# TCP/IP 통신

2024.04.25_노하람

# TCP (Transmission Control Protocol, 전송 제어 프로토콜)

- TCP는 네트워크를 통해 디바이스에서 서버로 데이터를 전송하는 네트워크 프로토콜.
- 채팅, 이메일, 온라인 동영상, 웹 검색 등
  TCP 프로토콜을 사용
- TCP는 연결 기반이므로 데이터를 전송하는 동안
  수신자와 발신자 사이에 연결을 설정하고 이를 유지
  데이터가 완전히 온전하게 도착하도록 보장
- TCP/IP 프로토콜이라고 칭하는 이유는 IP를 이용한 프로토콜이기때문

# TCP (Transmission Control Protocol, 전송 제어 프로토콜)

- 장점

1. 데이터를 전송할 때 오류를 검사하여 전송된 데이터가 목적지에 온전하게 도달하도록 보장
   : Header CheckSum
2. 수신자의 용량에 따라 데이터를 전송하는 속도를 최적화하고 변경합니다.
3. 데이터가 목적지에 도달했는지 확인하고 첫 번째 전송이 실패한 경우 재전송을 시도합니다.

# TCP (Transmission Control Protocol, 전송 제어 프로토콜)

- 단점

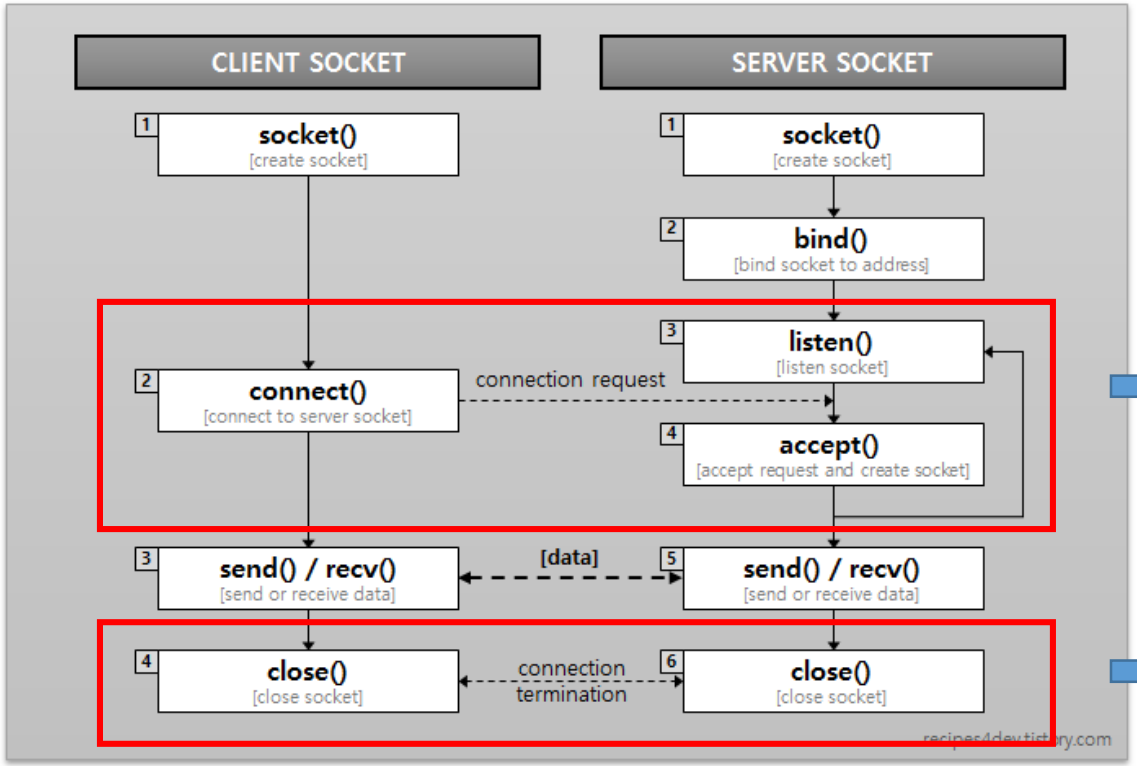
1. TCP는 상당히 많은 대역폭을 사용하며, UDP보다 속도가 느림
2. 전송 중에 소량의 데이터라도 손실되면 TCP는 다른 정보를 로드하지 못할 수 있음.
   예를 들어, 페이지에서 이미지나 동영상과 같은 한 요소가 로드되지 않으면
   나머지 페이지 데이터도 로드되지 않을 수 있음.
   *Zero Window 발생 시 미송신하기 때문에

