Chapter 01

윈도우 프로그래밍 기초

* 학습목표

- 윈도우 운영체제와 윈도우 응용 프로그램의 특징을 이해한다.
- SDK 응용 프로그램의 작성 과정과 기본 구조 및 동작 원리를 이해한다.
- MFC 응용 프로그램의 작성 과정과 기본 구조 및 동작 원리를 이해한다.
- 비주얼 C++ 개발 환경의 사용법을 익힌다.

Section

1 윈도우 프로그래밍 개요

윈도우 프로그래밍이란 윈도우 운영체제(Windows Operating System)에서 구동되는 응용 프로그램을 만드는 것이다. 따라서 윈도우 프로그래밍을 하기 전, 먼저 윈도우 운영체제와 윈도우 응용 프로그램의 특징을 이해해야 한다.

❶ 윈도우 운영체제의 특징

윈도우는 컴퓨터를 사용하는 사람이라면 누구나 익숙한 환경이다. 여기서는 윈도우 응용 프로그램을 제작하는 관점에서 윈도우의 특징을 살펴본다.

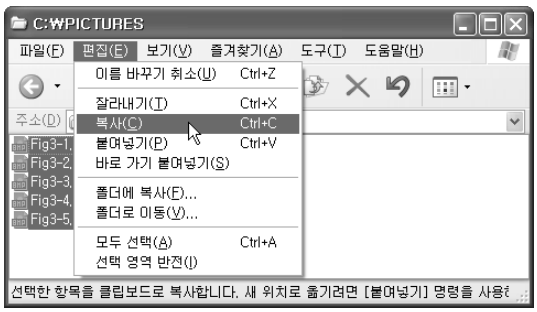
그래픽 사용자 인터페이스

그래픽 사용자 인터페이스(GUI, Graphical User Interface)는 도스(DOS, Disk Operating System) 같은 텍스트 기반 운영체제와 구분되는 외형적인 특징이다. 예를 들어, 파일을 복사하는 작업이 그림과 같이 다르다.

```
C:\PICTURES>COPY *.BMP C:\TEMP
Fig3-1.bmp
Fig3-2.bmp
Fig3-3.bmp
Fig3-4.bmp
Fig3-5.bmp
        5 file(s) copied.

C:\PICTURES>_
```

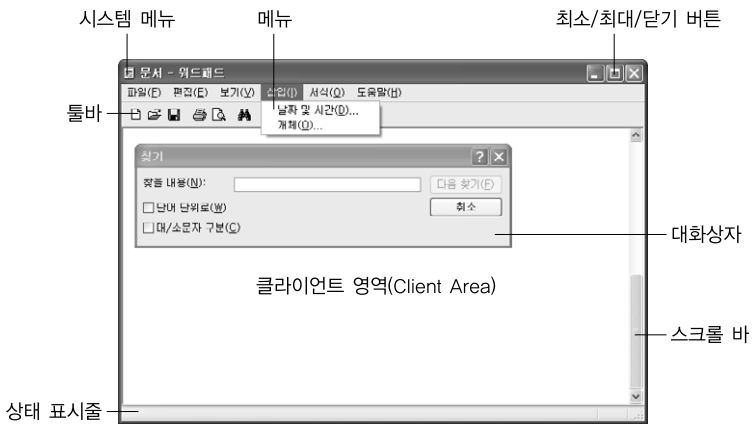
[그림 1-1] 텍스트 기반 운영체제-도스



[그림 1-2] 그래픽 기반 운영체제-윈도우

프로그래밍 관점에서 GUI 환경은 화면에 보이는 다양한 사용자 인터페이스 구성 요소(윈도우, 버튼, 스크롤 바, 대화상자 등)를 다루는 방법을 새로 익혀야 하는 불편함이 있다. 하지만 한 번 배워두면 다른 프로그램을 만들 때도 동일하게 적용할 수 있다는 장점도 있다. 윈도우 응용 프로그램은 외형적으로 [그림 1-3]과 같은 구성 요소로 이루어졌다.

※MFC에서는 사용자 인터페이스 구성 요소를 쉽게 다룰 수 있도록 C++ 라이브러리를 제공함으로써 프로그래머가 직접 다루어야 할 많은 부분을 자동으로 처리해 준다.



[그림 1-3] 사용자 인터페이스 구성 요소

메시지 구동 구조

윈도우 운영체제에서 실행되는 대부분의 응용 프로그램은 메시지 구동 구조(Message-Driven Architecture)로 동작한다. 여기서 메시지는 운영체제가 프로그램의 외부 또는 내부에 변화가 발생했음을 해당 프로그램에 알리기 위한 개념이다.

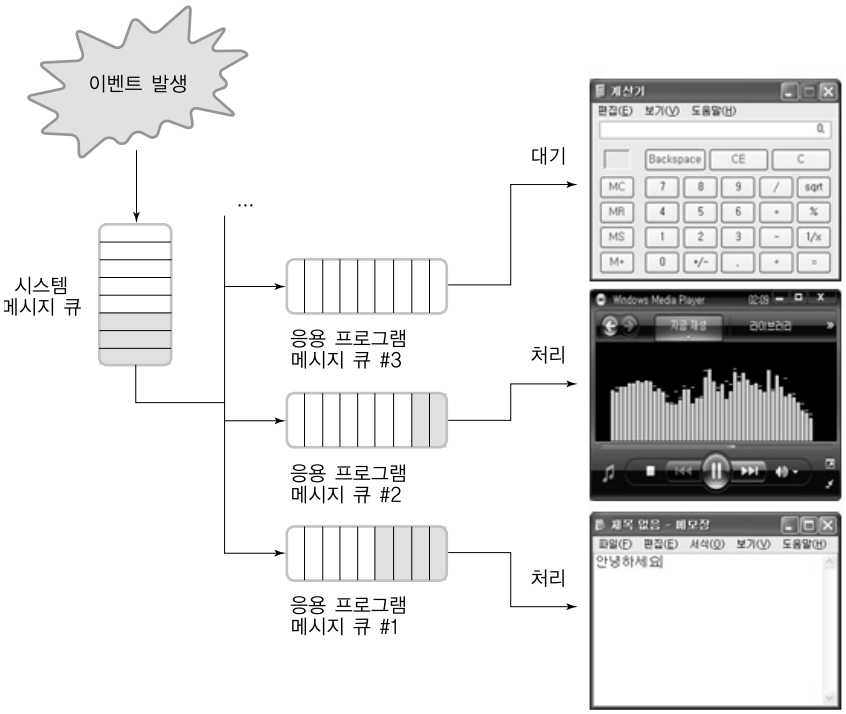
일반적으로 도스용 프로그램이나 윈도우의 콘솔 응용 프로그램은 프로그래머가 정한 순서대로 실행된다. 따라서 순차적인 방식에 의존해야 하므로 알고리즘과 논리의 흐름에 중점을 두고 프로그래밍한다. 하지만 대부분의 윈도우 응용 프로그램은 순차적으로 실행되지 않고 어떤 메시지를 받는가에 따라 코드의 실행 순서가 달라진다. 또한 메시지에 어떻게 반응하는가에 따라 동작이 달라진다.

