**네트워크 게임 프로그래밍**

**레포트**

**게임공학과**

**2017180010**

**민혜원**

1. 프로젝트 명 : 대전 오목 게임

2. 주요 기능 : 방 참가 기능, 승패 판정 기능(서버)

3. 구현 내용

- 멀티 스레드 TCP 서버를 이용해 알맞은 비밀번호를 입력하면 게임 방에 참가할 수 있게 한다.

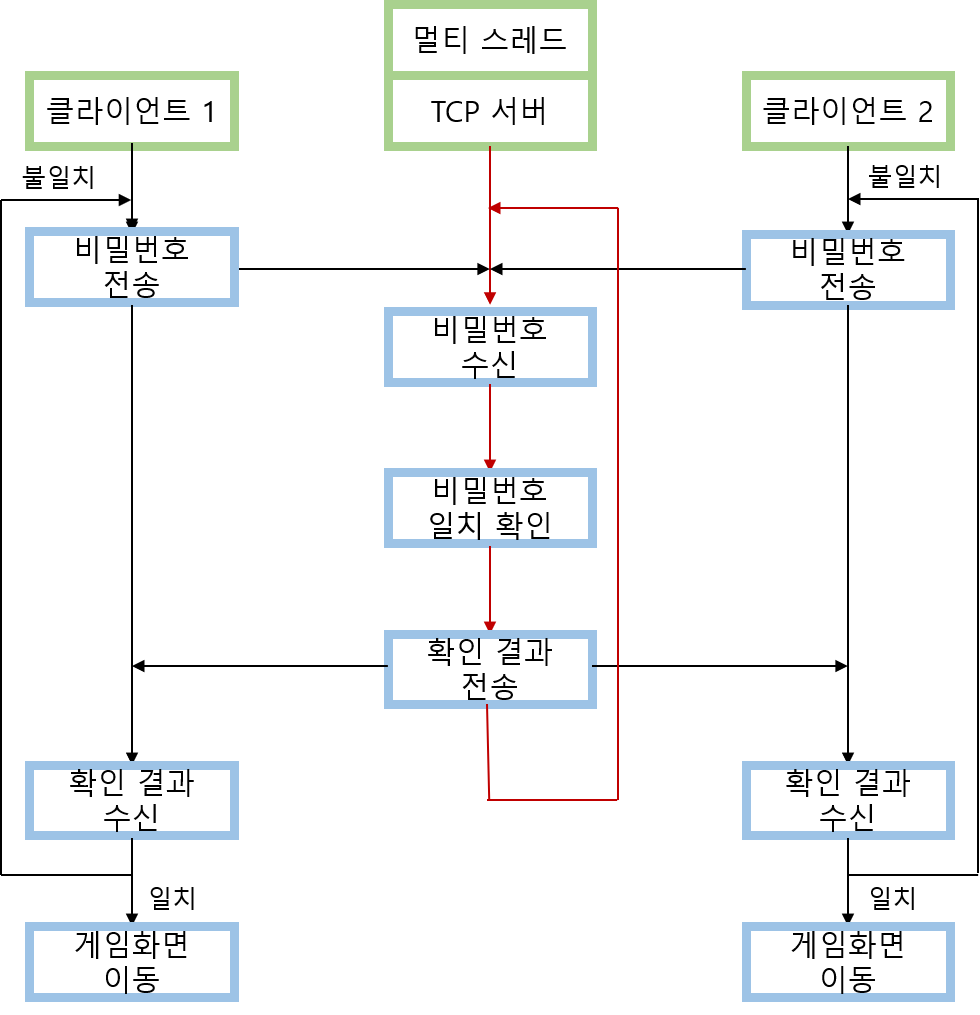
- 게임이 시작되면 멀티 캐스트를 이용해 클라이언트 통신과 승패 판정 통보를 한다.

4. 파일 git 주소 : https://github.com/mwon0326/Net.git

5. 상세 정보

1) 방참가 (서버 : Project\_TCP\_Server – login\_Server.cpp, 클라이언트 : Project\_Client – login.cpp)

**(1) 구조**



**(2) 구현 내용 설명**

- 멀티 스레드 기반 TCP 서버-클라이언트 구조를 이용해 게임 방에 참가하는 부분을 구현했다.

- TCP 프로토콜을 사용한 이유 : 비밀번호를 전송하고 결과를 수신하는 역할만 하기 때문에 빠른 속도가 필요없고, 신뢰성이 높아야 되기 때문에 TCP 프로토콜을 사용했다.

- 멀티 스레드 기반이기 때문에 두 클라이언트에게서 데이터를 입력하고 각각의 처리 결과를 전송한다.

- 클라이언트는 비밀번호를 전송하고 결과가 불일치일 때만 다시 비밀번호를 입력받아 서버에 전송한다. 결과가 일치하면 클라이언트는 게임 화면으로 넘어간다.

