WINDOW NETWORK -CHAPTER 1-

SOULSEEK

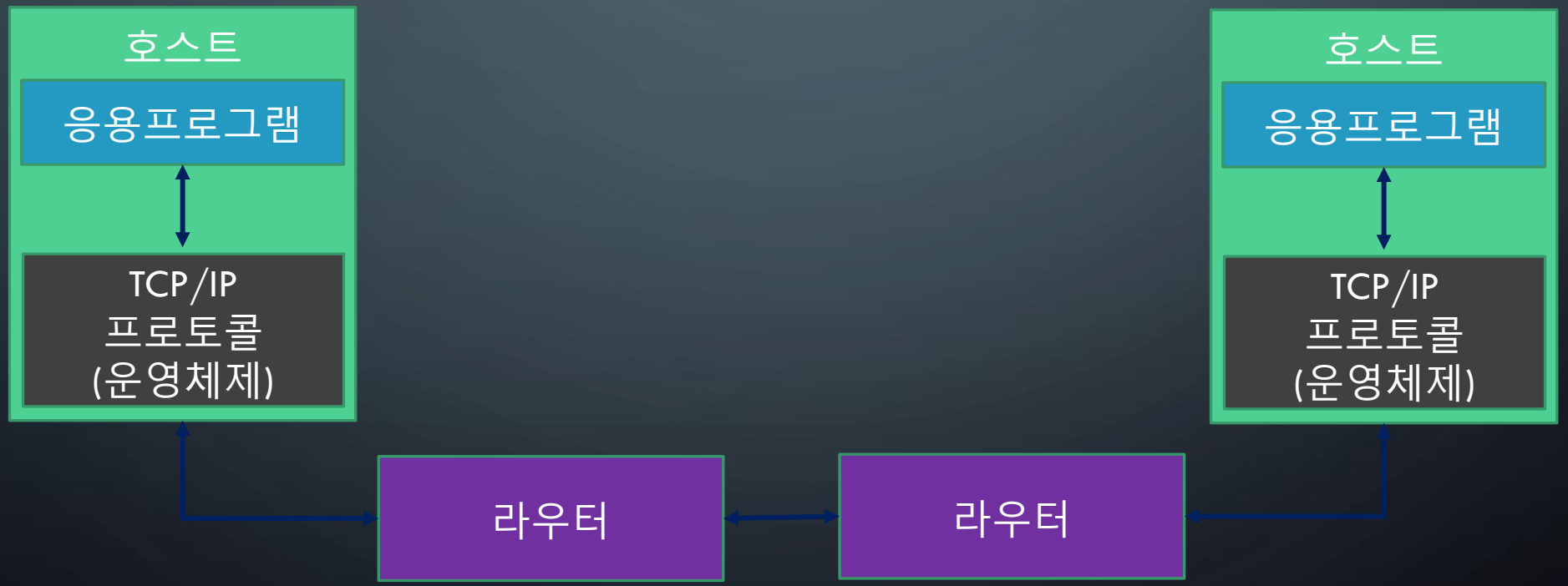
목차

1. NETWORK & SOCKETPROGRAMING

1. NETWORK & SOCKET PROGRAMING

TCP/IP 프로토콜

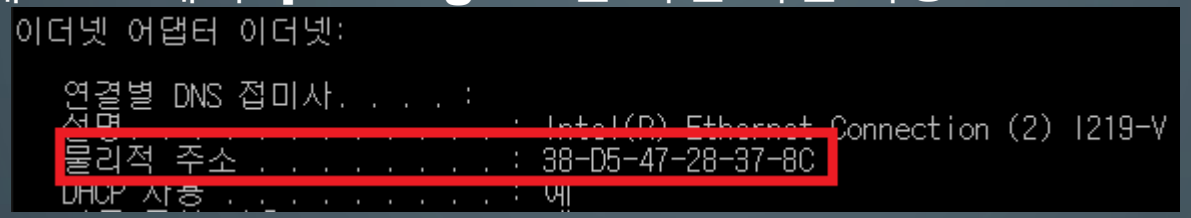
- 인터넷을 통해 통신을 수행하는 개체는 크게 호스트와 라우터로 나눌 수 있다.
- **호스트(host):** 응용 프로그램을 수행하는 주체. ➔ **PC,** 노트북, 휴대전화, **PDA**등.
- **라우터(router):** 호스트에서 생성된 데이터를 여러 네트워크를 거쳐 전송함으로써 서로 다른 네트워크에 속한 호스트 간에 데이터를 교환할 수 있게 하는 장비.
- **통신 프로토콜(communication protocol):** 라우터와 라우터 그리고 호스트와 호스트가 통신하기위해 이행하는 절차와 방법.