[그림 1-4]는 윈도우의 메시지 구동 구조를 간략하게 나타낸 것이다. 외부에서 메시지를 발생시키는 이벤트(Event)가 발생하면 운영체제가 관리하는 시스템 메시지 큐(Message Queue)에 정보가 저장된다. 각각의 응용 프로그램은 운영체제로부터 독립적인 메시지 큐를 할당받으며, 운영체제는 시스템 메시지 큐에 저장된 메시지를 적절한 응용 프로그램 메시지 큐에 보낸다. 응용 프로그램은 자신의 메시지 큐를 감시하다가 메시지가 발생해서 큐에 들어오면 하나씩 꺼내서 처리하고 메시지가 없을 때는 대기한다.

여기서 잠깐

새로운 사용자 인터페이스?

직사각형에 타이틀 바가 있고 시스템 메뉴와 종료 버튼이 있는 모습이 윈도우 운영체제에서 볼 수 있는 일반적인 사용자 인터페이스다. 이는 운영체제에서 공통적인 부분을 제공받기 때문인데, 필요에 따라 요소 일부를 조정할 수 있다. 물론 식상한 윈도우 인터페이스를 벗어나고자 그래픽을 이용하는 화려한 사용자 인터페이스를 제공하는 경우도 있다. 하지만 새로운 기술은 아니기 때문에 윈도우 프로그래밍 기본 지식을 쌓으면 모두 구현할 수 있다. 윈도우 운영체제 안에 있는 한 기본 절차를 따를 수 밖에 없기 때문이다.



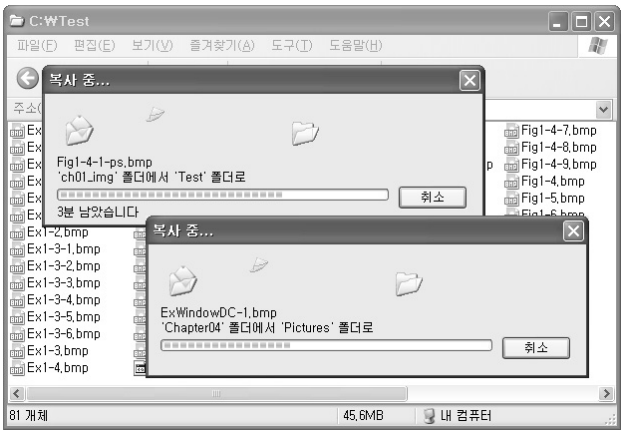
[그림 1-4] 메시지 구동 구조

멀티태스킹과 멀티스레딩

멀티태스킹(Multitasking)은 운영체제가 사용자에게 서로 다른 프로그램이 동시에 수행되는 것처럼 보이게 하는 기술이다. 일반적으로 CPU가 하나인 시스템에서 프로그램을 둘 이상 동시에 실행하는 것은 불가능하다. 하지만 운영체제가 CPU 시간을 적절히 분할해서 각 프로그램에 할당하면 멀티태스킹이 가능해진다. 멀티태스킹을 지원하는 운영체제에서는 프로그램 간 상호작용이 가능하므로 훨씬 다양한 일을 할 수 있다.

한편 윈도우 운영체제에서는 하나의 응용 프로그램 내에서도 동시에 진행되는 여러 개의 실행 흐름을 만들 수 있는데, 이러한 개념을 멀티스레딩(Multithreading)이라 한다. 예를 들면, 하나의 응용 프로그램이 마우스나 키보드로 입력된 내용을 처리하는 동시에 맞춤법 검사 루틴을 수행하고 네트워크를 이용해 통신할 수 있다. 윈도우 운영체제는 CPU 시간을 분할해서 각 스레드에 할당함으로써 멀티스레딩을 구현한다.

[그림 1-5]는 윈도우 XP에서 탐색기를 이용해 파일을 복사하는 모습이다. 파일을 복사하면서 다른 폴더로 이동하거나 또 다른 파일도 복사할 수 있다.



[그림 1-5] 멀티스레딩의 예

❷ 윈도우 응용 프로그램의 특징

소프트웨어 개발자 관점에서 윈도우 응용 프로그램의 특징을 정리해 보면 다음과 같다.

API 호출문 집합

윈도우 API(Application Programming Interface)는 유닉스의 시스템 콜(System Call)과 유사한 개념으로 윈도우 운영체제가 응용 프로그램을 위해 제공하는 각종 함수의 집합이다. 16비트 윈도우에서는 Win16 API, 32 또는 64비트 윈도우에서는 Win32 API라고 부른다(윈도우 95 이상은 모두 32비트 또는 64비트다). 윈도우 응용 프로그램 코드의 많은 부분이 API를 호출하여 구현한다.

여기서 잠깐

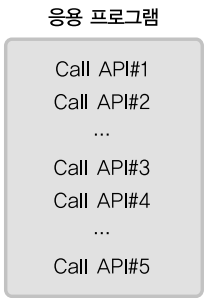
멀티태스킹과 멀티스레딩

운영체제 입장에서는 메모리 관리 때문에 멀티태스킹과 멀티스레딩이 상당히 다른 의미다. 하지만 사용자 입장에서는 둘 다 '동시에 두 가지를 실행한다'는 맥락으로 이해할 수 있다. 차이는 멀티태스킹은 '서로 다른 응용 프로그램 사이', 멀티스레딩은 '한 응용 프로그램 내부'에서 발생한다는 것이다. 따라서 멀티태스킹은 운영체제에서, 멀티스레딩은 해당 응용 프로그램에서 관리한다.

도스에서는 파일 복사 도중에 다른 일을 할 수 없었다. 반면 윈도우는 멀티태스킹을 지원하기 때문에 파일을 복사하면서 인터넷 익스플로러를 띄워 웹 검색을 할 수 있다. 인터넷 익스플로러는 멀티스레딩을 지원하는 응용 프로그램이므로 다시 웹 검색을 하면서 메뉴를 조작하거나 파일을 다운로드할 수 있다.