- 사용한 소켓 옵션

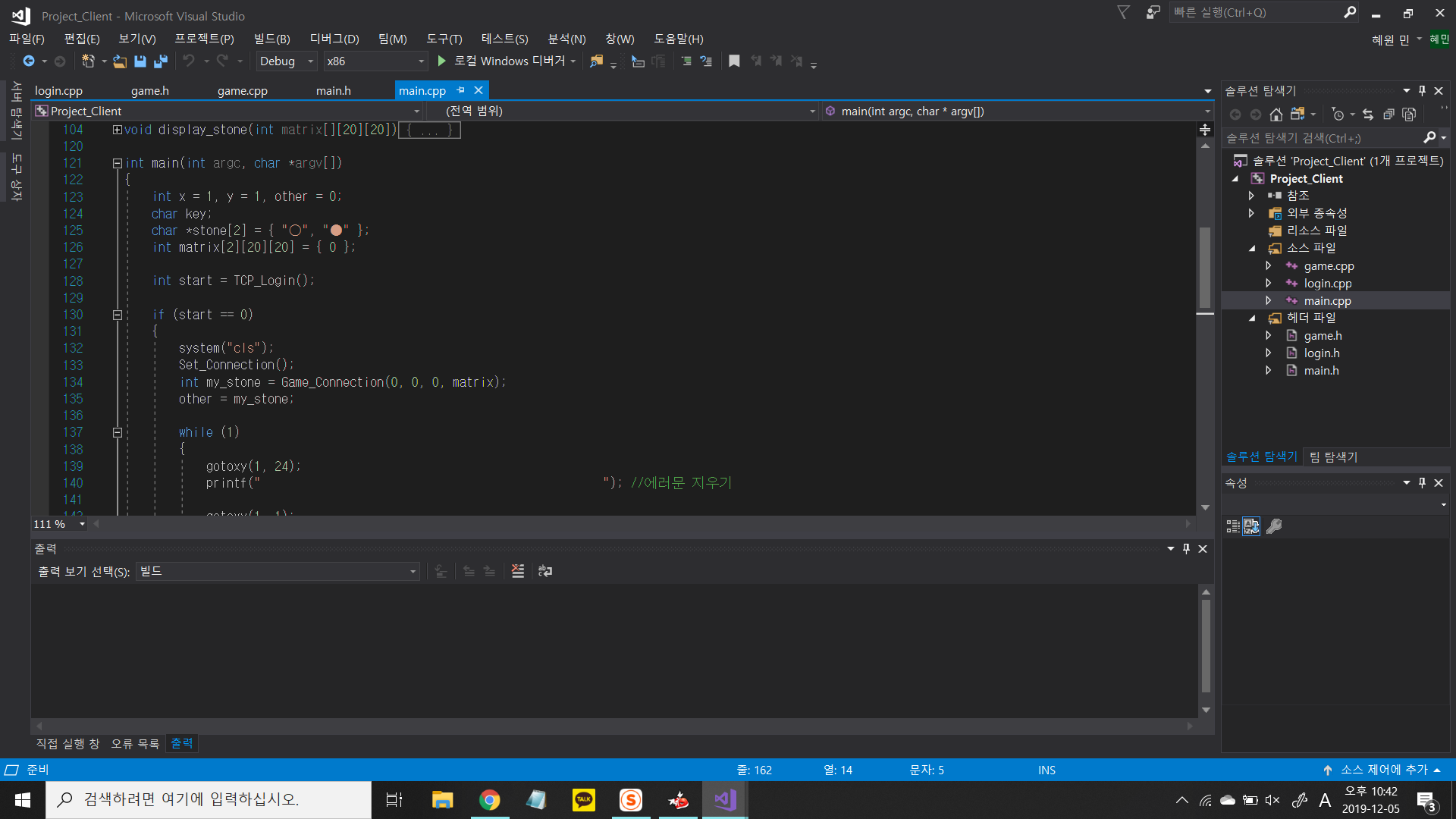
=> 서버 : SO\_SNDBUF, 클라이언트 : SO\_RCVBUF

=> 서버는 전송받은 비밀번호가 올바른지 아닌지만 판단해 참/거짓만 반환하면 된다. 따라서 서버는 전송 버퍼의 크기가 1이면 되고, 클라이언트는 수신버퍼의 크기가 1이면 되기 때문에 각각 버퍼 크기를 소켓 옵션으로 설정해준다.

**(3) 소스코드**

**- 클라이언트 (Project\_Client – main.cpp)**

<1> 방 참가를 위해 로그인 통신(TCP\_Login 함수)을 실행한다.

