C#
-CAHPTER12-

SOUL SEEK

목차

네트워크 프로그래밍

1. 네트워크 프로그래밍

TCP/IP

- 인터넷에서 사용하는 표준 프로토콜
  ➔**PC**들이 네트워크를 통해 데이터를 주고받기 위한 **“통신규약”**을 프로토콜이라고 한다**.**
- 데이터를 주고 받기 위한 일련의 프로토콜의 모음
- 물리적인 구성으로부터 독립적이다**(Lan**에서 **WiFi,** 이더넷**,** 모뎀 등등의 물리적인 구성과 무관하게 연결된다**.)**

TCP/IP Stack

- 링크**,** 인터넷**,** 전송**,** 애플리케이션 이 **4**계층을 말한다**.**

링크계층(Link Layer)

- 물리 계층**(Physical Layer),** 네트워크 계층**(Network Interface Layer),** 미디어 접근 계층**(Media Access Layer)**등으로 불리운다**.**
- 네트워크를 통해 가장 먼저 만나게 되는 구간이다**.**
  ➔네트워크의 물리적인 연결 매체를 통해 패킷을 주고 받는 작업을 담당해주기 때문에 **TCP/IP**는 물리적인 구성으로 부터 독립성을 가지게 된다**.**
  ➔패킷에서 물리적 데이터 전송에 사용되던 부분을 제거하고 인터넷 계층으로 패킷을 전달 한다**.**

패킷(Packet)

- 네트워크에서 오고 가는 데이터를 말한다**,** 소포랑 비슷한 면을 가졌기 때문에 패킷이라 붙여졌다**.**
  ➔데이터를 그냥 보내기만 하면 받는 쪽은 누가 보냈는지 몰라 처리 하거나 돌려주는게 힘들고**,** 전달하는 쪽은 어떤 방식**(**당일 배송**?**등기**?**일반우편**?)**인지 헷갈려서 어떻게 보낼지 몰라 한다**.** 그렇기 때문에 소포와 같이 데이터를 어떤 방식으로 전달하고 누가 보냈는지 어떤 데이터인지가 포함된다**.**

1. 네트워크 프로그래밍

인터넷 계층(Internet Layer)

- 패킷을 수신해야 할 상대의 주소를 지정하고, 나가는 패킷에 대해서는 적절한 크기로 분할하며 들어오는 패킷에 대해서는 재조립을 수행한다.
- 인터넷 계층에서 사용하는 프로토콜이 **IP**이다.
  ➔인터넷 주소체계를 **IP**라고 부르고 이것을 이용해서 어떤 곳에 보내는지 어떤 곳에서 왔는지를 구분한다.
  ➔**IP**는 내보낸 패킷을 누구 한테 보내는지에 대한 정보의 지정만 있을 뿐이지 중간과정에서의 손실 및 파손, 정상적으로 도착하였는지 받았는지에 대한 보장이 없다.

전송 계층(Transport Layer)

- 패킷의 운송을 담당하는 프로토콜들이 정의 되어 있다.
  ➔ 전송 제어 프로토콜**(TCP : Transmission Control Protocol)**은 송신 측과 수신 측간의 연결성을 제공하며, 신뢰할 수 있는 패킷 전송 서비스를 제공한다.
  ➔ **TCP**는 여러 개의 패킷을 송신하는 경우 패킷 사이의 순서를 보장하며, 패킷이 유실되기라도 하면 재전송을 해주기까지 하지만, 그 만큼 성능에서 손실을 볼 수 밖에 없다.
  →순서와 손실의 우려가 없는 작은 양의 데이터는 하나의 패킷으로 보내면 되기 때문에 **TCP**를 구지 활용하지 않아도 된다.
  →이를 위해 **UDP(User Datagram Protocol)**을 정의되어 있다. 성능이 **TCP**에 비해 우수하다.

1. 네트워크 프로그래밍

애플리케이션 계층(Application Layer)

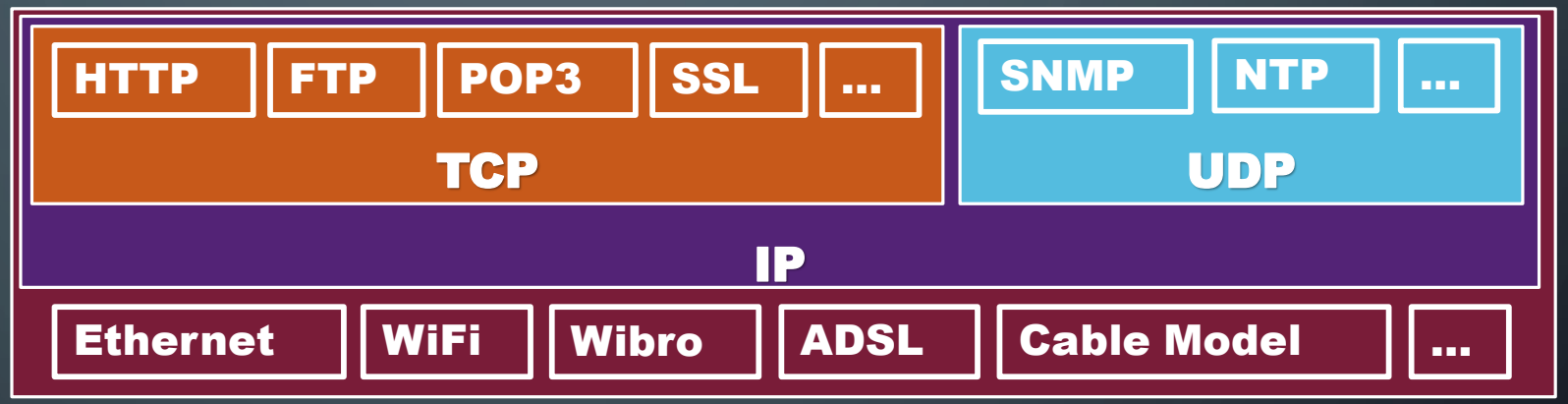
- 응용프로그램들의 프로토콜이 정의 되는 곳, **HTTP(Hyper Text Transfer Protocol), FTP(File Transfer Protocol), SNMP(Simple Network Management Protocol)**이 대표적인 예.
- 표준화된 프로토콜 뿐만 아니라 나름대로의 프로토콜을 정의해서 사용할 수 있다.
- 실제 우리가 구축하고 설정할 수 있는 단계.

