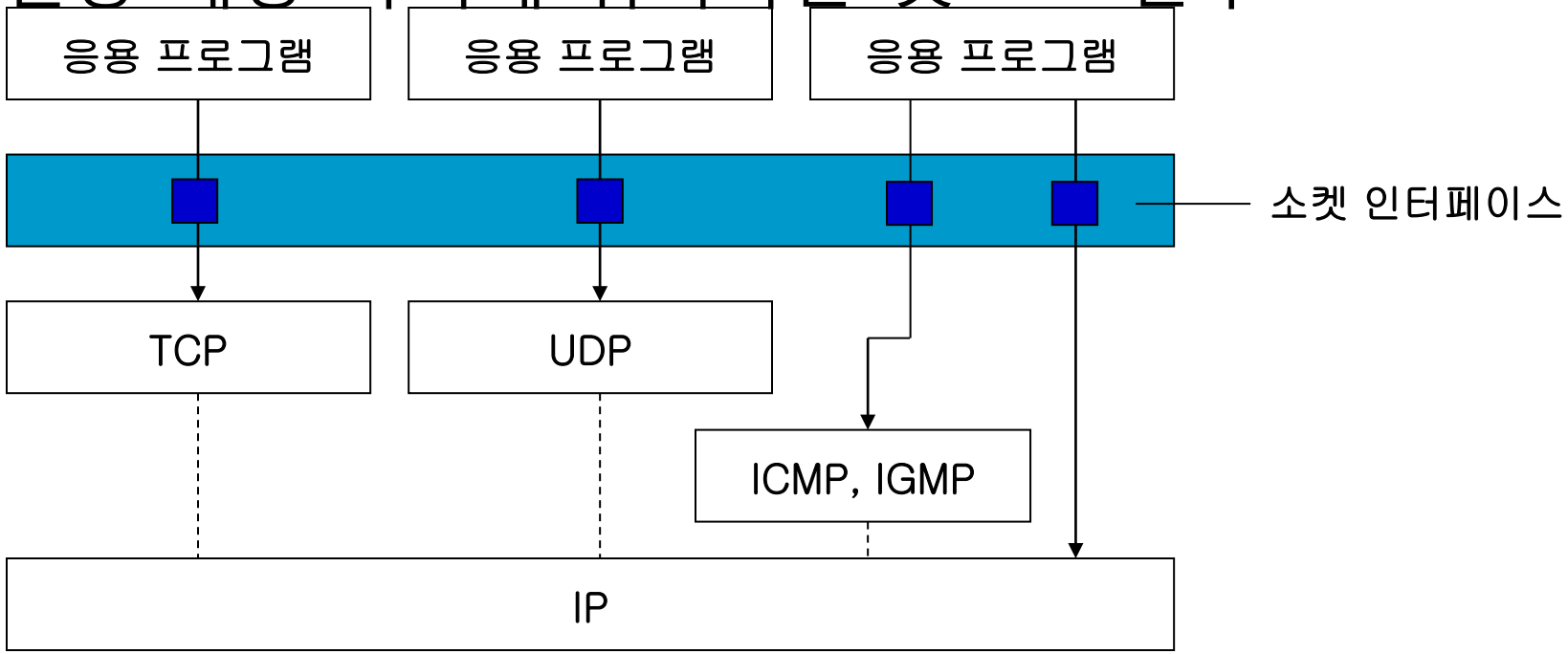
소켓프로그래밍 시작하기

편집: 김혜영

소켓의 개념

- 네트워크 프로그래밍 인터페이스
  - 통신 양단이 모두 소켓을 사용할 필요는 없음
  - TCP/IP 프로토콜에서 (일반적으로) 응용 계층과 전송 계층 사이에 위치하는 것으로 간주



윈도우 소켓 (1)

- 윈도우 소켓(윈속)
  - 버클리 유닉스에서 개발한 네트워크 프로그래밍 인터페이스를 윈도우 환경에서 사용할 수 있게 만든 것
  - 윈도우 95 버전부터 API에 정식으로 포함하여 제공

윈도우 소켓 (2)

- 윈도우 소켓과 유닉스 소켓의 차이점
  - 윈도우 소켓은 DLL을 통해 대부분의 기능이 제공되므로 DLL 초기화와 종료 작업을 위한 함수가 필요
  - 윈도우 프로그램은 대개 GUI를 갖추고 메시지 구동 방식으로 동작하므로 이를 위한 확장 함수가 존재
  - 윈도우는 운영체제 차원에서 멀티스레드를 지원하므로 멀티스레드 환경에서 안정적으로 동작하는 구조와 이를 위한 함수가 필요

윈도우 소켓 (3)

- 윈도우 운영체제의 윈속 지원

운영체제	윈속 버전
윈도우 95	1.1 (2.2)
윈도우 98/Me, 윈도우 NT/2000/XP/2003 서버, 윈도우 비스타/2008 서버/7	2.2
윈도우 CE	1.1 (2.2)

- 윈속에서 지원하는 통신 프로토콜
  - TCP/IP(윈도우 95 이상, 윈도우 CE 2.1 이상)
  - IPv6(윈도우 XP SP1 이상, 윈도우 CE .NET 4.1 이상)
  - IrDA(윈도우 98 이상, 모든 윈도우 CE 버전)
  - Bluetooth(윈도우 XP SP2 이상, 윈도우 CE .NET 4.0 이상)

윈도우 소켓 (4)

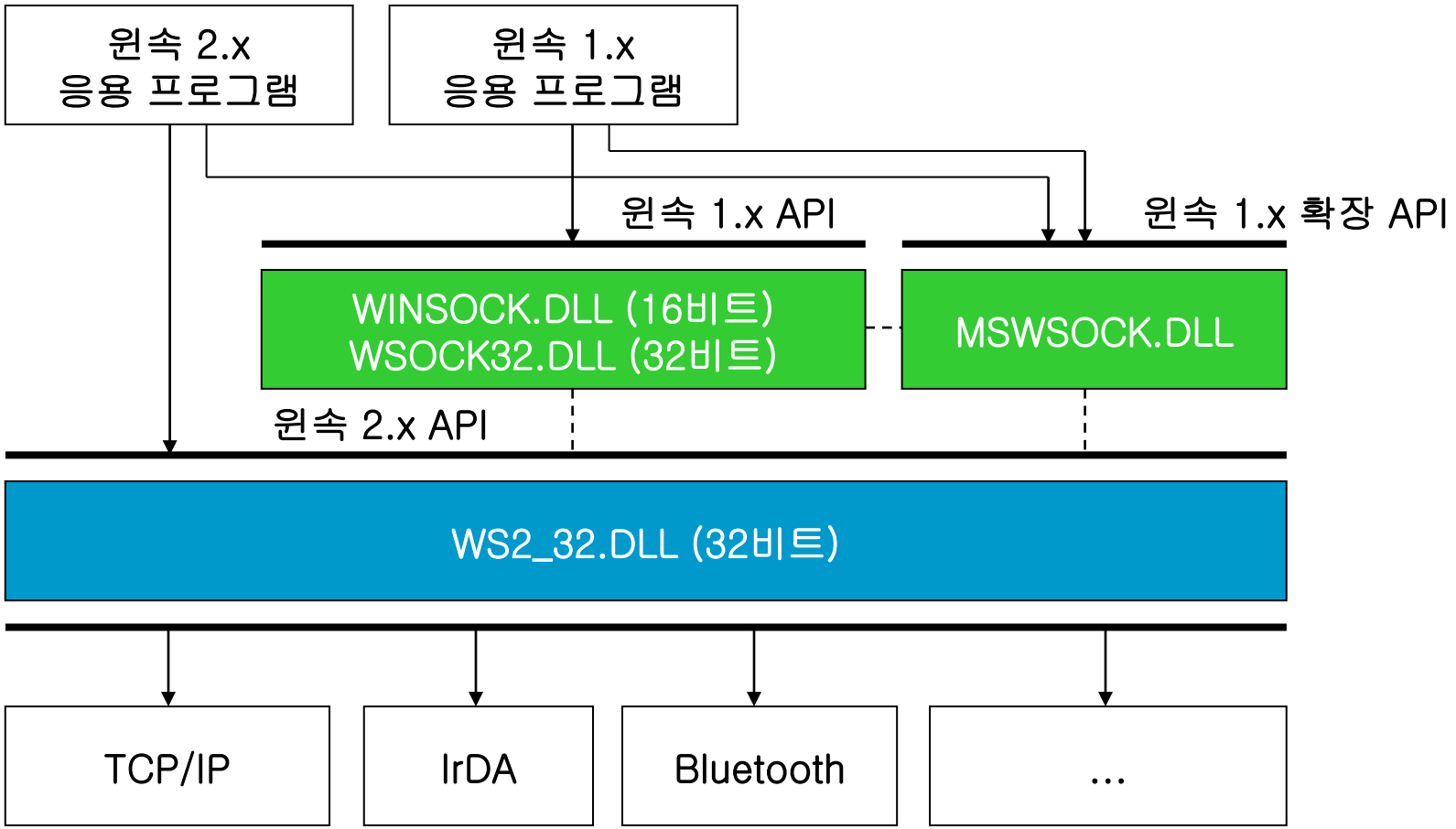
- 윈속의 장점
  - 유닉스 소켓과 소스 코드 수준에서 호환성이 높으므로 기존 코드를 이식하여 활용하기 쉬움
  - 가장 널리 사용하는 네트워크 프로그래밍 인터페이스이므로 한번 배우면 여러 운영체제(윈도우, 리눅스 등)에서 사용 가능
  - TCP/IP 외의 프로토콜도 지원하므로 최소 코드 수정으로 응용 프로그램이 사용할 프로토콜 변경 가능
  - 비교적 저수준 프로그래밍 인터페이스이므로 세부 제어가 가능하며 고성능 네트워크 프로그램 개발 가능

윈도우 소켓 (5)