개발자는 운영체제가 관리해 주는 멀티태스킹보다 직접 처리할 멀티스레딩에 관심을 가져야 한다. 응용 프로그램을 개발하다 보면 동시에 여러 일을 수행해야 하는 경우가 있다. 급한 일이 생겼을 때 혼자 하던 일을 동료에게 부탁하는 것처럼 멀티스레딩을 사용하여 이를 적절히 처리할 수 있다.

윈도우 운영체제는 화면에 점을 찍거나 선을 그리는 간단한 동작부터 파일 입출력, 네트워킹 같은 복잡한 기능에 이르기까지 다양한 API 함수를 제공한다. 이 책에서 공부할 MFC도 내부적으로는 API를 호출하여 많은 기능을 구현하고 있다. MFC는 수많은 API 중 일부만 C++ 라이브러리 형태로 제공하기 때문에 MFC를 이용하여 프로그램을 제작하더라도 API를 직접 호출하는 경우가 생긴다는 사실을 염두에 두자.



[그림 1-6] API 호출문 집합

※이 책에서는 1장을 제외하고는 API를 거의 언급하지 않지만, API를 잘 배워두면 MFC나 기타 다른 개발 방식으로 프로그래밍할 때 많은 도움이 된다. API 학습을 원한다면 추천도서 2번과 3번을 읽어보기 바란다.

메시지 핸들러 집합

윈도우 운영체제의 특징에서 살펴본 것처럼 윈도우 응용 프로그램은 메시지 구동 구조로 동작한다. 메시지 구동 구조에서는 받는 메시지에 따라 실행 순서가 달라지며, 해당 메시지에 어떻게 반응하는가에 따라 동작이 달라진다. 메시지를 받았을 때 동작을 결정하는 코드를 편의상 메시지 핸들러(Message Handler)라 부르자(MFC에서 자주 사용하는 용어다). 프로그래머는 키보드 메시지 핸들러, 마우스 메시지 핸들러, 메뉴 메시지 핸들러 같은 다양한 메시지 핸들러를 작성하게 된다.

[그림 1-7] 메시지 핸들러 집합

메시지 핸들러의 집합을 윈도우 프로시저(Window Procedure)라 부른다. 윈도우 운영체제는 프로그램이 처리하지 않은 메시지를 자동으로 처리할 수 있도록 운영체제 차원의 메시지 핸들러를 제공한다.

실행 파일과 DLL 집합

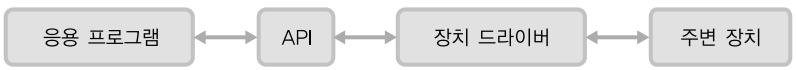
DLL(Dynamic-Link Library)은 프로그램이 실행 중에 결합하여 사용할 수 있는 코드와 리소스 집합이다. 일반적으로 디스크에서는 확장자가 .DLL인 파일로 존재한다. 자주 사용하는 기능을 하나의 DLL로 구현해 두면 여러 프로그램이 공유할 수 있으므로 메모리와 디스크 자원을 효율적으로 사용할 수 있다.

[그림 1-8] 실행 파일과 DLL 집합

윈도우 운영체제가 제공하는 API는 DLL 형태로 제공되며, 응용 프로그래머는 필요한 기능을 DLL로 직접 제작하기도 한다. 따라서 윈도우 응용 프로그램의 기능은 실행 파일 외에도 운영체제가 기본으로 제공하는 DLL과 사용자 정의 DLL에 분산되어 있다고 볼 수 있다. 윈도우의 실행 파일과 DLL은 코드는 물론이고 비트맵, 아이콘, 메뉴 같은 리소스(Resource)도 포함할 수 있다.

장치 독립성

윈도우 응용 프로그램은 모니터, 비디오 카드, 프린터, 네트워크 카드 같은 주변 장치가 바뀌어도 장치 드라이버(Device Driver)만 설치하면 프로그램을 수정하지 않고 그대로 실행할 수 있다. 응용 프로그램은 주변 장치를 직접 다루지 않고 API를 통해 장치 드라이버와 간접적으로 통신하므로 장치 독립성(Device-Independency)이 가능하다.



[그림 1-9] 장치 독립적으로 동작하는 응용 프로그램

3 윈도우 응용 프로그램의 개발 방식

앞서 살펴본 바와 같이 윈도우 응용 프로그램은 도스 같은 운영체제에서 실행되는 프로그램과 다른 독특한 특징이 있다. 이러한 윈도우 응용 프로그램을 개발하기 위해 현재 사용되는 방식을 분류하면 다음과 같다.

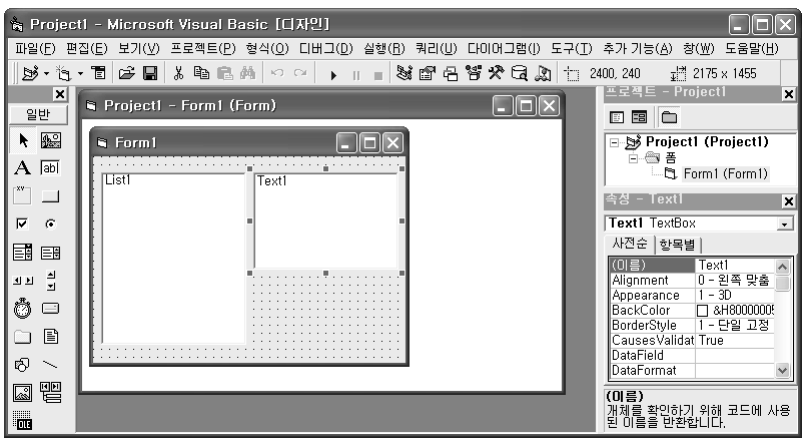
SDK

SDK(Software Development Kit ; 윈도우 2003까지는 Platform SDK, 윈도우 비스타부터는 Windows SDK라 부름)란 마이크로소프트 사에서 윈도우 응용 프로그램 제작을 위해 배포하는 컴파일러를 비롯한 각종 개발툴, 헤더 파일, 라이브러리 파일, 도움말, 소스 코드의 집합을 말한다. SDK는 마이크로소프트 사의 웹사이트에서 무료로 받을 수 있으며, 새로운 윈도우 버전이 발표될 때마다 업데이트된다.

SDK를 이용해 프로그래밍한다는 것은 C/C++ 언어로 윈도우 API를 직접 호출해서 프로그램을 구현함을 뜻한다. API를 직접 다루기 때문에 세부 제어가 가능하고, 윈도우 운영체제가 제공하는 모든 기능을 사용할 수 있다. 또한 생성 코드의 크기도 작고 속도도 빠르다. 하지만 다른 개발 방식에 비해 생산성이 매우 낮으므로 일반적인 데스크탑 응용 프로그램 개발용으로는 적합하지 않다.