*Zero Window : 수신자의 Socket Buffer가 꽉찰것같은 경우 수신자는 Window size를 0으로 채워 보냄.
*재전송 타이머 만료 후에도 확인 응답을 받지 못한경우 세그먼트를 재전송하고 재전송 타이머는 2배로 증가한다.
보통 최대 5회로 재전송 시도 후 5회 이상 모두 실패 시 전송오류 발생
*재전송타이머는 RTT에 따라 정해짐. 튜닝가능.

TCP 데이터 단위: segment
IP 데이터 단위: Packet
Socket 데이터 단위: Stream

# TCP (Transmission Control Protocol, 전송 제어 프로토콜)

- TCP 소켓 처리 흐름도

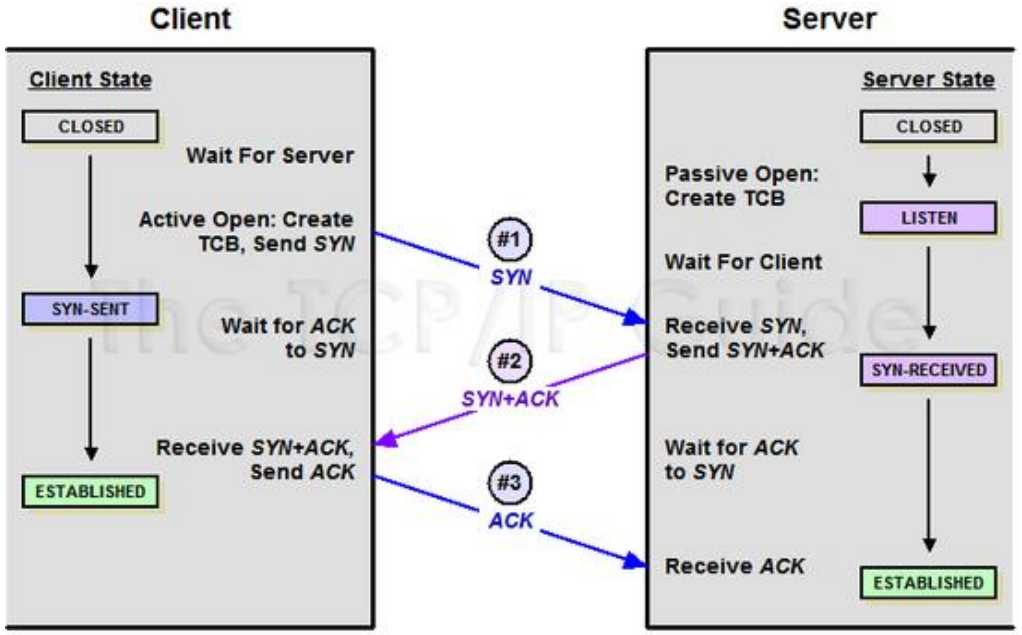



3Way Hand Shaking

4Way Hand Shaking

# TCP (Transmission Control Protocol, 전송 제어 프로토콜)

- 연결방식 (3Way Hand Shaking)



