**HYBRID P2P DESIGN assignment #3**

학번 : 201720768 성명 : 김수영

1. 목적

Concurrent 한 방식을 이용해서 UDP 와 TCP를 혼합한 프로그램을 개발한다. Concurrent한 방식을 통해 TCP 연결 방식으로 서버와 client 여러 개가 한번에 통신해서 peer로서 서로 통신할 client의 정보를 얻고, 얻은 정보를 통해 client 끼리 peer to peer 로 UDP 통신을 하도록 만든다. UDP로 연결된 client가 다른 client에게 데이터를 보내면서 동시에 데이터를 받을 수 있도록, thread를 사용한다.

1. 프로그램 설계

**Syntax**:

(1) TCP

-client 에서 server로 보내는 syntax :

server에서 받은 메시지에 대한 명령어를 전송하기 위해서 해당 명령어의 길이만큼 문자열로 전송하는 syntax가 있다. 또 Client가 server에 자신의 정보를 등록하기 위해서 보내는 syntax는 ‘client(number)@ajou.ac.kr’로 이루어진 길이 19의 id 문자배열 ULONG형태의 ip 주소 USHORT형태의 포트 넘버로 이루어져 있는 구조체를 문자열로 변환한 것이다.

-server에서 client로 보내는 syntax :

client의 개인 정보를 물어보는 syntax와 client의 명령어를 얻기 위해서 전송하는 syntax으로 이루어져 있다. 이 syntax는 각 명령어에 해당하는 문자열의 길이만큼 문자열을 client로 전송한다.

Client가 server에 자신의 정보를 등록한 경우 client에게 전송하는 echo message 는 ‘client(number)@ajou.ac.kr’로 이루어진 길이 19의 id 문자배열과 ULONG 형태인4byte의 ip 주소 그리고 USHORT형태의 2byte 포트 넘버로 이루어져 있는 구조체이다. client에게 recv받은 길이 만큼을 문자열의 형태로 전송한다.

Client가 요구하는 해당 client정보를 전송하는 syntax는 위의 구조체 형태와 같다. 전체 client정보가 구조체 배열의 형태로 저장되기 때문에 해당 구조체를 선택해서 다시 문자열의 형태로 전송한다.

(2) UDP

-client 에서 client로 보내는 syntax :

UDP client입장에서 UDP server로 전송하려는 입력 문자열의 길이만큼 문자열 syntax를 전송한다. UDP server는 수신된 syntax를 echo message로 syntax를 재전송한다.

**Semantics:**

**(대부분 semantics에 option을 걸지 않기 때문에 flag는 0으로 고정한다.)**

**(1) TCP**

*-retval = connect(sock, (SOCKADDR\*)&serveraddr, sizeof(serveraddr))*

Client가 server와 TCP통신을 하기 위해서 server에 connect를 맺으려는 try의 semantics로 TCPsocket으로 정해둔 sock를 통해서 이전에 설정해둔 server의 주소로 연결을 한다. 이때 지정해둔 serveraddr의 정보는 portnumber 9000에 aws상에서 정해둔 ip주소인 “15.164.232.220”로 이루어져 있다.

***-*** *retval\_send = send(client\_sock, (char\*)buf, 22,0)*

TCPserver가 TCPclient의 명령어를 얻기 위해서 보내는 목적으로 이전에 생성한 client socket으로 문자열을 전송한다. 전송이 성공했을 때 보낸 해당 메시지의 byte크기를 반환하고, 실패 시 error -1를 출력한다. Client의 입력정보를 얻기 위한 모든 semantics는 다음과 같이 구성된다.

*-msglen = recv(client\_sock, buf, BUFSIZE, 0)*

연결된 client의 주소와 연결된 socket를 통해서 client가 전송한 메시지를 buf에 받아온다. 이때 client가 보내는 메시지의 크기는 가변적이므로 최대 크기 1500으로 BUFSIZE를 정했다. 수신에 성공하면 client가 server로 전송한 전체 메시지 길이가 msglen에 출력되고 수신 실패시 error -1를 출력한다.

*-retval\_send = send(client\_sock, (char\*)&clientInfos[id], msglen,0)*

Client 가 자신의 정보를 등록했을 때 확인 echo message를 보내는 semantics로 연결된 client socket를 통해서 client 정보 구조체 배열 중 해당 구조체를 문자열로 변환시킨 것을 두번째 parameter로 전송 이 부분은 echo message를 전송하는 부분으로 이전에 recv()받은 msglen의 크기만큼 재전송한다.

