Socket Programming with MFC in Win32 Environment

2002년 5월 9일

윤성진(chaos@ChaosClub.net)

http://ChaosClub.net

서론

네트웍 프로그래밍이라는 것은 단순히 데이터를 처리하는 수준의 application과는 좀 다른 면이 있다. 어쩌면 어느 정도 시스템 프로그래밍이라고 할 수 있다. 이 말은 그 네트웍 프로그램이 실행될 환경, 즉 운영 체제를 이해해야 할 뿐만 아니라, TCP/IP에 대한 기본적인 지식이 필요하다는 뜻이다.
필자 역시 여러 언어로 네트웍 프로그래밍을 해봤는데, 역시 네트웍 프로그래밍은 쉽지만은 않다. 기본적으로 알아야 할 것도 많고 실제 프로그래밍을 할 때에 생각해야 할 것도 많다. 네트웍 프로그램은 한대의 컴퓨터에서만 실행되는 것이 아니라, 최소한 2대 이상의 컴퓨터에서 상호 작용을 통해 작동하고, 컴퓨터 외적인 요소에도 영향을 많이 받는다. 예를 들면 네트웍 회선이 불량하다든가, 잘못된 방화벽 설정, 라우터 설정 등으로 인한 문제, RFC 문서[1]를 준수하지 않는 상대측 네트웍 프로그램과의 호환성 문제 등 실제로 프로그래밍을 하다보면 외적인 요소에 많은 영향을 받게된다.
따라서 필자 역시 그러한 부분에서 많은 어려움을 겪었고 네트웍 프로그래밍에 입문하는 다른 분들에게 도움이 되고자 이 글을 남기기로 결심했다.
만약 이 글을 읽는 여러분이 인터넷 사용에 익숙하고 유닉스/리눅스 등을 어느 정도 다룬다면 이 글을 보는 데에 많은 도움이 될 것이다. 물론 이 글은 MS-Windows 환경에서의 네트웍 프로그래밍을 다루지만 리눅스 등을 많이 사용해 보았다면 IP 주소라든가 DNS, 포트 번호 등과 같은 단어에 익숙할 것이다.
이 글을 읽는 독자의 수준은 C/C++ 중급 이상의 수준, 그리고 기본적인 네트웍 상식과 MFC를 이용한 간단한 프로그램 작성 수준이면 된다. 너무 높은 수준을 요구한다고 생각되면 미리 공부를 좀 더 해볼 필요가 있겠다.

필자는 현재 대전 대덕연구단지에 위치한 모 벤처기업의 구석 자리에 앉아서 이 글을 쓰고 있다. 병역특례 자리를 마련해준 서울의 박사장님, 그리고 여기 대전 벤처기업의 변사장님께 이렇게 여유있는 시간을 주심에 대해 감사드리는 바이다.

2002년 5월 8일

윤성진 (chaos@chaosclub.net)

[1] RFC 문서란 Request For Comments란 뜻인데, 인터넷과 관련된 여러 규약을 정리한 문서라고 생각하면 된다. 쉽게 말해 인터넷에서 사용되는 많은 프로토콜들의 표준을 정의한 문서이다. 이 문서는 http://www.ietf.org에 가면 검색이 가능하다. 대중적인 프로토콜을 이용한 프로그램을 개발할 때에는 꼭 RFC 문서를 참조하여 표준을 지켜야 한다.

1. 네트워크 기초 및 TCP/IP

여기서는 간단한 TCP/IP 이론에 대해 소개한다. 꼭 필요한 수준만 논하고 있으니 더 깊은 이해를 원하는 사람은 기타 서적을 참고하기 바란다.

1.1 서버(server)와 클라이언트(client)

"클라이언트" 와 "서버" 라는 말이 자주 나오게 될텐데, 초보자를 위해 설명하자면 다음과 같다. 서버나 클라이언트는 모두 네트워크에 연결된 컴퓨터를 가리킨다. 다만 네트워크에 묶여져 있을 때 그 컴퓨터가 하는 "역할" 에 따라 서버와 클라이언트를 구분한다. 서버(server)는 하인이라는 뜻의 영어 단어인데, 어떤 작업 요청을 받아들여서 그것을 처리한 뒤에, 결과를 되돌려주는 일을 하는 컴퓨터를 말한다. 클라이언트는 이러한 서버에게 작업을 의뢰하고 결과를 받아가는 역할을 하는 컴퓨터를 말한다. 어떤 책에서는 "세션을 초기화하는 쪽"을 클라이언트라고 정의한다. 이것도 좀 난해하긴 하지만 그럴듯한 정의이다.

일반적으로 서버용 컴퓨터는 클라이언트보다 성능이 상당히 뛰어나지만 꼭 그런 것만은 아니다. 그 역할에 따라 서버와 클라이언트를 구분한다는 것을 명심하자. 예를 들어 보면 우리가 야후 사이트에 접속해서 웹페이지를 받아오는 것도 서버와 클라이언트의 관계가 성립되는데, 우리의 컴퓨터가 야후에 접속해서 웹페이지를 "요청" 하므로 우리의 컴퓨터는 클라이언트가 되고 야후의 컴퓨터가 서버가 되는 것이다. 이런 경우에 야후 서버는 웹 서비스를 제공하므로 특별히 "웹서버" 라는 말도 사용한다.

요즘 특이한 경우로 PtoP라는 것이 있다. 이것은 개인용 컴퓨터끼리 직접 연결해서 서로 자료를 주고 받는다든가 하는 것을 말하는데, 구루구루나 소리바다 같은 것이 PtoP(Peer to Peer)이다. 이런 방식의 경우에는 각 컴퓨터가 서버의 역할과 클라이언트의 역할을 동시에 한다.

꼭 이런 것이 아니더라도 우리의 컴퓨터가 서버의 역할과 클라이언트의 역할을 동시에 할 수 있다. 예를 들면 우리가 윈도우 2000 컴퓨터에서 FTP를 열어놓고 친구에게 자료를 받아가게 해놓으면서 동시에 그 컴퓨터를 이용해서 웹서핑을 할수도 있는데, FTP 서비스면에서 보면 서버의 역할을, 웹서핑 관점에서 보면 클라이언트의 역할을 하는 것이다.

참고로 서버와 클라이언트 외에 "호스트" 라는 말도 네트워크 서적에서 많이 나오는데, 이 말은 네트워크의 말단에 연결된 컴퓨터를 말하는 것이다. 따라서 일반적으로 서버나 클라이언트들은 모두 호스트이다. 그러나 네트워크 중심(core)에 위치한 스위치라든가 라우터와 같은 장비들은 호스트가 아니다.

1.2 프로토콜 스택

일단 프로토콜이라는 말에 대해 정의해 보면 다음과 같다. 프로토콜이란 "네트워크 소프트웨어의 핵심으로 둘 이상의 통신 개체 사이에 교환되는 메시지의 형태, 의미, 전송순서, 그리고 메시지 송수신 및 기타 사건에 수행할 동작을 정의한 규약" 이다. 흔히 전송 규약이라고 많이 해석을 하는데, 서버와 클라이언트가 서로 메시지를 어떻게 주고 받으며, 또 특정한 메시지를 받았을때에는 어떤

동작을 취해야 하는지를 모두 정의해 놓은 것을 전송 규약이라고 한다. 프로토콜에는 아래에서 설명할 TCP, UDP, IP와 같이 인터넷 프로토콜 스택을 구성하는 것도 있지만 응용 계층에서 설계되고 동작하는 응용 계층 프로토콜들도 있다. 응용 계층 프로토콜에 해당하는 것이 HTTP, FTP, SMTP, POP3, IMAP 등이다. 이러한 응용 계층 프로토콜은 여러분도 얼마든지 독자적으로 설계할 수 있다.