애플리케이션 계층

전송 계층

인터넷 계층

링크 계층



TCP/IP의 주소체계

- 인터넷에서 패킷을 보내기 위해 필요한 정보, 인터넷도 주소가 있으며 **IP**주소**(IP Address)**라고 부른다.
- **IP**주소는 부호 없는 **8**비트 정수필드, **16**비트 정수 필드로 구성한다. **IPv4, IPv6**형식이 있으며, **IPv4**는 우리가 익히 알고 있는 **8**비트의 정수 필드 **4**개을 점**(ex. 192.128.0.1)**으로 이어서 만들어지고 **IPv6**는 **16**비트의 수 **8**개를 콜론**(3FFE:FFFF:7546:FEDA:1245:BA98:3210:4562)**으로 연결한다.
- 현재 시점에서는 보안적인 측면때문에 점점 **IPv6**로의 전환이 시작되고 있다.

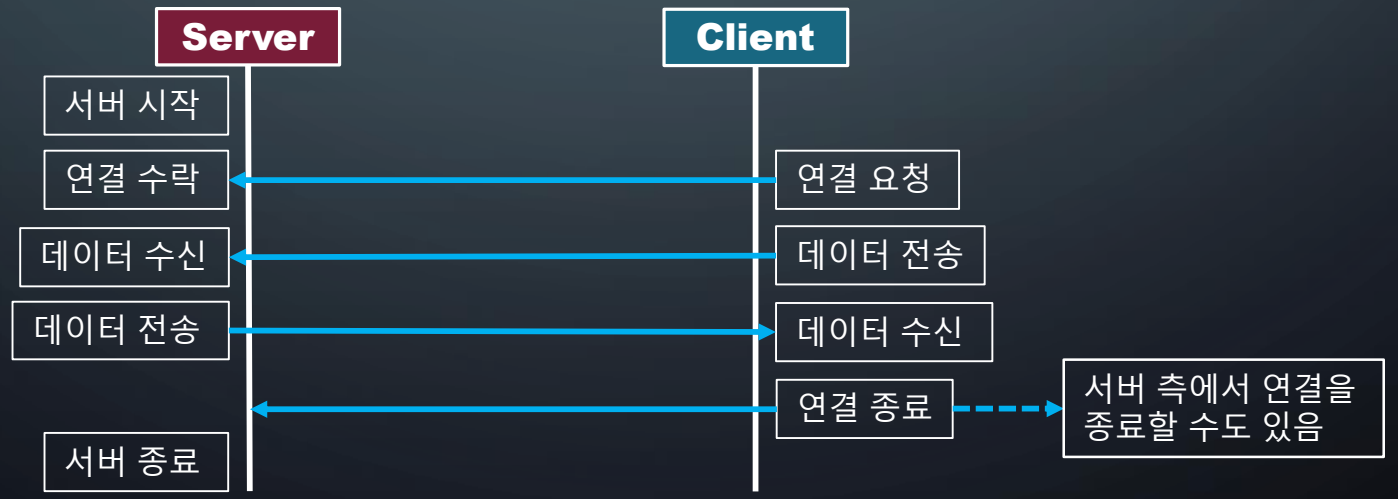
1. 네트워크 프로그래밍

포트(PORT)

- 네트워크 패킷이 드나드는 출입문을 말한다.
  ➔주소로 치면 **xxx**번지가 **IP**에 해당하고 세부사항이 **PORT**에 해당한다.
  ➔**IP : xxx** 아파트 **xx**동, **PORT : xxx**호
- 부호 없는 **16**비트 정수로 **0 ~ 65535** 사이의 값을 이용한다.
- 이미 우리가 잘 알려진 포트들이 있으며 그것을 제외한 포트들을 사용할 수 있다. **1 ~ 1023**
- **HTTP : 80, HTTPS : 443, FTP : 21, Telnet : 23, SMTP : 25, IRC : 194, IIOP : 535**
- 사용하고 있는 포트만 열어 두는 것이 보안 적인 측면에서도 도움이 된다.