이때 (char\*)&clientInfos[id]의 id부분은 처음에 client가 server에 id와 password를 적을 때 받아온 값이고, 같은 semantics가 client가 새로 요구하는 다른 client의 정보 숫자에 해당하는 값을 id대신 client\_num으로 넣어준다.

**(2) UDP**

*-retval = sendto(UDPclientsock, buf, len, 0, (SOCKADDR\*)&peerServeraddr, sizeof(peerServeraddr))*

TCP통신에서 server를 통해 얻었던 상대 client 정보인 상대 client의 IP주소와 portnumber를 peerServeraddr에저장해서 이 정보를 기반으로 UDP통신을 하는데 처음에 UDPclient가 UDPserver로 전송하는 semantics로 이전에 정해줬던 UCP client 소켓을 UDPclientsock의 변수로 저장해 이 소켓을 사용해서 입력한 메시지를 담은 문자열 buf를 입력한 메시지 길이 len만큼 이전에 설정해둔 peerServeraddr를 이용해서 peer client 에 전송한다. 마찬가지로 성공하면 보낸 문자열의 길이가 retval 에 출력, 송신에 실패하면 retval이 -1의 값을 가진다.

*-retval = recvfrom(UDPclientsock, buf, len, 0, (SOCKADDR\*)&peeraddr, &addrlen)*

Peer UDPserver로부터 echo 메시지를 받는 부분으로 보냈던 메시지를 다시 받는 것으로 보냈던 메시지 길이인 len 만큼 받아와 buf에 저장한다. 이때 echo message를 받는 부분은 따로 특정 UDPServer에 전송했었던 앞의 semantics와 달리 peeraddr즉 이전에 지정한 임의의 socket address, peer 주소의 앞부분을 가르키는 포인터를 통해서 sizeof peeraddr 즉 addrlen만큼(주소 길이) 받아온다. 수신 성공 시 retval에 전송받은 메시지의 길이가 실패 시 -1 error가 출력된다.

*-retval = recvfrom(UDPserversock, buf, BUFSIZE,0,(SOCKADDR\*)&clientaddr, &addrlen)*

UDP server입장에서 thread로 형성한 함수 내에서 존재하는 semantics로 UDPserver소켓을 통해서 이전에 지정한 임의의 소켓주소 변수인 clientaddr포인터와 그 해당 주소의 길이 addrlen를 통해 메시지를 BUFSIZE만큼 받아와 buf에 저장한다. 수신 성공 시 retval에 UDPclient가 전송한 메시지 길이가 실패 시 -1 error가 출력된다.

*-retval = sendto(UDPserversock, buf, retval, 0, (SOCKADDR\*)&clientaddr, sizeof(clientaddr))*

UDPclient를 통해 받아온 메시지를 echo 하여 다시 전송하는 UDPserver 측면에서 사용하는 semantics로 UDPserver 소켓을 통해서 이전에 메시지를 받아온 client address에게 &clientaddr 포인터를 이용해서 echo message(buf)를 이전에 수신받은 retval 만큼 전송한다. 이때 전송 성공 시 전송한 echo message의 길이 즉 parameter로 쓰인 retval값이 그대로 출력되고, 실패 시 error -1이 출력된다.

**Timing**:

이번에 프로그램에 timing은 적용하지 않았다.

3.프로그램 구현

1. TCP Client 구현(+ UDP client, UDP server구현)

먼저 TCP서버에 등록할 자신의 id와 IP주소 그리고 portnumber를 담을 구조체와 server에서 원하는 상대 client정보를 받아와 저장할 구조체를 생성한다. Main 함수에 TCP 통신을 하기 위해서 *sock = socket(AF\_INET, SOCK\_STREAM,0);*으로 해당 소켓을 생성하고 UDPclient로서 UDP통신을 *하기위해 SOCKET UDPclientsock = socket(AF\_INET, SOCK\_DGRAM,0);*로 소켓을 생성한다. 서버와 TCP통신을 하기 위해서 connect시킬 TCPserver 정보를 설정하고, TCP와 syntax를 주고 받을 부분과 UDPclient로서 UDPserver와 syntax를 주고 받을 부분, UDPserver thread함수를 생성하는 부분을 main 함수에 구현한다.