- 윈속의 단점
  - 응용 프로그램 수준의 프로토콜을 프로그래머가 직접 설계해야 함
    - 주고받는 데이터 형식이나 전송 절차 등을 고려해 프로그래밍해야 하며, 설계 변경 시에는 코드 수정이 불가피함
  - 서로 다른 바이트 정렬 방식을 사용하거나 데이터 처리 단위가 서로 다른 호스트끼리 통신할 경우, 응용 프로그램 수준에서 데이터 변환을 처리해야 함

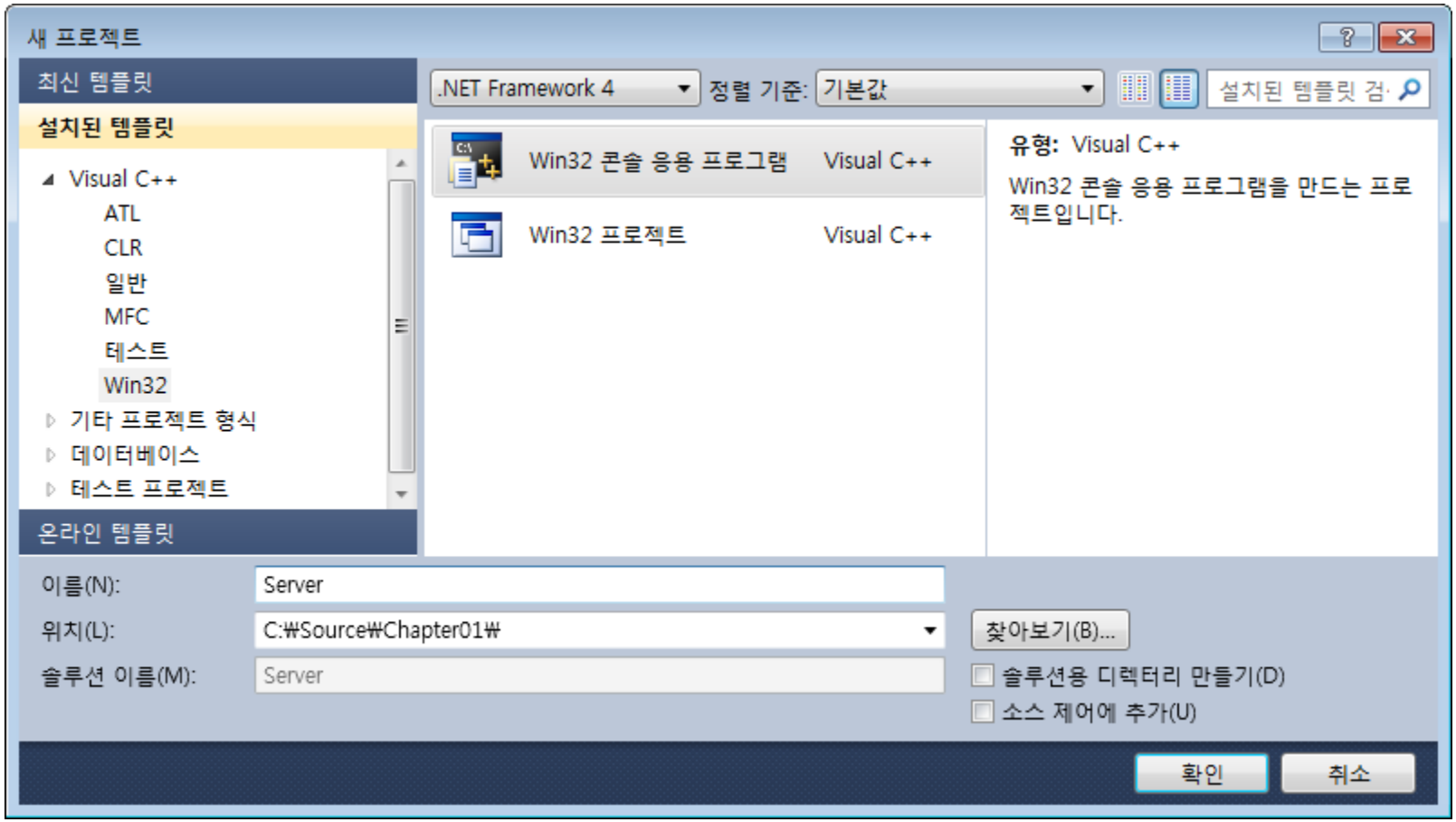
윈도우 소켓 (6)

- 윈속 구조


윈속 2.x
응용 프로그램
윈속 1.x
응용 프로그램
윈속 1.x API
윈속 1.x 확장 API
WINSOCK.DLL (16비트)
WSOCK32.DLL (32비트)
MSWSOCK.DLL
윈속 2.x API
WS2_32.DLL (32비트)
TCP/IP
IrDA
Bluetooth
...

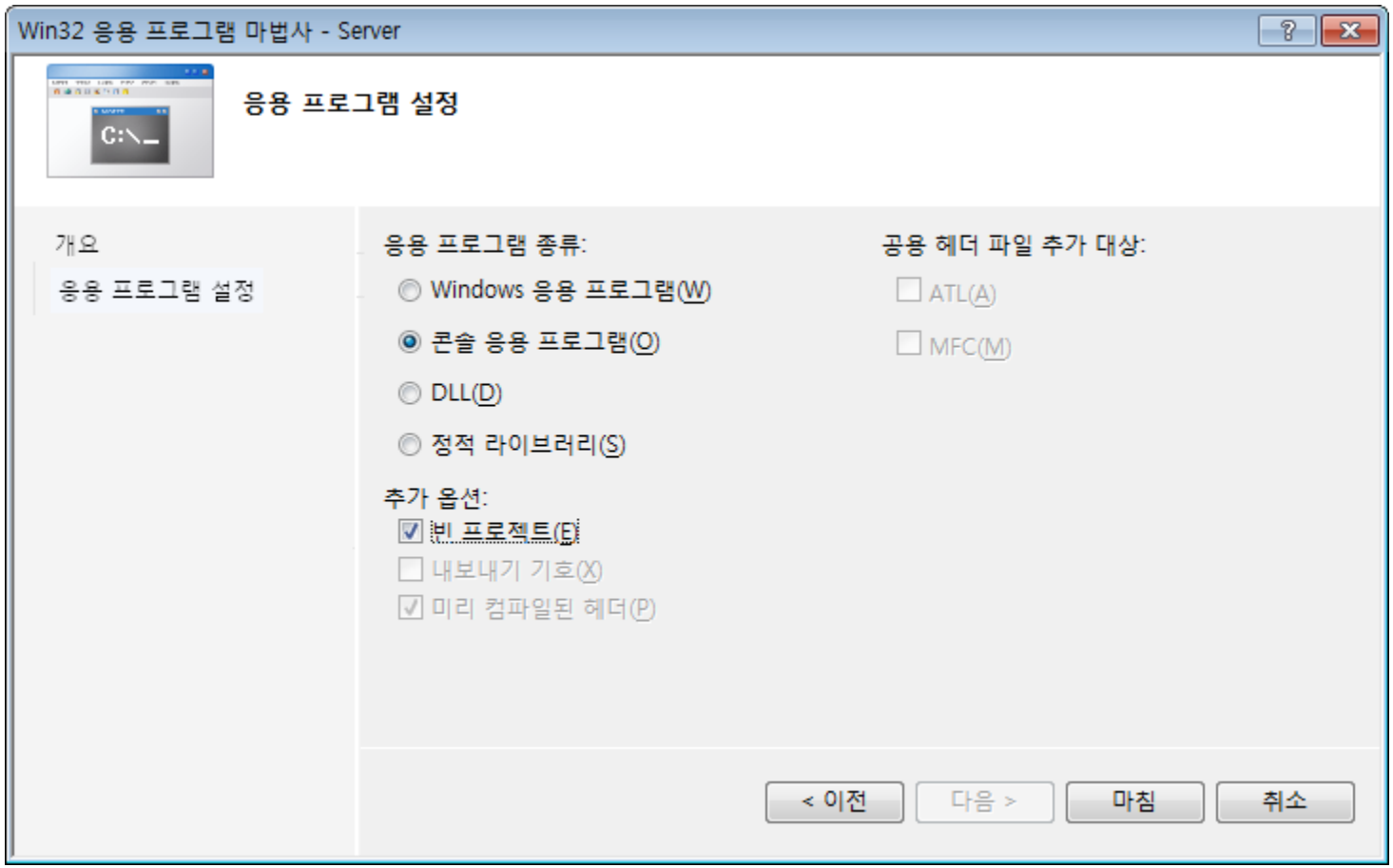
윈도우 소켓 프로그램 맛보기 (1)