TCP/IP의 동작 과정

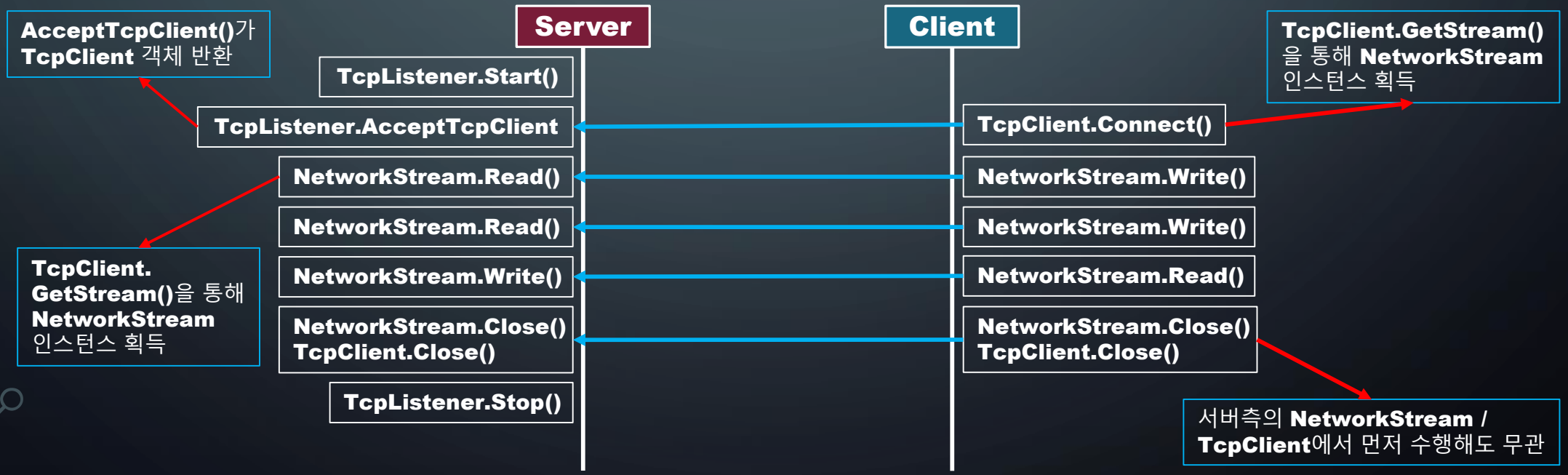
- 서버/클라이언트 방식으로 동작한다.
  ➔통신을 수행하는 양단 중에서 적어도 한쪽은 다른 한쪽에 서비스를 제공해야한다.(주고 받아야 한다)
- 서비스를 제공하는 쪽을 서버이며 서버가 서비스 제공을 준비하며 클라이언트가 접속할 준비를 한다.
- 클라이언트는 서버의 서비스가 준비되었으면 접속을 시도하고 서버가 승인하면 이제 데이터를 주고 받을 수 있다.



1. 네트워크 프로그래밍

TcpListener & TcpClient

- **.NET Framework**가 **TCP/IP** 통신을 위해 제공하는 클래스
  ➔**System.Net.Sockets** 네임스페이스를 사용한 **Sockets** 통신도 있지만 개념적인 이해를 위해 해당 클래스를 사용해보자**!**
- **TcpListener**클래스는 서버 애플리케이션에서 사용되며**,** 클라이언트의 연결 요청을 기다리는 역할을 한다**.** **TcpClient**는 서버 애플리케이션과 클라이언트 애플리케이션 양쪽에서 사용된다**.** **TcpClient**가 서버에 연결 요청을 하는 역할을 수행하며**,** 서버에서는 클라이언트의 요청을 수락하면 클라이언트와의 통신에 사용할 수 있는 **TcpClient**의 인스턴스가 반환된다**.**
- 서버와 클라이언트에서 각각이 **TcpClient**를 사용하고 있고 **GetStream()** 메소드를 갖고 있어서**,** 양쪽의 응용 프로그램은 이 메소드가 반환하는 **NetworkStream** 객체를 통해 데이터를 주고 받는다**.**
- 보낼 때는 **NetworkStream.Write(),** 읽을 때는 **NetworkStream.Read()**를 호출한다**.**
- 서버와 클라이언트의 연결을 종료할 때는 **Close()** 메소드를 호출한다**.**



1. 네트워크 프로그래밍

TcpListener와 TcpClient의 메소드

클래스	메소드	설명
TcpListener	**Start()**	연결 요청 수신 대기를 시작한다.
	AcceptTcpClient()	클라이언트의 연결 요청을 수락한다. 이 메소드는 **TcpClient** 객체를 반환한다.
	Stop()	연결 요청 수신 대기를 종료한다.
TcpClient	**Connect()**	서버에 연결을 요청한다.
	GetStream()	데이터를 주고 받는데 사용하는 매개체인 **NetworkStream**을 가져온다.
	Close()	연결을 닫는다.