RAD

RAD(Rapid Application Development)는 비주얼 베이직(Visual Basic)이나 델파이(Delphi)처럼 화면을 시각적으로 디자인하고 여기에 코드를 추가하는 방식으로 프로그램을 빠르게 개발할 수 있는 방식을 총칭하는 용어다. RAD 툴을 이용한 방식은 간편하게 직관적으로 프로그래밍할 수 있어 빠른 시간 내에 원하는 기능의 프로그램을 개발할 수 있다. 그러나 SDK나 클래스 라이브러리를 이용한 개발 방식보다 생성 코드 크기가 크고 실행 속도도 떨어지는 편이다. 또한 운영체제가 제공하는 모든 기능을 활용해 세부적으로 제어하기 어려운 경우도 있다.



[그림 1-10] RAD 개발툴의 예 – 비주얼 베이직

클래스 라이브러리

클래스 라이브러리(Class Library)는 윈도우 응용 프로그램 개발에 필수적인 기능을 C++와 같은 객체 지향 언어를 이용하여 클래스화한 것이다. 대표 예가 이 책에서 다룰 마이크로소프트 사의 MFC(Microsoft Foundation Class Library)며, 현재는 사용되지 않는 볼랜드 사의 OWL(Object Windows Library)도 여기에 속한다.

클래스 라이브러리를 이용해 프로그램을 개발하려면 객체 지향 프로그래밍에 익숙해야 한다. 또한 클래스 라이브러리의 구조와 각 클래스의 기능 및 관계를 잘 알아야 하므로 초기 학습에 필요한 기간이 긴 편이다. 그러나 일단 사용법을 익혀 활용할 수 있게 되면 SDK를 이용한 방식보다 빠른 속도로 프로그래밍할 수 있고, API를 직접 사용해서 세부적으로 제어할 수도 있다. 또한 RAD 개발 방식보다 생성 코드의 크기나 실행 속도 면에서도 유리하다.

.NET 프레임워크

.NET(닷넷) 프레임워크는 윈도우 운영체제에 설치할 수 있는 소프트웨어 개발 및 실행 환경으로, 윈도우 API와는 별도로 발전하는 차세대 소프트웨어 개발 플랫폼이다. .NET은 공용 언어 런타임(CLR, Common Language Runtime)이라는 일종의 소프트웨어 가상 머신을 제공하며, 가상 머신 제어 하에서 응용 프로그램이 구동된다(장치 독립성). 또한 윈도우 API에 버금가는 방대한 라이브러리를 제공하는데, 언어에 상관없이 이 라이브러리를 사용할 수 있다는 특징이 있다(언어 독립성). .NET은 단독으로 실행되는 전통적인 윈도우 응용 프로그램 작성은 물론이고 효율적인 웹 응용 프로그램/서비스 제작에도 부응하며, 빠른 속도로 새 기술이 추가되면서 발전하고 있다. API나 MFC와는 전혀 다른 범주로 간주할 수 있는 분야라 이 책에서는 더 이상 언급하지 않지만, 마이크로소프트 사에서 권장하는 차세대 소프트웨어 개발 및 실행 플랫폼이므로 관심을 갖고 필요하다면 학습하길 바란다.

여기서 잠깐

SDK(API)와 MFC의 관계

SDK는 C 언어로 윈도우 API를 사용할 수 있게 제공되는 라이브러리고, MFC는 이를 C++ 클래스 형태로 캡슐화하여 만든 클래스 라이브러리다. 마이크로소프트 사에서 제공하는 API는 수없이 많으며, 범주 하나만 가지고도 책 한 권을 쓸 수 있을 만큼 내용이 방대하다. MFC는 결국 API로 개발할 때 '자주 반복되는 기본 구조'를 C++ 클래스의 재사용성을 바탕으로 정리해 놓은 라이브러리인 것이다.

MFC는 새로운 비주얼 C++ 제품군이 발표될 때마다 기존 클래스를 업그레이드하거나 새로운 클래스를 추가함으로써 최신 기술을 지속적으로 지원하고 있다. 하지만 각 분야에서 저마다 개성을 내는 API를 하나로 묶는 데는 한계가 있으므로, 모든 API를 C++ 라이브러리로 만들어 놓지는 못한다. 그렇다고 일부 API 옹호론자들이 주장하는 것처럼 MFC를 배우는 것 자체가 낭비라고 하는 것은 합당하지 않다. 의미 없는 부분을 매번 반복적으로 작성하면서까지 MFC를 배제할 필요는 없기 때문이다.

따라서 현 프로그래밍의 주류는 기본 틀은 MFC를 사용하여 간단히 처리하고 '중요하게 처리할 부분'은 관련 API를 연구하고 활용하는 방식이다. 특정 분야의 API를 사용할 일이 많다면 자신이 C++ 클래스를 작성하기도 하고, 웹사이트에서 다른 개발자가 작성한 클래스나 라이브러리를 가져다 사용할 수 있다. 이 시점이 되면 자신이 능동적으로 라이브러리를 생성하고 다른 개발자에게 도움이 되는 능력을 갖추게 된다.

Section

2 SDK 프로그램 기본 구조

이제부터 비주얼 C++를 이용해 간단한 형태의 SDK 프로그램을 작성하고, 이를 분석하면서 윈도우 응용 프로그램의 기본 구조를 이해해 보자. SDK 프로그램의 동작 원리를 이해하면 MFC를 학습할 때 많이 도움된다. 3절에서 살펴볼 MFC 프로그램도 형태는 다르지만 기본 동작 원리는 SDK 프로그램과 크게 다르지 않기 때문이다. MFC를 이용한 프로그램에서도 SDK 프로그램처럼 종종 API 함수를 직접 호출하여 사용하기 때문에 API에 대한 기본 지식은 필수라 할 수 있다.

SDK 프로그램의 기본 골격은 다음과 같다.

① 윈도우 클래스를 정의(초기화)하고 운영체제에 등록한다.

② 윈도우를 생성하고 화면에 보이게 한다.

③ 메시지 루프를 구동한다.

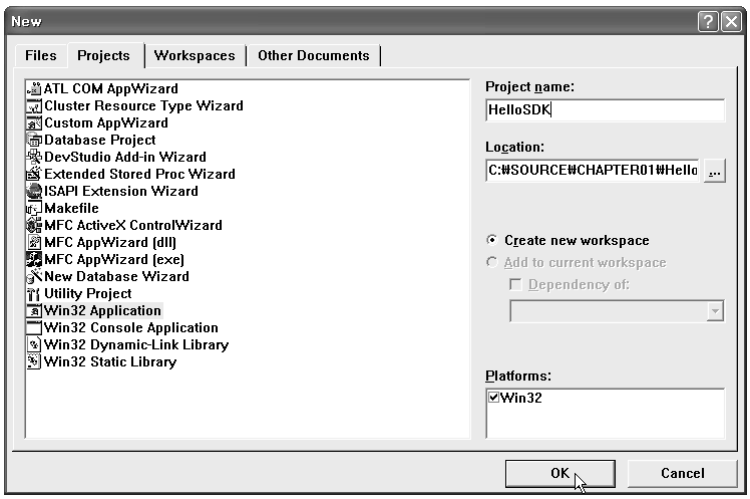
④ 윈도우 프로시저에서 메시지를 처리한다.