인터넷에서 사용되는 프로토콜은 TCP/IP라고 한다. 이것은 인터넷에서 사용되는 프로토콜인 TCP와 IP를 함께 부르는 말이다. 이 인터넷 프로토콜은 다섯 계층으로 분리되는데, 하부에서부터 물리(physical), 링크(link), 네트워크(network), 전송(transport), 응용(application) 계층으로 나뉜다. 어떤 자료에는 물리 계층과 링크 계층을 합쳐서 그냥 링크 계층이라고 하기도 한다. 이런 경우에는 4 계층이 된다. 이렇게 인터넷 프로토콜은 계층 구조를 이루는데, 이런 계층 구조를 프로토콜 스택이라고 한다. 인터넷 프로토콜 스택을 상위에서부터 살펴보면 다음과 같다.

▪ 응용 계층 (application layer)

응용 계층이란 우리가 흔히 사용하는 네트웍 프로그램들을 뜻한다. 여기에는 인터넷 익스플로러, 알FTP, 구루구루 등이 해당한다. 우리가 만드는 프로그램도 바로 이 응용 계층에 해당한다.

▪ 트랜스포트 계층 (transport layer)

트랜스포트 계층은 응용 계층에서 넘겨준 데이터를 다시 하위 계층인 네트워크 계층에 넘겨준다. 이 계층에는 2가지 프로토콜이 있는데, TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)이 있다. TCP는 연결지향형(connection-oriented) 서비스라고 하는데, 이것은 목적지로의 데이터 전달 보장과 흐름제어(flow control) 기능을 제공한다. 그리고 UDP는 비연결형(connectionless) 서비스로서 데이터가 목적지에 반드시 전달된다거나 순서대로 전달된다는 보장을 하지 않는다.

일반적으로 TCP를 많이 사용한다. 우리가 제작할 프로그램도 TCP를 사용한다. UDP는 매우 한정적으로 사용되는데 대표적으로 온라인 음악 방송이나 동화상 서비스 등이 UDP를 사용한다. UDP는 데이터의 오류도 체크하지 않고 패킷의 순서조차 보장하지 않음에도 장점이 한가지 있는데 속도가 TCP에 비해 빠르다는 것이다. 그래서 패킷[1] 몇 개가 없어져도 큰 상관이 없는 실시간 방송 등에는 UDP가 사용되는 것이다. 이런 특수한 경우를 제외하면 모두 TCP를 사용한다고 보면 된다.

우리가 TCP를 이용하여 네트웍 프로그램을 만들면 우리가 전송하는 데이터는 틀림없이 수신측에 도착하며 패킷의 순서도 보낸 순서와 동일함을 보장받을 수 있다. 이 알고리즘은 놀라울 정도로 정교하고 치밀하게 설계되어 있다.

---

[1] 패킷(packet)이란 네트워크를 통해 데이터를 전달할 때 일정한 크기의 묶음으로 전달을 하게 되는데 이것을 패킷이라고 한다. 패킷에는 헤더와 데이터 부분이 있어서 헤더에는 패킷의 송신지 정보, 수신지 정보 등이 담겨지게 된다.

▪ 네트워크 계층 (network layer)

네트워크 계층 프로토콜은 2가지 요소를 갖는다. 우선 IP 데이터그램의 필드를 정의하고 종단 시스템(end system)과 라우터가 이들 필드에 어떻게 동작하는지를 정의하는 프로토콜을 가지는데, 이것이 IP(Internet Protocol)이다. 또한 소스와 목적지 사이에 데이터그램이 가야하는 경로를 결정하는 라우팅 프로토콜을 수행한다. 이 계층을 흔히 IP 계층이라고도 한다.

▪ 링크 계층 (link layer)

링크 계층은 패킷을 실제로 다음 노드로 전송해주는 계층이다. 예를 들면 랜과 같은 이더넷(Ethernet)같은 것이 있다.

▪ 물리 계층

간혹 인터넷 프로토콜 스택을 4계층으로 구분할 때 이 물리 계층은 링크 계층에 포함되기도 한다. 이것은 데이터 통신의 가장 밑바닥으로서 동축케이블이라든가, 광케이블과 같은 물리적인 계층에서 비트 수준의 데이터 전송을 뜻한다.

링크 계층과 물리 계층은 다분히 하드웨어적인 분야로서 우리가 어떻게 다룰 수 있는 것은 아니다. 실제로 우리가 다룰 수 있는 것은 응용 계층 뿐이며, 특별히 드라이버 개발자라든가 운영체제 개발자에 한해 전송 계층이나 네트워크 계층은 접근할 수가 있을 것이다.

1.3 소켓(socket)

우리가 흔히 네트워크 프로그래밍을 한다는 것을 소켓 프로그래밍이라고 얘기를 한다. 소켓이란 응용 계층과 전송 계층 사이의 API(Application Programming Interface)[1]이다. 즉 우리가 응용 프로그램을 개발할 때 에러 검출이라든가 패킷의 순서를 맞춘다든가 하는 일을 우리가 하는 것이 아니라, 소켓을 사용하여 전송 계층으로 넘겨주기만 하면 그 다음부터는 알아서 흐름 제어라든가 에러 검출 및 재전송 등이 자동으로 이루어지게 된다. 물론 정확한 순서로 틀림없이 데이터가 전송된다는 것은 TCP를 사용할 때의 이야기이다. UDP를 사용하면 그런 서비스를 이용할 수 없다.

이 소켓이라는 것은 처음에 BSD(Berkeley Software Distribution) UNIX에서 처음 소개된 개념인데, 그래서 버클리 소켓이라고 부르기도 한다. 유닉스에서 소켓 프로그래밍을 할때의 소켓이 바로 버클리 소켓이다. MS-Windows에서는 Winsock이라는 이름으로 소켓을 지원한다. 버클리 소켓과 정확히 똑같지는 않지만 개념이라든가 함수 호출법 등에 있어서 거의 비슷하다.

1.4 기타 용어들

---

[1] API라는 것은 어떤 OS 등이 사용자에게 제공하는 함수 집합을 말한다. 또는 DBMS같은 것도 DBMS에 접근할 수 있는 함수 집합을 제공하는데 이런 것도 API라고 한다. 여기서 말하는 API는 소켓이 응용 프로그램에게 제공하는 함수 집합을 말한다.

▪ IP주소라는 것은 4바이트의 숫자로 구성된 주소를 말한다. 예를 들면 211.96.5.10과 같은 식의 주소가 IP 주소이다. 전 세계적으로 공인 IP 주소는 유일하다. 정확히 말하자면 IP 주소는 NIC(Network Interface Card)와 1:1로 대응된다. 이 말은 쉽게 말하면 랜카드가 한장 꽂혀 있는 컴퓨터는 IP 주소를 하나를 갖지만 랜카드가 두장 꽂혀 있는 컴퓨터라면 IP 주소를 2개 가질 수도 있다는 것이다. 간혹 컴퓨터 상식 문제 등을 보면 "IP 주소와 컴퓨터는 1:1로 대응된다"고 나와 있고 이것이 맞는 것으로 되어 있는 경우를 보는데, 엄밀히 말하면 틀린 말이다. IP 주소는 컴퓨터와 대응하는 것이 아니라 컴퓨터 안에 꽂혀 있는 랜카드와 1:1 대응된다. 그리고 랜카드라는 말은 범용적으로 쓰이는 말이긴 하지만 좋은 표현은 아니다. "랜카드"는 약간 용산의 분위기가 나는 용어이고, 정식으로는 Ethernet Card라고 해야 맞다. 왜냐하면 ethernet card가 꼭 랜 환경에서만 사용되는 것은 아니기 때문이다. WAN, CAN 환경 등에도 모두 Ethernet card를 사용한다. 물론 최근 WAN이니 LAN이니 하는 개념은 그다지 의미가 없다.
NIC(Network Interface Card)라는 것은 이더넷 카드보다 약간 더 포괄적인 의미로서 네트워크 연결을 할 수 있는 카드들을 통칭해서 NIC이라고 한다.