1. NETWORK & SOCKET PROGRAMING

TCP/IP 프로토콜의 구조

- **네트워크 접근 계층(Network Access Layer):** 물리적 네트워크를 통한 실제 데이터 송수신 담당. 물리적인 주소 체계 **cmd**에서 **ipconfig /all**을 하면 확인 가능



- **인터넷 계층(Internet Layer):** 네트워크 접근 계층의 도움을 받아 데이터를 목적지 호스트까지 전달(라우팅), **IP**주소 사용 **“192.168.10.100”**
- **전송 계층(Transport Layer):** 최종 통신 목적지를 지정하고 오류 없이 데이터를 전송 **TCP**와 **UDP, Port** 사용 **“9000”**
- **응용 계층(Application Layer):** 전송 계층을 기반으로 한 다수의 프로토콜과 응용 프로그램을 포괄 **FTP, HTTP, SMTP**등

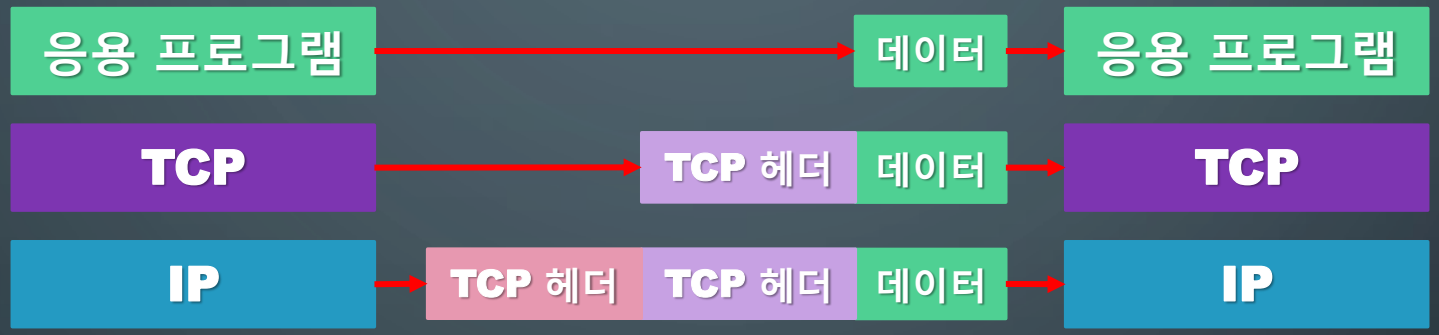
TCP	UDP
연결형 프로토콜	비연결형 프로토콜
신뢰성 있는 데이터 전송	신뢰성 없는 데이터 전송
일대일 통신	일대일, 일대다 통신
데이터 경계 구분 안 함	데이터 경계 구분함

