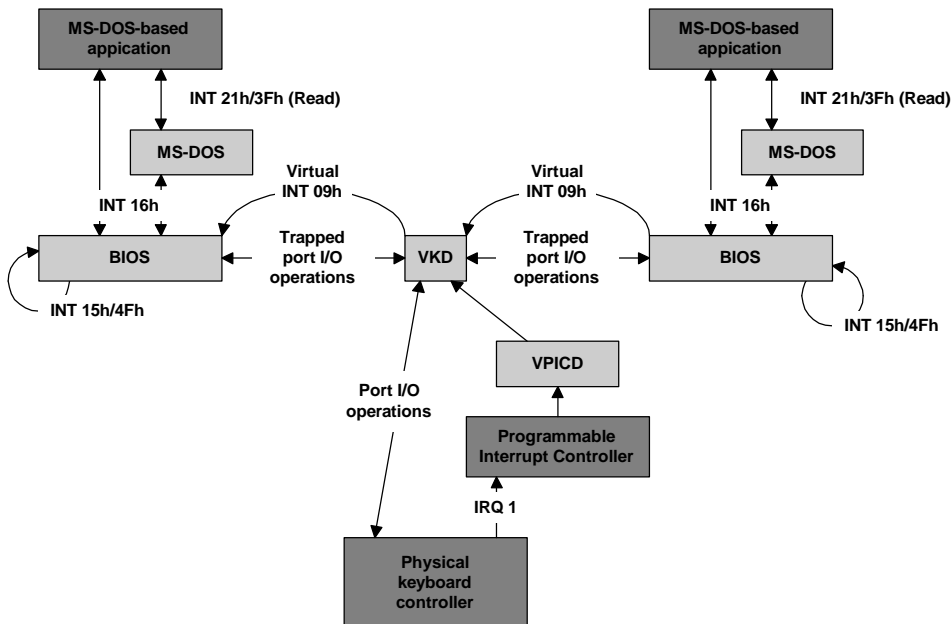


그림 3-3. 키보드 동작의 하드웨어 가상화

하드웨어 가상화는 인텔 80386 칩과 그 후속 칩의 몇 가지 특징과도 관련이 있다. 이런 특징의 하나인 **I/O 허용 마스크(I/O permission mask)**는 특정 I/O 포트의 모든 IN 명령이나 OUT 명령에 대하여 운영체제가 트랩 할 수 있도록 해준다. 하드웨어 인터럽트 핸들러에 있어서 어려운 부분은 핸들러가 PIC(Programmanle Interrupt Controller)로 혹은 PIC로부터 I/O 동작이 일어나기 때문인데, 이 포트 “트랩동작”은 소프트웨어가 인터럽트 하부 시스템을 쉽게 시뮬레이트 할 수 있도록 해준다. 또 다른 특징은 하드웨어 공조(hardware-assisted) 페이징인데, 페이징은 원래 운영체제가 가상 메모리를 지원하도록 하기 위한 것이지만, 메모리 위치에 대한 액세스를 가로챌 수 있다. 비디오 램을 가상화 하는 방법으로 페이징이 이를 가능케 해준다. 마지막으로, 프로세서의 V86 모드는 프로세서가 진짜는 보호모드에 있을 때, MS-DOS 기반 응용 프로그램에게 마치 리얼모드에 있는 것처럼 해준다. (역자 주 : 실제로V86 모드와 보호모드는 서로 다른 프로세서 모드이지만, V86 모드가 보호부분에서 보호모드와 많이 닮아있기 때문에 보호 리얼모드라 생각할 수 있다) V86 모드는 익스텐디드 메모리를 MS-DOS와 BIOS가 상주하는 리얼모드 어드레스 공간에 블록 단위로 맵핑 할 수 있으며, I/O 액세스 트랩, 필드 하드웨어, 소프트웨어 인터럽트 등을 허용해 준다.

3.1.3 가상 디바이스 드라이버 (Virtual Device Driver)

VMM과 커다란 VxD 집단이 바로 하드웨어 가상화의 가능성을 여는 열쇠이다. VxD에는 두 가지 종류가 있는데, 바로 **스태틱 VxD**와 **다이내믹 VxD**이다. 윈도우즈 95에서, VMM이 스태틱 VxD의 이름의 리스트를 얻는 곳은 시스템 레지스트리, SYSTEM.INI, 대략적으로 INT 2Fh 함수 1605를 바탕으로 한 소프트웨어적인 방법 등이다. VMM은 가상 머신을 지원하기 위한 뼈대를 만드는 동안 스태틱 VxD를 로드 한다. 로드 된 스태틱 VxD는 윈도우즈 세션동안 상주하게 된다.

물론 윈도우즈 95는 요구에 따라 VxD를 다이내믹하게 로드하고 해제할 수 있다. 이러한 기능의 1차원적인 사용자는 바로 윈도우즈 95의 구성 관리자(configuration Manager)와 입출력 감시자(Input/Output Supervisor; IOS)인데, 그들은 그 자체가 스태틱 VxD이다. 레지스트리와 하드웨어가 자신을 나타내는 다양한 프로토콜을 이용하여 구성관리자는 실제의 하드웨어에 맞는 다이내믹 VxD를 선택하고 로드 한다. 입출력 감시자는 VxD 프로그램 파일의 물리적 위치를 바탕으로 물리 디스크 드라이브를 위한 드라이버를 로드하기 위해 간단한 체계를 사용한다.