▪ 도메인 주소란 yahoo.com과 같이 우리가 알아보기 쉬운 글자로 되어 있는 컴퓨터의 주소를 말한다. 이러한 도메인 주소는 오로지 사람을 위한 것일뿐, 컴퓨터는 이러한 주소를 모두 IP 주소로 변환해서 처리한다. 이러한 변환을 처리해 주는 것이 바로 DNS(Domain Name Service)이다.

▪ FQDN이라는 표현도 가끔 나오는데, 이것은 Fully Qualified Domain Name(완전하게 표기된 도메인 이름)을 말한다. 일반적으로 yahoo.com이라는 도메인 앞에 www를 붙여서 쓰는데, www는 yahoo.com 도메인에 속한 컴퓨터의 이름이다. 컴퓨터의 이름은 어떤 것이라도 될 수 있다. mail.yahoo.com도 될수 있고, ftp.yahoo.com도 될 수 있다. 이렇게 컴퓨터의 이름과 도메인 이름을 합쳐서 표기하는 것을 FQDN이라고 한다. 그러나 많은 경우 www.yahoo.com과 같이 단어로 쓴 주소를 그냥 "도메인"이라고 부르는 경우가 많다. 그러나 엄밀히 얘기하면 도메인 주소는 yahoo.com까지만이고 www.yahoo.com은 FQDN 표기법이다.

▪ 포트(port)라는 것은 하나의 컴퓨터에서 실행되고 있는 많은 네트웍 프로그램을 구분하기 위해 부여된 번호이다. 잘 알려진(well-known) 응용 계층 프로토콜은 특정 포트 번호가 할당되어 있다. 예를 들면 FTP는 21번, TELNET은 23번, SMTP는 25번, DNS는 53번, HTTP는 80번, POP3는 110번, IMAP은 143번 등이다. 우리가 야후 사이트에 접속할때에는 주소만 입력하고 포트 번호는 명시하지 않지만, 웹브라우저에 의해 자동적으로 80번 포트로 접속을 하는 것이다. 이 번호는 1부터 65535까지 사용할 수 있으나, 1024번까지는 잘 알려진(well-known) 포트 번호이므로 사용하지 않는 것이 좋다. 특히 유닉스 계열에서는 1024번 이하의 포트 번호를 사용하려면 특권이 있어야 하는 경우도 있다. 0번 포트는 예약된 포트 번호로서 사용되지 않는다.

1.5 다중화(multiplexing)와 역다중화(demultiplexing)

서버에는 각기 다른 포트 번호를 사용하는 여러 서버 프로그램이 실행중일 수 있고, 클라이언트는 이 서버 프로그램에 동시에 접속하여 서비스를 받는 상황을 가정할 수 있다. 편의를 위해 클라이언트 측의 프로세스를 각각 C1, C2라고 하고, 서버측의 프로세스를 S1, S2라고 하자. C1은 S1 프로세스에 접속하여 서비스를 받는 중이고, C2는 S2 프로세스에 접속하여 서비스를 받는다고 가정하자. 이 경우에 클라이언트에서 서버로 수많은 패킷이 그냥 보내지게 되는데, 서버측에서는 이 패킷들을 S1 프로세스에게 전달해야할지 S2에게 전달해야할지 판단을 해야한다. 이때 사용하는 정보가 IP 주소와 포트 번호의 쌍이다. IP 주소 하나만으로는 서버로의 패킷 전달은 할수 있지만, 서버로 전달된 패킷을 적절한 응용 프로그램에 전달할 수는 없다. 이때 포트 번호를 같이 사용하는 것이다.

용어를 정의하면 다중화란 응용 계층의 데이터를 네트워크 계층으로 넘겨주는 것을 말하며, 역다중화란 전송 계층의 데이터를 올바른 응용 프로그램에 전달하는 것을 말한다. 다중화와 역다중화의 목적은 패킷을 올바른 응용 프로그램에게 전달하기 위함이고, 이를 위해 IP 주소와 포트 번호를 사용한다.

2. 응용 계층 프로토콜의 예: HTTP

실제 프로그램 개발에 들어가기에 앞서 간단한 응용 계층 프로토콜에 대해 알아보기로 하자. 이후에 개발할 프로그램도 HTTP를 이용한 것이 될 것이므로 HTTP에 대해 간단히 소개를 한다. 응용 계층 프로토콜을 연구해 보는 것은 추후에 여러분이 직접 독자적인 프로토콜을 설계할 때에도 많은 참고가 될 것이다. 특히 HTTP는 비교적 간단하기 때문에 처음 연구하기에도 적당하다.

HTTP는 Hyper Text Transfer Protocol로서 응용 계층에서 작동하는 프로토콜이다. 용도는 잘 알다시피 웹에서 URL(Uniform Resource Location)을 이용하여 HTML 파일이나 그림 파일 등을 전송할 수 있는 프로토콜이다. 이 HTTP는 현재 버전 1.0과 1.1이 있는데, 현재의 웹브라우저나 웹서버는 모두 버전 1.1을 지원한다. HTTP 버전 1.1은 버전 1.0과 하위 호환성이 있기 때문에 결국은 현재의 웹브라우저나 웹서버는 버전 1.0과 1.1을 모두 지원하는 셈이다. 물론 버전 1.1이 좀더 강력하고 효율적인 기능들이 있지만 아무래도 약간 더 복잡하다. 따라서 우리는 HTTP 1.0에 한해 알아보고, 이후에도 HTTP 1.0 규격으로 프로그램을 제작하도록 하자. 물론 HTTP 1.0에 맞추어 제작을 해도 HTTP 1.1을 지원하는 애플리케이션과 호환 가능하다. 여기서 제작해볼 프로그램 이상의 수준을 원한다면 RFC 문서를 더 참고하기 바란다.

우리가 야후 사이트에 접속하려면 웹브라우저를 열고 주소란에 http://kr.yahoo.com을 입력한다. 그러면 웹브라우저가 야후 웹서버에 접속하여 html 문서 및 관련 그림들을 받아서 여러분의 모니터로 보여주는 것이다. 이 과정을 telnet을 이용하여 직접 해볼 수 있다. 일단 MS-Windows 환경에서 해보려면 도스창을 열고 유닉스 환경이라면 터미널 창을 하나 열도록 한다. 그리고 다음과 같이 명령을 내려본다.

```
[root@chaos ~]# telnet kr.yahoo.com 80
Trying 211.32.119.135...
Connected to kr.yahoo.com.
Escape character is '^]'.
GET / HTTP/1.0
```

굵은 글자로 된 것을 입력하면 된다. 먼저 쉘 프롬프트에서 “telnet kr.yahoo.com 80”을 입력하면 kr.yahoo.com 서버의 80번 포트로 접속을 하게 된다. 접속이 되면 “GET / HTTP/1.0”을 입력하고 엔터를 두번 친다. 이 명령은 HTTP 프로토콜의 일부로서 웹페이지를 요청하는 명령이다. 형식은 “[method] [URL] [HTTP version]”이다. Method는 GET, POST, HEAD 등이 올수 있으며 대문자가 원칙이다. URL은 원하는 주소의 URL을 입력하면 된다. 그 다음에 HTTP 버전은 HTTP/1.0 혹은 HTTP/1.1이 올수 있는데, 우리는 버전 1.0을 사용할 것이므로 HTTP/1.0을 사용하면 된다. 이렇게 입력하고 엔터를 두번치면 야후 웹서버로부터 html 문서가 전송됨을 볼 수 있을 것이다. 이것이 가장 간단한 HTTP 1.0의 사용 예이다. 우리가 제일 먼저 만들 프로그램은 방금 telnet으로 해본 것을 자동화하는 프로그램에 지나지 않는다.