1. NETWORK & SOCKET PROGRAMING

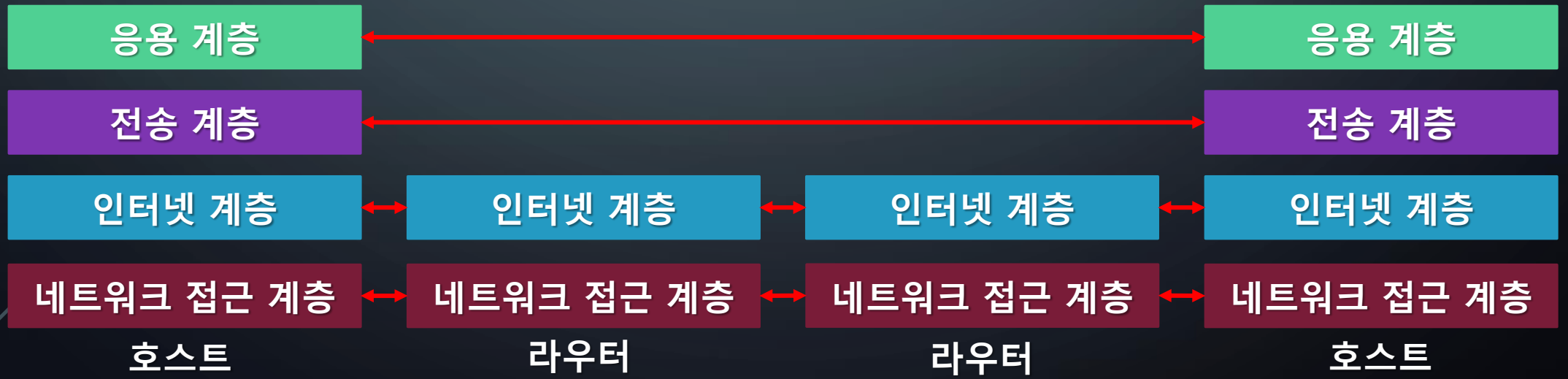
데이터 전송 원리

- 각 프로토콜에서 정의한 **제어정보(IP 주소, PORT 번호, 오류 체크 코드 등)**를 기준으로 송수신 측의 호스트의 응용 프로그램에 정보를 전달한다.
- 제어 정보는 **헤더(header)와 트레일러(trailer)**로 나뉜다.
- 데이터를 이러한 제어정보와 결합한 형태로 전송되며, **패킷(packet)**이라 부른다.

패킷 전송 형태 : 계층별



패킷 전송 형태 : 인터넷



1. NETWORK & SOCKET PROGRAMING

IP주소, PORT번호

IP

- **IPv4, IPv6** 두 종류가 있다.
- 인터넷에 있는 호스트를 구분하는데 사용한다.

PORT

- **IP**가 구분하지 못하는 프로세스를 식별한다.
- **TCP**와 **UDP**는 포트 번호로 부호 없는 **16**비트 정수를 사용한다. **0~65535**범위를 사용가능.
- **0~1023**은 용도가 정해져 있으므로 함부로 사용하면 안된다.
- **1024~49151**의 범위 안의 것을 사용한다.

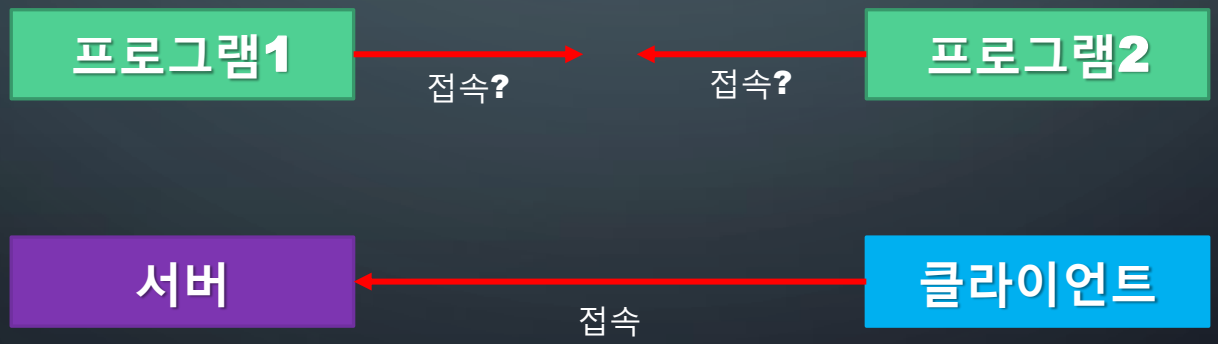
포트 번호	분류
0 ~ 1023	알려진 포트**(well-known ports)**
1024 ~ 49151	등록된 포트**(registered ports)**
49152 ~ 65535	동적/사설 포트**(dynamic and/or private ports)**