3.2 프로세스와 쓰레드 (Processes and Thread)

윈도우즈 95는 프로세스와 쓰레드의 개념에 있어서 윈도우즈 NT와 비슷하다. 즉, 모든 윈도우즈 기반 응용 프로그램은 하나의 프로세스를 가지고 있으며, 이 프로세스는 주어진 어드레스 공간과 하나 이상의 실행 쓰레드로 이루어져 있다. 각 쓰레드는 프로그램 진행 순서나 이 순서와 관계되어 있는 레지스터와 시스템 오브젝트의 전개 상태에 대응된다. 윈도우즈 95는 선점적 멀티태스킹 쓰레드들에게 우선권을 바탕으로 하는 체계를 사용한다.

가상머신, 프로세스, 쓰레드를 바탕으로 하는 완전히 일반적인 시스템에서 가상 머신은 여러 개의 프로세스를 가질 수 있으며, 프로세스는 여러 개의 쓰레드를 가질 수 있다. 그러나 윈도우즈 95에서는 이런 완전히 일반적인 모델은 지원하지 않는다. 윈도우즈 95 시스템에서 시스템 VM(System VM)이란 특별한 가상 머신인데, 이것은 모든 Win16과 Win32 응용 프로그램이 실행되는 곳이기 때문이다. VMM은 항상 시스템 VM을 생성하는데, 이 생성은 윈도우즈 95 세션을 시작하는 프로세스에서 매우 초기에 일어난다. 시스템 VM외에도 다른 추가적인 VM도 있을 수 있다. 이 추가적인 가상머신은 MS-DOS 기반 응용 프로그램을 위한 것이며, 각각 완전히 한 개의 프로세스와 한 개의 쓰레드를 가지고 있다.

Win16 응용 프로그램(Win16 Application): 16비트 윈도우즈 응용 프로그램

Win32 응용 프로그램(Win32 Application): 32비트 윈도우즈 응용 프로그램

앞에서도 언급했듯이 윈도우즈 95가 MS-DOS 가상 머신을 제한하기는 하지만, 명령 셸은 기존과 좀 다르게 만들어졌다. 즉, 윈도우즈 95의 MS-DOS 프롬프트에서 16비트나 32비트 응용 프로그램을 시작시킬 수 있다는 것이다. 윈도우즈 95는 MS-DOS 프롬프트에서 GUI 응용 프로그램을 일단 시작시키고, MS-DOS VM은 또 다른 명령 프롬프트가 나오도록 다시 제어권을 얻는다. 더구나 Win32 응용 프로그램은 그것이 어떤 방법으로 시작되었던지 상관없이 새로운 쓰레드를 자유롭게 생성할 수 있다. 그래서 이것은 갑자기 MS-DOS가 그래픽 수준으로 향상된 것인가 아니면 혹시 MS-DOS VM이 진짜 여러 개의 프로세스를 가지거나 프로세스마다 여러 개의 쓰레드를 가지고 있는가?

만약 이 질문에 “아니다”라고 생각했다면, 맞는 대답이다. 그러나 여기에 대하여 위험을 무릅쓰고 실험을 하기도 전에 또 다른 혼란스런 실험결과를 얻게 될 것이다. 소위 말하는 *콘솔모드(console-mode)*라고 하는 Win32 응용 프로그램을 만들 수 있는데, 이것은 윈도우즈 그래픽 인터페이스와 관계된 부분만 제외하면 모든 Win32 API를 사용할 수 있다. 부록 CD의 /CHAP03/THREAD 디렉토리에 있는 THREAD.EXE와 같이 멀티쓰레딩 실행을 하는 콘솔모드 응용 프로그램을 실행시켜 보라. THREAD.C의 내용은 다음과 같다. (물론 부록 CD에도 있다.)

```

THREAD.C
#include <windows.h>
#include <stdio.h>

DWORD WINAPI mythread(LPVOID junk)
{
    // mythread
    puts("Hello from the thread!");
    return 0;
}
// mythread

int main(int argc, char *argv[])
{
    // main
    DWORD tid;
    HANDLE hThread;

    hThread = CreateThread(NULL, 0, mythread, NULL, 0, &tid);
    if( hThread)
    {
        // thread has been started
        puts("Hello from the main program!");
        WaitForSingleObject(hThread, INFINITE);
    }
    // thread has been started
    return 0;
}
// main

```

윈도우즈 95의 MS-DOS 프롬프트에서 이 응용 프로그램을 실행시키면 *mythread* 쓰레드가 생성되고 올바로 동작할 것이다. 출력은 MS-DOS 창에 나타날 것이며, 마치 실제로 윈도우즈 NT를 실행시키는 것 같을 것이다. 그리고 이 응용 프로그램이 종료하기 전까지 MS-DOS 프롬프트로 되돌아오지 않을 것이다. 이것은 Win32 응용 프로그램 실행이 MS-DOS 기반 응용 프로그램과 꼭 닮아 있으며, 이제까지 필자가 설명한 MS-DOS 가상 머신의 제한과는 분명히 다르게 새로운 쓰레드를 만든다는 것이다.