3. CAsyncSocket과 CSocket

이 글을 읽는 여러분은 최소한 MFC 프로그래밍에 대한 기초는 있다고 가정한다. 따라서 MFC에 대한 기본 개념이라든가 MFC를 이용하여 간단한 프로그램 만드는 법 등은 따로 설명하지 않는다.
MS-Windows 환경에서 소켓 프로그래밍을 하는 방법은 크게 2가지로 나눌 수 있다. Win32 API를 직접 이용하는 방법과 MFC[1]에서 제공하는 클래스를 이용하는 방법이다. 여기서는 MFC를 이용하여 소켓 프로그래밍을 해 볼 것이다.
MFC에서 소켓 프로그래밍을 위해 제공되는 클래스는 2개가 있는데 CAsyncSocket과 CSocket이다. CAsyncSocket 클래스는 CObject에서 바로 상속된 클래스로서 비동기 소켓을 지원한다. 그리고 CSocket은 다시 CAsyncSocket에서 상속된 클래스로서 동기 소켓이며, CArchive를 이용한 입출력도 가능하다. 비동기 소켓이란 데이터 송/수신 함수를 호출했을 때 호출을 하자마자 바로 리턴이 되는 것을 말하고, 동기 소켓이란 데이터 송/수신 함수를 호출했을 때 함수가 끝날때까지 반환하지 않는 것을 말한다. CAsyncSocket과 CSocket 중 어떤 것을 사용할 것인지를 작업의 성격과 개인의 프로그래밍 스타일 등에 영향을 받을 수 있겠는데, 여기서는 CSocket을 이용하여 프로그래밍을 해 보겠다. CSocket이 아무래도 CAsyncSocket보다 더 추상화된 클래스이기 때문에 우리가 실제로 사용하기엔 좀더 편하다.

---

[1] MFC란 Microsoft Foundation Class로서 MS사에서 제공하는 클래스 모음이다. 마치 자바에서 제공되는 클래스 집합과 비슷한 것이다.

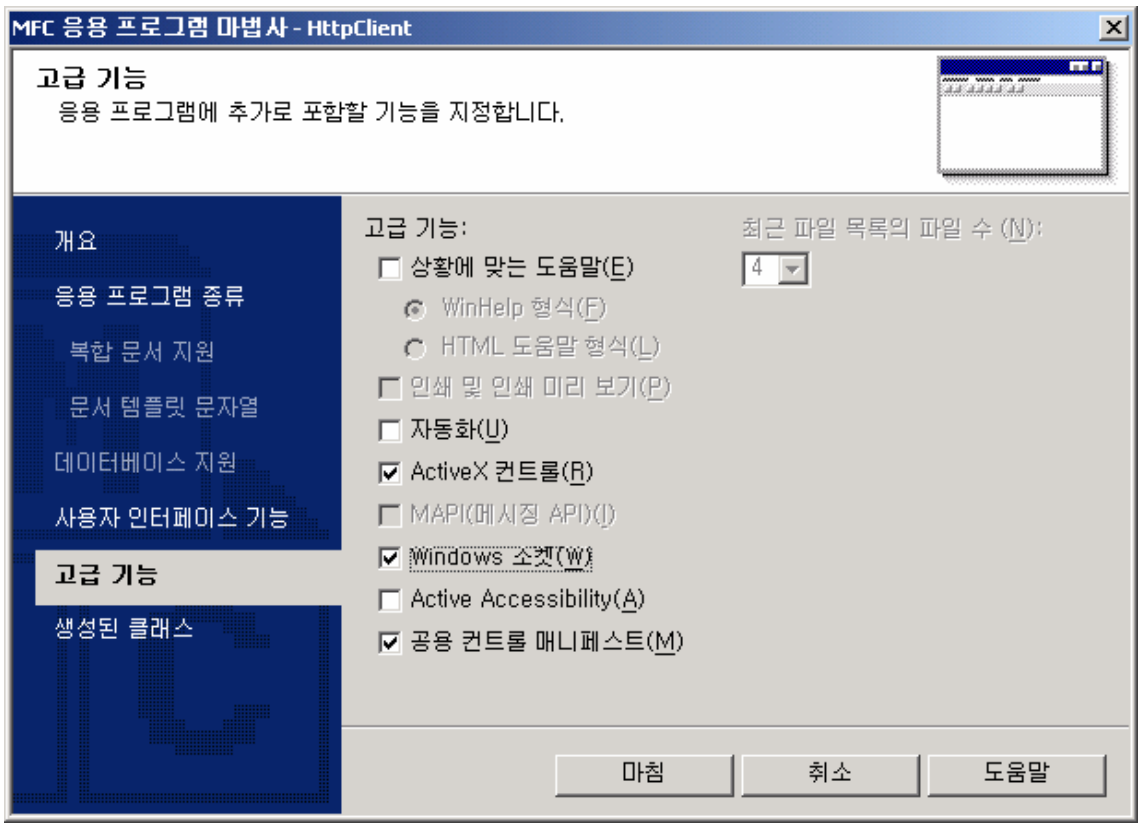
4. 첫 소켓 프로그래밍: HTTP Client 초기버전

일단 필자의 개발환경은 MS-Windows 2000 Professional 및 XP Professional에 Visual Studio C++.NET이다. Visual Studio 6.0을 써도 아무 상관은 없으나, 기본적으로 클래스 상속 방법과 같은 기본적인 통합개발환경의 사용법은 알고 있어야 할 것이다.

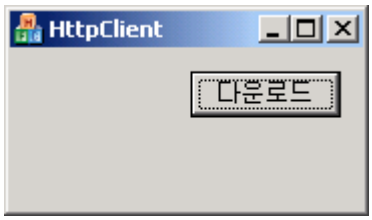
여기서는 네트워크 프로그램을 처음 만들어보는 사람을 위해 가장 간단한 클라이언트 프로그램을 만들어 볼 것이다. 만약 이 정도 수준은 간단히 만들 수 있다면 이 장은 그냥 건너뛰어도 좋다.

여기서 만들어 볼 프로그램은 HTTP 프로토콜을 준수하여 웹에 위치한 특정 파일을 받아오는 클라이언트 프로그램을 만들 것이다. 즉 우리가 만들어볼 프로그램은 웹 상에 위치하는 그림이라든가 html 파일, mp3 파일 등을 HTTP 프로토콜을 이용하여 전송받아 파일로 저장해주는 프로그램이다.

Visual Studio를 실행하여 MFC 응용 프로그램을 만들되 Dialog based(대화상자 기반)로 하자. 그리고 “고급 기능”에서 “Windows 소켓”도 선택하자. Visual Studio .NET을 기준으로 설명하지만 Visual Studio 6.0에서도 AppWizard에서 소켓 지원을 선택하는 것이 있다. 프로젝트 이름은 필자의 경우 HttpClient로 하였으나, 이것은 독자들 마음대로 해도 된다. 만약 지금 설명하는 것도 따라하지 못하겠다면 MFC 기본을 더 공부해야 될 것이다. 이 정도가 이해가 안 된다면 더 이상 이 글을 읽어봐도 별 의미가 없을 것이다.



위와 같이 “Windows 소켓”을 선택하면 된다. 그리고 다음과 같이 컨트롤들을 위치시키자. 1개의 버튼이 있다. 이보다 더 간단할 수가 없다.