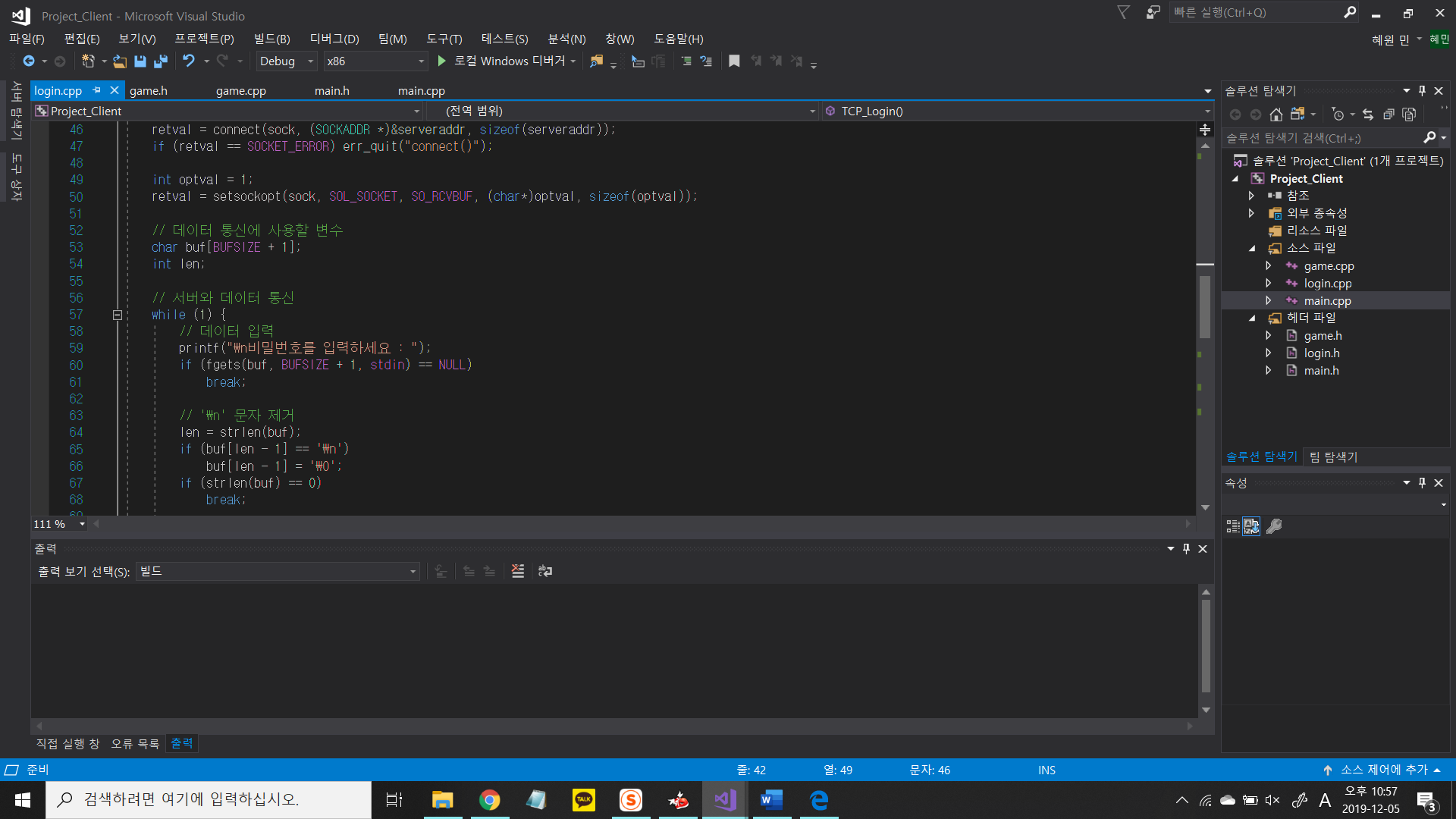


**- 클라이언트 (Project\_Client – login.cpp) : TCP\_Login 함수가 위치한 파일**

<1> 윈속 초기화 -> 소켓 생성( socket()함수 ) -> 서버 접속( connect()함수 )

=> 기존 TCP\_Client 예제 활용

<2> 소켓 옵션 설정 : SO\_RCVBUF 옵션을 설정해 수신 버퍼의 크기를 1바이트로 지정한다.

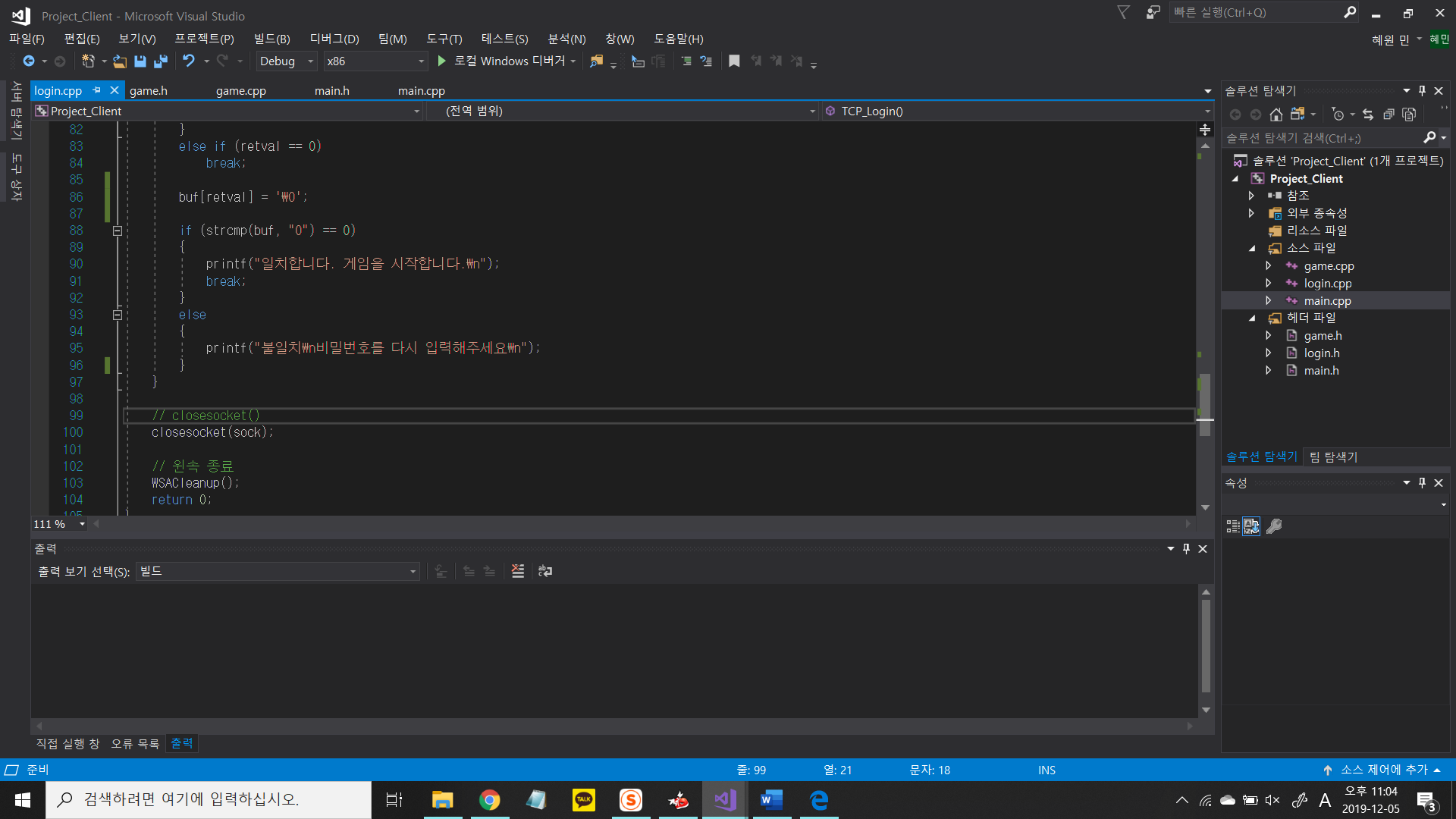


<3> 비밀번호를 입력받기 -> 받은 문자열 데이터 마지막에 ‘\n’은 ‘\0’으로 바꾸기 -> 데이터 보내기( send()함수 ) -> 데이터 받기

=> 기존 TCP\_Client 예제 활용

=> loop문 내에서 이루어지고 있다.