TcpListener 서버 시작 코드

IPEndPoint는 **IP**통신에 필요한 **IP**주소와 출입구(포트)를 나타낸다.

```
IPEndPoint localAddress = new IPEndPoint(IPAddress.Parse("192.168.100.17"), 5452);

TcpListener server = new TcpListener(localAddress);

Server.Start();
```

Server 객체는 클라이언트가 **TcpClient.Connect()**를 호출하여 연결 요청해오기를 기다리기 시작한다.

1. 네트워크 프로그래밍

클라이언트의 연결 요청

```
IPEndPoint clientAddress =
                        new IPEndPoint(IPAddress.Parse("192.168.10.18"), 0);
TcpClient client = new TcpClient(clientAddress);
IPEndPoint serverAddress =
                        new IPEndPoint(IPAddress.Parse("192.168.100.17"), 5452);

Client.Connect(serverAddress);
```

포트를 **0**으로 지정하면 **OS**에서 임의의 번호로 포트를 할당해준다.

서버가 수신 대기하고 있는 **IP** 주소와 포트 번호를 향해 연결 요청을 수행한다.

서버가 클라이언트의 연결 수락

```
TcpClient client = server.AcceptTcpClient();
```

NetworkStream을 이용해서 읽고 쓰기

```
NetworkStream stream = client.GetStream();

int length;
string data = null;
byte[] bytes = new byte[256];

while((length = stream.Read(bytes, 0, bytes.Length)) != 0)
{
        data = Encoding.Default.GetString(bytes, 0, length);
        Console.WriteLine(String.Format("수신: {0}", data))
        byte[] msg = Encoding.Default.GetBy(data);`

        stream.Write(msg, 0, msg.Length);
        Console.WriteLine(String.Format("송신: {0}", data));
}
```

TcpClient를 통해 **NetworkStream** 객체를 얻는다.

NetworkStream.Read() 메소드는 상대방이 보내온 데이터를 읽어 들인다. 한편, 상대와의 연결이 끊어지면 이 메소드는 **0**을 반환한다. 즉, 이 루프는 연결이 끊어지기 전까지는 계속된다.

NetworkStream.Write() 메소드를 통해 상대방에게 메시지를 전송한다.

1. 네트워크 프로그래밍

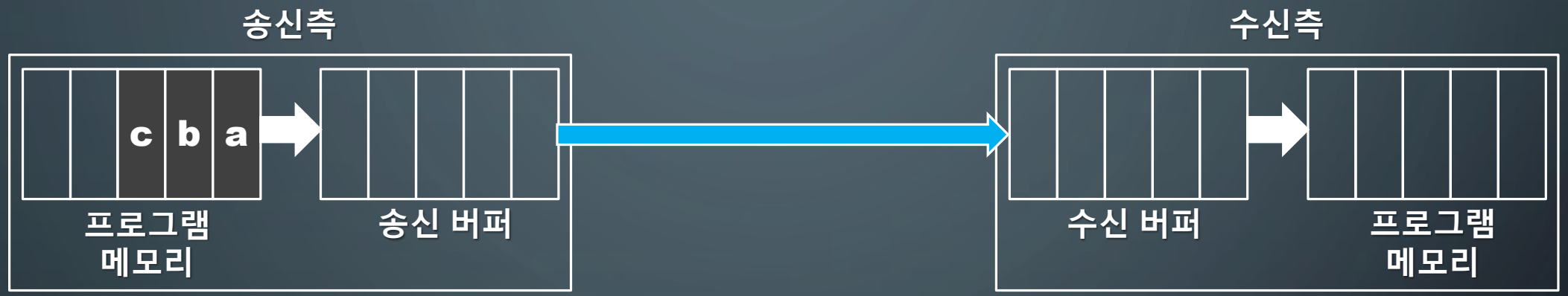
패킷이 전송되는 원리

- **TCP**는 연결 지향, 흐름 지향 프로토콜이다.
  ➔송신 측에서 **Write()**를 할 때마다 메시지가 만들어지며 이 메시지를 수신 측에서 **Read()**를 통해 하나씩 읽어오는 식의 방식이 아니다.
  ➔전기가 흘러가는 모양과 비슷하다.
- **TCP** 통신 애플리케이션은 보내거나 받는 데이터를 저장할 공간을 마련해 두는데 이를 **buffer**라고 하고 **buffer**에 있는 데이터를 전기 흐르듯 흘려 보낸다.
- **서로의 주고 받을 수 있는 공간에 데이터가 있는지 없는지를 지속적으로 체크하고 있는 상황이기 때문에 데이터를 다 만들어서 주는 것이 아니라 지속적으로 보내고 있어야 하며 온전한지를 체크해야 하며 이 데이터가 어떤 데이터인지 구분도 필요하다.**