이제 여러분이 많은 MFC 기본 서적에서 했던 것과 같이 여기에서는 “다운로드” 버튼의 클릭(BN_CLICKED)에 대한 이벤트 핸들러를 만들면 된다. 다음이 이 버튼에 대한 이벤트 핸들러의 전부이다.

```
void CHttpClientDlg::OnBnClickedButtonDownload() {
    CSocket socket;
    socket.Create(0);
    socket.Connect("kr.yahoo.com", 80);

    // 야후 서버로 "GET / HTTP/1.0" + 엔터 두번 메세지 보내기.
```

```
    char *request = "GET / HTTP/1.0WrWnWrWn";
    socket.Send(request, strlen(request));

    // 야후가 보내오는 정보 받아서 저장하기.
    CFile file;
    file.Open("test.html", CFile::modeCreate | CFile::modeWrite);

    int cbRcvd;
    char buf[4000];
    while ((cbRcvd = socket.Receive(buf, 4000)) > 0) {
        file.Write(buf, cbRcvd);
    }

    file.Close();
    //.

    socket.Close();
}
```

바로 이 코드가 가장 간단한 소켓 프로그램의 형태이다. 간단하지만 작동은 잘 된다. 첫 줄부터 설명을 하면 다음과 같다.

여기서는 가장 간단한 형태를 만들어 보는 것이 목적이므로 CSocket 클래스의 인스턴스를 "CSocket socket;"과 같이 직접 생성하여 사용하였다. 이렇게 CSocket의 인스턴스를 하나 만든 후에, 이것을 가지고 여러 멤버 함수들을 호출하면 되는 것이다.

가장 먼저 호출한 함수가 Create() 함수이다. 이것은 소켓을 초기화해주는 함수라고 보면 된다. 원래 이 함수는 인자를 4개까지 받을 수 있는데, 보통 맨 처음 인자인 포트 번호만 주면 된다[1]. 여기서는 포트 번호로 0을 넘겨주었다. 형식적으로 0을 포트 번호로 넘겨주었지만 클라이언트 측의 프로그램을 만들 때는 특별히 포트 번호를 주지 않는다. 사실 포트 번호 0은 사용하지 않는 포트 번호이다. 그런데 여기서 분명히 알아야 할 사실이 있는데, 포트 번호를 주지 않았다고 해서 클라이언트측에서 포트 번호가 사용되지 않는 것이 아니다. 다만, 운영체제가 현재 사용하지 않는 포트 번호 중 하나를 임의로 할당해 줄 뿐이다. 클라이언트 측에서 사용하는 포트 번호는 보통 알 필요는 없다. 그냥 운영체제가 알아서 포트 번호를 할당해주면 그냥 쓰면 된다. 물론 어떤 포트 번호가 자동으로 할당되었는지 꼭 알아야겠다면 알아낼 수는 있다.

이렇게 소켓을 초기화한 이후에는 Connect 함수를 호출한다. 이 함수를 호출하면 서버측으로 접속을 하게 되는데, 인자로는 접속할 서버의 주소와 포트 번호를 적어주면 된다. 서버의 주소는 도메인 주소가 될 수도 있고 IP 주소가 될 수도 있다. 여기서는 "kr.yahoo.com" 서버의 80번 포

[1] 사실 인자를 포트 번호도 주지 않고 아무것도 넘겨주지 않아도 된다. 이 경우 포트 번호로 0을 넘긴 것과 동일하게 처리된다. Create 함수의 프로토 타입을 보면 소켓 인자를 생략했을 경우 0을 대입해 주도록 되어 있다.

트[1]로 접속하였다. 다시 한번 말하지만 여기서 Connect 함수에서 적어준 포트 번호는 접속할 서버의 포트 번호이다. 클라이언트의 포트 번호는 위에서 Create 함수를 호출할 때 운영체제에 의해 이미 임의로 할당되었다. 여기까지 했으면 야후의 웹서버에 연결이 된 상태이다. telnet으로 테스트해본 것과 비교해보자면 "telnet kr.yahoo.com 80"이라고 치고 엔터를 친 상태까지 구현된 것이다.

텔넷으로 야후 서버에 접속한 뒤에 웹페이지를 얻어오기 위해 "GET / HTTP/1.0[엔터 2번]"과 같은 메시지를 보내준 것을 기억할 것이다. 이제 그 메시지를 야후 웹서버로 보내줄 차례이다. 그것은 Send 함수를 이용하면 된다.
Send 함수는 보낼 데이터와 그 데이터의 크기를 인자로 받는다. 보낼 데이터와 크기를 별도로 받는 이유는 바이너리 데이터 전송을 위해서이다. Send 함수의 원형을 잘 살펴보면 알겠지만 첫번째 인자인 데이터는 void *형이다. 이 말은 문자열 뿐만 아니라 어떤 바이너리 형태라도 모두 보낼 수 있다는 뜻이다. 그래서 널 종료(null terminated) 문자열이 아닌 데이터도 제대로 전송하기 위해 별도로 데이터의 크기도 인자로 받는 것이다. 그리고 엔터에 해당하는 기호로 "\r\n"을 사용했는데 이것은 CR+LF[2] 기호이다. 보통 많은 경우에 \n 기호만 사용해도 엔터로 인식이 되는데, 네트워크 프로그래밍을 할 때에는 명시적으로 \r\n을 모두 써 주기를 권장한다. 이기종간의 호환성을 극대화하기 위한 목적이다.

이제 서버로 문서 요청을 했으므로 서버는 요청한 문서를 클라이언트에게 보내줄 것이다. 그러면 클라이언트 측에서 Receive 함수를 호출하여 서버가 보내주는 데이터를 받으면 된다. Receive 함수를 while loop 안에서 호출하면서 전송받는 대로 족족 파일에 기록을 했다. Receive 함수는 데이터를 받을 버퍼와 한번에 받을 수 있는 최대 크기를 인자로 받는다. 여기서는 char형의 버퍼 4000바이트를 마련해서 그곳에 데이터를 받았다. 그리고 Receive 함수는 실제로 받은 데이터수를 반환한다. 우리는 최대 4000바이트까지 받으라고 했지만 항상 4000바이트를 모두 다 받는 것은 아니다. 그래서 항상 몇 바이트를 받았는지 체크를 해서 그 바이트만큼만 파일에 기록한 것이다. 그리고 특별한 경우로 이 함수가 0을 리턴한다면 접속이 끊긴 것이다.
위에서 보듯이 while loop의 조건으로 ((cbRcvd = socket.Receive(buf, 4000)) > 0)을 검사하는데, 이 의미는 서버로의 접속이 끊길때까지 계속 받으라는 의미이다. HTTP/1.0은 그 특성상 요청한 데이터가 모두 전송되면 서버측에서 접속을 끊어버리게 되어 있다.
여기서 한가지 의문점이 생기는 독자도 있을 것 같다. 만약 네트웍 접속이 매우 느려서 while loop의 조건문 안에서 Receive 함수를 호출하고 다음번 Receive 함수를 호출할때까지 1바이트도 수신되지 않았다면 어떻게 될 것인가 하고 말이다. 전송받을 것은 더 있지만 접속이 워낙 느려서 아직 전송받은 데이터가 없을 경우 말이다. 이 경우에 Receive 함수가 0을 반환할까? 방금