1. NETWORK & SOCKET PROGRAMING

클라이언트 – 서버 모델

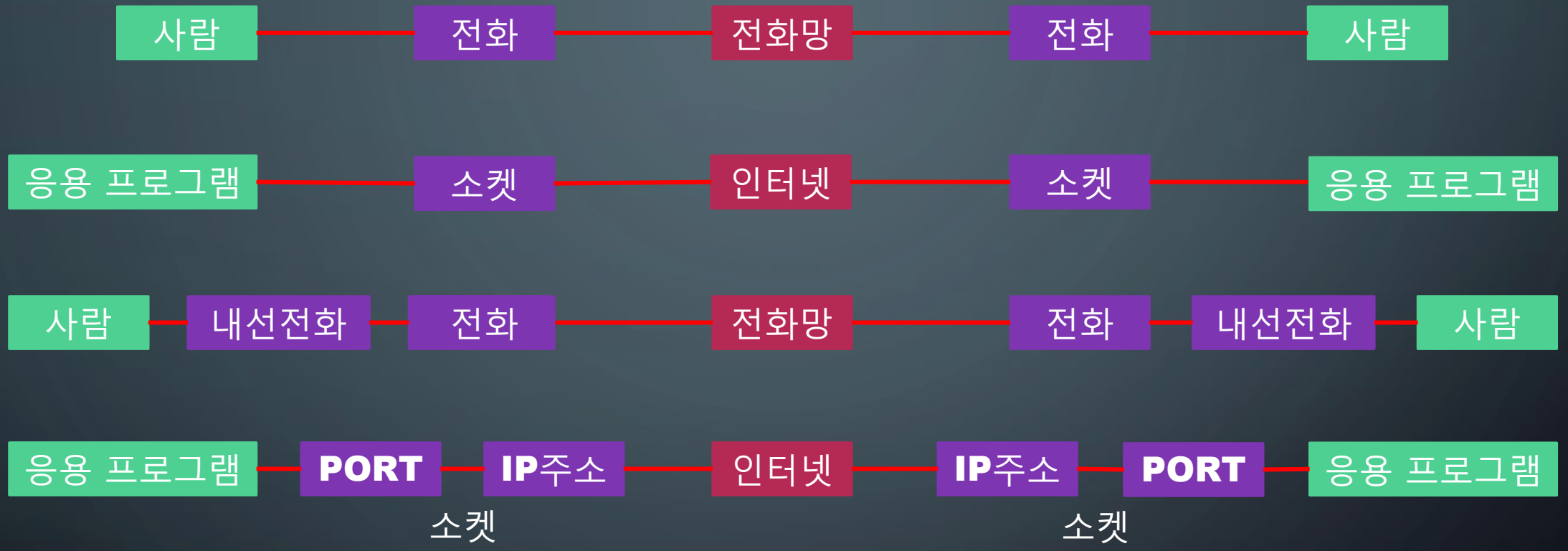
- 네트워크에서 일반적으로 작성하는 네트워크 모델.
- 서비스를 요청하는 쪽이 클라이언트, 클라이언트가 요청하는 서비스를 처리하는 쪽을 서버라고 부른다.
- 로컬 시스템에서 프로세스 간 통신기법을 사용해 데이터를 주고 받을 수 있다.
- 네트워크로 연결된 상황에서 통신 프로토콜을 사용해서 데이터를 주고 받을 수 있다.

아무리 통신 프로토콜을 이용한 연결을 할 경우 서로 연결을 시도만 하고 받아주지 않는 상황이라면 서로 통신을 할 수 없기 때문에 한쪽을 연결을 기다리고 한쪽은 연결을 시도하는 방법을 택해야 한다.