먼저 UDPserver의 역할을 하는 thread함수를 생성하는 부분인 *hTread = CreateTread(NULL, 0, ProcessServer,NULL, 0, &ThreadId);* 를 실행해서 해당 함수와 밑에 구현된 부분이 동시에 concurrent하게 실행되게 만든다.

UDP통신을 끊기 위해서 QUIT메시지를 전송할 때 까지 전체적으로 while(quit !=1)문으로 감싸고, 제일 상위에 UDPclient의 기준에서 UDPserver에 메시지를 전송하는 부분을 구현하는데, 보내려는 문자열을 buf에 저장한 이후에 *retval = sendto(UDPclientsock, buf, len, 0,(SOCKADDR\*)&peerServeraddr, sizeof(peerServeraddr));*를 통해서 메시지를 전송하고 *retval = recvfrom(UDPclientsock, buf, len,0,(SOCKADDR\*)&peeraddr, &addrleln);*를 통해 UDPserver에서 전송한 echo message를 받아온다.

TCP와 연결을 끊은 option을 선택할 때까지 while(optnum !=3)문으로 *감싸고 msglen = recv(sock, buf, BUFSIZE,0);*를 통해서 TCPserver가 전송한 질문 혹은 명령문들을 받아온다.

각 명령문들에 대해서 대답 혹은 명령문 전송을 *retval = send(sock, (char\*)buf, len, 0);*를 통해서 전송한다. 우선 처음에 TCPserver가 전송한 메시지가 id를 물어보거나 password를 물어보는 부분에 대한 대답 처리를 구현하고, option number 선택 후에 option number를 1로 선택했을 때 현재 IP주소와 해당 client의 특정 port number, id를 가지는 구조체 데이터 client\_Info를 생성해서 *retval = send(sock, (char\*)&client\_Info, 26, 0);*를 통해 26byte 만큼 server에 등록한다. 다음 *msglen = recv(sock, buf, BUFSIZE,0);* 을 통해 confirm message와 echo 데이터를 받아오고 confirm 을 할지 말지를 clientServer에 send()한다.

Client가 옵션 숫자를 2를 선택했다면 server로부터 원하는 cliendID를 묻는 메시지가 오고 이를 if문안에서 원하는 clientID를 send() 그 이후 TCPserver를 통해서 id가 존재하지 않는지, id가 존재한다면 해당 client정보를 받아와 *memcpy(&client\_Info\_Recv, buf, sizeof client\_Info\_Rev);*를 통해 client\_Info\_Recv이름의 구조체에 데이터를 저장하고, 나중에 UDP로 통신할 peerServeraddr의 정보를 저장한다.

옵션 수를 3을 선택했을 때 TCPsocket을 만들었을때의 sock 를 close하고 이후에 UDP의 thread가 닫히고 모두 완료되면 UDPclientsock를 종료한다.

1. TCP Server 구현

먼저 global 변수로 id문자배열과 4byte의 IP주소, 2byte의 portnumber 가지는 구조체 생성해서 client에서 등록할 데이터를 담는 구조체 배열을 선언한다. TCPServer면에서

메인 함수에서 Listen 소켓 생성후에 전체 interface와 bind시키고, server에서 따로 데이터베이스가 저장되어 있다고 설정했기 때문에 client의 id와 password를 저장할 client\_identity 2차원 배열을 생성하고 데이터를 저장한다. Server가 직접 종료할때까지 listen socket를 열어두게 하기 위해서 while문 안에 client\_sock 를 accept()하고 이후에 *hTread = CreateThread(NULL, 0,ProcessClient, (LPVOID)client\_sock, 0, &ThreadId);* 를 통해서 TCP Client와 통신할 부분을 구현하는데 그 이유는 TCPclient가 TCP server와 통신하는데 동시에 각각의 process가 종료되지 않고 통신하게 하기 위해서 concurrent한 방식을 이용하기 위해서이다.