---

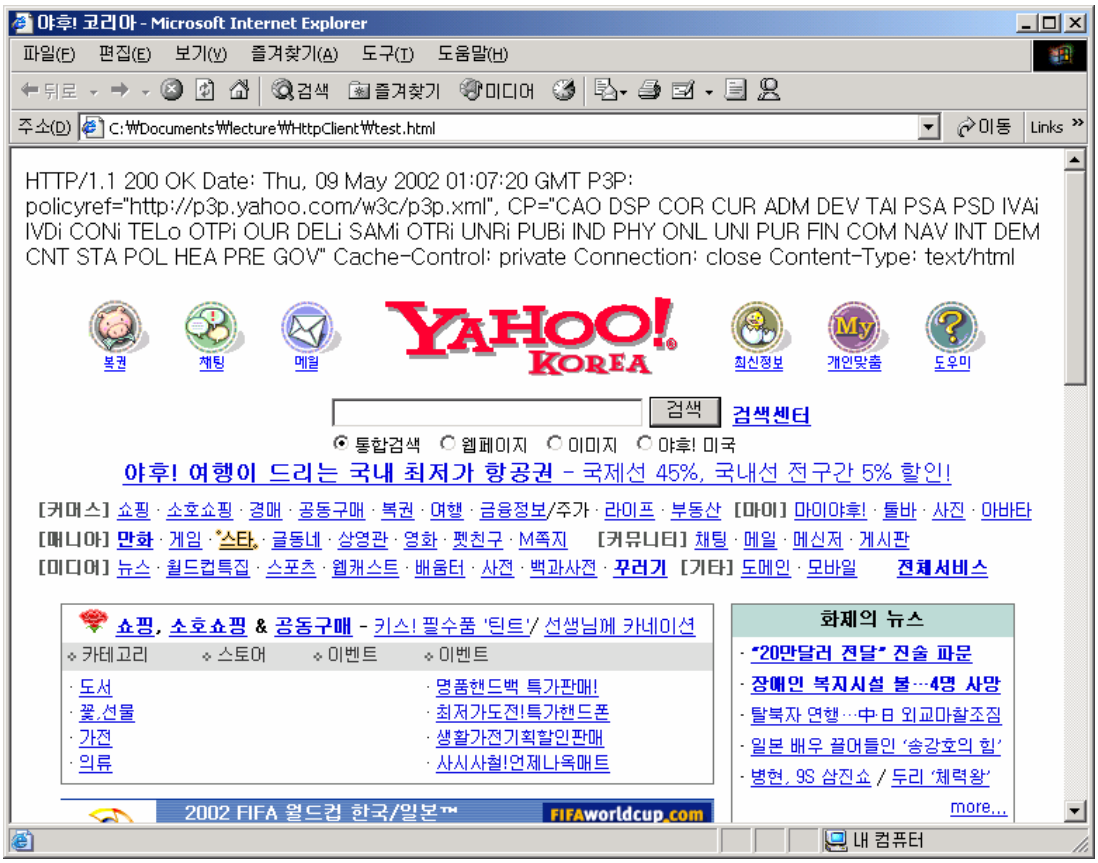
[1] 우리가 만들고 있는 프로그램이 HTTP 클라이언트를 구현하는 프로그램이기 때문에 당연히 HTTP가 사용하는 포트 번호인 80번 포트로 접속을 해야 한다.
[2] Carriage Return + Line Feed

얘기했지만 이 함수의 반환값 0은 접속이 끊겼을 경우에만이다. 정답은 “블록(block)”된다는 것이다. 쉽게 말하면 1바이트라도 서버가 보내줄때까지 Receive 함수는 반환을 하지 않는다는 것이다. 즉 Receive 함수에서 그냥 멈춰있는 것이다. 이것을 블록된다고 하고 동기화 소켓의 특징이다. 앞에서도 얘기했지만 우리가 지금 사용하고 있는 CSocket은 동기화 소켓이다. 따라서 이러한 문제점이 생기는 것이다. 그러나 인터넷 접속 환경이 좋은 경우 이런 것은 별 문제가 되지 않는다. 아마 여러분도 대개 인터넷 접속 환경이 양호하기 때문에 위의 코드는 아무런 문제도 일으키지 않을 것이고, 전혀 막힌다거나 멈춘다는 느낌은 받기 어려울 것이다. 그러나 어떠한 환경에서도 견고(robust)하고 안정적으로 작동하는 프로그램을 만들기 위해서는 프로그램이 계속 멈춰 있어서는 안되며 특별한 처리, 즉 타임아웃 처리를 해줘야 한다. 이것에 대해서는 다음에 다시 알아볼 것이다.

이렇게 모두 전송을 받은 후에는 “socket.Close();”와 같이 Close 함수를 호출하여 소켓을 닫으면 된다. 물론 파일도 다 사용했으니 파일도 닫아주었다.

지금까지 모두 잘 따라했다면 여러분은 이제 소켓 프로그래밍을 할 수 있는 것이다. 필자는 항상 이런 초간단한 코드로 설명된 책이 거의 없다는 것이 항상 안타까웠다. 서점에 가보면 Visual C++에 대해 논하는 많은 책에서 소켓 프로그래밍에 대한 예제를 제시하면서 터무니없이 복잡한 코드를 제시하는 경우가 많다. 심지어 채팅 예제와 같은 것을 그냥 바로 제시해버리는 경우도 있는데 초보자가 그것을 따라하기란 정말 어렵지 않나 생각된다. 필자는 무언가를 설명할 때에는 핵심만을 나타내고 나머지는 모두 생략해 버려서 설명하고자 하는 것만을 최대로 부각시키고 최대한 간단히 설명하는 것을 좋아한다. 응용은 여러분의 몫이지 필자가 해줄 수 있는 것은 아니다.

이제 위의 프로그램을 컴파일해서 실행해보면 야후 웹서버로 접속해서 초기 웹페이지를 받아온 후에 그것을 test.html 파일로 저장할 것이다. test.html 파일을 웹브라우저로 열어보면 다음과 같이 보일 것이다.



야후 첫 페이지가 맞긴 맞는데 위에 뭔가 이상한 코드들이 있는 것을 볼 수 있다. 프로그램이 잘못된 것이 아니라 HTTP 헤더가 붙어있는 것 뿐이다. 우리가 프로그램에서 특별히 HTTP 헤더와 본문(body) 부분을 나누지 않고 모두 저장했기 때문에 이렇게 된 것이다. 이 test.html 파일을 텍스트 에디터로 열어보면 HTML 코드가 시작되기 전에 다음과 비슷한 헤더가 보일 것이다.

```
HTTP/1.1 200 OK
Date: Thu, 09 May 2002 01:07:20 GMT
Cache-Control: private
Connection: close
Content-Type: text/html
```

이것이 웹서버가 클라이언트에게 보내주는 HTTP 헤더이고, 실제 본문(body)는 헤더 다음에 공백 한줄이 위치한 다음에 이어진다. 실제로 telnet을 이용하여 야후의 웹서버로 접속해서 테스트해봐도 잘 보면 위와 같은 헤더가 먼저 보내진 이후에 본문이 보내지는 것을 볼 수 있을 것이다. 헤더와 본문을 구분하는 기준은 맨 처음 나오는 공백 줄(기호로는 "\r\n\r\n")이다.

여기까지 알아본 것만 가지고도 여러분은 인터넷 게시판에서 이메일을 긁어오는 프로그램이라든가 인터넷에 올려져 있는 MP3 파일(물론 실시간 방송을 말하는 것은 아니다.) 등을 받는 프로그램을 만들 수 있을 것이다. 물론 견고한 프로그램, 즉 중간에 멈춰버리거나 하지 않는 프로그램을

만들기 위해서는 추가적인 코드가 필요하다.