1 간단한 SDK 프로그램 작성

[실습 1-1]을 통해 간단하지만 핵심 구조를 갖춘 Hello 프로그램을 작성해 보자.

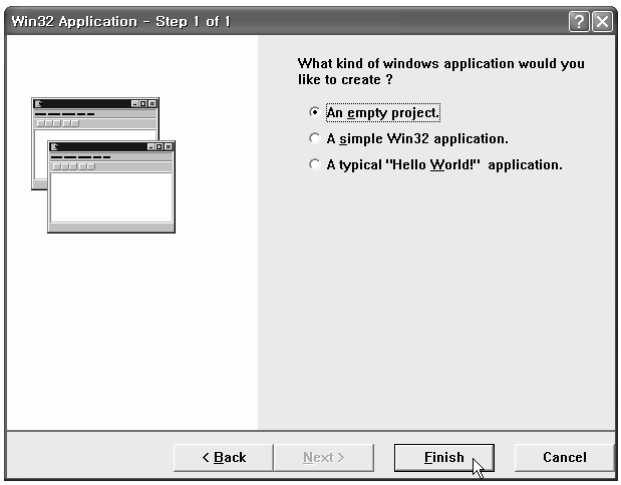
실습 1-1 간단한 SDK 프로그램 작성하기

1 비주얼 C++를 실행한 후 [File]-[New...] 메뉴를 선택하면 [New] 대화상자가 열린다. 프로젝트 종류 중 'Win32 Application'을 선택한다. 'Location'은 프로젝트가 생성될 위치를 나타내며, 오른쪽에 있는 '...' 버튼을 누르면 원하는 폴더를 선택할 수 있다. 프로젝트 이름으로 "HelloSDK"를 입력하면 'Location'에 'HelloSDK'가 자동으로 추가된다. 프로젝트 이름을 입력하였으면 〈OK〉 버튼을 누른다.



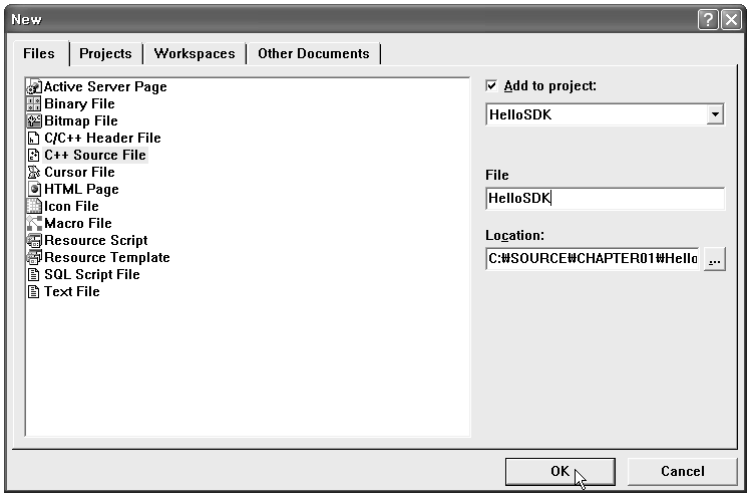
[그림 1-11] 프로젝트 종류 선택

2 'An empty project'가 기본으로 선택되어 있는데 그대로 두고 〈Finish〉 버튼을 누르면, 프로젝트에 대한 정보를 보여주는 대화상자가 열리는데 역시 〈OK〉 버튼을 누른다.



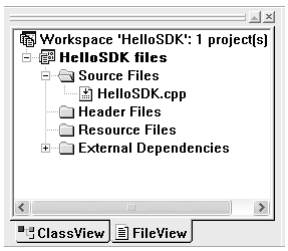
[그림 1-12] 1단계

3 다시 [File]-[New...] 메뉴를 선택하면 [New] 대화상자가 열리는데 이번에는 자동으로 [Files] 탭으로 이동해 있다. 'C++ Source File'을 선택한 후 오른쪽 중간의 'File' 부분에 "HelloSDK"를 입력하고(.cpp 확장자는 자동으로 붙음) 〈OK〉 버튼을 누른다. 이때 'Add to project'가 체크되어 있음에 유의한다.



[그림 1-13] 소스 파일 추가

4 다음과 같이 워크스페이스 창 아래에 있는 [FileView]을 클릭하면 'HelloSDK.cpp' 파일이 프로젝트에 추가된 것을 볼 수 있다. 'HelloSDK.cpp' 파일을 선택하여 예제 파일을 입력한다. ※워크스페이스 창이 보이지 않으면 [View]-[Workspace] 메뉴를 선택한다.