- 프로젝트 생성



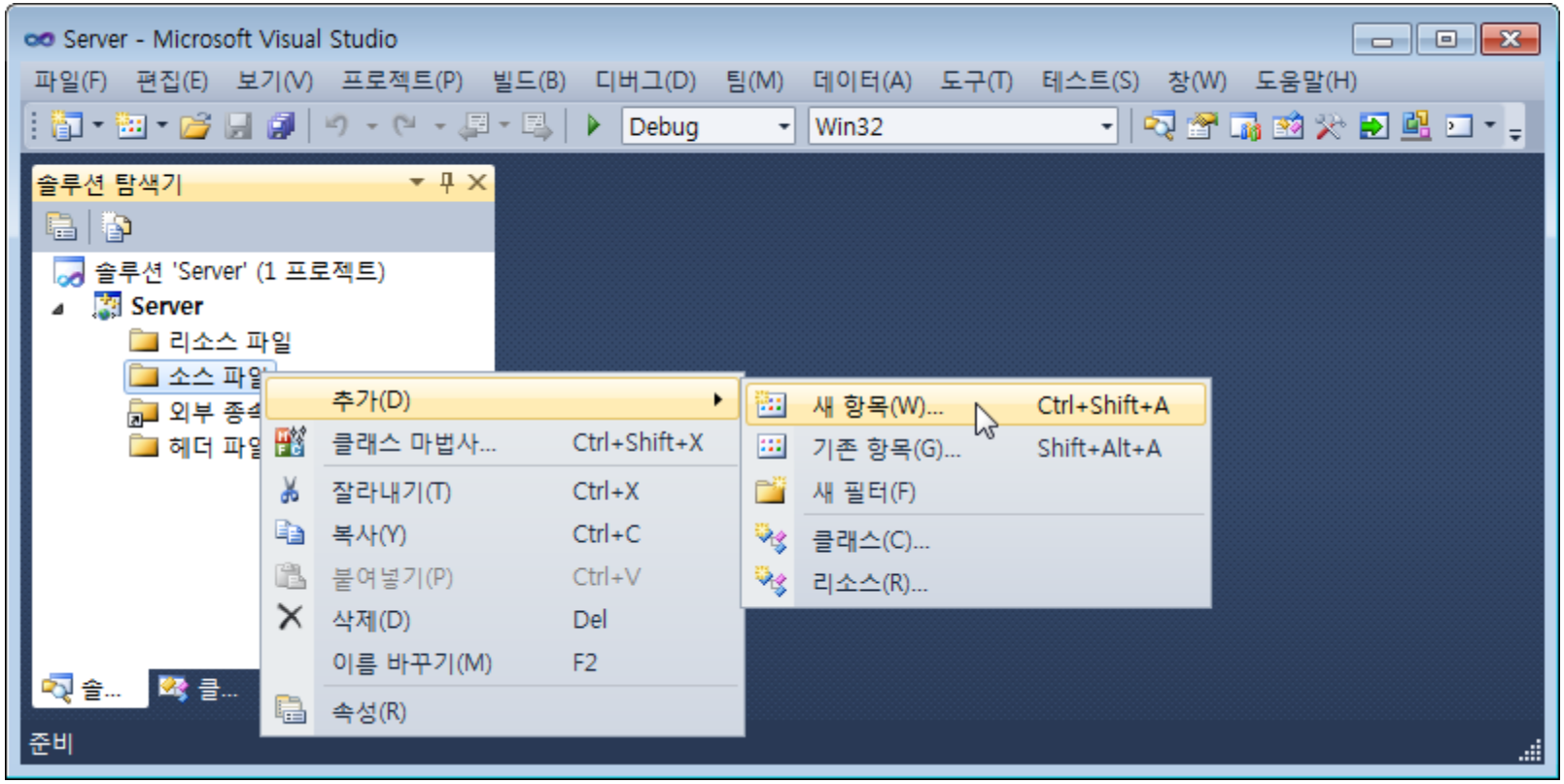
윈도우 소켓 프로그램 맛보기 (2)

- 설정 변경



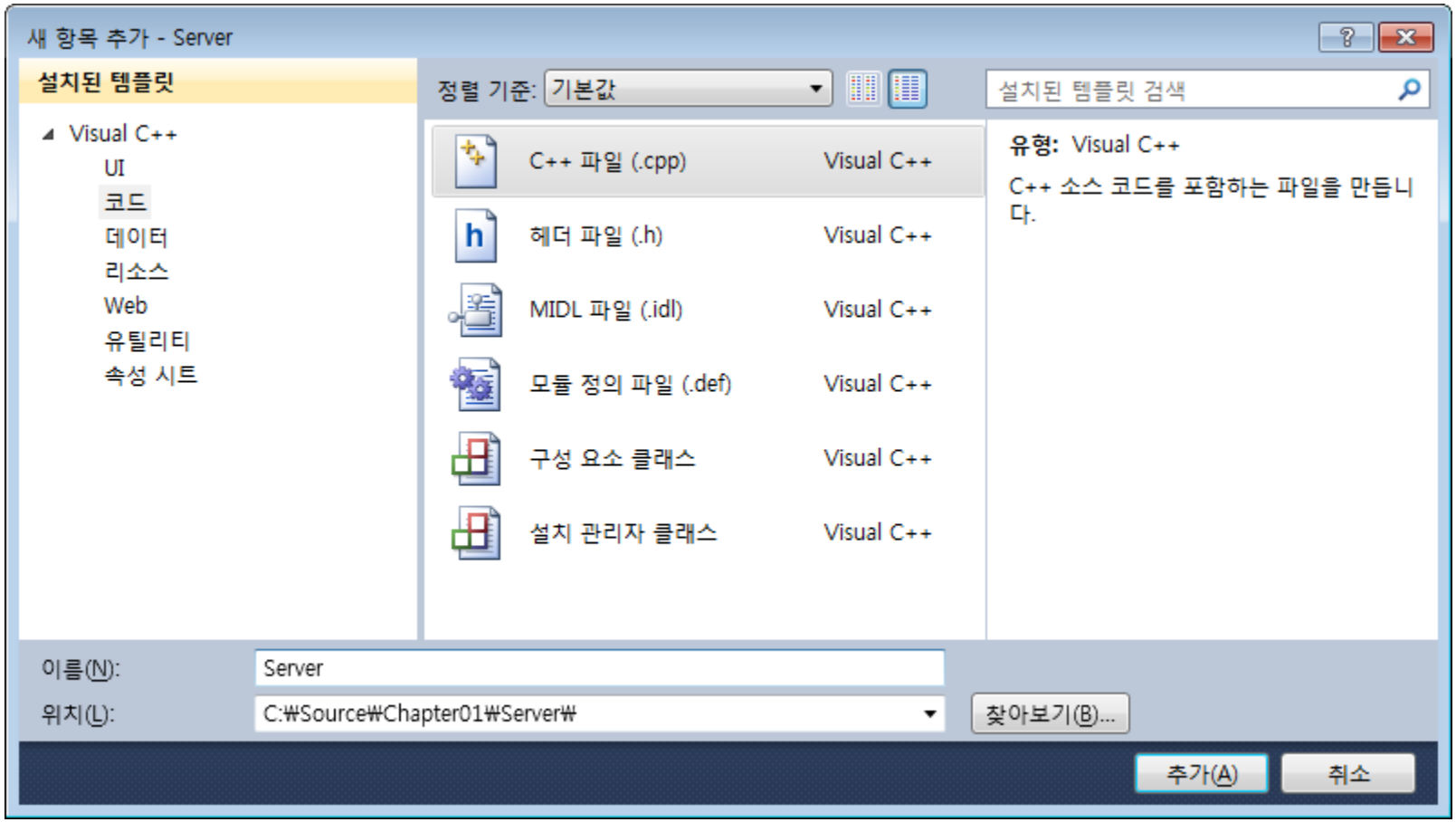
윈도우 소켓 프로그램 맛보기 (3)

- 소스 파일 추가 (1/2)



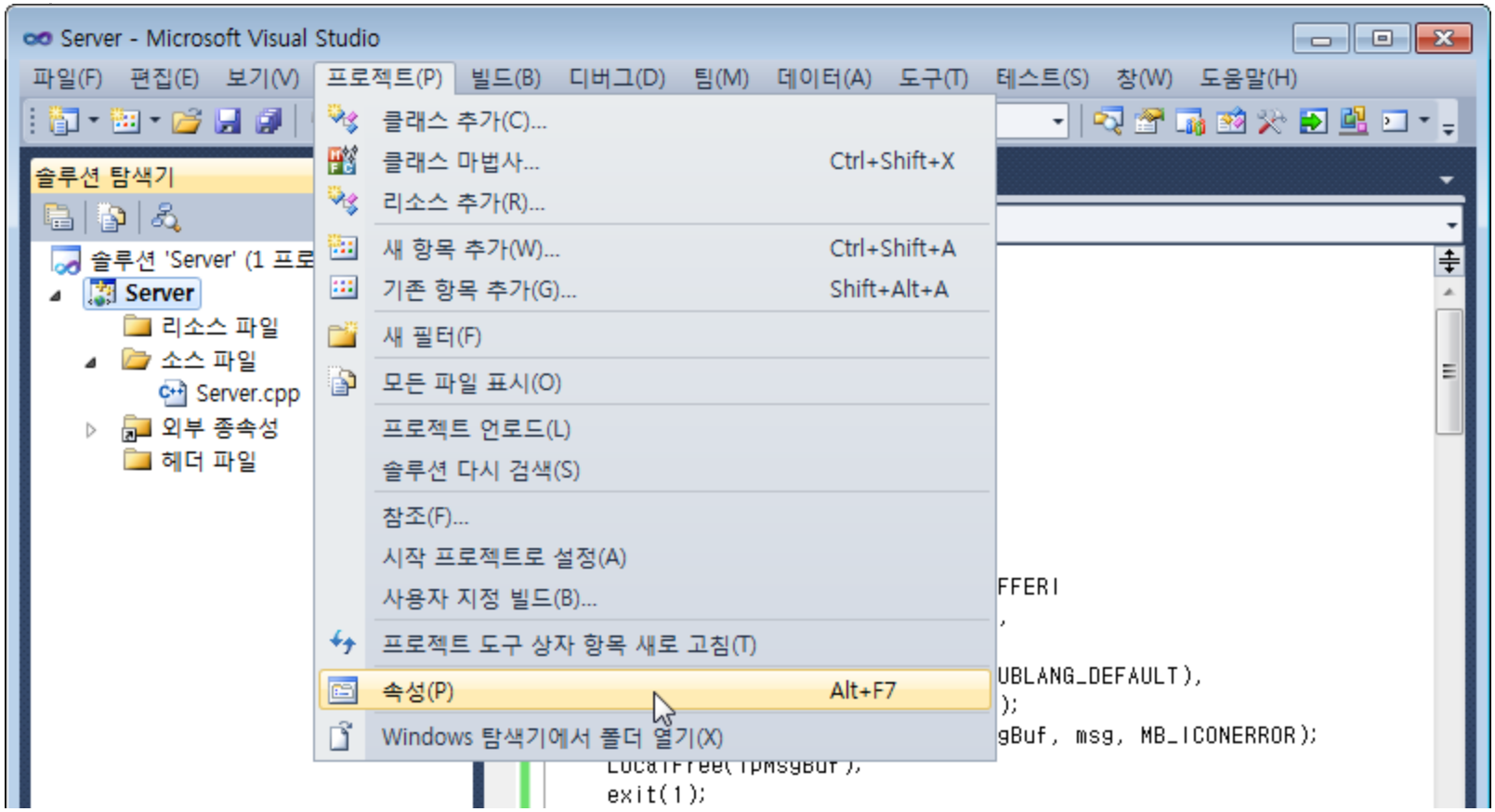
윈도우 소켓 프로그램 맛보기 (4)