마지막으로 소켓 생성(Create 함수)과 서버 접속시(Connect 함수) 위의 코드는 에러 처리를 하나도 하지 않았는데, 실제로는 다음과 같이 에러 처리를 해 줘야 좋을 것이다. 물론 이 정도는 독자 여러분이 알아서 할 것이라 생각한다. 참고로 Create는 성공했고 Connect만 실패할 경우에 Close 함수를 호출해서 소켓을 닫아주어야 한다.

```
// 소켓 생성.
if (!socket.Create(0)) {
    MessageBox("소켓 생성에 실패했습니다.", "에러", MB_OK | MB_ICONEXCLAMATION);
    return;
}

// 접속 시도
if (!socket.Connect("kr.yahoo.com", 80)) {
    socket.Close();
    MessageBox("서버 접속에 실패했습니다.", "에러", MB_OK | MB_ICONEXCLAMATION);
    return;
}
```

5. 타임아웃 기능 구현

이번에는 타임아웃 처리가 된 클라이언트 프로그램을 만들어 보도록 하겠다. 이 장을 제대로 이해하기 위해서는 C++의 클래스 상속 및 재정의(overriding)에 대한 이해가 필수적이다. 이것은 C++의 기초에 해당하는 것이므로 여기서는 별도로 설명하지 않겠다.

이전에 만들어본 프로그램은 정말로 가장 간단한 형태를 구현했기 때문에 CSocket의 인스턴스를 직접 생성해서 사용했지만, 타임 아웃 처리 등을 하기 위해서는 그렇게 해서는 안된다. CSocket 클래스를 상속받아 새로운 클래스를 만들고 이 클래스의 몇가지 함수를 overriding(재정의)해야 한다. 그리고 이 새로운 클래스의 인스턴스를 생성시켜서 사용해야 한다.

CSocket 클래스는 전에도 얘기했지만 동기화 소켓이고 Connect, Send, Receive 함수 등을 호출했을 때 바로 처리가 되지 않으면 멈춰진 상태(블록된 상태)로 있게 된다. 그래서 CSocket에는 블록된 상태에서 어떤 메시지가 발생하면 이를 감지하고 OnMessagePending이라는 함수를 호출해 주는 기능이 있다. 그런데 우리가 이 OnMessagePending 함수 내에서 어떤 처리를 해주기 위해서는 이 함수를 overriding해야 하고, 그러기 위해서는 CSocket 클래스를 상속받아 자신만의 클래스를 새로 만들어야 하는 것이다.

함수를 호출했을 때 블록될 수 있는 함수는 Connect, Send, Receive 정도이다. 먼저 어떤 경우에 타임 아웃이 필요한지 실제로 예를 들어 보여주도록 하겠다.

만약 야후 서버로 접속하되 실수로 1000번 포트로 접속을 시도했다고 가정하자. 지금 바로 도스 창이나 터미널창을 열어서 "telnet kr.yahoo.com 1000"을 입력하고 엔터를 쳐보자. "Trying 211.32.119.135..."이라고 나오더니 아무리 시간이 지나도 아무런 응답이 없을 것이다[1]. Ctrl-C 를 눌러 그냥 종료하도록 하자. 물론 1~2분 정도가 지나면 텔넷 프로그램이 타임아웃을 걸어서 종료될 수도 있지만 그것은 텔넷 프로그램에서 접속을 포기하고 타임아웃 처리를 한 것이다. 실제 우리가 만든 프로그램에서는 뭔가가 잘못되었을 때 영영 블록된 상태에서 빠져나올 수가 없다. 예를 들면 Connect 함수를 호출하여 서버로 접속을 시도했는데, 성공도 실패도 아닌 아무런 응답도 없는 경우에 블록될 수 있다. 또한 서버는 아무런 데이터도 보내줄 생각을 안하는데 클라이언트에서 Receive 함수를 호출한 경우에도 블록된다. Send 함수 호출 역시 블록될 수 있다. 정말 무한정 대기 상태에 빠지게 된다.

이런 경우에 대한 해결책으로 타임아웃을 처리하는 예제를 만들어 보기로 한다. 우리가 제일 먼저 만들어볼 프로그램은 야후에 1000번 포트로 접속하되 5초 동안 Connect 함수가 반환되지 않는다면 강제로 종료시켜 버리는 것이다. 방금 테스트해보아서 알겠지만 야후 웹서버로 1000번 포트에 접속하면 아무런 응답이 없이 블록될 것이다. 이 자리를 빌어 좋은 테스트 환경을 제공해준 야후 코리아 측에도 감사를 표하는 바이다.

Visual C++을 이용해서 새로운 프로젝트를 만들되 Dialog based(대화상자 기반)으로 하고 역시 버튼 하나만 위치시키도록 하자. 필자는 프로젝트 이름을 TimeOut이라고 하였다. 혼동을 피하기 위해 되도록이면 필자와 같은 이름을 사용하도록 하자. 물론 처음 프로젝트를 생성할 때 "Windows 소켓"에 체크하는 것을 잊지 말자. 다음과 같이 각자 재량에 따라 만들면 되겠다.

역시 정말 간단한 프로그램이다. "접속 시도" 버튼을 누르면 야후 웹서버의 1000번 포트로 접속을 시도하고 5초가 지나면 타임아웃이 되는 프로그램이다. 그런데 실제 코드를 작성하기전에 해야할 일이 있는데 CSocket 클래스를 상속하여 새로운 소켓을 하나 만드는 일이다. 필자는 CSocket을

[1] 야후에 1000번 포트로 접속했을 때 아무런 응답이 없는 이유는 야후 측의 방화벽 설정 때문이다. 야후 웹서버의 80번 포트를 제외한 포트는 보안을 위해 모두 접속을 막아버렸고, 이로 인해 아무런 응답이 없다. 방화벽 설정은 보통 거부와 무시가 있는데 이 경우는 무시이다. 더 자세한 것은 방화벽 관련 문서(iptables 등)를 참조하기 바란다.

상속한 클래스를 CDataSocket이라는 이름으로 만들었다. 그러면 CDataSocket 클래스의 선언 및 구현 파일인 DataSocket.h와 DataSocket.cpp 파일이 생겼을 것이다.

우리가 CSocket 클래스를 상속한 이유는 OnMessagePending 함수를 overriding하기 위해서 이다. 그 함수를 다음과 같이 재정의하자.

```
BOOL CDataSocket::OnMessagePending() {
    MSG Message;
    if (::PeekMessage(&Message, NULL, WM_TIMER, WM_TIMER, PM_NOREMOVE)) {
        if (Message.wParam == 10) {
            ::PeekMessage(&Message, NULL, WM_TIMER, WM_TIMER, PM_REMOVE);
            CancelBlockingCall();
            Close();
        }
    }

    return CSocket::OnMessagePending();
}
```

만약 여러분이 Win32 API 프로그래밍에 익숙하다면 위의 코드를 아무런 부담없이 이해할 것이지만 MFC로만 프로그래밍하던 분이라면 이해하기 좀 힘들지도 모르겠다. 만약 위의 코드가 잘 이해가 되지 않는다면 김상형씨의 "윈도우 API 정복" 책의 윈도우, 메시지 파트 부분을 참고하기 바란다. 참 잘 나와있는 책이다. 한마디 덧붙이자면 아무리 MFC로 프로그래밍을 하더라도 Win32 API를 제대로 모르고서는 프로그래밍을 잘 할 수가 없다. 방금 소개한 김상형씨의 "윈도우 API 정복"은 초/중급 수준에서 필자가 추천하는 책이다. Win32 API를 처음 공부해보겠다면 그 책을 보는 것도 괜찮을 것이다.