[그림 1-14] 워크스페이스

HelloSDK.cpp

```
01  #include <windows.h>                                    ❶ 헤더 파일
02
03  //WinMain() 함수에서 참조하므로 함수 원형을 선언한다.
04  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
05
06  int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
07                     LPSTR lpCmdLine, int nCmdShow)       ❷ 메인 함수
08  {
09      WNDCLASS wndclass;
10      HWND hwnd;
11      MSG msg;
12
```

```
13      // 윈도우 클래스를 초기화하고 운영체제에 등록한다.                    ❸ 윈도우 클래스 초기화
14      wndclass.style = CS_HREDRAW | CS_VREDRAW; // 스타일 지정
15      wndclass.lpfnWndProc = WndProc;          // 윈도우 프로시저 이름
16      wndclass.cbClsExtra = 0;                 // 여분 메모리(0바이트)
17      wndclass.cbWndExtra = 0;                 // 여분 메모리(0바이트)
18      wndclass.hInstance = hInstance;          // 인스턴스 핸들
19      wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION); // 아이콘 모양
20      wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);   // 커서 모양
21      wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH); // 배경색(흰색)
22      wndclass.lpszMenuName = NULL;            // 메뉴(NULL->메뉴 없음)
23      wndclass.lpszClassName = "HelloClass";   // 윈도우 클래스 이름
24      if(!RegisterClass(&wndclass)) return 1;
25
26      // 윈도우를 생성하고 화면에 보이게 한다.                              ❹ 윈도우 생성
27      hwnd = CreateWindow("HelloClass", "HelloSDK", WS_OVERLAPPEDWINDOW,
28             CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
29             NULL, NULL, hInstance, NULL);
30      ShowWindow(hwnd, nCmdShow);
31
32      // 메시지 큐에서 메시지를 하나씩 꺼내서 처리한다.                     ❺ 메시지 루프
33      while(GetMessage(&msg, NULL, 0, 0) > 0){
34          TranslateMessage(&msg);
35          DispatchMessage(&msg);
36      }
37
38      return msg.wParam;
39  }
40
41  LRESULT CALLBACK WndProc(HWND hwnd, UINT message,                  ❻ 윈도우 프로시저
42                           WPARAM wParam, LPARAM lParam)
43  {
44      HDC hdc;
45      PAINTSTRUCT ps;
46      char *str = "Hello, SDK";
47
48      // 발생한 메시지의 종류에 따라 적절히 처리한다.
49      switch(message){
50      case WM_CREATE:
51          return 0;
52      case WM_LBUTTONDOWN:
53          MessageBox(hwnd, "마우스를 클릭했습니다.", "마우스 메시지", MB_OK);
54          return 0;
55      case WM_PAINT:
56          hdc = BeginPaint(hwnd, &ps);
57          TextOut(hdc, 100, 100, str, lstrlen(str));
```

```
58          EndPaint(hwnd, &ps);
59          return 0;
60      case WM_DESTROY:
61          PostQuitMessage(0);
62          return 0;
63      }
64
65      // 응용 프로그램이 처리하지 않은 메시지는 운영체제가 처리한다.
66      return DefWindowProc(hwnd, message, wParam, lParam);
67  }
```

5 입력이 끝난 후 [Build]-[Execute HelloSDK.exe] 메뉴(또는 ! 나 Ctrl+F5)를 선택하면 대화상자가 뜬다. 〈예(Y)〉 버튼을 누르면 컴파일과 링크를 거쳐 프로그램이 실행된다.

실습 과정이 성공적으로 끝나면 다음과 같은 프로그램이 실행된다. 이 프로그램은 윈도우를 최소화 혹은 최대화할 수 있으며, 마우스로 테두리 부분을 잡고 드래그하면 크기가 변경된다. 화면에 표시된 Hello, SDK 문자열이 지워지지 않고 항상 클라이언트 영역의 일정한 위치에 표시되는 것을 눈여겨 보기 바란다. 또한 마우스로 클라이언트 영역을 클릭하면 간단한 메시지 상자가 뜬다.

[그림 1-15] 실행 화면

❷ HelloSDK 예제 분석

❶ 헤더 파일

windows.h는 여러 헤더 파일을 포함하는 헤더 파일이다. 윈도우 API 함수 원형, 데이터 타입, 구조체, 매크로 상수 등이 여기에 선언되어 있다. HINSTANCE, LPSTR, HWND 같은 생소한 데이터 타입의 정의를 알고 싶다면 windows.h에 포함된 헤더 파일을 살펴보면 된다.

❷ 메인 함수

WinMain() 함수는 C 프로그램의 main() 함수에 해당하며 프로그램을 실행하는 시작점이다. HINSTANCE형 변수인 hInstance, hPrevInstance는 인스턴스 핸들이라 부른다.

hInstance는 메모리에 로드된 실행 파일의 위치를 나타내는 주소값이다. 이 값은 실행 파일에 포함된 리소스(비트맵, 아이콘, …)에 접근할 때 종종 사용된다. hPrevInstance는 16비트 윈도우의 잔재로 항상 NULL 값을 가진다. lpCmdLine은 명령행 인자를 담고 있는 문자열이다. 예를 들어 'HelloSDK TEST1.TXT TEST2.TXT'와 같이 프로그램을 실행하면 lpCmdLine은 'TEST1.TXT TEST2.TXT' 값을 가진다. nCmdShow는 프로그램이 시작할 때 윈도우 모양을 결정한다. 이 값에 따라 윈도우는 최소화, 최대화 혹은 보통 상태로 시작한다.

※MFC에서는 프로그램에서 hInstance 값에 직접 접근할 수 있도록 AfxGetInstanceHandle( ) 함수를 제공한다.

❸ 윈도우 클래스 초기화와 등록

윈도우 클래스는 윈도우를 생성하는 데 필요한 다양한 정보를 담고 있는 구조체다. 실제 윈도우는 27행의 CreateWindow( ) 함수를 사용해 생성한다. CreateWindow( ) 함수의 첫 번째 인자로 윈도우 클래스 이름이 전달되므로, 윈도우 생성 전에 윈도우 클래스를 운영체제에 등록하는 과정이 필요하다.

윈도우 클래스는 앞으로 생성될 윈도우의 특성을 나타내는 다양한 요소로 이루어져 있는데, 이 중 가장 중요한 것은 lpfnWndProc이다. lpfnWndProc는 일종의 함수 포인터로, 윈도우가 운영체제로부터 받은 메시지를 처리할 사용자 정의 함수(윈도우 프로시저라 부름)의 주소를 담고 있다. 윈도우 클래스가 등록되면 응용 프로그램은 특성이 동일한 윈도우를 여러 개 생성할 수 있는데, 이렇게 생성된 윈도우에서 발생한 메시지는 하나의 윈도우 프로시저에서 처리한다.

❹ 윈도우 생성

CreateWindow( ) 함수는 24행에서 등록한 윈도우 클래스를 기반으로 실제 윈도우를 생성한다. 여기서 만드는 윈도우는 응용 프로그램의 최상위 윈도우로, 메인(main) 윈도우라 부른다. 첫 번째 인자인 윈도우 클래스 이름은 운영체제가 윈도우를 생성하는 데 필요한 정보를 제공하는데, 특히 윈도우 프로시저의 주소를 운영체제에 알려주는 중요한 역할을 한다.

CreateWindow( ) 함수의 리턴값은 윈도우 핸들이라 부르는데 운영체제 내부에서 윈도우를 유지하는 데 필요한 데이터 구조체를 가리키는 일종의 포인터다. 응용 프로그램을 작성하는 관점에서 핸들값의 숫자 자체는 그다지 중요하지 않다. 중요한 것은 이 핸들을 가지고 있으면 윈도우 API를 호출함으로써 해당 윈도우를 다양한 방식으로 제어할 수 있다는 점이다. CreateWindow( ) 함수의 전달 인자는 '**클래스 이름, 윈도우 이름, 윈도우 스타일**, x좌표, y좌표, 윈도우의 폭, 윈도우의 높이, 부모 윈도우 핸들, 메뉴 핸들, **인스턴스 핸들**, 옵션 데이터' 순이다. 진하게 표시한 부분을 제외하고는 모두 기본값을 사용하고 있다.