- 소스 파일 추가 (2/2)



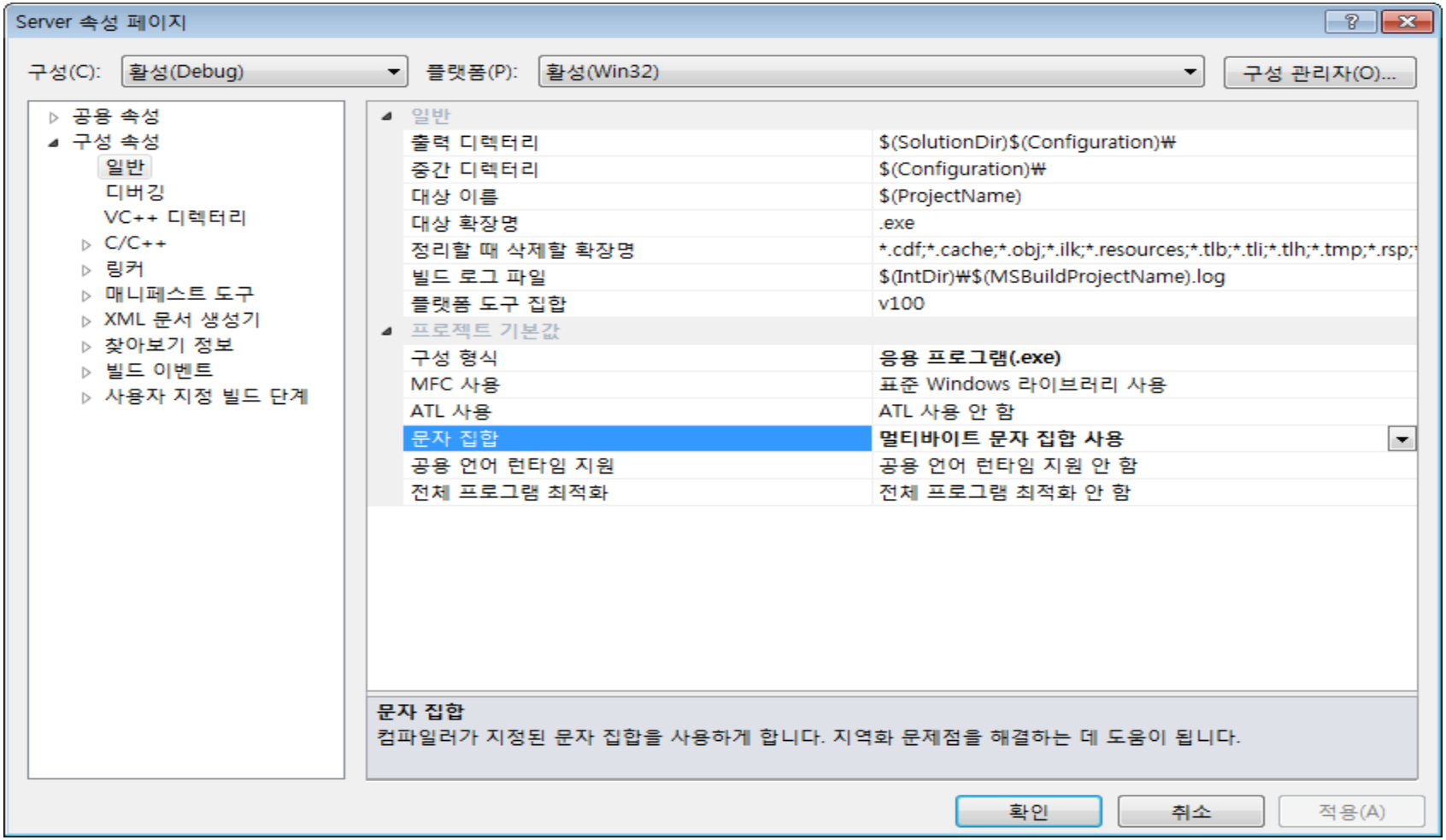
윈도우 소켓 프로그램 맛보기 (5)

- 문자 집합 변경과 윈속 라이브러리 추가



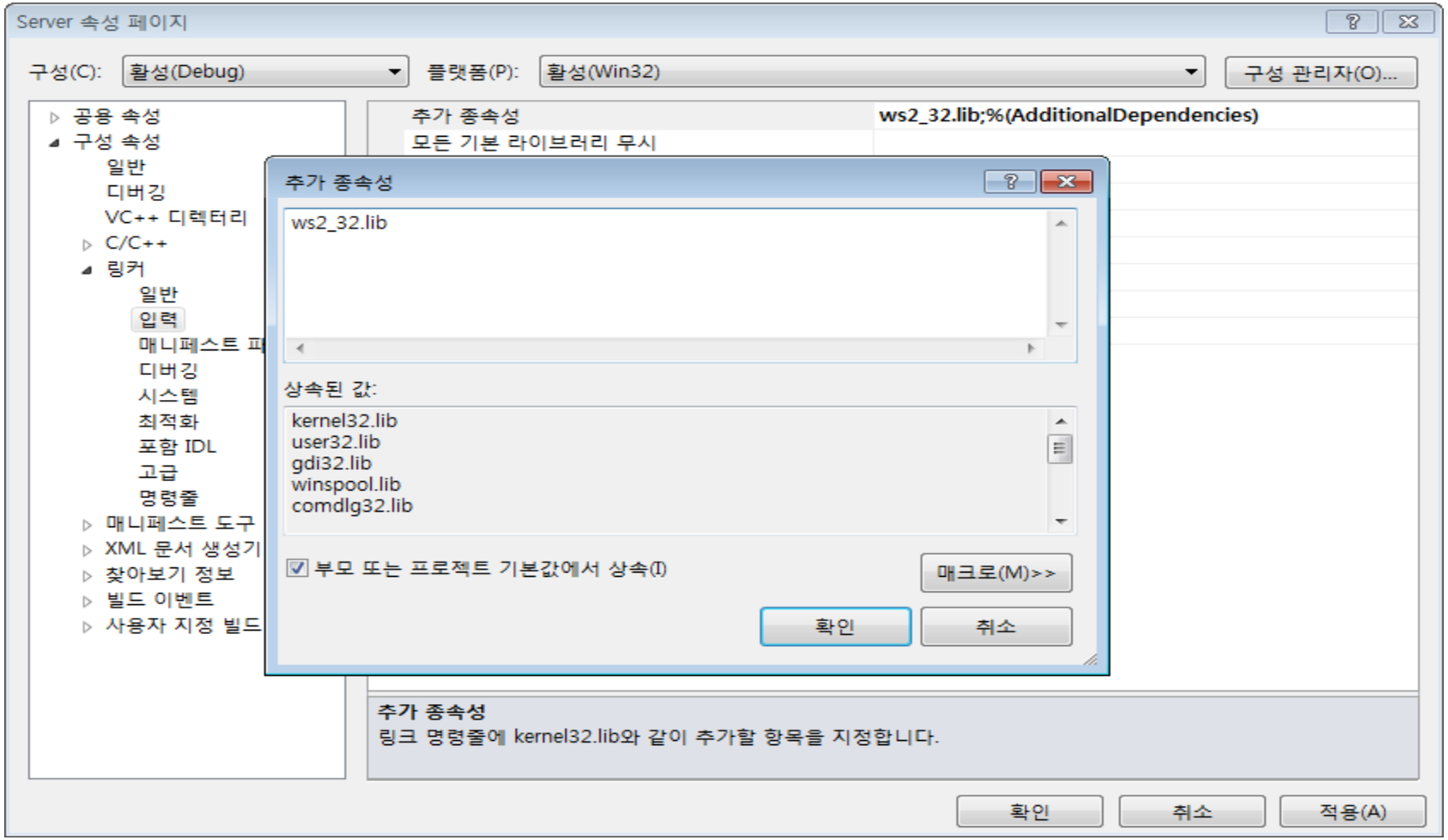
윈도우 소켓 프로그램 맛보기 (6)

- 문자 집합 변경과 윈속 라이브러리 추가



윈도우 소켓 프로그램 맛보기 (7)