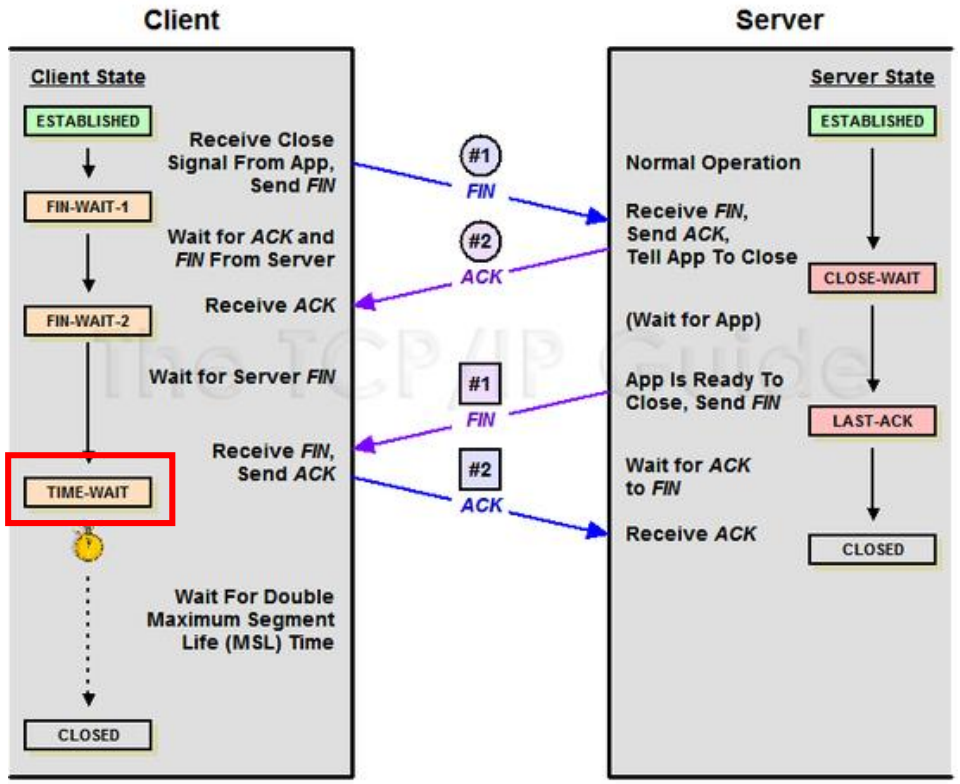

Transmission Control Block (TCB): 네트워크에서 TCP 연결의 상태를 관리하기 위해 사용되는 데이터 구조

SYNC : 시퀀스 넘버 (고유번호라 생각하면 됨)
보안의 이유로 랜덤으로 32bit 결정

클라이언트 : SYN(1000) 송신 (자신의 시퀀스 넘버 송신)
서버: SYN(4000) + ACK(1001) 송신 (자신의 시퀀스 넘버 송신 + ACK)
클라이언트 : ACK (4001) 송신

# TCP (Transmission Control Protocol, 전송 제어 프로토콜)

- 종료방식 (4Way Hand Shaking)




*종료를 요청한 쪽에서 Time Wait가 걸림
서버에서 time wait가 생기면,
굉장한 딜레이+ 소켓낭비가 생길 수 있으니
항상 클라이언트에서 종료요청 하는 식으로 설계 필요

# TCP 서버_1.초기화

```
WSADATA m_wsa;
```

```
bool ServerSocket::init()
{

    m_listClient.clear();       //클라이언트 리스트

    //윈속 초기화
    m_wsa = { 0 };
    if (::WSAStartup(MAKEWORD(2, 2), &m_wsa) != 0)
    {
        puts("윈속 초기화 불가");
        return false;
    }
    return true;


}
```

WSADATA : Windows Socket Address
WSAStartup : wsaData 초기화 함수
Parameter 1 : 버전
Parameter 2 : wsaData

MAKEWORD : 윈속 버전 지정 매크로

Return : OK (0) / SOCKET_ERROR (-1)

| 플랫폼 | 윈속 버전 |
|---|---|
| Windows 95 | 1.1 (2.2) |
| Windows 98 | 2.2 |
| Windows Me | 2.2 |
| Windows NT 4.0 | 2.2 |
| Windows 2000 | 2.2 |
| Windows XP | 2.2 |
| Windows CE | 1.1 |