따라서 이를 통해 client의 명령을 concurrent하게 받아드릴 thread함수를 만들고, 데이터를 서로 교환하는 부분을 while(1)문으로 담는다. 먼저 *retval\_send = send(client\_sock, (char\*)buf, 22, 0);*를 통해 각각의 client의 명령문을 받을 메시지를 TCPclient에 전송하는데 3번째 parameter은 전송하려고 하는 메시지의 길이만큼 을 담는다. 각각을 *msglen = recv(client\_sock, buf, BUFSIZE, 0);*를 통해 TCPclient로부터 recv()해서 buf에 저장하고 필요한 데이터를 global 변수 혹은 지역 변수에 저장한다. ID Password를 받아오고 나서 각각을 if문을 통해 검사를 한 후에 option number를 받아오는데 받아온 option\_number을 값을 이용해서 switch(option\_number){case1 : {} ..}를 통해서 각 조건에 해당하는 client의 선택에 따라 TCP client와 데이터 교환을 한다.

Option\_number이 1에 해당하는 경우 *memcpy(&clientInfos[id], buf, sizeof clientInfos[id]);*를 통해서 이전에 선언한 구조체 배열안에 해당하는 id의 데이터를 저장하고, echo message를 *retval\_send = send(client\_sock, (char\*)&clientInfos[id], msglen, 0);*를 통해서 전송한다. 이 부분은 전체적으로 while문으로 묶어서 confirm 에 대한 TCP client의 대답이 ‘yes’일 경우 while문을 나가는 break;를 설정하고 아닐 경우 다시 while문에서 command가 반복된다.

Option\_number가 2일 경우 TCPclient가 원하는 client의 ID를 받아오고 그 ID가 존재하지 않을 경우 존재하지 않는다고, 존재한다면 *retval\_send = send(client\_sock, (char\*)&clientInfos[client\_num], 26, 0);*를 통해서 TCPclient 가 보낸 clientID의 client(client\_num)@ajou.ac.kr에 해당하는 값을 가져와 그 값의 구조체 데이터를 전송한다.

이 thread함수 내에서 client socket와의 연결을 해제하는 close()를 while문 바깥에 구현한다.

TCP가 직접 종료를 할때는 main 함수에서 listen socket를 종료하는 부분을 구현한다.