- 문자 집합 변경과 윈속 라이브러리 추가

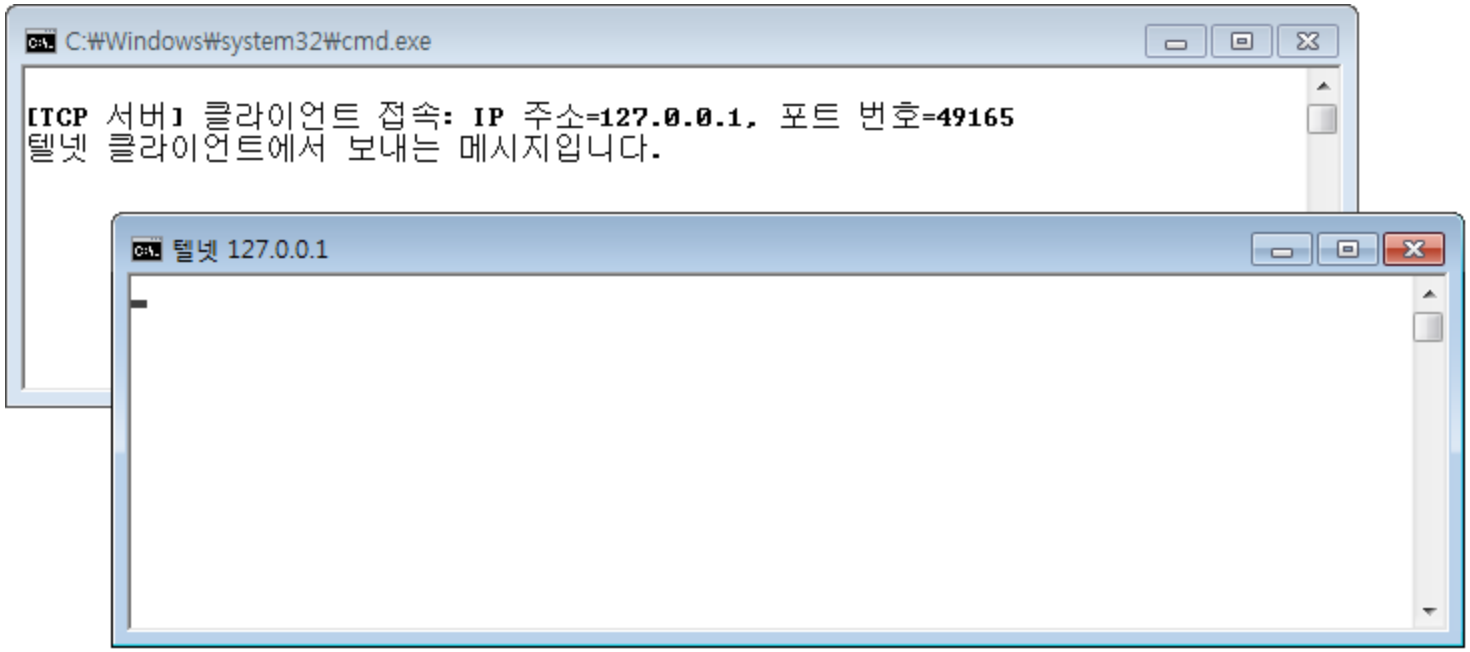


윈도우 소켓 프로그램 맛보기 (8)

- 문자 집합 변경과 윈속 라이브러리 추가 (4/4)
  - 비주얼 C++ 6.0을 사용하는 경우에는 [Project]→[Settings]→[Link]→[Object/library modules] 부분에"ws2_32.lib"를 입력한다.
  - 비주얼 C++ 버전에 따라 윈속 라이브러리를 추가하는 방식이 달라서 번거롭다면 소스 코드의 임의 위치에 #pragma comment(lib,"ws2_32") 한 줄을 넣으면 된다. 2장 이후의 모든 코드는 이 방식을 사용한다.

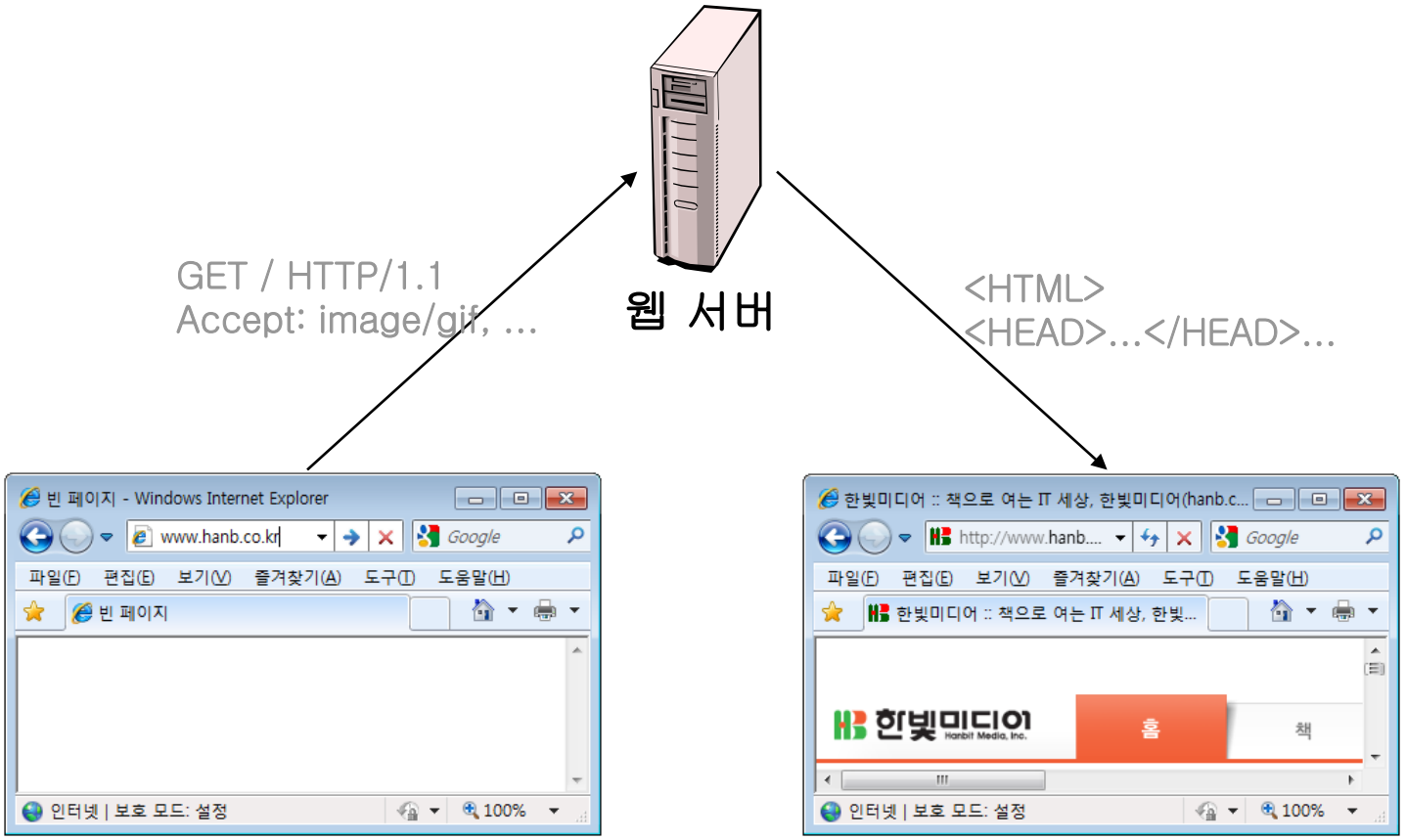
윈도우 소켓 프로그램 맛보기 (9)

- 실행 화면



TCP 서버-클라이언트 개념 (1)

- TCP 서버-클라이언트 동작



웹 클라이언트

윈속 함수 오류 처리 (1)

- 오류 처리 유형
  - ① 오류를 처리할 필요가 없는 경우
    - 리턴값이 없거나 호출 시 항상 성공하는 일부 소켓 함수
  - ② 리턴 값만으로 오류를 처리하는 경우
    - WSAStartup() 함수
  - ③ 리턴 값으로 오류 발생을 확인하고, 구체적인 내용은
    오류 코드로 확인하는 경우
    - 대부분의 소켓 함수

윈속 함수 오류 처리 (2)

- 오류 코드 얻기

```
int WSAGetLastError(void);
```

- 사용 예

```
if (소켓함수(...) == 실패) {
    int errcode = WSAGetLastError();
    printf(errcode에 해당하는 오류 메시지);
}
```

오류 코드를 문자열로 바꾸기 (1)

- FormatMessage() 함수

```
DWORD FormatMessage (
    DWORD dwFlags, // 옵션
    LPCVOID lpSource, // NULL
    DWORD dwMessageId, // 오류 코드
    DWORD dwLanguageId, // 언어
    LPTSTR lpBuffer, // 오류 문자열 시작 주소
    DWORD nSize, // 0
    va_list* Arguments // NULL
) ;
```

성공: 오류 메시지의 길이, 실패: 0

오류 코드를 문자열로 바꾸기 (2)

- err_quit() 함수 정의

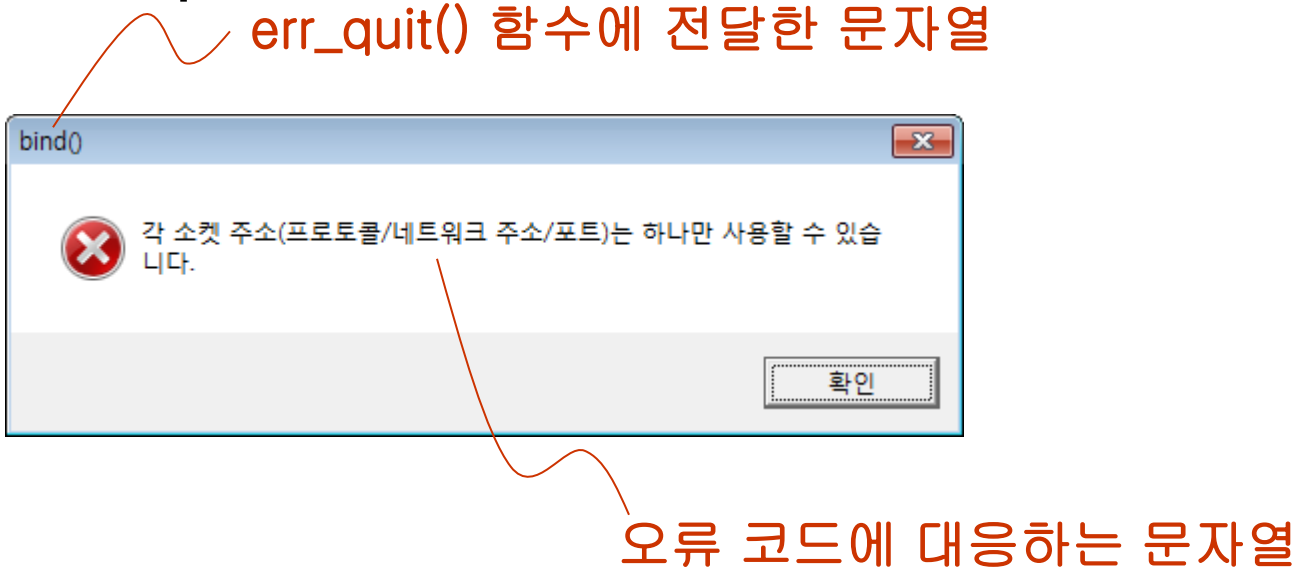
```
void err_quit(char *msg)
{
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER
         | FORMAT_MESSAGE_FROM_SYSTEM,
        NULL, WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        (LPTSTR)&lpMsgBuf, 0, NULL);
    MessageBox(NULL, (LPCTSTR)lpMsgBuf, msg, MB_ICONERROR);
    LocalFree(lpMsgBuf);
    exit(1);
}
```