<4> 비밀번호 확인 : 서버에서 받은 값이 ‘0’이면 게임을 시작하기 위해 loop문을 빠져나오고, 그외의 값을 받으면 다시 loop문 처음으로 돌아가 비밀번호를 다시 입력한다.



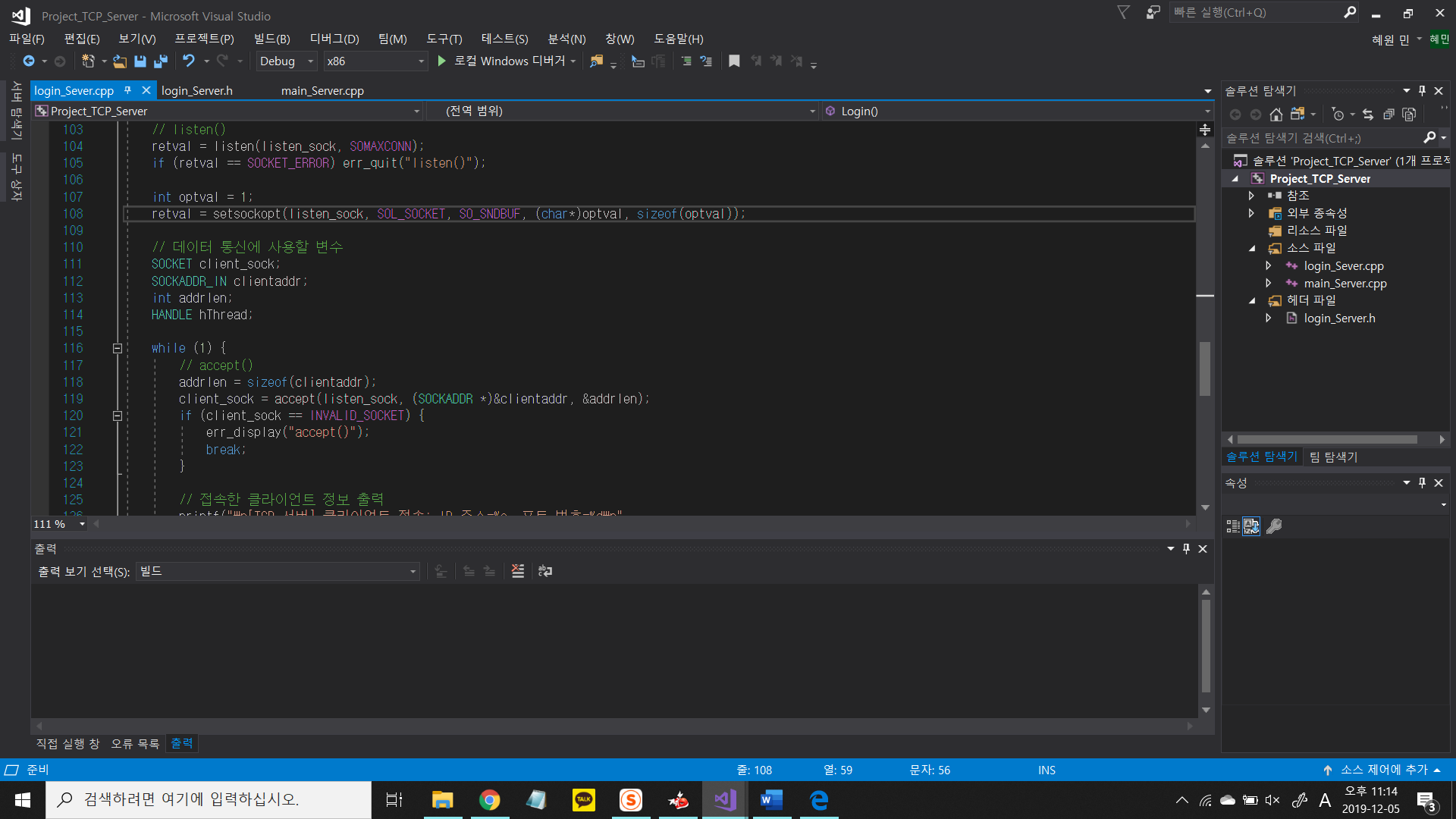
<5> loop문을 빠져나왔으면 소켓 닫기( closesocket()함수 ) -> 윈속 종료( WSACleanup()함수 )

**- 서버 (Project\_TCP\_Server – login\_Server.cpp)**

<1> 윈속 초기화 -> 소켓 생성 -> 지역 IP/port 번호 결정 -> listen상태로 변경( listen()함수 )

=> 기존 MultiThread\_TCP\_Server 예제 활용

<2> 소켓 옵션 설정 : SO\_SNDBUF 옵션을 설정해 전송 버퍼의 크기를 1로 지정한다.



<3> 원격 IP/port 번호 결정( accept()함수 ) -> 스레드 생성

=> loop문 내에서 실행된다.