1. NETWORK & SOCKET PROGRAMING

소켓의 개념



소켓의 의미

- 데이터 타입에서의 의미
- 통신 종단점에서의 의미
- 네트워크 프로그래밍 인터페이스에서의 의미

1. NETWORK & SOCKET PROGRAMING

데이터 타입

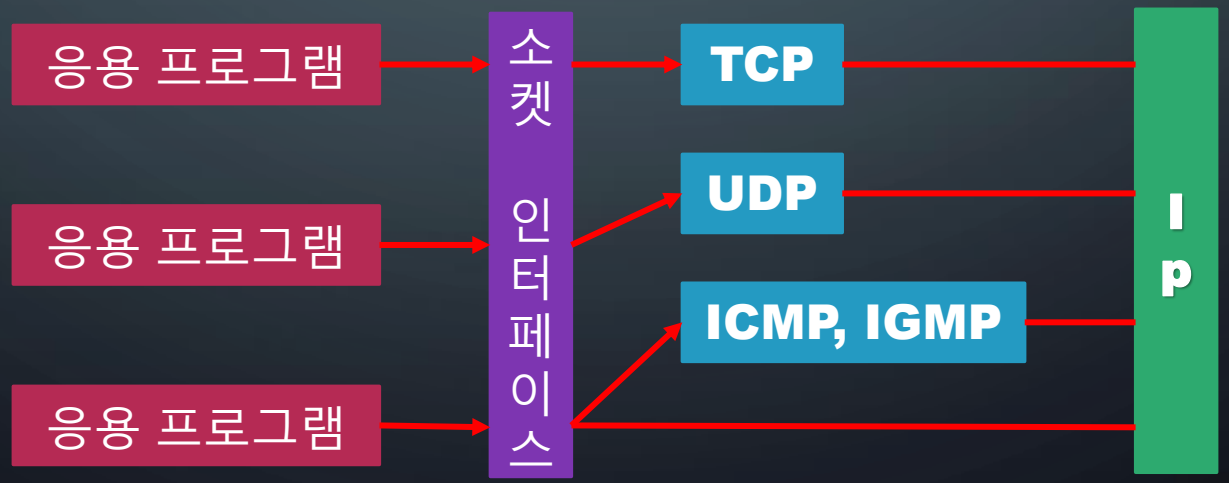
- 파일 디스크립터**(file descriptor)** 혹은 핸들**(handle)**과 유사한 개념
- 파일 입출력의 코드 흐름과 유사하다**.**
- 생성과 설정과정이 끝나면 운영체제의 통신 관련 정보를 참조해 다양한 작업을 편리하게 할 수 있는 데이터 타입으로 볼 수 있다**.**

통신 종단점

- 통신의 출발점이자 도착점이다**.**
- 소켓을 통해 서로 연결을 간주해서 데이터를 주고 받는다**.**

네트워크 프로그래밍 인터페이스

- 정해진 형태와 절차만 따른다면 인터페이스를 통해 데이터를 주고 받을 수 있다**.**



2. WINDOW SOCKET

2. WINDOW SOCKET

윈도우 소켓의 특징

- 유닉스 소켓에 기반을 둔 네트워크 프로그래밍 인터페이스
- **DLL(Dynamic-Link Library)**을 통해 대부분의 기능이 제공되므로 **DLL** 초기화와 종료 작업을 위한 함수가 필요하다.
- **GUI(Graphical User interface)**를 갖추고 메시지 구동 방식으로 동작하므로 이를 위한 확장 함수가 존재한다.
- 운영체제 차원에서 멀티쓰레드**(multithread)**를 지원하므로 멀티쓰레드 환경에서 안정적으로 동작하는 구조와 이를 위한 함수가 필요하다.
- 윈도우 소켓에는 버전이 있는데 **Win98**이상에서는 **2.2**버전을 사용한다.