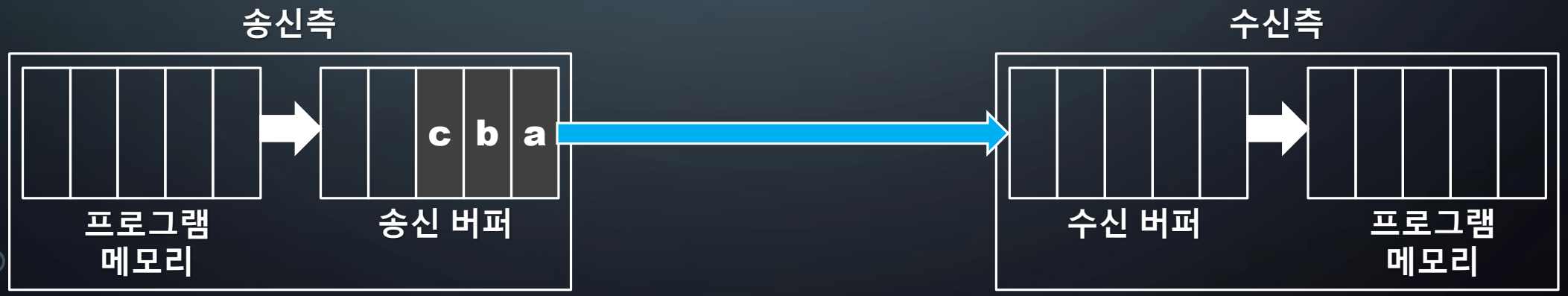
1. 네트워크 프로그래밍

TCP 패킷 흐름의 예

TCP 연결을 맺고 있고**,** 송신 **App**가 메모리에 들고 있는 데이터 **a, b, c**를 수신 **App**에 보내려고 한다고 할 때

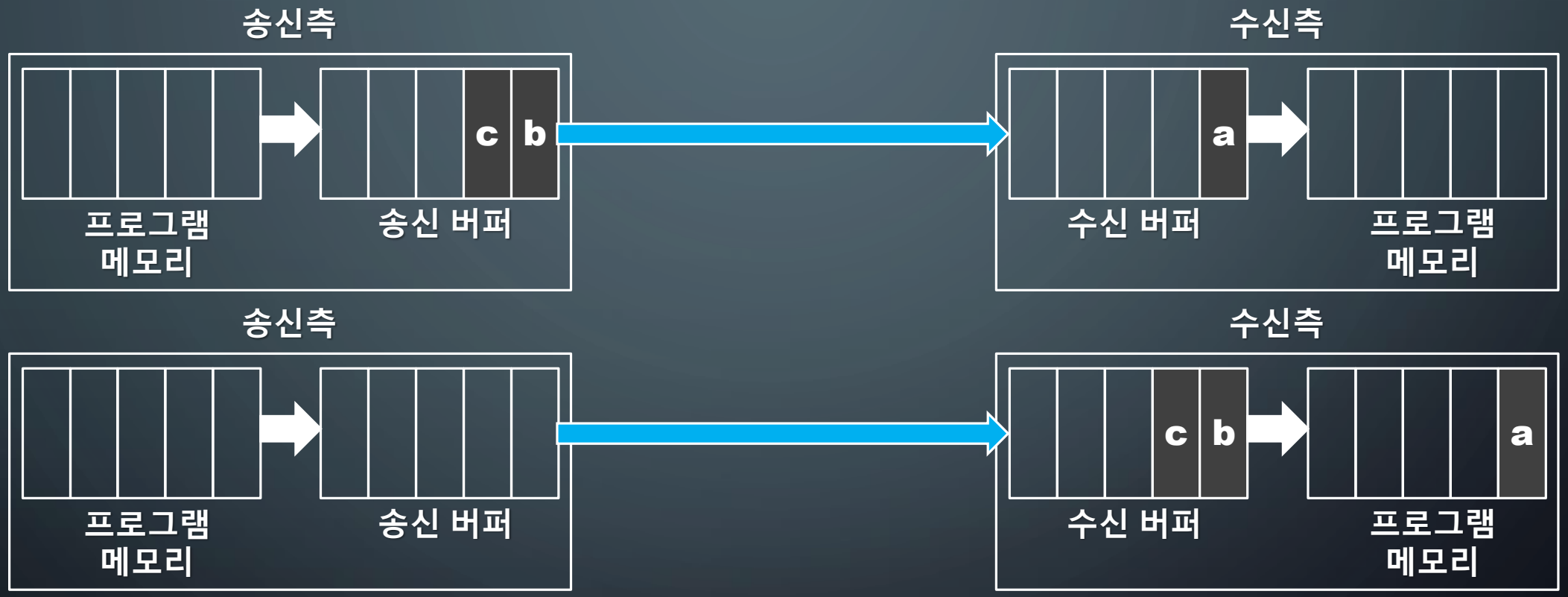


이미 **wBuffer**라는 이름의 바이트 배열에 담겨 있다고 가정하면**,** 송신측 **App**에서 **writer.Write(wBffer, 0, 3)**을 호출하면 데이터는 **App**의 메모리에서 송신 **Buffer**로 이동한다**.**