=> 기존 MultiThread\_TCP\_Server 예제 활용

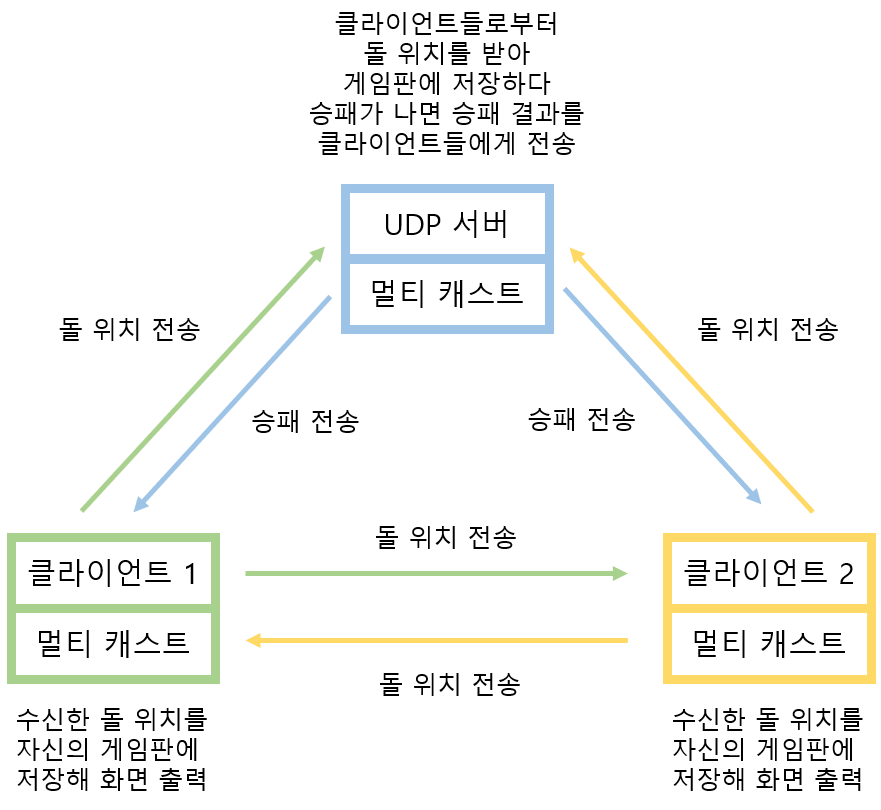
<4> 스레드 내에서 실행되는 클라이언트 데이터 통신

=> 기존 MultiThread\_TCP\_Server 예제 활용

=> 받은 데이터를 서버 내에 지정된 비밀번호와 비교해 맞으면 “0”, 틀리면 “1”을 전송 버퍼에 저장하는 부분만 추가

2) 게임 화면

**(1) 구조**



**(2) 구현 내용 설명**

- UDP 기반 멀티 캐스트를 사용해 게임 실행을 구현했다.

- UDP 프로토콜을 사용한 이유 : 대전 형식의 게임이기 때문에 전송한 결과가 빠르게 대전 상대에게 가야 되기 떄문에 속도가 빠른 UDP 프로토콜을 사용했다.

- 멀티 캐스트 방식을 사용한 이유 : 클라이언트1이 놓은 돌의 데이터는 클라이언트2에게도 전송되어야 하지만 그 사이에 굳이 서버를 거쳐야 될 필요성을 느끼지 못했다. 멀티 캐스트로 수신하는 소켓과 전송하는 소켓을 한번에 구현해 데이터를 주고 받는다면 빠르게 데이터를 주고 받을 수 있다.

- 서버의 의미 : 클라이언트끼리 돌의 데이터를 주고받을 수 있다면 서버가 의미 없을 수 있다. 클라이언트에서 승패 판정을 하게 될 경우 각자 데이터가 입력될 때마다 승패판정을 해야되기 떄문에 오래 걸린다. 따라서 클라이언트들은 돌의 데이터들만 주고 받고, 돌의 데이터가 추가될 때마다 판단하는 역할은 서버에게 맡긴다. 서버는 클라이언트들로부터 데이터를 받기만 하다 승패가 났을 때만 승패 결과를 클라이언트들에게 멀티캐스트로 뿌린다.

- 사용한 소켓 옵션

=> 서버, 클라이언트 : IP\_ADD\_MEMBERSHIP(멀트캐스트 그룹 가입), SO\_REUSEADDR(지역 주소 재사용)

- 가장 처음 서버와 통신해 서버는 클라이언트에게 흑돌과 백돌을 지정해준다.

**(3) 소스코드**

\* 현재 게임 화면의 소스코드는 정상적으로 돌아가지 않는다.

=> 이유 : 멀티캐스트의 수신부분과 송신부분을 한 곳에 구현했을 때, 수신부분과 송신부분을 각각 다른 프로세스에서 돌려야 한다. 송신은 플레이어로부터 데이터를 입력받은 순간에만 이루어지면 되지만, 수신은 데이터가 올 때까지 loop문을 돌면서 기다려야 되기 때문이다. 하지만 프로세스를 새로 생성해 돌리는 방법을 알지 못해 구현하지 못했다.