ShowWindow( ) 함수는 CreateWindow( ) 함수를 이용해 이전에 생성한 윈도우가 화면에 보이게 하는데, nCmdShow 값에 따라 윈도우는 최소화, 최대화 혹은 보통 상태로 시작한다.

이 함수의 첫 번째 인자로 윈도우 핸들을 전달하고 있음에 주목하자. SDK 프로그래밍에서는 윈도우, 메뉴, 비트맵 같은 사용자 인터페이스 구성 요소를 다룰 때 핸들을 자주 사용한다.

※MFC는 내부에 핸들을 숨기고 있기 때문에 핸들을 직접 다룰 일은 많지 않다. 그러나 특별한 기능 구현을 위해 윈도우 API를 직접 호출하는 경우에는 핸들이 필요한데, 다행히 핸들값이 MFC 클래스의 public 멤버로 공개되어 있는 경우가 많다.

❺ 메시지 루프

메시지 큐에서 메시지를 하나씩 꺼내서 처리하는 반복문을 메시지 루프(Message Loop)라 부른다. GetMessage( ) 함수는 메시지 큐에서 메시지 하나를 꺼내 msg 변수에 저장한다. 이때 꺼낸 메시지가 WM_QUIT면 0을 리턴하면서 while 문을 빠져 나간다. TranslateMessage( ) 함수는 msg 변수에 키보드 메시지가 들어 있을 경우 문자를 만들어내는 역할을 한다. DispatchMessage( ) 함수는 메시지를 윈도우 프로시저에 보내는 역할을 한다. 여기서 윈도우 프로시저는 15행에서 등록한 사용자 정의 함수를 말한다.

❻ 윈도우 프로시저

윈도우 프로시저는 윈도우 메시지를 처리하는 핵심 함수다. 하나의 윈도우 클래스를 기반으로 윈도우를 여러 개 생성한 경우 모든 윈도우가 윈도우 프로시저를 공유하므로 어느 윈도우에서 메시지가 발생했는지 구분할 필요가 있다. 첫 번째 인자인 hwnd가 이 역할을 한다. 두 번째 인자인 message는 발생한 메시지 종류를 나타내며 WM_으로 시작하는 상수값이다. wParam과 lParam은 메시지 종류에 따라 부가 정보를 전달하는 역할을 하는 32비트 값이다. 예제에 포함된 윈도우 메시지를 정리하면 다음과 같다.

[표 1-1] 예제에 포함된 윈도우 메시지

메시지 이름	메시지 발생 시점
WM_CREATE	CreateWindow( ) 함수를 호출할 때
WM_LBUTTONDOWN	클라이언트 영역에서 마우스 왼쪽 버튼을 클릭할 때
WM_PAINT	클라이언트 영역의 일부 또는 전체를 다시 그릴 필요가 있을 때
WM_DESTROY	종료 버튼을 눌렀을 때

WM_CREATE 메시지는 윈도우가 생성될 때(27행) 한 번만 발생하므로 각종 초기화 작업을 수행한다. 반면 WM_DESTROY 메시지는 윈도우가 파괴될 때 한 번만 발생하므로 각종 청소 작업을 수행한다. WM_DESTROY 메시지 처리의 마지막 단계로는 PostQuitMessage( ) 함수를 호출한다(61행). 이 함수는 응용 프로그램 메시지 큐에 WM_QUIT 메시지를 집어넣는 역할을 한다. GetMessage( ) 함수(33행)는 WM_QUIT 메시지 수신 시 0을 리턴하므로 메시지 루프를 빠져나가고 결국 응용 프로그램은 종료한다.

WM_LBUTTONDOWN 메시지는 클라이언트 영역에서 마우스 왼쪽 버튼을 클릭할 때 발생한다. wParam과 lParam에는 마우스/키보드 버튼의 상태와 마우스 커서 위치가 들어 있으므로 필요하다면 꺼내서 활용할 수 있다.

현장 팁

단축 아이콘과 WinMain( ) 함수

도스 시절에는 명령행 인자를 자주 사용했지만 윈도우에서는 마우스를 사용하면서 명령행 인자를 사용하지 않는다. 그래서인지 많은 사람들이 윈도우에서 명령행 인자가 없어졌다고 믿어버린다. 그런데 프로그래밍을 배우면서 갑자기 WinMain( ) 함수에서 명령행 인자를 처리할 수 있다고 하니 당황할 수 밖에 없다. 윈도우에도 그런 인자를 사용할 수 있다는 것인가? 다음은 레지스트리 편집기 화면이다. 데이터 항목에 notepad.exe와 함께 %1 명령행 인자가 보이는가? 윈도우에서는 아직도 명령행 인자를 사용한다.

›› 레지스트리 편집기

좀 더 구체적으로 살펴보기 위해 인터넷 익스플로러 단축 아이콘의 등록 정보를 살펴보자. 그림처럼 '대상(T)'에 URL을 명령행 인자로 추가한 후 단축 아이콘을 실행하면, 인터넷 익스플로러가 실행되면서 해당 URL로 이동한다. 이를 통해 명령행 인자를 어떻게 활용할 수 있는지 확인할 수 있다.

›› 단축 아이콘 등록 정보

그 뿐만이 아니다. 위의 그림에서 '실행(R)' 항목 내용이 WinMain( ) 함수에 전달되는 nCmdShow다. 현재는 '기본 창'으로 지정되어 있지만 이 값을 '최대화' 또는 '최소화'로 변경하면 인터넷 익스플로러는 설정한 대로 실행하게 된다.

이러한 값은 응용 프로그램의 WinMain( ) 함수에 인자로 항상 전달된다. 응용 프로그램의 성격에 따라 WinMain( ) 함수로 전달된 인자를 아예 무시하거나 색다르게 처리할 수도 있다. 중요한 것은 응용 프로그램에 명령행 인자를 넘겨서 처리해야 하는 상황이 반드시 있고 그때 이 내용을 떠올려서 활용하면 된다.

WM_PAINT 메시지는 특별한 의미가 있으므로 잘 알아두자. 윈도우 응용 프로그램이 화면에 무언가를 표시하고자 할 때는 주로 윈도우의 클라이언트 영역에 출력한다([그림 1-3]). 출력된 내용은 운영체제가 자동으로 저장/복원하지 않으므로 다른 윈도우가 가렸다가 다시 드러나면 지워진다. 윈도우 운영체제는 클라이언트 영역의 일부 또는 전체를 다시 그릴 필요가 있음을 응용 프로그램에 알리기 위해 WM_PAINT 메시지를 발생시킨다. 응용 프로그램은 WM_PAINT 메시지에 응답하여 자신의 클라이언트 영역 화면을 다시 그려야(=복원해야) 한다.