# TCP 서버_2.소켓생성

```
#define AF_UNSPEC       0               // unspecified
#define AF_UNIX         1               // local to host (pipes, portals)
#define AF_INET         2               // internetwork: UDP, TCP, etc.
#define AF_IMPLINK      3               // arpanet imp addresses
#define AF_PUP          4               // pup protocols: e.g. BSP
#define AF_CHAOS        5               // mit CHAOS protocols
#define AF_NS           6               // XEROX NS protocols
#define AF_IPX          AF_NS           // IPX protocols: IPX, SPX, etc.
#define AF_ISO          7               // ISO protocols
#define AF_OSI          AF_ISO          // OSI is ISO
#define AF_ECMA         8               // european computer manufacturers
#define AF_DATAKIT      9               // datakit protocols
#define AF_CCITT        10              // CCITT protocols, X.25 etc
#define AF_SNA          11              // IBM SNA
#define AF_DECnet       12              // DECnet
#define AF_DLI          13              // Direct data link interface
#define AF_LAT          14              // LAT
#define AF_HYLINK       15              // NSC Hyperchannel
#define AF_APPLETALK    16              // AppleTalk
#define AF_NETBIOS      17              // NetBios-style addresses
#define AF_VOICEVIEW    18              // VoiceView
#define AF_FIREFOX      19              // Protocols from Firefox
#define AF_UNKNOWN1     20              // Somebody is using this!
#define AF_BAN          21              // Banyan
#define AF_ATM          22              // Native ATM Services
#define AF_INET6        23              // Internetwork Version 6
#define AF_CLUSTER      24              // Microsoft Wolfpack
#define AF_12844        25              // IEEE 1284.4 WG AF
#define AF_IRDA         26              // IrDA
#define AF_NETDES       28              // Network Designers OSI & gateway
```

```
SOCKET m_hSocket;                       //서버의 리슨 소켓 (접속대기)
```

```
bool ServerSocket::createSocket()
{
    //접속대기 소켓 생성
    m_hSocket = ::socket(AF_INET, SOCK_STREAM, 0);
    if (m_hSocket == INVALID_SOCKET)
    {
        puts("ERROR: 접속 대기 소켓을 생성할 수 없습니다.");
        return false;
    }
    return true;
}
```

Socket :소켓 생성 함수

Parameter 1 : 주소 패밀리 사양
L3 에서는 IP

Parameter 2 : 타입
L4 에서는 SOCK_STREAM (TCP)
SOCK_DGRAM (UDP)

Parameter 3 : 프로토콜
*값이 0인 경우 호출자는 프로토콜을 지정하지 않으며
서비스 공급자는 사용할 *프로토콜* 을 선택합니다.

```
#define SOCK_STREAM     1               /* stream socket */
#define SOCK_DGRAM      2               /* datagram socket */
#define SOCK_RAW        3               /* raw-protocol interface */
#define SOCK_RDM        4               /* reliably-delivered message */
#define SOCK_SEQPACKET  5               /* sequenced packet stream */
```

# TCP 서버_3.포트 바인딩

```
bool ServerSocket::bindPort(int port)
{
    //포트 바인딩
    SOCKADDR_IN svraddr = { 0 };
    svraddr.sin_family = AF_INET;
    svraddr.sin_port = htons(port);
    svraddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
    if (::bind(m_hSocket,
        (SOCKADDR*)&svraddr, sizeof(svraddr)) == SOCKET_ERROR)
    {
        puts("ERROR: 소켓에 IP주소와 포트를 바인드 할 수 없습니다.");
        return false;
    }
    return true;
}
```

intel : little endian 사용
ex) INADDR_ANY, 포트번호 다 little endian

network : big endian 사용

htons / htonl : littel -> big 으로 변환하는 함수

*빅 엔디안 : 저장 시 큰 쪽을 먼저 저장하는 방식
*리틀 엔디안 : 저장 시 작은 쪽을 먼저 저장하는 방식

# TCP 서버_4.소켓 리슨

```
bool ServerSocket::listenSocket()
{
    //접속대기 상태로 전환
    if (::listen(m_hSocket, SOMAXCONN) == SOCKET_ERROR)
    {
        puts("ERROR: 리슨 상태로 전환할 수 없습니다.");
        return false;
    }
    return true;
}
```

Listen 함수 : 서버가 closed -> 열리면 Listen 이 되는데,
그 상태를 만들어 주는 함수

Parameter 1 : 상태변환 할 소켓 핸들

Parameter 2 : 백로그
listen -> accept 하는 사이에 다른 client의 요청이 올 수 있다.
그럴 경우 백로그가 없으면 접속 요청 자체가 날라가서 접속이 불가했다.
옛날엔 보통 5정도 넣어서 했다.
* 요즘은 OS에서 알아서 처리하므로, 신경쓸필요 없음