OnMessagePending 함수는 Connect, Send, Receive 등의 함수 호출로 블록된 도중에 어떤 메시지가 발생하면 자동으로 호출되는 함수이다. 이것은 CSocket 클래스에 의해 제공되는 기능이다.
OnMessagePending 함수 안에 overriding한 코드를 간략하게 설명해 보자면 PeekMessage(&Message, NULL, WM_TIMER, WM_TIMER, PM_NOREMOVE)[1] 함수는 현재 메시지 큐에 WM_TIMER 이벤트가 있는지 확인하는 것이다. 확인만 하고 WM_TIMER 메시지는 메시지 큐에 그대로 남겨둔다. 만약 WM_TIMER 메시지가 메시지 큐에 있다면 다시 이 메시지의 WPARAM값이 10인지 확인한다. WM_TIMER가 발생했을 때 WPARAM에는

[1] 첫번째 인자는 메시지 구조체를 입력받을 변수의 포인터, 두번째 인자는 메시지를 받을 윈도우의 핸들인데 NULL을 지정하면 현재 쓰레드의 모든 메시지를 대상으로 한다. 그리고 세번째와 네번째는 어떤 메시지를 체크할 것인지에 대한 일종의 범위를 지정한 것이다. WM_TIMER부터 WM_TIMER까지로 정의했으니 이것은 WM_TIMER 메시지만 정확하게 체크하라는 것이다. 마지막 PM_NOREMOVE는 메시지 확인만 하고 메시지 큐에서 메시지를 지우지는 말라는 의미이다. 자세한 것은 MSDN을 참고하기 바란다.

타이머의 ID가 들어간다. 즉 타이머의 ID가 10번인지 확인하는 것이다. 만약 타이머의 ID가 10이 맞으면 메시지 큐에서 WM_TIMER 메시지를 삭제하고 CancelBlockingCall과 Close 함수를 호출한다. CancelBlockingCall은 현재 블록된 상태로 있는 함수를 취소하는 함수이고 Close 함수는 소켓을 닫아버리는 것이다. 즉 타임아웃이 되면 블록된 함수를 취소하고 접속을 끊어버리는 것이다.

이렇게 OnMessagePending 함수를 재정의한 후에 실제로 타임아웃을 이용한 프로그램을 작성해 보도록 하자. 우선 TimeOutDlg.cpp(혹은 여러분의 다이얼로그 클래스 구현 파일)에 #include "DataSocket.h"를 써줘야 한다. CSocket의 인스턴스를 생성할 것이 아니라 CSocket을 상속받은 CDataSocket의 인스턴스를 만들 것이기 때문에 이 클래스의 선언 파일을 포함시켜줘야 한다.
이제 "접속 시도" 버튼에 대한 이벤트 핸들러를 만들고 다음과 같이 코드를 입력한다.

```
void CTimeOutDlg::OnBnClickedButtonConnect() {
    CDataSocket socket;

    if (!socket.Create(0)) {
        MessageBox("소켓 생성에 실패했습니다.", "에러", MB_OK | MB_ICONWARNING);
        return;
    }

    SetTimer(10, 5000, NULL);
    if (!socket.Connect("kr.yahoo.com", 1000)) {
        KillTimer(10);
        socket.Close();
        MessageBox("타임 아웃되었습니다.", "에러", MB_OK | MB_ICONWARNING);
        return;
    }
    KillTimer(10);
    MessageBox("서버에 접속되었습니다.", "알림", MB_OK);

    socket.Close();
}
```

위의 코드가 타임아웃을 구현한 간단한 예이다. 여기서는 Connect 함수가 블록된다는 가정하에 Connect 함수가 5초동안 반환하지 않으면 강제로 Connect 함수 호출을 취소하도록 한 것이다. 소켓을 생성(Create)하는 부분은 이전과 다를 것이 없다. 주의해서 볼 부분은 Connect 함수를 호출하는 전후이다. 즉 SetTimer 호출 부분부터 if 문 다음의 KillTimer 부분까지가 핵심이다.

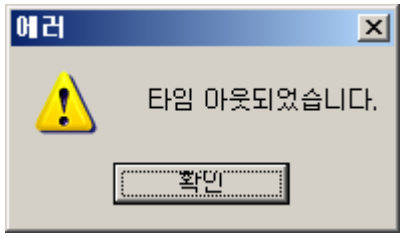
두가지 시나리오를 생각해 보자.
우선 Connect 함수가 성공적으로 호출되어 지체없이 TRUE를 반환하는 경우를 생각해 보자. 이 경우 SetTimer 함수로 ID 10번, 시간 간격 5초의 타이머를 설치하고, 바로 Connect 함수를 호출하게 된다. 그리고 이 Connect 함수가 TRUE를 반환하기 때문에 if 문 안의 코드는 실

행되지 않고 바로 if 문 다음으로 가서 ID 10번의 타이머를 파괴하고는 계속 다음의 작업을 계속할 것이다. 이런 경우 Connect 함수가 5초 안에 충분히 끝날 것이므로 WM_TIMER 메시지는 발생하지 않는다[1].

이번엔 Connect 함수를 호출했을 때 블록된다는 시나리오를 생각해 보자. 일단 SetTimer 함수로 타이머 ID 10번, 시간 간격 5초[2]의 타이머를 생성했다. 시간 간격을 5초로 했기 때문에 5초 뒤에 WM_TIMER 메시지가 발생할 것이다. 그리고 Connect 함수를 호출하였는데, 이 부분에서 블록된 상태로 있을 것이다. 그래서 5초가 지나도 Connect 함수가 반환하지 않고 블록되어 있기 때문에 5초 뒤에 WM_TIMER 메시지가 발생하게 된다. 블록된 상태에서 이렇게 메시지가 발생하면 OnMessagePending 함수가 바로 호출되게 된다. 그래서 그 메시지 안에서 WM_TIMER 메시지가 발생했고 ID가 10번임을 확인하고는 CancelBlockingCall을 호출하여 Connect 함수를 취소하고 Close 함수를 호출하여 소켓을 닫아버리게 되는 것이다. CancelBlockingCall을 호출하면 블록되어 있던 Connect 함수는 FALSE를 반환하게 된다. 참고로 Send나 Receive 함수의 경우엔 CancelBlockingCall을 호출하면 SOCKET_ERROR를 반환하게 된다. 어쨌든 이런 방식으로 동기화 소켓에서의 타임아웃을 처리하는 것이다.

Connect 함수에 대한 타임아웃만 예로 보였지만 Send, Receive 함수 모두 같은 방식으로 처리하면 된다.

위의 프로그램을 컴파일해서 실행해 보면 다음과 같이 버튼을 누른뒤 5초 뒤에 타임아웃이 되는 것을 볼 수 있을 것이다.



참고로 버튼을 누르고 나서 바로 윈도우를 이동시키려 하면 움직여지지 않을 것이다. 이것은 Connect 함수에서 블록되어 있기 때문이다. 소켓을 통해 데이터 송수신을 바쁘게 하는 도중에도

---

[1] SetTimer 함수로 타이머를 설치했을 때 첫 WM_TIMER 메시지는 함수를 처음 호출했을 때가 아니라, SetTimer를 호출하고 지정한 시간이 지난 다음에 발생한다. 타이머는 한번 설치해놓으면 지정한 시간 간격으로 계속 WM_TIMER 메시지를 발생시키므로 사용했으면 바로 KillTimer 함수로 파괴해주어야 한다.

[2] SetTimer의 2번째 인자는 WM_TIMER 메시지가 발생할 시간 간격인데 단위가 ms(밀리 세컨드, 1000분의 1초)이다. 따라서 5초 간격의 WM_TIMER 메시지를 발생시키려면 5000을 넘겨주면 된다. 참고로 윈도우 95/98/ME에서 최소 시간 간격은 55ms이고, 윈도우 NT/2k/XP에서는 10ms이다. 시간 간격에 한계가 있음을 참고로 알아두자.

윈도우 이동이라든가 최대/최소화 등이 부드럽게 되도록 하려면 데이터 송수신 작업을 쓰레드로 만들어야 한다. 이것은 나중에 다시 알아보도록 한다.