WM_PAINT 메시지를 처리할 때는 BeginPaint( )와 EndPaint( ) 함수를 시작 부분과 끝 부분에서 호출해야 한다. BeginPaint( )/EndPaint( ) 함수는 응용 프로그램이 WM_PAINT 메시지를 받아서 윈도우의 클라이언트 영역을 다시 그렸음을 운영체제에 알려주는 역할을 한다. 56~58행을 주석으로 처리한 후 프로그램을 다시 실행해 보자. 윈도우의 크기를 약간 변경한 후(이때 WM_PAINT 메시지가 발생함) 작업 관리자로 살펴보면 HelloSDK 프로그램의 CPU 사용률이 매우 높음을 확인할 수 있다. BeginPaint( )/EndPaint( ) 함수 호출을 생략했으므로 운영체제가 끊임없이 WM_PAINT 메시지를 발생시키기 때문이다.

BeginPaint( ) 함수의 리턴값을 디바이스 컨텍스트(Device Context) 핸들이라 부르는데, 화면 출력 함수는 이 값이 필요하다. 클라이언트 영역의 지정한 위치에 문자열을 출력하는 TextOut( ) 함수의 첫 번째 인자로 hdc가 사용됨에 주목하기 바란다. WM_PAINT 메시지를 받았을 때 화면을 제대로 그려주지 않으면 윈도우의 최소화, 최대화 및 크기 변경, 다른 윈도우가 클라이언트 영역을 가렸다가 다시 보이게 되는 경우 등이 발생할 때 화면 출력이 지워지는 현상이 발생한다.

프로그래머는 switch-case 문을 이용하여 메시지의 종류에 따른 적절한 처리 코드를 추가하되, 처리하지 않은 메시지는 DefWindowProc( ) 함수를 호출하여 운영체제가 자동으로 처리하게 한다. 좀더 복잡한 SDK 프로그램은 훨씬 더 많은 종류의 메시지를 처리하므로 switch-case 문이 더욱 길어진다.

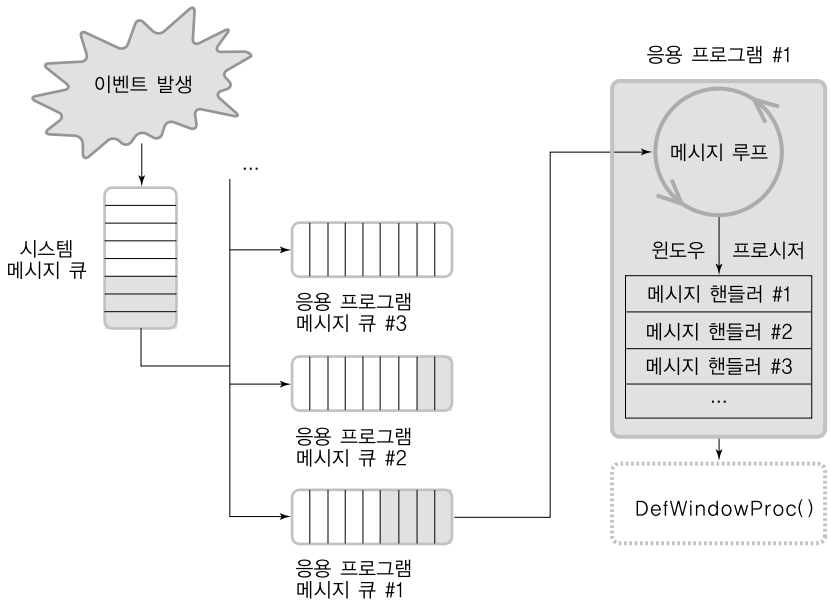
요약

지금까지 살펴본 SDK 프로그램의 구조를 간략하게 나타내면 [그림 1-16]과 같다. 운영체제는 각 응용 프로그램에 메시지 큐를 할당하며, 프로그램의 외부 또는 내부적인 요인에 의해 발생하는 메시지가 시스템 메시지 큐를 거쳐 응용 프로그램 메시지 큐에 저장된다. WinMain( ) 함수에 있는 메시지 루프는 응용 프로그램 메시지 큐에 저장된 메시지를 하나씩 꺼내서 윈도우 프로시저에 전달한다.

윈도우 프로시저는 받은 메시지의 종류에 따라 적절한 처리를 하되 자신이 처리하지 않은 메시지는 DefWindowProc( ) 함수에 넘겨서 운영체제가 자동으로 처리하게 한다.

윈도우 응용 프로그램의 핵심은 윈도우 프로시저라 할 수 있다. 윈도우 프로시저는 메시지 처리 코드 집합이므로, 결국 메시지 처리 코드(MFC에서는 메시지 핸들러라 부름)를 어떻게 작성하느냐가 응용 프로그램의 동작을 결정한다. 잠시 후 3절에서 살펴볼 MFC 프로그램도 결국 이와 동일한 구조로 동작한다.

※MFC로 프로그래밍할 때도 메시지의 의미와 해당 메시지가 발생하는 시점을 이해하는 것은 매우 중요하다. 비주얼 C++가 메시지 핸들러를 좀 더 편리하게 작성할 수 있는 구조와 툴을 제공하지만 처리할 메시지를 선택하는 것은 프로그래머의 몫이다.



[그림 1-16] SDK 응용 프로그램 동작 원리

여기서 잠깐

헝가리 표기법

많은 프로그래머들이 SDK나 MFC 프로그래밍을 할 때 헝가리 표기법(Hungarian Notation)을 사용해 변수를 표기한다. 헝가리 표기법은 변수 타입을 짐작할 수 있도록 접두사(Prefix)를 덧붙여 변수 이름을 정하는 방법이다. 예를 들면 hInstance라는 이름의 변수는 첫 글자인 h만으로 핸들 타입임을 알 수 있다. 헝가리 표기법을 적용해서 변수 이름을 정하면 타입 불일치로 인한 오류 발생 가능성을 줄일 수 있고 다른 사람들도 코드를 좀 더 쉽게 이해할 수 있다는 장점이 있다. 다음 표는 자주 사용하는 접두사다. 표에는 없더라도 이를 응용하여 다른 종류의 데이터 타입에도 비슷한 방식으로 적용할 수 있다.

접두사	데이터 타입	접두사	데이터 타입
c	char	sz	0으로 끝나는 문자열
n 또는 i	int	h	handle
b 또는 f	BOOL (b=bool, f=flag)	p	pointer
w	WORD	lp	long pointer
l	LONG	fn	function
dw	DWORD		