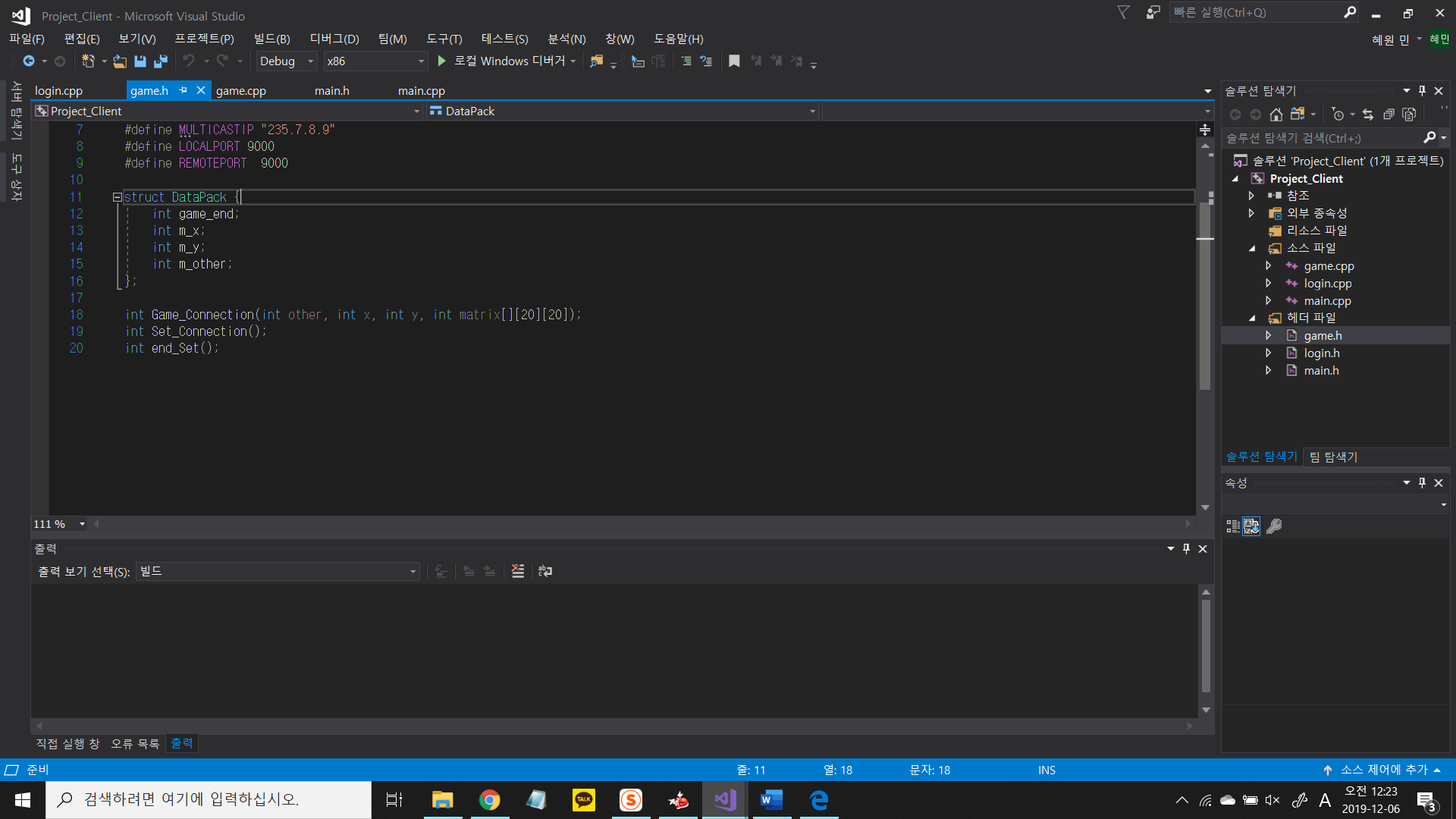
=> 서버라 칭했지만 서버와 클라이언트는 둘다 멀티캐스트로 데이터를 뿌리고, 뿌린 데이터를 받기 때문에 통신 구조가 동일하다.

=> 클라이언트들이 번갈아가며 돌을 놓는 부분을 구현하지 못해 임의로 서로 번갈아가며 돌을 놓는다 가정한다.

**- 공통 통신 부분(소켓 생성 및 멀티 캐스트 그룹 가입, 주소 할당)**

**서버 (Project\_UDP\_Server – UDP\_Server.cpp), 클라이언트 (Project\_Client – game.cpp)**

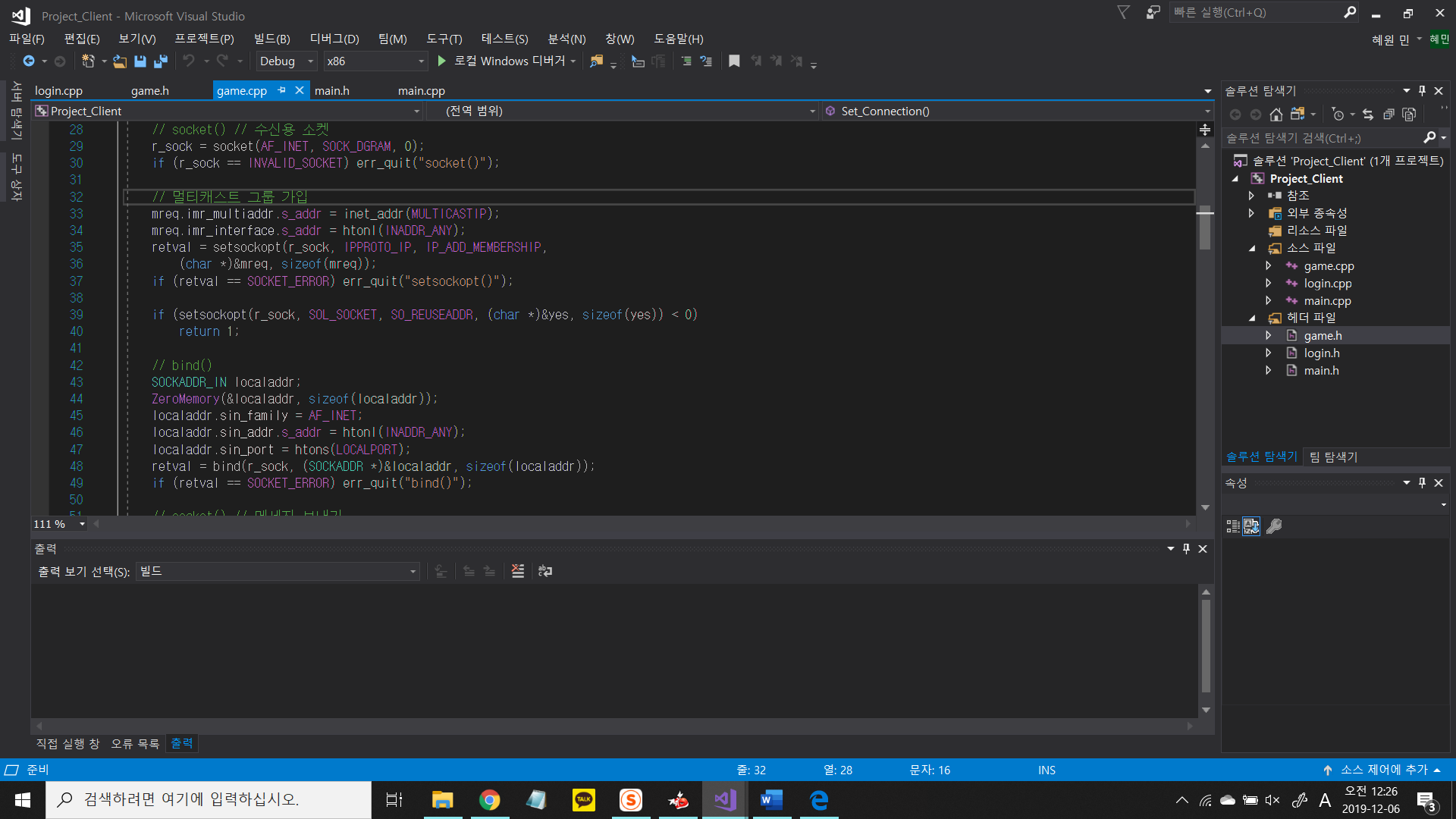
<1> 송수신하는 데이터는 구조체를 사용한다.