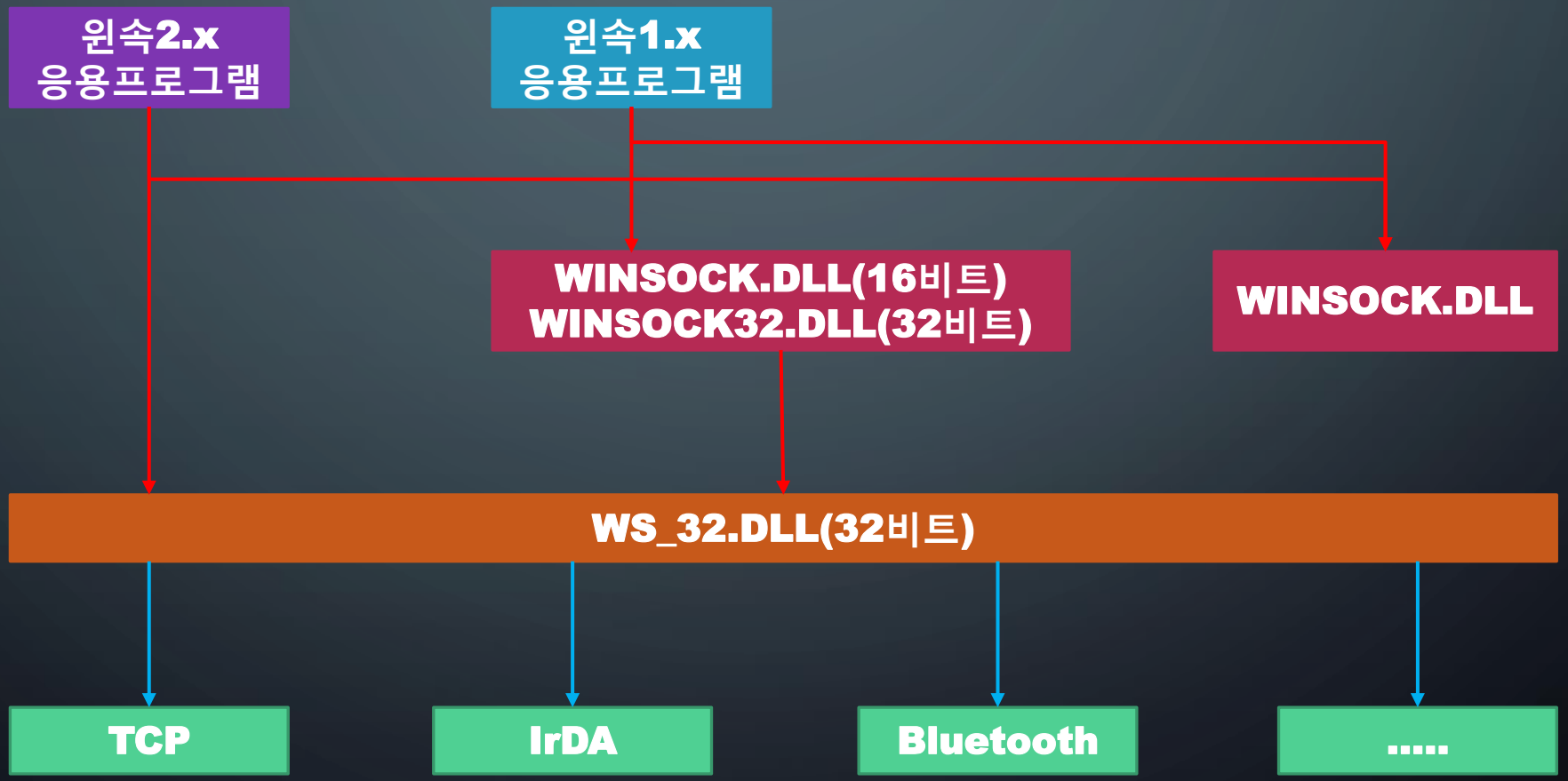
장점

- 유닉스 소켓과 소스 코드 수준에서 호환성이 높으므로 기존 코드를 이식하여 활용하기 쉽다.
- 가장 널리 사용하는 네트워크 프로그래밍 인터페이스므로 한번 배우면 여러 운영체제에서 사용 할 수 잇다.
- **TCP/IP** 외의 프로토콜도 지원하므로 최소 코드 수정으로 응용 프로그램이 사용할 프로토콜을 변경할 수 있다.
- 저수준 프로그래밍 인터페이스므로, 세부 제어가 가능하며 고성능 네트워크 프로그램을 개발할 수 있다.

2. WINDOW SOCKET

윈도우 소켓의 구조

- **WS2_32.DLL**로 제공, **WINSOCK.DLL**이나 **WSOCK32.DLL**을 통해 궁극적으로 **WS2_32.DLL**의 기능을 사용하게 되고 응용 프로그램이 실제로 사용할 하부 프로토콜은 **WS2_32.DLL**이 적절히 선택해 연결해 준다. ➔ **ws2_32.dll** 링크 필수!!