명령 셸은 이 샘플 프로그램을 처리하기 위해 몇 가지 정교한 기술을 이용한다. MS-DOS 명령을 입력하면,

MS-DOS의 COMMAND.COM 명령 셸은 이 입력된 실행파일을 로드하고 실행시키기 위해 MS-DOS 인터럽트(INT 21h, 함수 4Bh)를 발생시킨다. VMM은 이 인터럽트를 가로채서 Win32 응용 프로그램을 실행시키려 했는지를 알아낸다. VMM은 이 입력된 명령을 MS-DOS VM이 아니라 시스템 VM에서 실행시킨다. 만약 GUI 응용 프로그램을 시작시키면, MS-DOS는 즉시 제어권을 다시 얻게되고, 이 응용 프로그램은 자체의 독립적인 방법으로 실행된다. 만약 콘솔모드 응용 프로그램을 시작시키면, VMM은 이 응용 프로그램이 종료하기 전까지 INT 21h 함수 4Bh로부터 리턴하지 않는다. 그러나 어떤 경우든 응용 프로그램은 새로운 프로세스를 가지고 시스템 VM에서 실행된다.

3.3 윈도우즈 응용 프로그램 (Windows-based Applications)

윈도우즈 기반 응용 프로그램에는 16비트와 32비트가 있다. Win16 응용 프로그램은 윈도우즈 1.0에서부터 3.11까지 발전되어 장기간의 사용으로 검증이 충분히 이루어진(낡은 것은 아니다) 16비트 윈도우즈 API를 사용한다. Win32 응용 프로그램은 원래 윈도우즈 NT 버전 3.1에서 개발된 Win32 API의 서브세트를 사용한다. 두 종류의 응용 프로그램 모두 80386 이상 프로세서 칩의 보호모드를 사용하며, 이로 인해 가상 머신 관리자가 제공하는 가상 메모리의 전 영역을 액세스한다. Win32 응용 프로그램은 여러 개의 쓰레드를 가질 수 있는 반면, Win16 응용 프로그램은 시작 시에 만들어진 단 한 개의 쓰레드만 가질 수 있다. Win32 응용 프로그램은 VMM의 스케줄링 서브 시스템이 모두 제어하는 선점적 멀티태스킹에 속하지만, 반면 Win16 응용 프로그램은 그들끼리 협력하는 멀티태스킹을 수행한다. (역자 주 : Win16 응용 프로그램은 제어권을 얻은 후 이를 잠시 이용하고 다른 응용 프로그램이 사용할 수 있도록 즉시 제어권을 반납하는 형식으로 멀티태스킹을 구현하고 있으므로, 응용 프로그램간의 협력 하에 멀티태스킹이 이루어지고 있는 것이다. 따라서 이하 이를 협력 멀티태스킹이라 하겠다.)

일반적으로 32비트 응용 프로그램이 16비트 응용 프로그램보다 더 빠르고 메모리 사용에 있어 더 효율적이다. 이것은 32비트 프로그램은 플랫폼 메모리 모델을 사용하기 때문이며, 이로 인하여 한 개의 세그먼트로 모든 가상 메모리의 코드와 데이터를 주소 지정할 수 있다. 16비트 프로그램은 64KB보다 큰 메모리 영역을 액세스하기 위해 계속적으로 세그먼트 선택터를 세그먼트 레지스터에 로드 해야만 한다. 보호모드에서 선택터를 로드하는 것은 그냥 32비트 플랫폼 포인터(역자 주 : 32비트 오프셋)를 로드 하는 것 보다 7배의 비싼 대가를 치러야 한다. KERNEL, USER, GDI 모듈에 있는 기본적인 윈도우즈 API는 메모리에 있어 분명 64 KB보다 크며, 모든 응용 프로그램에서 자주 사용된다. 고리고 16비트 윈도우즈 멀티태스킹 모델에 의해 강요되는 협력 멀티태스킹(cooperative multitasking)은 분명 한 응용 프로그램에서 다른 응용 프로그램으로 빈번히 전환하기 때문에, 이것은 선택터를 로드 하는 일이 자주 발생할 가능성이 매우 높다.

3.3.1 Win32 응용 프로그램 (Win32 Applications)

32비트 응용 프로그램이 16비트 응용 프로그램보다 작업을 더 잘 수행할 뿐만 아니라 또 다른 이점이 있다. Win32 API는 윈도우즈가 실행되는 다른 모든 플랫폼에 이식되어 있다. (역자 주 : 실제로는 몇몇 플랫폼이다. 왜냐하면 아직 윈도우즈 NT처럼 하는 일없이 등치만 큰 운영체제를 돌릴 만큼 막강한 컴퓨터가 그리 많지 않기 때문이다.) 따라서 잘 만든 Win32 응용 프로그램은 간단히 다시 컴파일하고 링크만 해서 다른 모든 하드웨어 플랫폼에 이식할 수 있다. 또한 Win32 API는 16비트 API에 비해 더 풍부하고 더 논리적이며, 프로그래머가 시스템이나 다른 응용 프로그램과 상호 작용하도록 프로그램 하기가 더 쉽다. 한가지 예를 인용한다면, Win32 응용 프로그램은 디스크에 주소 지정된 파일을 가상 메모리로 만들어 주는 가상 파일 매핑을 바로 만들 수 있다. 이로써 데이터 베이스 관리, 실행파일 로드와 같은 다양한 일들을 엄청나게 간단히 할 수 있다.