<2> 윈속 초기화 -> 소켓 주소체 초기화 -> 수신용 소켓 생성 -> 멀티 캐스트 그룹 가입

=> MultiCast 예제를 활용

<3> 소켓 옵션 설정 : SO\_REUSEADDR 옵션을 설정해 수신용 소켓을 재활용한다.



<4> 지역 IP/port 번호 결정 -> 송신용 소켓 생성

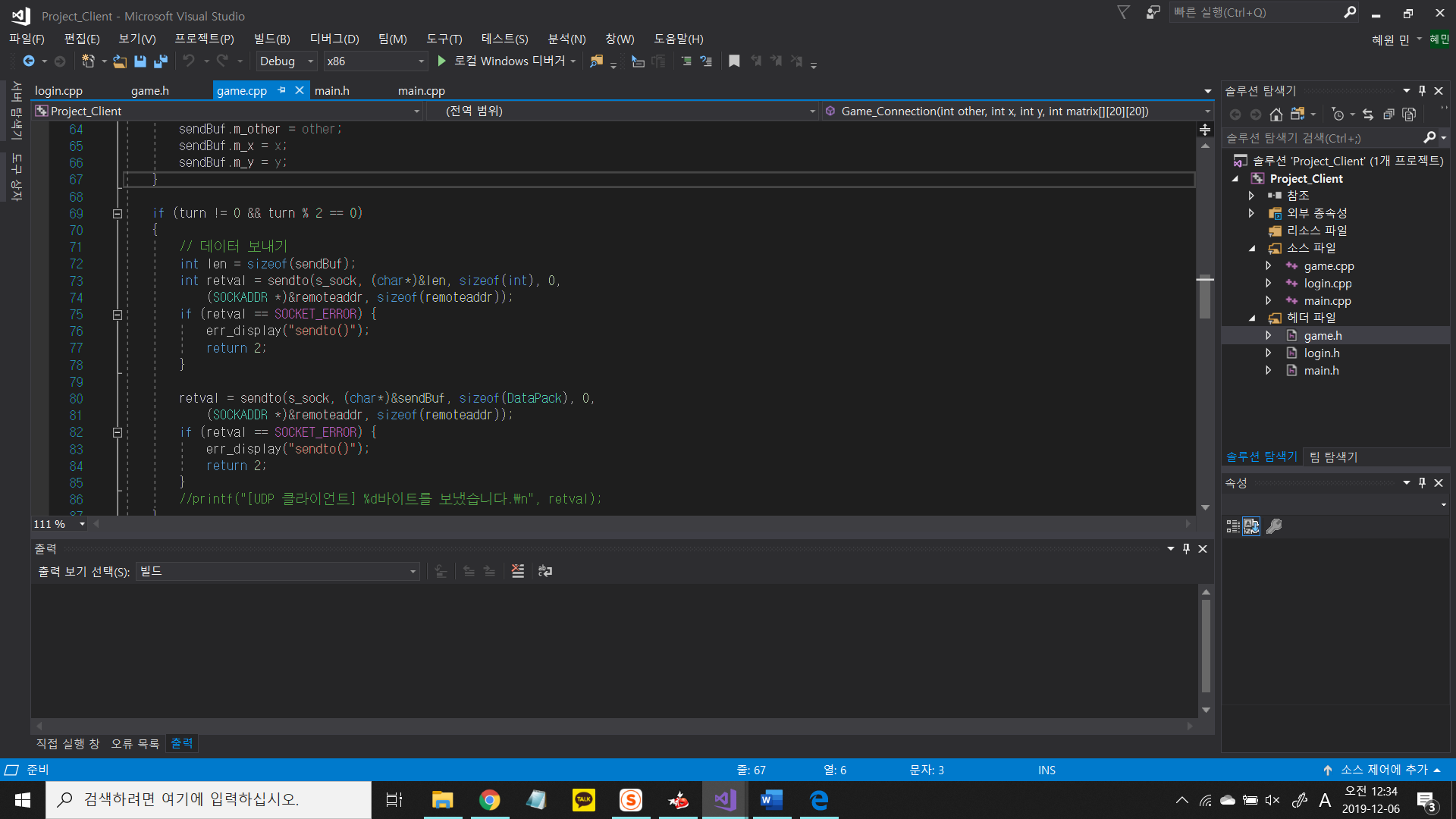
=> MultiCast 예제를 활용

=> 송신용 소켓은 멀티캐스트 데이터를 뿌리는 소켓이고, 수신용 소켓은 뿌린 데이터를 받는 소켓이다.