# TCP 서버_5.클라이언트 접속 처리 및 대응

```
void ServerSocket::acceptClient()
{
    //클라이언트 접속 처리 및 대응
    SOCKADDR_IN clientaddr = { 0 };     //remote info, IP, Port
    int nAddrLen = sizeof(clientaddr);
    SOCKET hClient = 0;                 //remote Client와 통신하는 소켓
    DWORD dwThreadID = 0;               //쓰레드 ID
    HANDLE hThread;                     //쓰레드 핸들

    //클라이언트 연결을 받아들이고 새로운 소켓 생성(개방)
    while ((hClient = ::accept(m_hSocket,
        (SOCKADDR*)&clientaddr, &nAddrLen)) != INVALID_SOCKET)
    {
        if (AddUser(hClient) == FALSE)
        {
            puts("ERROR: 더 이상 클라이언트 연결을 처리할 수 없습니다.");
            //서버 종료 처리 해야함
            //CtrlHandler(CTRL_C_EVENT);
            break;
        }

        //클라이언트로부터 문자열을 수신함.
        hThread = ::CreateThread(NULL,  //보안속성 상속
            0,                          //스택 메모리는 기본크기(1MB)
            m_Thread.ThreadConnect,     //쓰레드로 실행할 함수이름
            (LPVOID)hClient,            //새로 생성된 클라이언트 소켓
            0,                          //생성 플래그는 기본값 사용
            &dwThreadID);               //생성된 스레드ID가 저장될 변수주소
        ::CloseHandle(hThread);
    }
}
```

Accept 함수 : 클라이언트를 수락하는 함수

Parameter 1 : 상태변환 할 소켓 핸들

Parameter 2 : 백로그

listen -> accept 하는 사이에 다른 client의 요청이 올 수 있다.
그럴 경우 백로그가 없으면 접속 요청 자체가 날라가서 접속이 불가했다.
옛날엔 보통 5정도 넣어서 했다.
* 요즘은 OS에서 알아서 처리하므로, 신경쓸필요 없음

# TCP 서버_6.클라이언트 접속 처리 및 대응

```
void ServerThread::ThreadConnectImpl(SOCKET hClient)
{

    //클라이언트에게 채팅 메시지 서비스를 제공하는 스레드 함수
    //연결된 각각의 클라이언트마다 한 스레드가 생성

    char cBuff[128] = { 0 };
    int nRcv = 0;

    puts("새 클라이언트가 입장했습니다.");
    while ((nRcv = ::recv(hClient, cBuff, sizeof(cBuff), 0)) > 0)
    {
        puts("1. 클라이언트에게서 메시지 수신");
        puts(cBuff);    //수신한 문자열을 연결된 전체 클라이언트에게 전송
        SendMsg(cBuff);
        memset(cBuff, 0x00, sizeof(cBuff));
    }

    puts("클라이언트가 연결을 끊었습니다.");

    m_cs->Enter();
    //해당 클라이언트 삭제
    g_sock->getListClient().remove(hClient);
    m_cs->Leave();

    ::closesocket(hClient);

}
```

Buff : 수신한 데이터를 받을 버퍼크기
nRcv : 수신한 데이터 크기

Recv를 while로 처리함으로써,
클라이언트로부터 리시브 false 할때까지 진행

Recv == false 는 클라이언트 연결 끊김을 뜻함

# TCP 서버_7.서버 종료

```cpp
void ServerSocket::terminateServer()
{
    /*
    1. 연결된 모든 클라이언트 리슨 소켓 닫기
    2. 리스트 초기화
    3. 클라이언트와 통신하는 스레들 종료되길 기다림.
    : 추후엔 정확한 값 산측해서 기다림.
    : 왜냐면 임의의 값이므로, 종료를 반드시 감지할땐 다른방법을 써야함
    4. 윈속 해제
    */

    //연결된 모든 클라이언드 닫기
    if (IsSocketOpen(m_hSocket))
    {
        ::shutdown(m_hSocket, SD_BOTH);
        m_Thread.m_cs->Enter();
        for (auto it = m_listClient.begin(); it != m_listClient.end(); it++)
        {
            puts("모든 클라이언트 연결 종료");
            ::closesocket(*it);
        }

        //리스트 초기화
        m_listClient.clear();

        m_Thread.m_cs->Leave();     //진행중인 Critical Section이 있을수 있으니 모두 해제

        ::Sleep(100);               //종료 시간 임의 추정

        m_Thread.m_cs->Delete();    //섹션 삭제

        puts("terminateServer 섹션 삭제");

        //소켓 클로즈
        ::closesocket(m_hSocket);

        //윈속 해제
        ::WSACleanup();
    }
    exit(0);
}
```

1. 연결된 모든 클라이언트 리슨 소켓 닫기

2. 리스트 초기화

3. 클라이언트와 통신하는 스레들 종료되길 기다림.
: 추후엔 정확한 값 산측해서 기다림.
: 왜냐면 임의의 값이므로, 종료를 반드시 감지할 땐 다른방법을 써야함