윈도우즈 NT만이 Win32 API의 모든 요소를 제공한다. 윈도우즈 95도 상당량의 서브세트를 제공하지만 여기에는 보안기능, 이벤트 로깅, 유니코드 지원이 빠져있다. 또한 윈도우즈 95는 소수의 Win32에 있어서 윈도우즈 NT와는 좀 다르게 동작한다. 이것은 우선 윈도우즈 95의 이런 함수가 16비트 코드를 통해 수행되기 때문이다. 윈도우즈 95 Win32 API의 수행에 대하여는 *Programmer's Guide to Microsoft Windows 95* (마이크로소프트 출판, 1995)의

3조항에서 더 자세히 설명되어 있다.

응용 프로그램 프로그래머에게 있어 이러한 이점이 있음에도 불구하고, Win32 응용 프로그램은 가상 머신 관리자에게 문제가 되기도 한다. 이것은 믿든 믿지 않든 간에, VMM은 자신의 심장 깊숙한 부분에 시스템 VM이 16비트 코드로 실행하고 있다고 믿는다. 그로 인해 VMM이나 다른 VxD와 대화하기 위한 시스템 프로그래밍 인터페이스 대부분은 16비트 프로그램만이 가능하다. Win32 응용 프로그램이 이런 인터페이스를 사용하기 위해 소프트웨어 인터럽트를 발생시켰을 때, 곧바로 충돌이 발생한다. 마이크로소프트는 이 놀라운 결과가 이식 불가능한 너무 많은 Win32 응용 프로그램이 만들어질 것이라는 두려움으로 인하여 하부 디자인에 대한 결정을 바꾸고 싶어하지 않았다.

MS-DOS나 BIOS와 직접적으로 작업하기를 원하는 Win32 응용 프로그램은 Win32 API를 통해서 어떤 것을 보내려 하는 것에 대하여 제한을 받는다. 물론 API는 매우 강력하다. 응용 프로그램은 프로세스나 쓰레드를 다루거나 동기화하기 위해 자신의 뜻대로 호출할 수 있는 풍부한 API를 가지고 있다. 윈도우즈 95의 모양을 제어하는 시스템 레지스트리는 Win32 호출을 통한 때 액세스하고 변경하기가 쉽다. 어떤 응용 프로그램은 디버깅 프리미티브를 사용하여 다른 응용 프로그램을 제어할 수 있으며, 또한 광범위한 성능측정 API도 있다.

필요한 Win32 API가 없을 때, 응용 프로그램은 다른 VxD로 정보를 보내거나 받는데 *DeviceIoControl*도 사용할 수 있다. 예를 들어, Win32 응용 프로그램은 VWIN32 디바이스에 IOCTL 코드들을 보내서 저 수준 디스크 입출력(I/O)을 할 수 있다. 더욱이 VxD는 그들 스스로가 Win32 응용 프로그램의 콜백을 초기화하기 위해 *비동기 프로시저 호출(asynchronous procedure call)*을 사용할 수 있다. 이러한 메커니즘에 의존하는 응용 프로그램은 Win95나 Win95 향후 버전에서는 동작하지만, 다른 플랫폼에서는 제대로 동작하지 않는다는 것을 주의하라. 이러한 두 가지 방법이 보통 전통적인 방법보다 이식성이 좀 떨어지더라도 더 좋은 응용 프로그램을 만들 수 있는지는 필자도 잘 모르겠다. 그러나 필자의 경우, 이 방법들이 더 훌륭하다고 입증할 수 없더라도 어떠한 방법으로 동작하는지 설명할 수 있다면 이런 방법들로 일을 한다.

3.3.2 Win16 응용 프로그램 (Win16 Applications)

윈도우즈 95에서, 지금까지 인텔칩에서 해왔던 16비트 프로그래밍보다는 32비트 프로그래밍이 매우 매력적이기는 하지만, 16비트 프로그램을 계속적으로 돌릴 수 있게 하는 것은 중요한 부분이다. 호환성 견지에서만 본다면, 윈도우즈 95가 기존의 16비트 기반 응용 프로그램을 지원하지 않으면 상업적으로 실패하는 것은 명백하다. 물론 윈도우즈 95는 다소 깊숙한 부분까지 호환성을 제공한다. 예를 들어, 어떤 중요한 16비트 사용 프로그램들은 현재 사용하는 윈도우즈의 버전을 알아내기 위해 다음과 같은 코드를 사용한다.

```
BOOL bAtLeast_3_10 = ((WORD)GetVersion() >= 0x0A03); // 틀렸음
```