2. WINDOW SOCKET

윈도우 소켓 작성

오류처리

- 오류 발생 상황이 다양하고 발생률도 높기 때문에 오류가 어떤 것이 났는지 따로 처리해 줄 필요가 있다.
- **WSAGetLastError()** 함수로 에러 코드를 알아 낼 수 있고 **FormatMessage()**를 이용해서 오류 코드를 문자열로 변환하여 출력해준다.**(MSDN**에서 **FormatMessage()**의 다른 옵션 인자 값을 확인 할 수 있다.**)**

DWORD FormatMessage(DWORD dwFlags, LPCVOID lpSource, DWORD dwMessage,
DWORD dwLanguageIs, LPTSTR lpBuffer, DWORD nSize, va_list *Arguments)

- **첫 번째 인자 : FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM** 값을 사용한다**. FORMAT_MESSAGE_ALLOCATE_BUFFER**는 오류 메시지를 저장할 공간을 **FormatMessage()** 함수가 알아서 할당한다는 의미고**, FORMAT_MESSAGE_FROM_SYSTEM**은 운영체제부터 오류 메시지를 가져온다는 의미다**.**
- **두 번째 인자 : NULL**값
- **세 번째 인자 :** 오류 코드를 나타내며**, WSAGetLastError()** 함수의 리턴 값을 여기에 넣는다**.**
- **네 번째 인자 :** 오류 메시지를 표시할 언어를 나타내며**, MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT)**를 사용 하면 사용자가 제어판에서 설정한 기본 언어로 오류 메시지를 얻는다
- **다섯 번째 인자 :** 오류 메시지의 시작 주소가 여기에 저장된다**. LocalFree()** 함수를 사용해서 마지막에 해제를 해줘야 한다**.**
- **여섯 번째 인자 : 0**
- **일곱 번째 인자 : NULL**

2. WINDOW SOCKET

두 가지의 오류 체크 코드

MessageBox로 표현

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

printf로 표현

```
void err_display(char *msg)
{
    LPVOID lpMsgBuf;
    FormatMessage( FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
                   NULL, WSAGetLastError(), MAKELANGID(LANG_NEUTAL, SUBLANG_DEFAULT),
                   (LPTSTR)&lpMsgBuf, 0, NULL);
    printf("[%s] %s, msg, (char *)lpMsgBuf");
    LocalFree(lpMsgBuf);
    exit(1);
}
```

2. WINDOW SOCKET

WINSOCK초기화와 종료

- **WSAStartup()** 함수로 초기화**, WSACleanup()** 함수로 종료를 해준다**.**

Int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);

- **첫 번째 인자 :** 프로그램이 요구하는 최상위 윈속 버전이다**. 2.2**를 사용**.**
- **두 번째 인자 : WSADATA** 구조체를 전달하면 이를 통해 윈도우 운영체제가 제공하는 윈속 구현에 관한 정보를 얻을 수 있다**.**
- 성공 **: 0,** 실패 **:** 오류 코드

Int WSACleanup(void);

- 성공 **: 0,** 실패 **: SOCKET_ERROR**

ws2_32.lib링크 혹은 **#pragma comment(lib, “ws2_32”)**를 사용**.**

```
int main()
{
    WSADATA was;
    if(WSAStartup(MAKEWORD(2, 2), &wsa) != 0) return 1;
    MessageBox(NULL, “윈속 초기화 성공”, “알림”, MB_OK);
    WSACleanup();
    return 0;
}
```

2. WINDOW SOCKET

소켓 생성과 닫기

기본 요건은 통신 양단이 같은 프로토콜을 사용해야 한다.
➔ 같은 **(TCP / UDP)**인터페이스여야 통신이 가능하다.

소켓 생성

- **socket()** 함수를 이용해서 사용자가 요청한 프로토콜을 사용해 통신할 수 있도록 내부적으로 리소스를 할당하고, 이에 접근 할 수 있는 일종의 핸들 값**(socket**타입**)**을 리턴 한다.

SOCKET socket(int af, int type, int protocol);

- **첫 번째 인자 :** 주소 체계를 지정한다. **(IPv4**의 **TCP/UDP ➔ AF_INET)**
- **두 번째 인자 :** 소켓 타입을 지정한다.
  SOCK_STREAM : 신뢰성 있는 데이터 전송 기능 제공, 연결형 프로토콜
  SOCK_DGRAM : 신뢰성 있는 데이터 전송 기능 제공, 비 연결형 프로토콜
- **세 번째 인자 :** 사용할 프로토콜을 지정한다. **➔TCP/UDP**는 위 두 단계에서 이미 판단 가능하므로 **0.**
  IPPROTO_TCP : TCP
  IPPROTO_UDP : UDP

소켓 닫기

- 소켓을 사용한 통신을 마치면 관련 리소스를 반환해야 한다. **closesocket()** 함수 사용.