오류 코드를 문자열로 바꾸기 (3)

- err_quit() 함수 사용 예

```
if(socket(...) == INVALID_SOCKET) err_quit("socket()");
if(bind(...) == SOCKET_ERROR) err_quit("bind()");
```

- err_quit() 함수의 오류 메시지



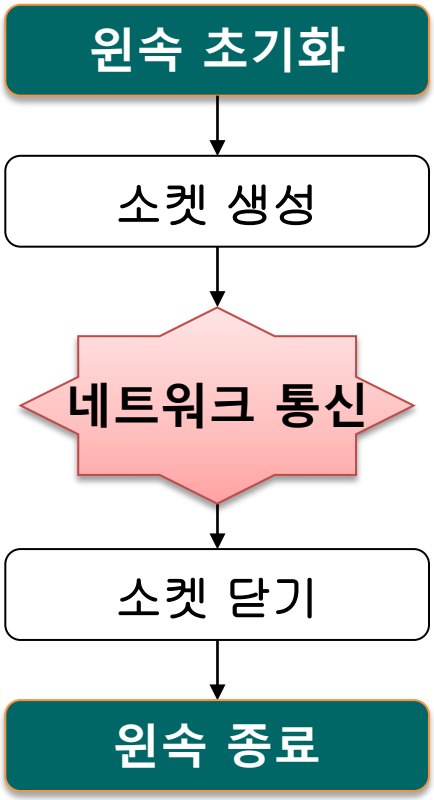
오류 코드를 문자열로 바꾸기 (4)

- err_display() 함수 정의

```
void err_display(char *msg)
{
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER
         | FORMAT_MESSAGE_FROM_SYSTEM,
        NULL, WSAGetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        (LPTSTR)&lpMsgBuf, 0, NULL);
    printf("[%s] %s", msg, (char *)lpMsgBuf);
    LocalFree(lpMsgBuf);
}
```

윈속 초기화와 종료 (1)

- 윈속 응용 프로그램의 공통 구조



윈속 초기화와 종료 (2)

- 윈속 초기화

```
int WSAStartup (
    WORD wVersionRequested,
    LPWSADATA lpWSAData
);
```

성공: 0, 실패: 오류 코드

  - wVersionRequested
    - 프로그램이 요구하는 최상위 윈속 버전. 하위 8비트에 주 버전을, 상위 8비트에 부 버전을 넣어서 전달
  - lpWSAData
    - 윈도우 운영체제가 제공하는 윈속 구현에 관한 정보를 얻을 수 있음(거의 사용 안 함)

윈속 초기화와 종료 (3)

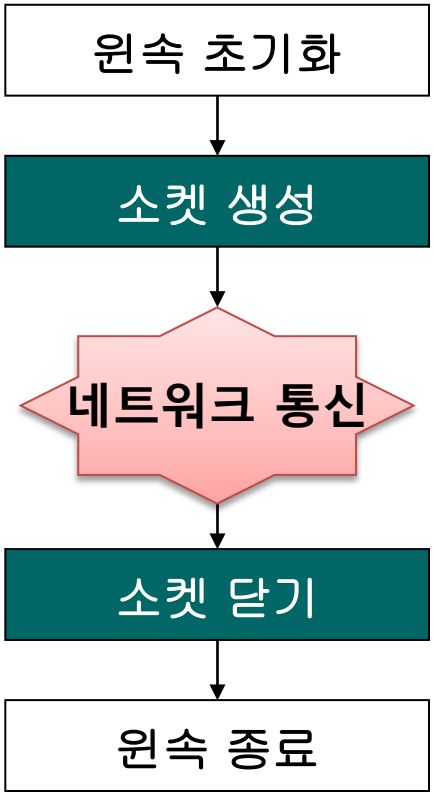
- 윈속 종료

int WSACleanup(void);

성공: 0, 실패: SOCKET_ERROR

소켓 생성과 닫기 (1)

- 윈속 응용 프로그램의 공통 구조



소켓 생성과 닫기 (2)

- 소켓 생성

```
SOCKET socket (
    int af, // 주소 체계
    int type, // 소켓 타입
    int protocol // 프로토콜
);
```

성공: 새로운 소켓, 실패: **INVALID_SOCKET**

  - 사용자가 요청한 프로토콜을 사용해 통신할 수 있도록 내부적으로 리소스를 할당하고, 이에 접근할 수 있는 일종의 핸들 값(SOCKET 타입, 32비트 정수)인 소켓 디스크립터(socket descriptor)를 리턴

소켓 생성과 닫기 (3)

- 주소 체계

```
...
#define AF_INET 2     // Internetwork: UDP, TCP, etc.
...
#define AF_INET6 23   // Internetwork Version 6
...
#define AF_IRDA 26    // IrDA
...
#define AF_BTH 32     // Bluetooth RFCOMM/L2CAP protocols
...
```

소켓 생성과 닫기 (4)

- 소켓 타입
  - 사용할 프로토콜의 특성

소켓 타입	특성
SOCK_STREAM	신뢰성 있는 데이터 전송 기능 제공, 연결형 프로토콜
SOCK_DGRAM	신뢰성 없는 데이터 전송 기능 제공, 비연결형 프로토콜

- TCP와 UDP 프로토콜 사용을 위한 설정(1)

사용할 프로토콜	주소 체계	소켓 타입
TCP	AF_INET 또는 AF_INET6	SOCK_STREAM
UDP		SOCK_DGRAM

소켓 생성과 닫기 (5)

- 프로토콜
  - 주소 체계와 소켓 타입이 같더라도 해당 프로토콜이 두 개 이상 존재할 경우 프로토콜을 명시적으로 지정
    - TCP와 UDP 프로토콜 사용을 위한 설정(2)

사용할 프로토콜	주소 체계	소켓 타입	프로토콜
TCP	AF_INET 또는 AF_INET6	SOCK_STREAM	IPPROTO_TCP
UDP		SOCK_DGRAM	IPPROTO_UDP

    - TCP와 UDP 프로토콜 사용을 위한 설정(3)

사용할 프로토콜	주소 체계	소켓 타입	프로토콜
TCP	AF_INET 또는 AF_INET6	SOCK_STREAM	0
UDP		SOCK_DGRAM	0

소켓 생성과 닫기 (6)