GetVersion API는 윈도우즈의 주버전(major version)을 하위 바이트에, 부 버전(minor version)을 상위 바이트에 넣어 리턴 한다. 버전이 같은지를 비교하는 것이 아니라면, 주 버전과 부 버전을 분리해서 비교하거나, 다음의 예와 같이 우선 주 버전이 상위에 가도록 바이트를 바꿔야 한다.

```
WORD wVersion = (WORD)GetVersion();  
wVersion = (LOBYTE(wVersion) << 8) | HIBYTE(wVersion);  
BOOL bAtLeast_3_10 = wVersion >= 0x030A; // 맞음
```

마이크로소프트는 원래 윈도우즈 95의 내부 명칭으로 4.00을 사용하려 했다. 실제로, Win32의 *GetVersion* 함수 호출자는 0x0004를 리턴 받으며, VxD의 *Get_VMM_Version* 호출자도 또한 4.00에서 실행하고 있다고 알게 될 것이다. 그러나 기존에 많이 사용하던 16비트 응용 프로그램들의 잘못된 버전 검사 때문에 마이크로소프트는 결국 16비트의 *GetVersion* API 호출은 0x0004 대신 0x5F03 (이것은 3.95)를 리턴 하도록 바꾸었다. 그래서 다음의 샘플 프로그램은 Win16 응용 프로그램인지 Win32 응용 프로그램인지에 따라 서로 다른 결과를 만들 것이다.

```

GETVER.C

#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance,
                  HINSTANCE hPrevInstance,
                  LPSTR lpCmd, int nShow)
{
    char msg[128];
    wprintf(msg, "Windows version is %4.4X", (WORD)GetVersion());
    MessageBox(GetFocus(), msg, "GetVersion Value",
               MB_OK | MB_ICONINFORMATION);
    return 0;
}

```

GETVER.C와 GETVER.EXE의 16비트와 32비트 버전은 각각 부록 CD의 /CHAP03/GETVER16과 /CHAP03/GETVER32 디렉토리에 포함되어 있다.

윈도우즈 95에서 16비트 응용 프로그램에 대한 메모리 관리와 16비트 응용 프로그램간의 멀티태스킹 처리는 호환성에 대하여 명백히 보여주고 있다. 윈도우즈 3.x에서, 모든 윈도우즈 기반 응용 프로그램과 MS-DOS 기반 응용 프로그램이 단일 4GB의 가상 어드레스를 서로 공유했기 때문에 메모리 블록의 선형 어드레스는 매우 무관심한 문제였다. 그러나 윈도우즈 95에서는 각각의 Win32 프로세스가 00400000h에서 80000000h까지의 선형 어드레스를 차지하는 자체적인 가상 메모리 영역을 소유하고 있다. 따라서 윈도우즈 NT같이, Win32 응용 프로그램은 파일 맵핑(file mapping)이라는 방법을 제외하고는 직접 메모리를 공유할 수 없다. 그러나 Win16 응용 프로그램은 서로간 이라든가 DPMI(DOS Protected Mode Interface)를 사용하는 확장 DOS 응용 프로그램간에 무차별적으로 메모리를 공유하려고 한다. 이러한 이유로 윈도우즈 95는 모든 Win16 응용 프로그램, *GlobalAlloc*를 통해 할당받은 메모리 블록, DPMI 호출을 통해 할당받은 메모리를 80000000h에서 C0000000h의 선형공간에 위치시킨다. 이 영역은 모든 쓰레드간에 공유된다.

Win16 응용 프로그램은 여전히 협력 멀티태스킹(cooperative multitasking)을 이루기 위해 자발적으로 제어권을 반납해야 한다. 몇몇 응용 프로그램은 특정하게 정의된 양보시점을 제외하고는 제어권을 놓지 않는 수법을 사용하기도 한다. (역자 주 : 다른 응용 프로그램이 서로간에 순차적으로 동작하기 위해 이런 수법을 많이 이용했었다.) 따라서 윈도우즈 95는 호환성을 위해 Win16 응용 프로그램에게 이전 버전의 윈도우즈가 했던 것과 같은 협력 멀티태스킹 모델을 제공한다. 실제로 각 Win16 응용 프로그램은 독특한 프로세스와 쓰레드 부분이 있다. (Soft-Ice/W 같은 디버거를 사용하여 0순위권한의 콘트롤 구조체를 검사하지 않는다면 이것을 알아낼 방법은 없다.) 그러나 Win16 쓰레드는 Win16Mutex 상호배제(mutual exclusion; mutex) 세마포어(semaphore)를 요구한 후 실행한다. 한번에 한 개의 Win16 프로그램만을 허용하기 위해 이전버전의 윈도우즈를 흉내내기 위해 뮤텍스를 사용하는 것이 VMM의 선점적 멀티태스킹에 의한 스케줄링에 적당하다.