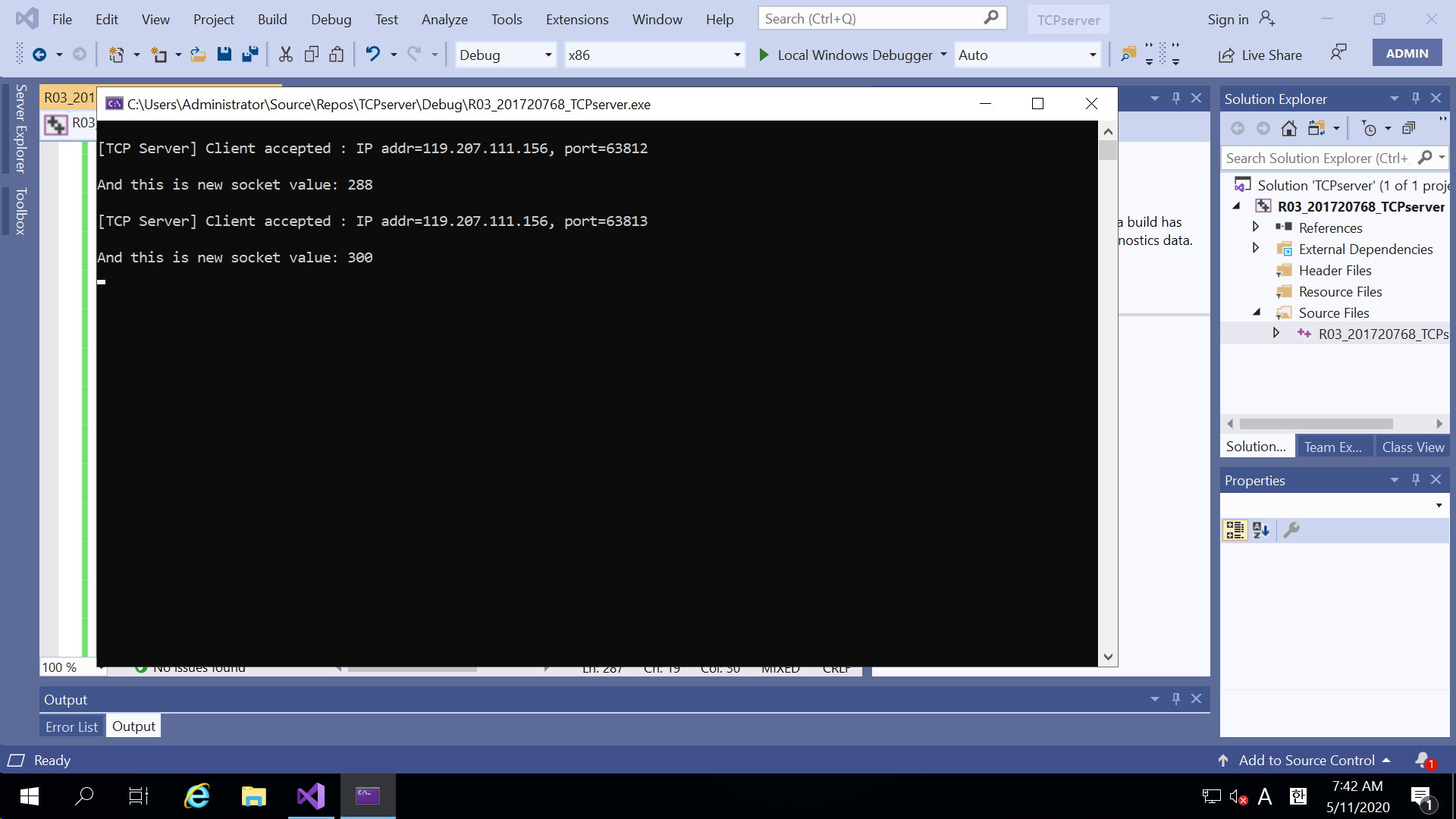
4.동작 결과

1. 동작환경

* OS환경 : windows 10
* 사용한 IDE : Visual studio 2019

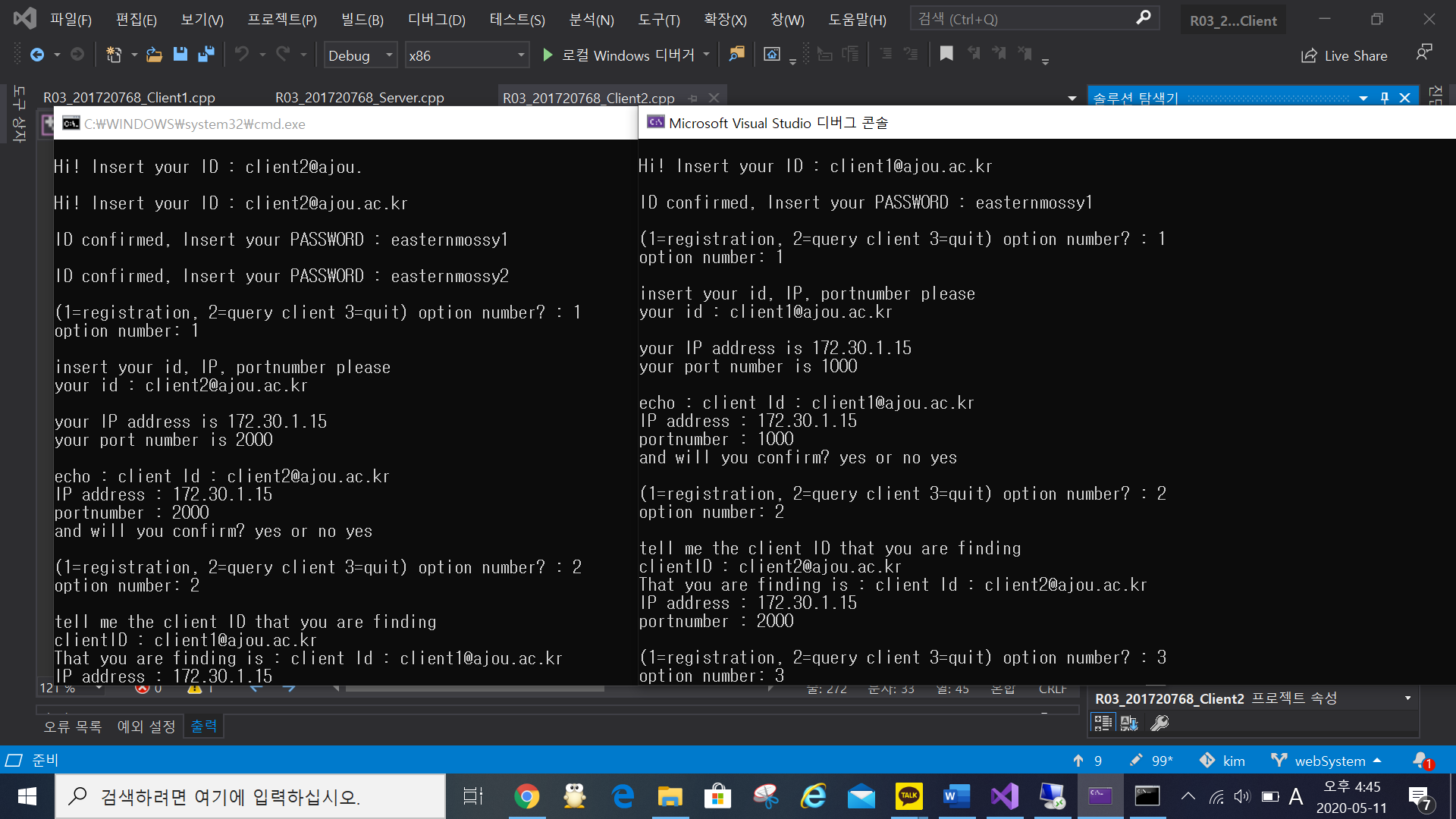
1. 동작결과