- 소켓 닫기

```
int closesocket (
    SOCKET s
);
```

성공: 0, 실패: SOCKET_ERROR

  – 소켓을 닫고 관련 리소스를 반환

네트워크 프로그램의 흐름

- 송신 측

  소켓 생성 -> 포트 부여 -> 상대편 IP/Port 주소로 연결 -> 통신 -> 종료
- 수신 측

  소켓 생성 -> 포트 부여 -> 상대편 연결 기다리기 -> 통신 -> 종료

1. 소켓 생성

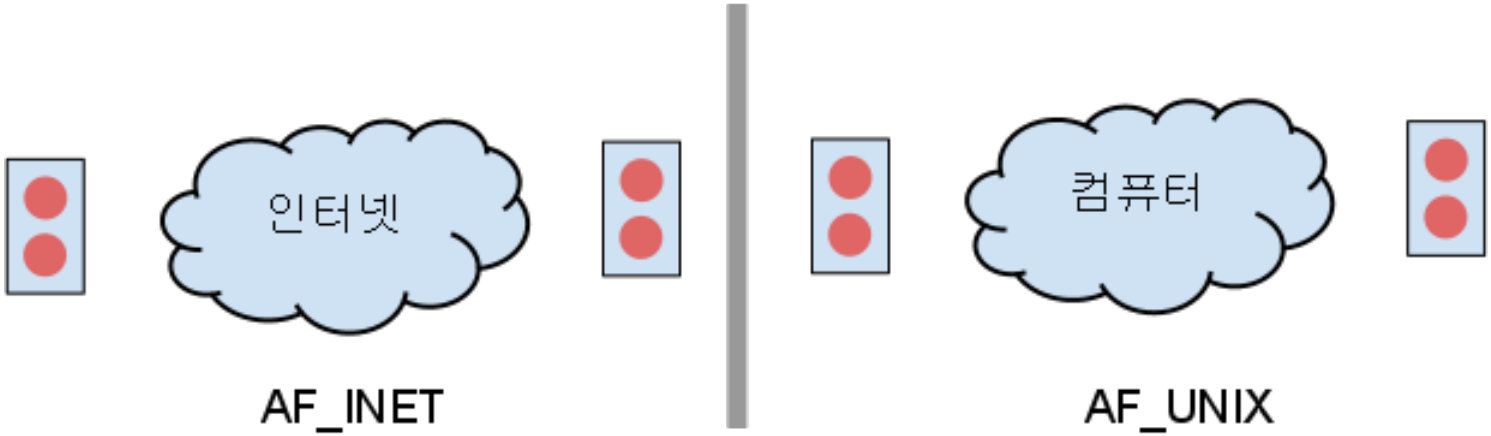
- 인터넷과 연결하기 위한 접점소켓(endpoint socket) 생성

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (int domain, int type, int protocol);
```

- domain : 소켓의 사용 영역을 정의한다.
- type : 소켓 유형을 정의한다.
- protocol : 소켓이 사용할 프로토콜을 정의한다.

1. 소켓 생성

- domain : 소켓이 사용되는 네트워크의 영역을 정의



TYPE	설명
AF_UNIX	프로세스간 통신(IPC)용
AF_INET	일반 TCP/IP 인터넷 통신
AF_IPX	노벨의 IPX
AF_X25	X.25 프로토콜

1. 소켓 생성

Type & Protocol

- Type : 통신에 사용할 패킷의 타입을 지정
- Protocol : 통신에 사용할 프로토콜 지정
- Tyep에 따라서 Protocol이 정해짐.

Type	Protocol
SOCK_STREAM	IPPROTO_TCP
SOCK_DGRAM	IPPROTO_UDP
SOCK_RAW	

- SOCK_STREAM & IPPROTO_TCP : TCP 기반의 통신에 사용
- SOCK_DGRAM & IPPROTO_UDP : UDP 기반의 통신에 사용
- SOCK_RAW & (원하는 프로토콜) : RAW Socket으로 저수준에서 프로토콜을 직접 다룰 때 사용

1. 소켓 생성

socket 함수를 이용한 소켓 생성의 예.

1. TCP 소켓 :
   - socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
2. UDP 소켓
   - socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)

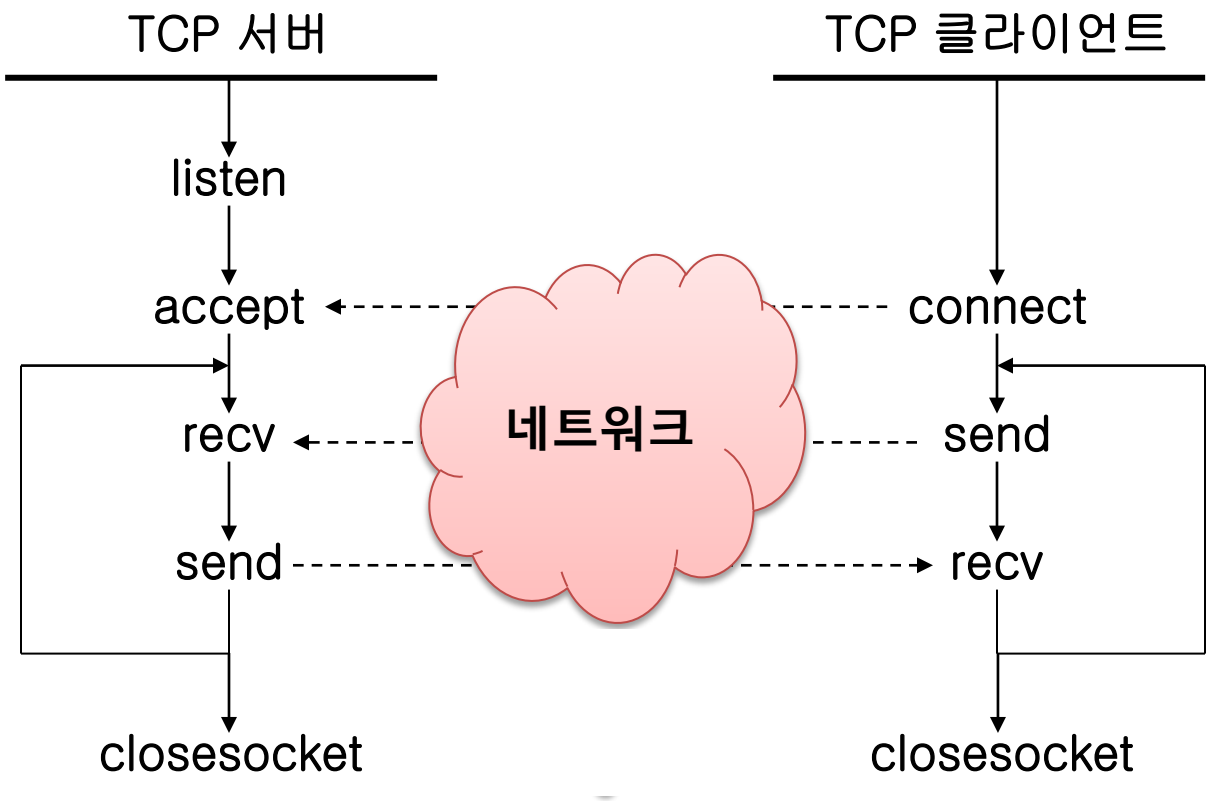
1. 소켓 생성

socket 함수 반환 값.

- 성공적으로 소켓을 만들면 0보다 큰 int 값을 반환
- 소켓지정번호, socket descriptor 라고 부른다.