3.3.3 32비트 코드와 16비트 코드와의 혼합 (Mixing 32-bit and 16-bit Code)

Win32 프로그램과 Win16 프로그램은 모두 시스템 VM에서 같이 실행되고 있으므로, 이 두 개의 프로그램이 같이 작업하게 하고 싶을 때가 있을 것이다. 그러나 윈도우즈 환경에서는 여러 가지 이유로 이들의 혼합 프로그램밍이 어렵다. Win16 프로그램에 의해 사용되는 NE(new executable) 파일 포맷은 16비트 세그먼트만을 허용하며, Win32 프로그램에 의해 사용되는 PE(portable executable) 파일 포맷은 32비트 세그먼트만을 허용한다. Win32 프로그램은 모두 32비트 오프셋을 사용하는 반면, Win16 프로그램은 세그먼트의 조합을 사용하며 16비트 레지스터와

16비트 오프셋을 사용한다. 그래서 Win32 프로그램이 Win16 프로그램을 성공적으로 호출하기 위해서는 **썹크(thunk)**가 필요하다. 여러 가지 썹크 중에서, 여기에서 필요한 썹크는 32비트 코드에서 16비트 코드로의 변환, 스택 주소 지정 변환, 32비트 정수에서 16비트 정수로 잘라내는 변환 등을 한다. 반대 방향으로의 작업(역자 주 : Win16 프로그램에서 32비트 코드를 호출하는 것)은 이와 비슷한 반대 작업이 필요하다. 이러한 방법에는 여러 가지가 있으며, 이 책에서는 설명할 수 없는 많은 복잡한 것이 관계되어 있다.

프로그래머로 하여금 윈도우즈 95에서 16비트와 32비트간의 자체적인 인터페이스를 쉽게 만들 수 있도록 한 **썹크 컴파일러(thunk compiler)**에 대해서는 *Programmer's Guide to Microsoft Windows 95*를 참고하면 된다. (역자 주 : 여기서 썹크 컴파일러는 흔히 생각하는 컴파일러가 아니라 16비트 프로그램과 32비트 코드간의 실행을 용이하도록 어드레사나 스택 변환을 해주는 일종의 번역기이다) 이 썹크 컴파일러는 윈도우즈 3.1의 Win32s 부속 시스템에서 실행되는 Win32 프로그램이 Win16 DLL을 호출할 수 있도록 해주는 **유니버설 썹크(universal thunk)**를 엄청나게 개량했으며, 또한 윈도우즈 NT에서 실행되는 **제너릭 썹크(generic thunk)**를 개량했다. 이전의 이 두 가지 썹크 메커니즘들(역자 주 : 유니버설 썹크와 제너릭 썹크의 메커니즘)은 특이하기도 하지만 무엇보다도 단방향이었다. 반면 윈도우즈 95 썹크 메커니즘은 강인하고 양방향이다. (그러나 불행히도 이전의 유니버설 썹크나 제너릭 썹크와 호환성이 없다.)

윈도우즈 95에서 16비트 코드와 32비트 코드 사이의 썹크는 매우 잘 동작하며, 이것은 마이크로소프트의 윈도우즈 95가 윈도우즈 프로그램을 실행시키는데 있어 썹크에 매우 깊이 의존하고 있음을 보여주는 것이다. 앞에서 주마간산 격으로 언급한 KERNEL, USER, GDI란 말은 이들 요소의 둘 중 한 개만을 말한 것이다. 사실 이들 시스템 요소에는 16비트 버전과 32비트 버전이 있으며, 이들은 서로 썹크로 연결되어 있다. 예를 들어, KERNEL32.DLL은 이런 일을 하기 위해 KRNL386.EXE를 호출하며, 그 반대도 마찬가지이다. 윈도우즈 오브젝트를 관리하는 것과 관련된 대부분의 작업은 USER.EXE (16비트)에서 이루어지며 USER32.DLL은 이를 약간 도우는 정도이다. 이와 비슷하게 GDI의 32비트 버전인 GDI32.DLL은 16비트 버전인 GDI.EXE와 작업을 공유한다.

그러나 16비트 시스템 요소들은 근본적으로 윈도우즈 3.1에서와 똑같이 그대로 남아 있으며 재진입이 불가능하기 때문에, 윈도우즈 95는 이들을 실행하는데 동기화가 필요하다. **Win16Mutex** 세마포어는 Win16 응용 프로그램을 동기화 할 뿐만 아니라 이러한 목적(역자 주 : 즉, 32비트 시스템 요소들이 16비트 시스템 요소들을 호출할 때의 동기화)을 지원하기도 한다. GDI32와 같은 32비트 시스템 요소는 이러한 16비트 모듈중의 하나를 호출해야 할 때마다 **Win16Mutex** 세마포어를 요구한다. 이 결과는 한번에 한 개의 쓰레드(16비트든 32비트든 간에)만이 시스템 VM에 있는 16비트 코드를 실행시킬 수 있다. Matt Pietrek가 쓴 *"Windows 95 System Programming Secrets"* (IDG Books, 1995)를 보면, 윈도우즈 95의 KERNEL, USER, GDI 서브시스템이 어떻게 동작하는지 많은 상황에 대하여 전반적으로 논의하고 있다.