1. 네트워크 프로그래밍

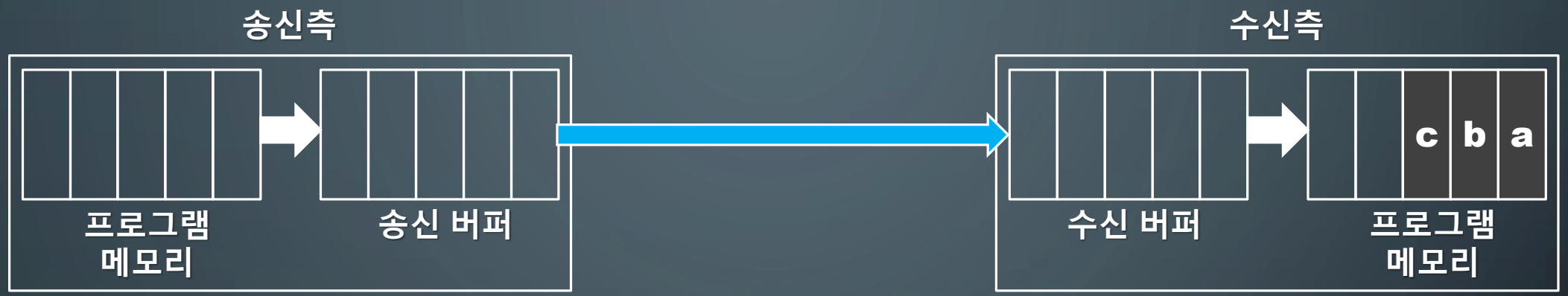
OS는 송신 **Buffer**에 있는 내용을 연결을 맺고 있는 수신 측으로 보내기 시작하고, 이 때 네트워크 대역폭이 넓고 품질도 좋다면 많은 데이터가 빠른 속도로 수신 측으로 이동할 것이고, 그렇지 않다면 조금씩 보내어 질것이다.



수신 측의 **App**에서는 데이터를 담기 위한 **rBuffer**를 선언해 두었다면 **reader.Read(rBuffer, 0, 16)**을 호출하는데 보통 보내는 데이터와 쌍을 이뤄서 보내는 만큼 받게 준비해 둔다. 만약 **16**바이트를 읽어오려고 시도하지만 실제 수신 **Buffer**에는 **a** 하나밖에 없으므로 **Buffer**에는 **a**가 담기고 **Read()** 메소드는 실제로 읽은 바이트 수 **1**을 반환하고 그 동안 **b, c**는 수신 **buffer**에 도착한다.

1. 네트워크 프로그래밍

수신 **Buffer**에서 **b, c**를 읽고 **Read()** 함수는 바이트 수 **2**를 반환하고 메모리에 전달하면서 **a, b, c**가 모두 메모리에 들어왔다**.**



프로토콜 설계

TCP 네트워크 프로그래밍은 요구하는 것이 많다 하지만 이런 것들은 무조건 필요한 부분이기 때문에 한번 설계해서 그 방식과 절차만 익힌다면 **Framework**처럼 사용**(**최소의 비용을 지불하고**)**할 수 있기 때문에 설계에 대한 방법을 찾는 것이 중요하다**.**
데이터가 한번만 왔다 갔다 하거나 한종류의 데이터만 주고 받으려고 만들어 지는 경우는 없다**.** 그렇기 때문에 **TCP**에서는 연속적인 통신을 하고 있기 때문에 패킷의 경계를 구분해 내는 것이 중요하고 대표적인 방법으로는 **고정 길이 형식과 가변 길이 형식** 두 가지가 있다**.** **고정길이는 어떤 패킷이든 항상 같은 크기**로 경계를 체크한다**,** **가변길이는 또 두가지 방식이 있는데 메시지를 두 부분으로 나눠서 길이가 고정된 앞부분에 뒷부분의 길이를 기입하는 방식과 메시지를 구분하는 특정 값을 이용하는 방식이 있다.**
고정방식은 바이너리 통신에 주로 사용하고 **가변길이는 텍스트 통신**에 주로 사용된다**.**

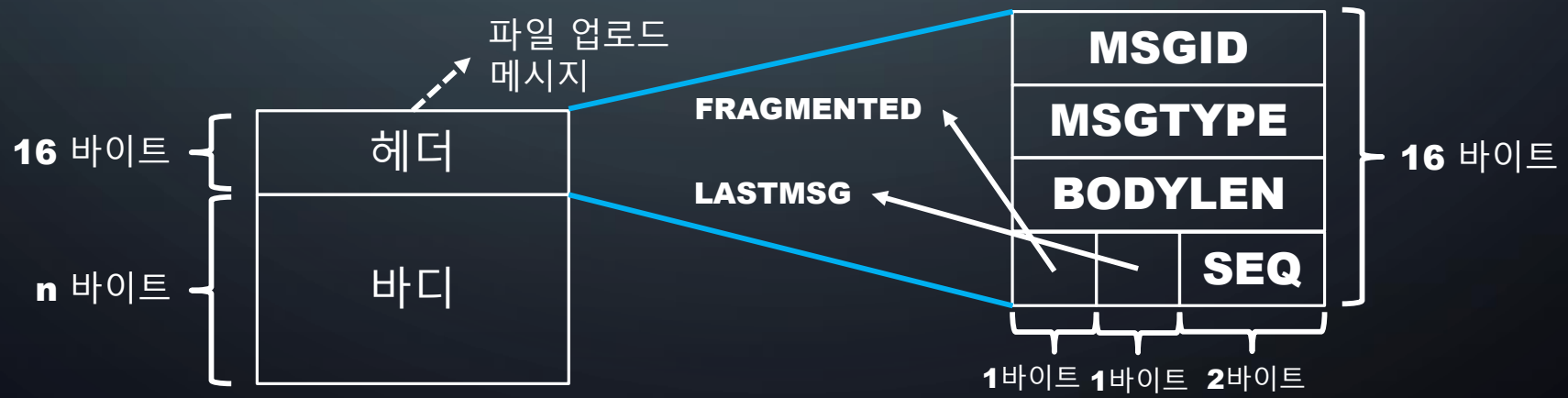
1. 네트워크 프로그래밍

파일 업로드 프로토콜 설계로 가변길이를 이해해보자!