1. AWS 상의 TCP server를 먼저 컴파일 🡪 각 여러 개의 Client(여기선 두개)를 컴파일 한다. (AWS상의 TCP server)



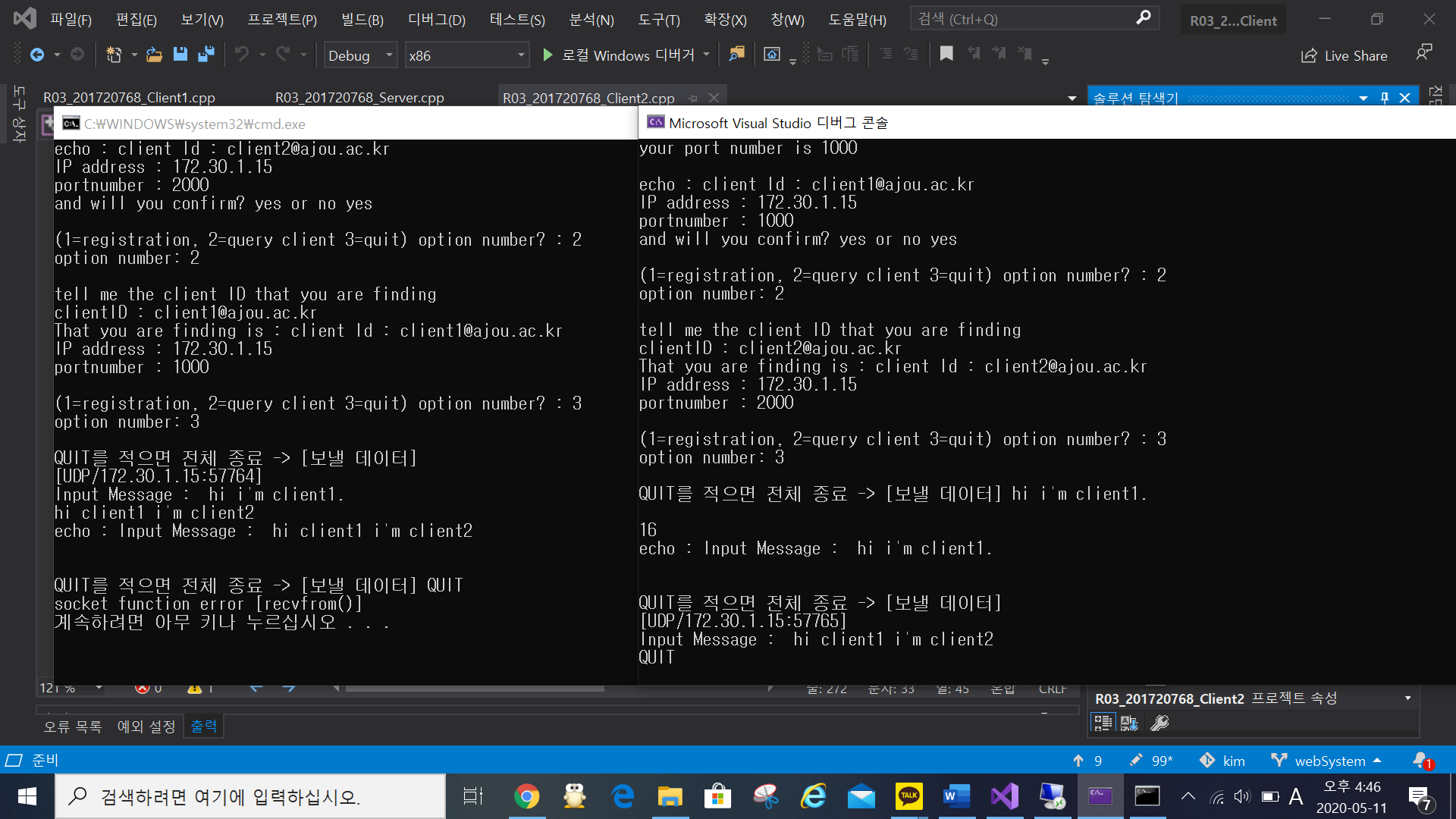
2. 왼쪽은 client2, ID와 password를 각각 한번씩 틀리고 option number를 1번을 선택해서 자신의 id를 직접 입력하고 코드상 IP주소와 portnumber를 받아와서 Server에 등록한다. 동시에 오른쪽 client1에서 같은 방식으로 server에 등록하고 🡪 먼저 등록한 client2는 client1을 query 하기 위해 option number2를 선택 🡪 정보를 얻어온다. 같은방식으로 client1도 client2를 query.