- 소켓을 지시하며, 이를 이용해서 소켓을 제어한다.

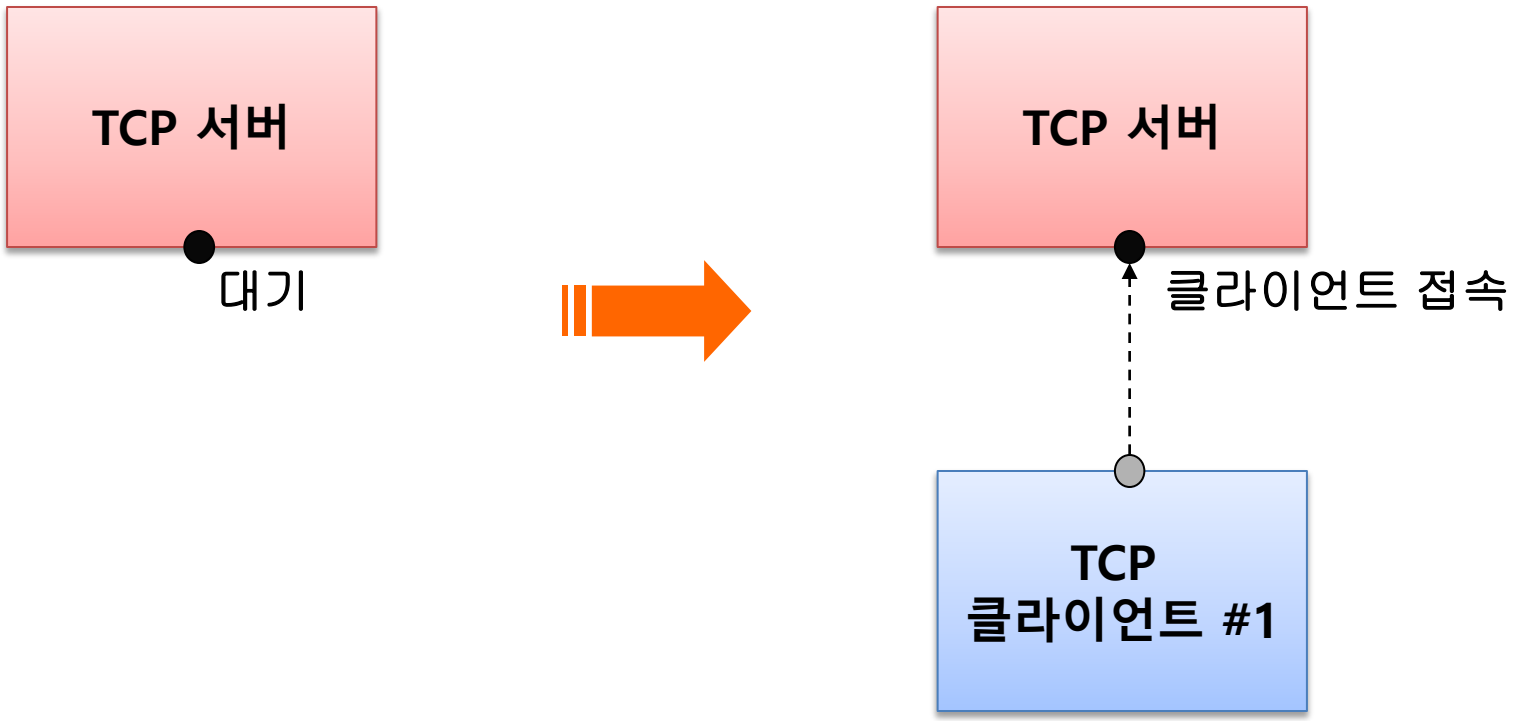
TCP 서버-클라이언트 개념 (2)

- TCP 서버-클라이언트 핵심 동작



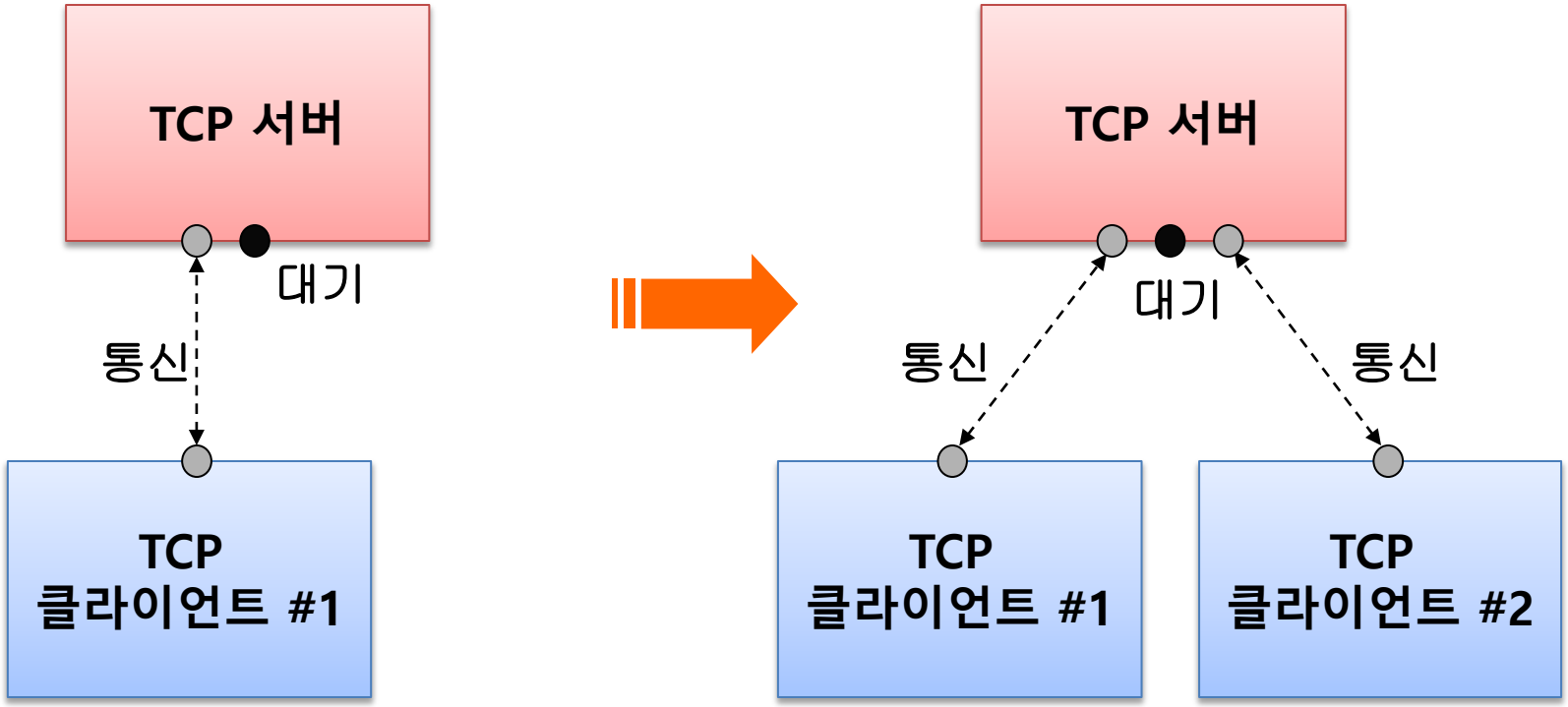
TCP 서버-클라이언트 동작 원리 (1)

- TCP 서버-클라이언트 동작 원리



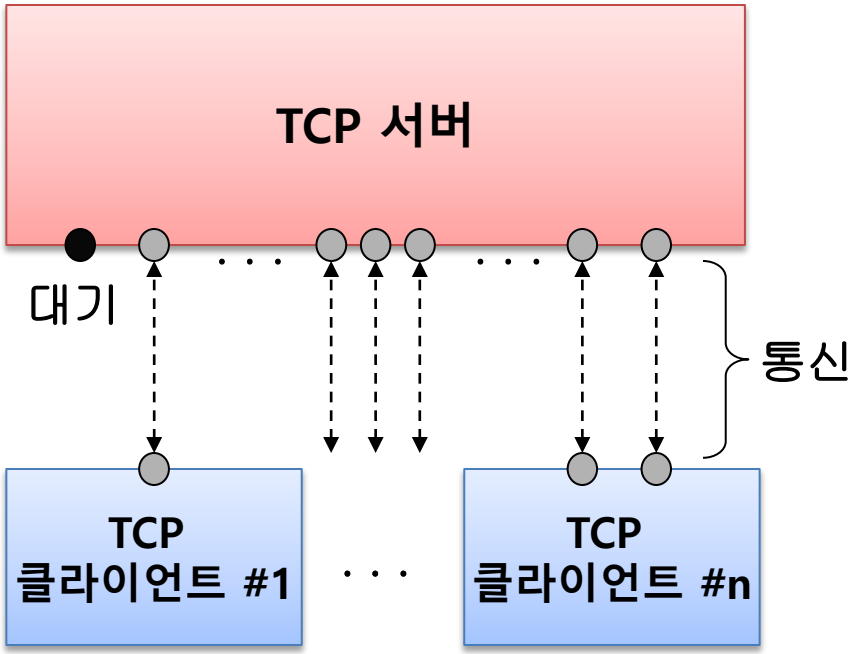
TCP 서버-클라이언트 동작 원리 (2)

- TCP 서버-클라이언트 동작 원리



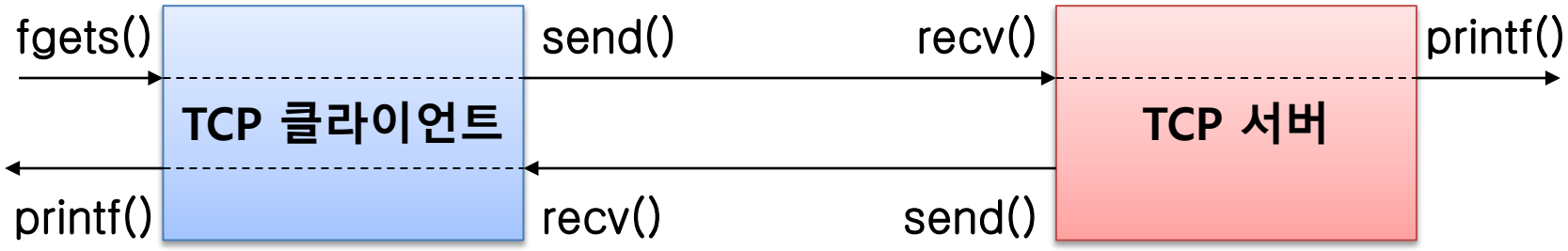
TCP 서버-클라이언트 동작 원리 (3)

- TCP 서버 하나와 여러 TCP 클라이언트의 통신



TCP 서버-클라이언트 실습

- TCP 서버-클라이언트 예제 동작

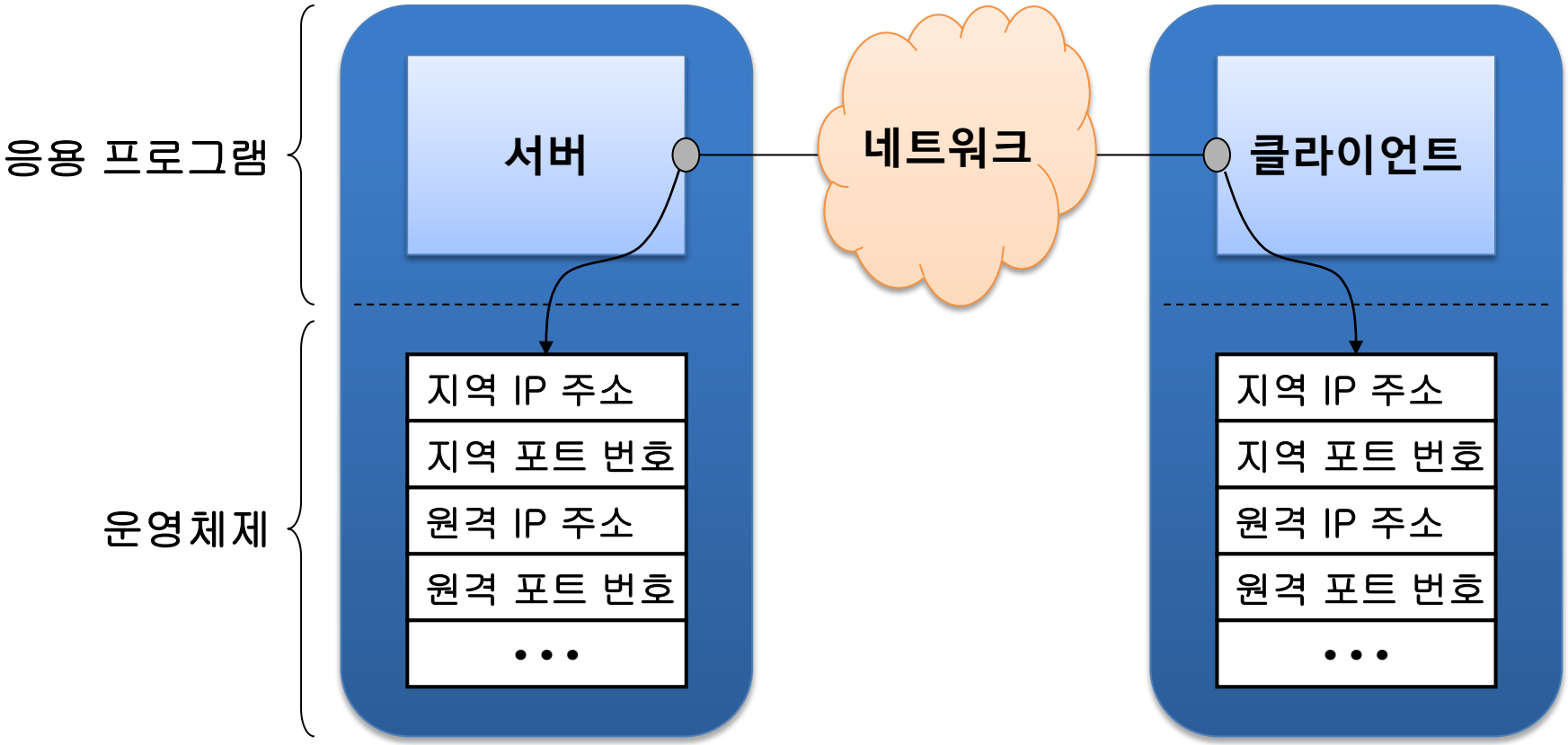


TCP 서버-클라이언트 분석 (1)

- 소켓 통신을 위해 결정해야 할 요소
  - ① 프로토콜
    - 통신 규약. 소켓을 생성할 때 결정
  - ② 지역 IP 주소와 지역 포트 번호
    - 서버 또는 클라이언트 자신의 주소
  - ③ 원격 IP 주소와 원격 포트 번호
    - 서버 또는 클라이언트가 통신하는 상대의 주소

TCP 서버-클라이언트 분석 (2)

- 소켓 데이터 구조체



실습

- 윈도우 소켓 프로그램 시작하기

```
int main(int argc, char *argv[])
{
    //윈속초기화
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
        return 1;
    MessageBox(NULL, L"윈속초기화 성공", L"쪽지", MB_OK);

    //소켓 생성 socket()
    SOCKET tcp_sock = socket(AF_INET, SOCK_STREAM, 0);      //socket( 버전4 , TCP , 0 )
    if (tcp_sock == INVALID_SOCKET) err_quit("socket()");   //소켓이 생성 되지 않고 INVALID_SOCKET 을 반환했을경우 err_quit 함수 실행
    MessageBox(NULL, L"TCP 소켓 생성 성공", L"쪽지", MB_OK); //소켓이 생성 되었다면 MessageBox 출력

    //소켓 종료 closesocket()
    closesocket(tcp_sock);

    //윈속 종료
    WSACleanup();
    return 0;

}
```