3.4 MS-DOS와 MS-DOS 응용 프로그램

(MS-DOS and MS-DOS-based Applications)

윈도우즈는 원래 MS-DOS 위에서 만들어진 운영 "환경"(operating "environment")에 지나지 않았다. 실제로 오래 전 윈도우즈는 커맨드 프롬프트보다는 좀더 쉬운 단지 장식적인 명령 셸(fancy command shell)이 지나지 않았다. 시간이 지나 윈도우즈가 발전함에 따라 MS-DOS에 의해 제공되는 운영체제 기능의 핵심들을 점점 더 포함하게 되었다. 윈도우즈 95 이전의 윈도우즈 구조에 대한 그림에서 아래쪽에 있는 하드웨어 근처에 MS-DOS와 많은 리얼모드 드라이버 집단을 볼 수 있다. (그림 3-4 참조) 윈도우즈 95에서는 아래쪽에 있는 하드웨어 근처에 가상머신 관리자와 많은 VxD 집단이 있으며, MS-DOS와 작은 양의 리얼모드 드라이버는 가상머신 관리자의 클라이언트임을 볼 수 있다. (그림 3-5 참조)

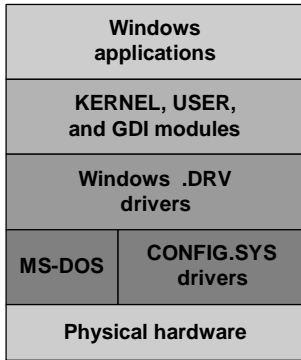


그림 3-4. 윈도우즈 95 이전 버전에서 MS-DOS의 위치

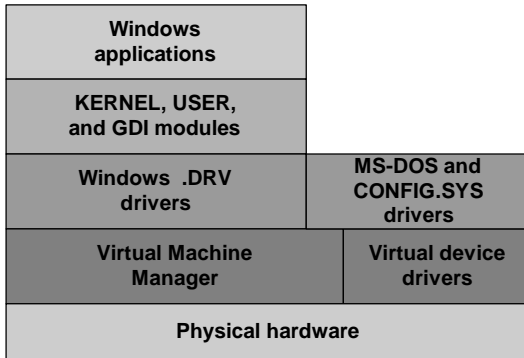


그림 3-5. 윈도우즈 95에서 MS-DOS의 위치

이러한 방향 결정이 MS-DOS에서 VMM으로 이동한다는 것은 PC 하드웨어와 소프트웨어 시장의 전개 상황을 반영하는 것이다. 과거에는 8086이나 80286 칩을 토대로 하는 낮은 능력의 컴퓨터가 데스크탑 컴퓨터의 대부분을 차지하고 있었으며, 이러한 컴퓨터는 윈도우즈보다는 MS-DOS가 더 잘 실행되었다. 응용 프로그램 제작자들은 충분한 메모리 성능이나 사용자 편의를 위해 아마도 도스 확장자(DOS Extender)나 상용 라이브러리를 사용하여 MS-DOS 프로그램을 작성하는 것이 더 쉽다는 것을 알고 있었다. 요즘에 팔리는 컴퓨터는 386 향상모드에서 윈도우즈를 실행할 수 있을 정도인데다가 더 이상 MS-DOS를 사용할 이유가 없을 정도로 매우 막강한 프로세서를 가지고 있다. 더욱이 문서나 교육을 통한 마이크로소프트의 도움과 사용자(end user)들의 상당한 압력의 결과로 인하여, 요즘의 응용 프로그램 제조회사들은 그래픽 윈도우즈를 겨냥한 훌륭한 응용 프로그램을 일상적으로 만들어 내고 있다.

대부분의 새 응용 프로그램은 윈도우즈를 목표로 하며, 대부분의 컴퓨터가 32비트 보호모드 프로그램을 실행시킬 수 있는데, 왜 굳이 MS-DOS에 대하여 걱정하는가? 필자는 이 대답이 단지 호환성과 친밀성이라고 생각한다. 필자가 이 장에서 얘기했듯이, 필자는 보통 수준의 컴퓨터에 윈도우즈 95를 사용하고 있다. 그러나 필자는 네트워크를 실행하는데 리얼모드 드라이버를 사용하고 있다. 왜냐하면 윈도우즈 95의 베타버전이 나온 지 2년이 지났음에도 불구하고, 필자가 사용하는 네트워크 제조회사가 이 드라이버를 보호모드로 포팅하지 않았기 때문이다. 필자의 컴퓨터는 DIR, GREP, 기타의 명령을 입력할 수 있도록 데스크탑에 MS-DOS 박스를 가지고 있다. 필자는 또한 여기에 이따금 MS-DOS에서만 실행되는 게임 프로그램을 실행시키기도 한다. 마이크로소프트가 OS/2와 윈도우즈 NT의 경험에서 가장 뼈저리게 배운 교훈은 그 운영체제에서 실행되는 괜찮은 응용 프로그램이 많이 없다면 쉽게 시장을 뚫을 수 없다는 것이었다. 윈도우즈 95는 윈도우즈 3.1보다 더 뛰어나고 MS-DOS와 이들이 실행시키던 모든 프로그램을 실행시킬 수 있기 때문에 성공할 것이다. (역자 주 : 성공했다.) 또한 윈도우즈 95는 32비트 응용 프로그램 개발을 촉진시킬 것이다. 왜냐하면 32비트 응용 프로그램을 실행시킬 수 있는 중저가 플랫폼을 제공하기 때문이다. (필자는 앞에서 윈도우즈 3.1의 Win32s 서버 시스템을 짚아 내렸는데, 대부분의 제작자들은 이것을 장난감 정도로 치부해 온 것 같다.)