- 해당 업로드 프로토콜의 이름을 **FUD**라 지을 것이다.
- 패킷은 가변 길이 형식을 사용하고 가변 길이 형식 중에서 앞부분이 뒷부분의 길이를 알려주는 형식으로 만들 것이다.
- 프로토콜 부분은 클라이언트와 서버의 규약이 될 것이기 때문에 클래스 라이브러리**(.dll)**를 이용 할 것이다.

패킷 구성

- 헤더와 바디 두부분으로 나뉜다.
- 바디에는 실제로 전달하고자 하는 데이터를 담고, 헤더에는 본문의 길이를 비롯하여 메시지의 속성 몇 가지를 담을 것이다. 바디는 담는 데이터에 따라 달라지고 헤더는 항상 일정한 **16**바이트로 고정 할 것이다.
  ➔헤더가 고정인 이유는 헤더를 먼저 온전하게 읽고 어떤 데이터이며 크기는 얼마나 되는지를 파악해야 바디를 읽을 수 있기 때문



1. 네트워크 프로그래밍

FUP의 헤더가 갖고 있는 각 속성 필드

필드 이름	크기(바이트)	설명
MSGID	**4**	메시지 식별 번호
MSGTYPE	**4**	메시지의 종류 • **0x01 :** 파일 전송 요청 • **0x02 :** 파일 전송 요청에 대한 응답 • **0x03 :** 파일 전송 데이터 • **0x04 :** 파일 수신 결과
BODYLEN	**4**	메시지 본문의 길이**(**단위 **:** 바이트**)**
FRAGMENTED	**1**	메시지의 분할 여부 • 미분할 **: 0x0** • 분할 **: 0x1**
LASTMSG	**1**	분할된 메시지의 마지막인지의 여부 • 마지막 아님 **: 0x0** • 마지막 **: 0x1**
SEQ	**2**	메시지의 파편번호

1. 네트워크 프로그래밍

MSGTYPE의 바디구조

파일 전송 요청(0x01)

필드이름	크기(바이트)	설명
FILESIZE	**8**	전송할 파일 크기(단위 : 바이트)
FILENAME	**BODYLEN – FILESIZE(8 byte)**	전송할 파일의 이름

파일 전송 요청에 대한 응답(0x02)

필드이름	크기(바이트)	설명
MSGID	**4**	파일 전송 요청 메시지(**0x01**)의 메시지 식별 번호
RESPONSE	**1**	파일 전송 승인 거부 • 거절 **0x0** • 승인 **0x1**