4. 윈속 해제

# TCP 클라이언트_1.초기화 (동일)

```
bool ClientSocket::init()
{
    //윈속 초기화
    m_wsa = { 0 };
    if (::WSAStartup(MAKEWORD(2, 2), &m_wsa) != 0)
    {
        puts("윈속 초기화 불가");
        return false;
    }
    return true;
}
```

WSADATA : Windows Socket Address
WSAStartup : wsaData 초기화 함수
Parameter 1 : 버전
Parameter 2 : wsaData

MAKEWORD : 윈속 버전 지정 매크로

Return : OK (0) / SOCKET_ERROR (-1)

| 플랫폼 | 윈속 버전 |
|---|---|
| Windows 95 | 1.1 (2.2) |
| Windows 98 | 2.2 |
| Windows Me | 2.2 |
| Windows NT 4.0 | 2.2 |
| Windows 2000 | 2.2 |
| Windows XP | 2.2 |
| Windows CE | 1.1 |

# TCP 클라이언트_2.소켓생성 (동일)

```
bool ClientSocket::createSocket()
{
    m_hSocket = ::socket(AF_INET, SOCK_STREAM, 0);
    if (m_hSocket == INVALID_SOCKET)
    {
        puts("에러: 윈속 초기화 불가");
        return false;
    }
    return true;

}
```

# TCP 클라이언트_3.포트 바인딩

```
bool ClientSocket::bindPort(int port)
{
    //포트 바인딩
    SOCKADDR_IN svraddr = { 0 };
    svraddr.sin_family = AF_INET;
    svraddr.sin_port = htons(PORT);
    svraddr.sin_addr.S_un.S_addr = inet_addr(SERVER_IP);

    if (::connect(m_hSocket, (SOCKADDR*)&svraddr, sizeof(svraddr)) == SOCKET_ERROR)
    {
        puts("에러: 서버에 연결할 수 없습니다.");
        return false;
    }
    return true;

}
```

inet_addr : 문자를 숫자로 변경하는 함수

*client port 지정 안할 시 OS가 임의로 포트를 열어줌.

# TCP 클라이언트_4. 데이터 송신

```
void ClientSocket::sendMsg()
{
    char cBuff[128];
    puts("채팅을 시작합니다. 메시지를 입력하세요");
    while (true)
    {
        //사용자로부터 문자열을 입력 받는다.
        memset(cBuff, 0x00, sizeof(cBuff));
        gets_s(cBuff);
        if (strcmp(cBuff, "EXIT") == 0)
        {
            break;
        }

        //사용자가 입력한 문자열을 서버에 전송
        ::send(m_hSocket, cBuff, strlen(cBuff) + 1, 0);
    }
}
```

# TCP 클라이언트_5.서버 데이터 수신 처리

```
DWORD WINAPI ClientThread::ThreadReceive(LPVOID pParam)
{
    //서버가 보낸 메시지를 수신하고 화면에 출력하는 쓰레드
    SOCKET hSocket = (SOCKET)pParam;
    char cBuff[128] = { 0 };

    std::string name = "";
    name = "상대방";
    while (::recv(hSocket, cBuff, sizeof(cBuff), 0) > 0)
    {
        printf("%s:\t%s\n", name.c_str(), cBuff);
        memset(cBuff, 0, sizeof(cBuff));
    }

    puts("수신 스레드가 끝났습니다");

    return 0;

}
```

쓰레드로 수신 처리를 따로 진행

# TCP 클라이언트_6.종료

```
void ClientSocket::terminateClient()
{
    /*
    1. 연결된 모든 클라이언트 리슨 소켓 닫기
    2. 리스트 초기화
    3. 클라이언트와 통신하는 스레들 종료되길 기다림.
    : 추후엔 정확한 값 산측해서 기다림.
    : 왜냐면 임의의 값이므로, 종료를 반드시 감지할땐 다른방법을 써야함
    4. 윈속 해제
    */

    //연결된 모든 클라이언드 닫기
    if (IsSocketOpen(m_hSocket))
    {

        //7.소켓을 닫고 종료
        ::closesocket(m_hSocket);

        //스레드가 종료될 수 있도록 잠시 기다려줌
        //임시로 100. 추후에 종료 시간 계산해보자
        Sleep(100);

        //윈속해제
        WSACleanup();
    }

    exit(0);

}
```