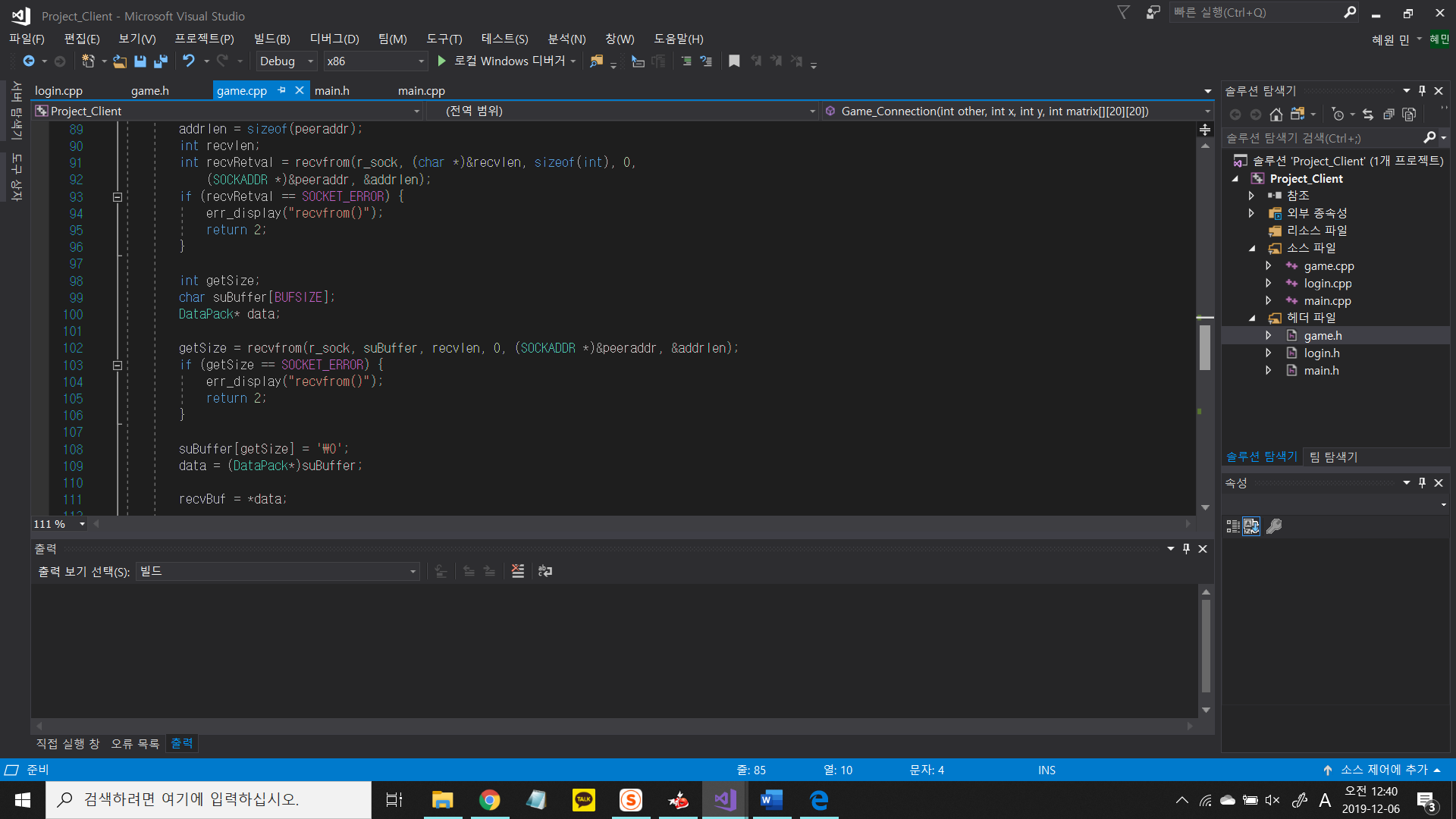
=> <4>번까지는 한 번만 실행

**- 공통 통신 부분 (데이터 송신, 수신)**

<1> 데이터 송신 : 구조체 데이터를 보내기 위해 고정 데이터(구조체 크기에 대한 정보)를 보내고 가변 데이터(구조체)를 보낸다.



<2> 데이터 수신 : 수신과 마찬가지로 고정 데이터(데이터 크기)를 먼저 받고, 가변 데이터(문자열)을 받는다.



=> 받은 문자열을 구조체 포인터로 변환해 저장한다.

- 공통 통신 부분(멀티캐스트 그룹 탈퇴, 소켓 종료, 윈속 종료)

=> 한 번만 실행

=> 소켓을 반환할 때 수신용 소켓과 송신용 소켓을 둘 다 반환하기 위해 closesocket()을 두번 실행한다.

=> MultiCast 예제를 활용

**- 클라이언트 부분 (Project\_Client – main.cpp)**

<1> 멀티캐스트 그룹 가입등 공통 통신부분1을 실행 -> 한 번 서버와 통신해 흑돌, 백돌을 지정받음

<2> 바둑판 출력 -> 플레이어에게 돌을 입력받음 -> 통신부분2로 통신해 내가 놓은 돌의 데이터를 전송함 -> 상대편이 놓은 데이터를 전송 받음 -> 데이터 구조체 중 end\_game 변수에 1또는 2가 들어가면 승패가 난 것이므로 공통 통신부분3에 멀티캐스트 그룹 탈퇴 및 윈속 종료를 실행한다.

**- 서버 부분(Project\_UDP\_Server – UDP\_Server)**

<1> 클라이언트들로부터 데이터를 받아 승패를 판정한다 -> 승패가 판정나면 end\_game 변수에 1(흑돌 승리), 2(백돌 승리)를 넣어 승패를 클라이언트들에게 멀티캐스트로 전송한다. -> 멀티캐스트 그룹 탈퇴 및 윈속 종료한다.