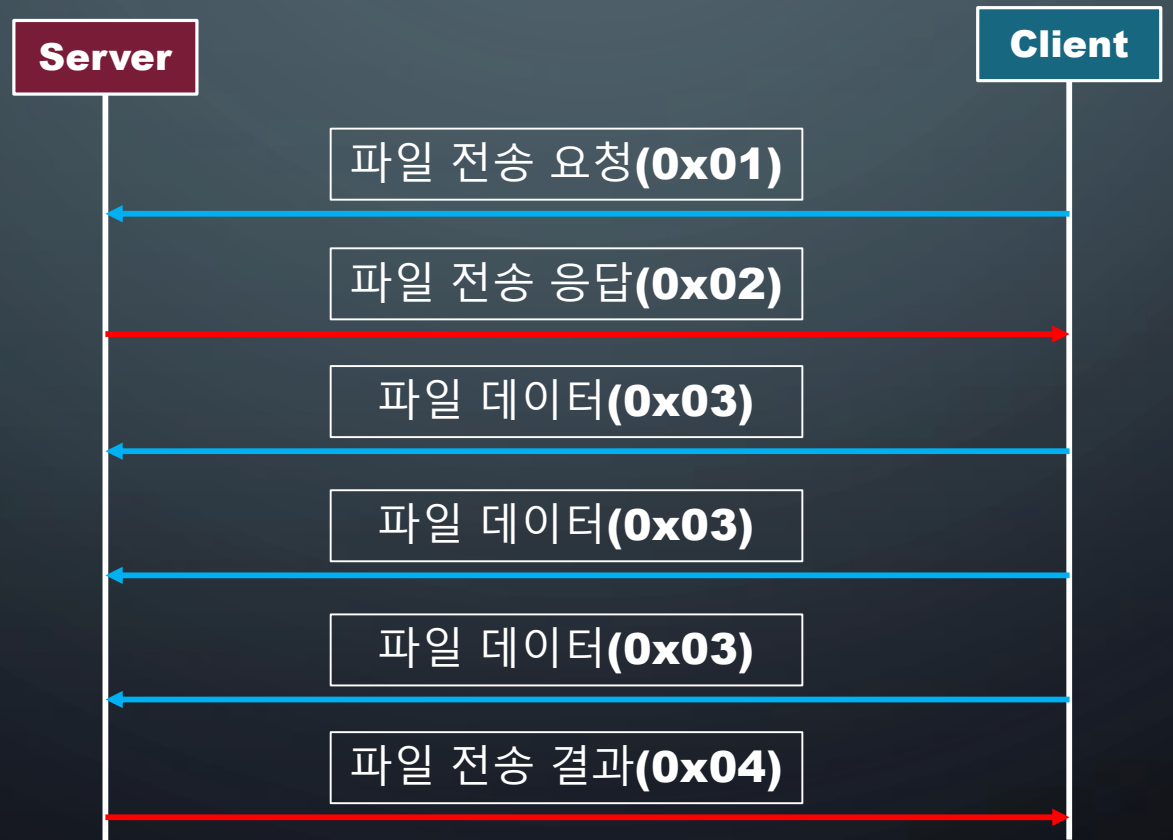
파일 전송 데이터

필드이름	크기(바이트)	설명
DATA	헤더의 **BODYLEN**	파일 내영

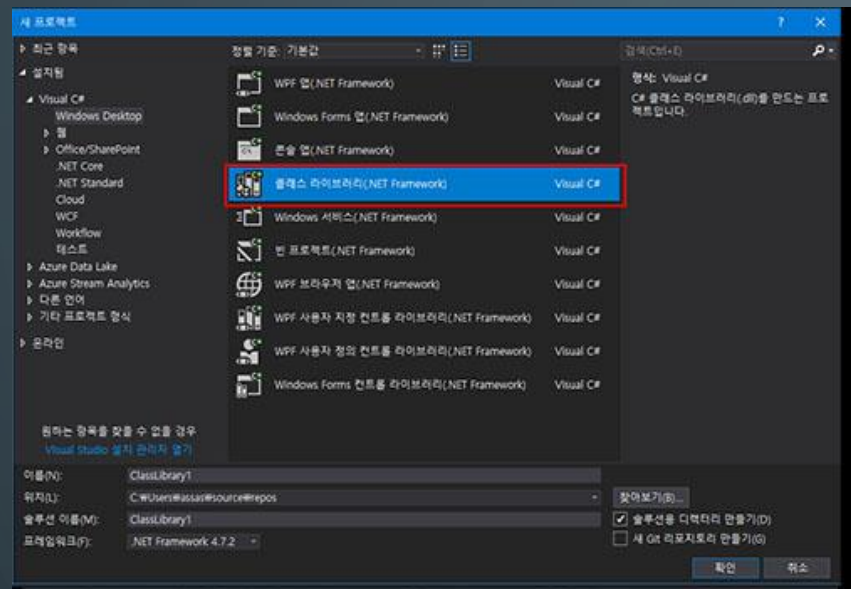
1. 네트워크 프로그래밍

파일 수신 결과 메시지(0x04)

필드이름	크기(바이트)	설명
MSGID	**4**	파일 전송 데이터**(0x03)**의 식별 번호
RESULT	**1**	파일 전송 승인 거부 • 실패**: 0x0** • 성공**: 0x1**

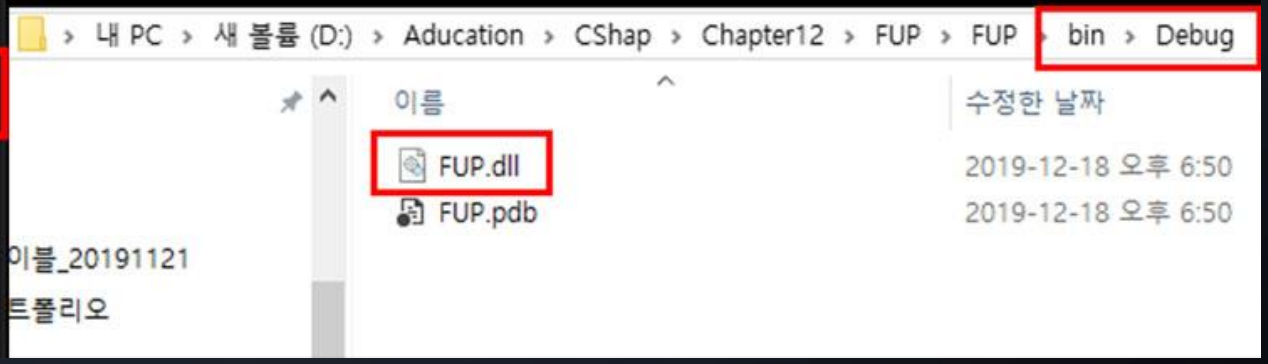


1. 네트워크 프로그래밍



서버와 클라이언트가 공통으로 공유할 수 있는 부분은 클래스 라이브러리형태로 제작해서 배포하는 것이 유용하다.

클래스라이브러리 형식으로 프로젝트를 생성하고 프로젝트를 빌드하면 **.dll 파일이 생성**된다
해당파일의 위치를 확인하고 알아두자.
(참조 추가에 활용 할 수 있다.)



1. 네트워크 프로그래밍

.dll 파일의 경로를 검색해서 직접 추가를 해주면 사용할 수 있다.