(client2) (client1)

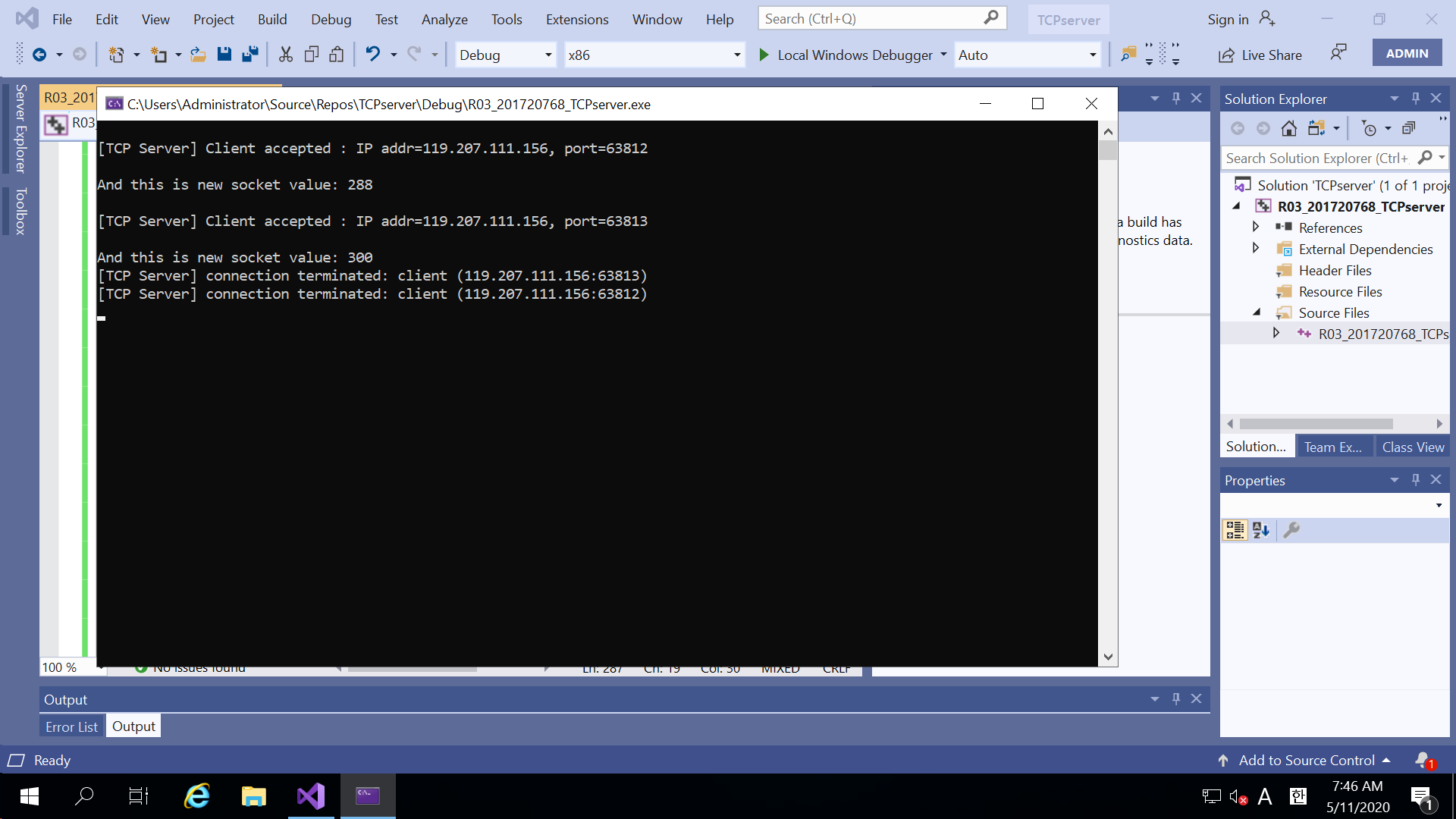


3. 두개의 client를 option number 3을 통해서 종료하면 UDP 통신하는 메시지 입력 command가 뜨고 client1에서부터 먼저 “hi I’m client1”이 전송되어 client2에 표기 client1에 echo message로 표기. Client2도 “hi client1 I’m client2”를 전송해서 이를 반복하는 모습이다. 🡪 두개의 client에서 “QUIT”를 통해 각각 종료

(client2) (client1)



4. client 2개를 모두 option number3를 통해서 tcp 연결 종료를 했을 때의 aws상 server화면 (server)



5. 프로그램 내역

R03\_201720768\_HybridP2P 압축파일 안에

client 부분 소스파일 & exe파일

1. R03\_201720768\_Client.cpp
2. R03\_201720768\_Client.exe

server 부분 소스파일

1. R03\_201720768\_Server.cpp

**<빌드 동작 시 주요사항>**

* 소스파일과 exe파일만 압축
* 링커: ws2\_32.lib, 메니페스트 포함: 아니요, 특정 경고 사용 안함: 4996
* Server는 aws서버에서 컴파일 했기 때문에 client에서

//====================

//나의 aws IP address

//====================

serveraddr.sin\_addr.s\_addr = inet\_addr("15.164.232.220");

이 부분을 바꿔줘야 한다.

* Client 부분에서 서로 udp 통신하기 위해서

//==========================

//현재 내 wifi 의 IP address

//==========================

printf("\nyour IP address is %s", "192.168.0.6");

client\_Info.IP = inet\_addr("192.168.0.6");

이 부분을 현재 wifi 주소로 바꿔줘야 한다.

* Client를 여러 개를 복사해서 실행을 시킬 때 각 port number이 달라야 하기 때문에

//==========================

//client에 해당하는 port number

//===========================

UDPserveraddr.sin\_port = htons(1000);

udpServerPort = htons(1000);

여기서 1000값을 임의로 바꿔줘야 한다.

* Client의 id는 [client1@ajou.ac.kr](mailto:client1@ajou.ac.kr), [client2@ajou.ac.kr](mailto:client2@ajou.ac.kr), …등으로 [client9@ajou.ac.kr](mailto:client9@ajou.ac.kr) 까지 가능하다.
* Client 의 password는 easternmossy1, easternmossy2, …로 각 client뒤에 해당하는 숫자값을 맞춰서 설정했다.

Ex) [client1@ajou.ac.kr](mailto:client1@ajou.ac.kr) 🡪 easternmossy1

[client2@ajou.ac.kr](mailto:client2@ajou.ac.kr) 🡪 easternmossy2

* 모두 option number1 을 통해서 client정보를 등록한 후에 꼭 client에서 option number2를 선택하고 3을 선택해서 tcp종료를 해야 다른 client의 정보와 함께 메시지를 보내는 것이 가능하고, 다른 경우에는 메시지 받는 것이 모두 가능하다.
* 현재 커서의 위치를 무시하고 다른 client로부터 udp 통신을 통해 메시지메 받아 오기 때문에 메시지를 보내는 입력란이 생기고 메시지를 받아올 수 있는데 이는 그냥 무시하고 메시지를 보내도 된다.