윈도우즈 95와 MS-DOS의 공생은 컴퓨터가 부팅 할 때 시작된다. 하드디스크에 있는 부트스트랩(bootstrap) 로더는 MS-DOS 버전 7.0을 가지고 있는 IO.SYS 파일을 로드하고 실행한다. 아무튼 이 버전이란 아래에서 보여주는 결과와 같이 도스의 INT 21h 함수 30h가 알려주는 버전이다. (버전 번호는 부 버전/주 버전의 형태로 AX 레지스터를 통해 리턴 된다)

```
C:/>debug
-a 100
2D01:0100 mov ah, 30
2E01:0102 int 21
2E01:0104 int 3
2E01:0105
-g

AX=0007 BX=FF00 ...
DS=2E01 ES=2E01 ...
2E01:0104 CC INT 3
-q
```

MS-DOS 7.0은 대체로 이전의 MS-DOS에 사용했던 CONFIG.SYS와 AUTOEXEC.BAT를 처리하고 나서, 윈도우즈 95 디렉토리에 있는 WIN.COM을 실행시키기 위해 WIN 명령을 자동으로 발생시킨다. 윈도우즈 95의 가상 머신 관리자가 발전하고, 이 가상 머신 관리자는 기계를 장악하고 그래픽 운영 환경을 실행시킨다. 여기에 이 처리를 가로막고 MS-DOS에 남아 있게 하는 몇 가지 방법이 있다. 오랜 동안 필자는 WIN.COM보다 앞에 있는 패스에 모조품 WIN.BAT 파일을 두었다. 이 BAT 파일은 MS-DOS가 자동으로 생성된 WIN 명령을 처리할 때 WIN.COM 대신 제어권을 가지게 된다. 또한 부팅 처리를 하는 동안 F8키를 누른 다음 커맨드 프롬프트에서 중지하도록 지정한다. 요즘은 필자는 MS-DOS.SYS 파일 (보통 히든 파일이다)에 다음과 같이 설정해 놓고 실행시킨다.

```
[Options]
BootGUI=0
```

이렇게 하면 이전 버전의 윈도우즈를 실행시키는 것을 포함하여 커맨드 프롬프트에서 해왔던 것을 DOS 모드 처럼 할 수 있다. 물론 여전히 그렇게 하는 것이 좋다면 말이다. 그러나 이와 같이 좀 색다른 방법을 사용하지 않고서는 MS-DOS에서 정지시킬 수 있는 지점이 정말로 그렇게 많지 않다. MS-DOS에서 윈도우즈 95를 실행시키기 위해서는, 윈도우즈 95 디렉토리의 WIN.COM을 실행시키기만 하면 된다.

일단 그래픽 셸이 나타나게 되고 실행하게 되면 VMM이나 윈도우즈 95의 다른 부분이 MS-DOS를 어떻게 실행하는지 가까이에서 검사해 본다면 좀더 흥미로운 광경이 나타난다. 앤드류 슐먼(Andrew Schulman)은 *Unauthorized Windows 95* (IDG Books, 1994)에서 이 주제에 대하여 매우 깊이 탐구했었다. 앤드류가 발견한 정보중의 한 토막 중에는, 윈도우즈 95가 여전히 모든 응용 프로그램에 대해 다소 평범한 PSP(program segment prefix)를 만들며 이 작업을 하는데 MS-DOS에 의존한다는 것이다. 그러나 이제 MS-DOS는 32비트 보호모드에서 발생하는 인터럽트 핸들링이나 I/O 동작과 같은 대부분이 시스템 서비스에서만끔은 저 뒤로 밀려나 있다. MS-DOS 파일 시스템까지도 이제는 (보통) VxD 코드에 의해 수행된다.

그러나 호환성에 있어서 MS-DOS 드라이버나 램 상주 프로그램(terminate-and-stay-resident; TSR)이 계속적으로 작업할 수 있도록 하는 것이 필요하다. 호환성에 있어서 또한, Win16 프로그램은 여전히 운영체제 서비스를 요청하기 위해 INT 21h를 포함한 소프트웨어 인터럽트를 사용할 수 있도록 하는 것이 필요하다. VMM이나 VMM과 함께 동작하는 VxD 집합체들이 MS-DOS 그 자체를 가상화 하여 호환성을 보장하고 있다. 다시 말하면, 어떤 프로그램이 (리얼모드 MS-DOS 프로그램이라고 하더라도) 파일 시스템 서비스를 요청하기 위해 INT 21h를 사용하면, VxD는 이 요청을 서비스하고 디스크 I/O를 수행할 것이다. 리얼모드 드라이버가 디스크를 관리하는 포트에 I/O 동작을 수행하면, VxD는 이 액세스를 가로채서 관리할 것이다. 따라서 실제로 있어서 윈도우즈 95는 MS-DOS를 응용 프로그램처럼 다루고 있는 것이다.