2. WINDOW SOCKET

소켓 생성과 닫기

기본 요건은 통신 양단이 같은 프로토콜을 사용해야 한다.
➔ 같은 **(TCP / UDP)**인터페이스여야 통신이 가능하다.

소켓 생성

- **socket()** 함수를 이용해서 사용자가 요청한 프로토콜을 사용해 통신할 수 있도록 내부적으로 리소스를 할당하고, 이에 접근 할 수 있는 일종의 핸들 값**(socket**타입**)**을 리턴 한다.

SOCKET socket(int af, int type, int protocol);

- **첫 번째 인자 :** 주소 체계를 지정한다. **(IPv4**의 **TCP/UDP ➔ AF_INET)**
- **두 번째 인자 :** 소켓 타입을 지정한다.
  SOCK_STREAM : 신뢰성 있는 데이터 전송 기능 제공, 연결형 프로토콜
  SOCK_DGRAM : 신뢰성 있는 데이터 전송 기능 제공, 비 연결형 프로토콜
- **세 번째 인자 :** 사용할 프로토콜을 지정한다. **➔TCP/UDP**는 위 두 단계에서 이미 판단 가능하므로 **0.**
  IPPROTO_TCP : TCP
  IPPROTO_UDP : UDP

소켓 닫기

- 소켓을 사용한 통신을 마치면 관련 리소스를 반환해야 한다. **closesocket()** 함수 사용.

WinsockInit, WinsockSocketInit 프로젝트를 참고하자.

3. SOCKET ADDRESS STRUCT

3. SOCKET ADDRESS STRUCT

소켓 주소 구조체

- 네트워크 프로그램에서 필요한 주소 정보를 담고 있는 구조체, 다양한 소켓 함수의 인자로 사용.
  ➔ 반드시 포인터 형으로 변환해야 한다.
- 프로토콜 체계에 따라 주소 지정 방식이 다르다.
- **SOCKADDR**구조체가 가장기본 형식이다.

```
typedef struct sockaddr
{
    u_short sa_family;
        ➔ 주소 체계를 나타내는 16비트 정수 값, TCP/IP 프로토콜을 사용한다면 AF_INET 값이 된다.
    char        sa_data[14];
        ➔ 주소 체계가 사용할 주소 정보를 담는다.
} SOCKADDR;
```

IPv4 소켓 주소 구조체

```
typedef struct sockaddr_in
{
    short           sin_family;
        ➔ 호스트 바이트 정렬이 필요
    u_short         sin_port;
        ➔ 포트번호, 네트워크 바이트 정렬이 필요.
    struct in_addr  sin_addr;
        ➔ IP주소, 네트워크 바이트 정렬이 필요.
}SOCKADDR_IN;
```

3. SOCKET ADDRESS STRUCT

바이트 정렬

- 메모리에 데이터를 저장할 때 바이트 순서를 나타내는 용어.
- 빅 엔디안**(big-endian),** 리틀 엔디안**(little-endian)** 방식이 있다. **➔CPU**와 운영체제에 따라 다르다.
  - **빅 엔디안 :** 최상위 바이트부터 차례로 저장하는 방식.
  - **리틀 엔디안 :** 최하위 바이트 부터 차례로 저장하는 방식.

- 호스트와 라우터간의 **IP**주소체계를 파악하는데 사용. ➔ 빅 엔디안을 사용하기로 약속 되어 있다.
- 빅 엔디안을 네트워크에서는 **네트워크 바이트 정렬**이라고 한다.
- 윈속에서는 윈속 함수가 바이트 정렬을 지원해준다.
- **hton*()** 함수는 호스트 바이트 정렬로 네트워크 바이트 정렬로 저장된 값을 입력으로 받아서 호스트 바이트 정렬로 변환한 값을 리턴 한다.
- **noth*()** 함수는 네트워크 바이트 정렬로 저장된 값을 입력으로 받아서 호스트 바이트 정렬로 변환한 값을 리턴 한다.

```
u_short htons(u_short hostshort);
u_long htonl(u_long hostlong);
u_short ntohs(u_short netshort);
u_long ntohl(u_long netlong);
```

ByteOrder, IPAddrTrans 프로젝트를 참조한다.