Socket Programming: Gomoku Game

2018-14385 권지웅

1. Problem Statement

This project is to understand the basics of socket programming. By implementing both client and server side, we hope to understand how each of the client and server works. I have used basic TCP protocol for communication and multithreading for handling multiple clients. The expected result of this project is to make a simple multi-user Gomoku Game using client-server network. In addition to basic rules few specifications were added, namely:

1. Upon initiation, a client can open a new game room and wait for another client to join or inquire the server if there are any game room to join.

2. The game is determined as "Draw" if more than equal to 50 stones are exchanged.

3. A client lose the game if he/she waits the game more than 60 seconds.

4. A client is regarded as a looser if he/she put stones out of range more than one time.

5. The game start when both players are ready

The server is specified by IP address and a chosen port. Each clients' information should be on the Server while connected. Server and Client is expected to transfer data using protocols. The transfer data format should be a socket as the name of this project. The language could be anything. I used c# on Client, c++ on server. C# supports socket programming by 'System.Net.Sockets' header and threading by 'System.Threading' header. C++ also supports socket programming by sys/socket.h and threading by pthread.h.

The simpe UI is also recommended since this is a game. I used Winform(c#) for UI.

My Server file can be compiled and ran by g++ compiler (g++ -std=c++11 -pthread -o). C# + winform uses Visual Studio. So, the 'build' by VS is highly recommended. Please read README for more information.

2. Implementation

As I mentioned above there are 2 sides in this project: Server (c++) and Client(c#).

0) Protocol

I've used basic TCP(Transmission Control Protocol) for communication.

On C++ in <sys/socket.h> there exists:

ssize_t send(int *socket*, **const void ***_buffer_, **size_t** *length*, **int** *flags*);

ssize_t recv(int socket, void *buffer, size_t length, int flags);

I changed string(which I want to send) into char* by c_str() and used send function to send.

When receiving, I used recv function and formatted the char* data into vector(array) of strings.

The connection is made by bind -> listen -> accept, and then data transfer, which is kind of a common sense.

On C# in Systems.Net.Sockets it supports:

private **TcpClient** tcpClient;

private **NetworkStream** stream;

When transferring data, we change string into bytes by Encoding function and writing it to stream (The stream should connected to TcpClient). The TcpClient then sends it to the server (The opposite is same). It was simpler than c++ which I had to made connection manually.

I used 4.6.1 net framework in this project.

1) Server

The given server is linux server. So the socket programming follows the linux convention. The <netinet/in.h>, <sys/socket.h> header supports linux c++ socket programming.

```
int main() {
    sigignore(SIGPIPE);
    pthread_t thread;
    int status;
    serverSocket = socket(AF_INET, SOCK_STREAM, NULL);
    serverAddr.sin_addr.s_addr = inet_addr("147.46.240.42");
    serverAddr.sin_port = htons(20385);
    serverAddr.sin_family = AF_INET;
    cout << "[ 오목 서버 가동 ]" << endl;
    bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    listen(serverSocket, 32);

    int addressLength = sizeof(serverAddr);
    while (true) {
        int clientSocket = socket(AF_INET, SOCK_STREAM, NULL);
        if (clientSocket = accept(serverSocket, (struct sockaddr*)&serverAddr, (socklen_t *) &addressLength)) {
            Client* client = new Client(nextClientID, clientSocket);
            cout << "[ 새 사용자 접속 ]" << endl;
            pthread_create(&thread, NULL, ServerThread, (void *)client);
            connected.push_back(*client);
            nextClientID++;
        }
    }
    pthread_join(thread, (void**)&status);
}
```

The main function looks like this.

Sigignore is to ignore SIGPIPE signal that terminates server when signaled. (Server should be open. Always)

The specified address is given to inet_addr and prot is given to htons. We then bind the socket with the given addr, listen and accept. We make new thread on new connection and let ServerThread handle the rest. The array connected saves client information. Pthread_join is to get terminated threads.

The ServerThread below mainly receives and handles 5 kinds of message:

Receives: Ask, Enter, Ready, Put, Play.

Sends: Ask, Enter, Put, Play, Exit

```
void* ServerThread(void * client2) {
    struct Client * client = (struct Client *) client2;
    char* sent = new char[256];
    char* received = new char[256];
    int size = 0;
    int board[11][11];
    for (int i = 0; i < 11; i++) {
        for (int j = 0; j < 11; j++) {
            board[i][j] = 0;
        }
    }
    while (true) {
        memset(received, 0, 256);
        if ((size = recv(client->getClientSocket(), received, 256, NULL)) > 0) {
            string receivedString = string(received);
            vector<string> tokens = getTokens(receivedString, ']');
            if(receivedString.find("[Ask]")!= -1){ ... }
            else if (receivedString.find("[Enter]") != -1){ ... }
            else if(receivedString.find("[Ready]")!=-1){ ... }
            else if (receivedString.find("[Put]") != -1){ ... }
            else if (receivedString.find("[Play]") != -1){ ... }
        }
        else{ ... }
    }
```

We receive by recv function and format data into string by getTokens.

```
if(receivedString.find("[Ask]")!= -1) {
    for (int i = 0; i < connected.size(); i++) {
        if(connected[i].getClientSocket() == client->getClientSocket()){
            string data = getRooms();
            memset(sent, 0, 256);
            sprintf(sent, "%s", data.c_str());
            send(connected[i].getClientSocket(), sent, 256, 0);
        }
```

When received Ask, we should give room information to client. We get data by getRoom which searches the connected array (which saves client info) then sends it to client.

```
else if (receivedString.find("[Enter]") != -1) {
    /* 방에 접속 */
    for (int i = 0; i < connected.size(); i++) {
        string roomID = tokens[1];
        int roomInt = atoi(roomID.c_str());
        if (connected[i].getClientSocket() == client->getClientSocket()) {
            memset(sent, 0, 256);
            string data;
            int clientCount = clientCountInRoom(roomInt);
            cout << clientCount << endl;
            /* 방에 둘 이상 이미 차 있을 시 */
            if (clientCount >= 2) data = "[Full]";
            /* 방이 비었거나 한명만 있을시 */
            else {
                cout << "사용자 [" << client->getClientID() << "]: " << roomID << "번 방으로 접속" <<
                Client* newClient = new Client(*client);
                newClient->setRoomID(roomInt);
                connected[i] = *newClient;
                data = "[Enter]";
            }
            sprintf(sent, "%s", data.c_str());
            send(connected[i].getClientSocket(), sent, 256, 0);
```

When client tries to Enter the room, we first count the number of clients in the specified room, if its over 2, we send [Full] message then rejects the call, if its 1 or 0, we update 'connected' array (user info saver) then sends back data that approves the call.

Ready function is simple. If the user sends [Ready] we check the user's room and if both of the users are ready, we send the [Play] message that starts the game.

When received Put, we updates the board. The put is normally called when user make a move. We update that move globally. On special cases like out of range click, timeout, or possibly Win/Lost, Draw situation, we decides the result (Win, Lost, Draw) and send the results to users.

Play is for starting the game. It notifies that the Game has started to clients.

Exit is when the user ends the connection. We search the unconnected user, delete it from the connected array(user info saver) then notifies to other client in the room (if other client exists).

Well this is all for users.

2) Client

The client uses C# + winform. So the style is bit different with that of c++;

First there are simple board_paint function that paints the board.

MultiPlayForm_FormClosed is called when user closes the game window. This function calls closeNetwork that terminates thread and connection with the server.

There are 3 pre-game buttons : Ask, Enter, Ready

```
참조 1개
private void askButton_Click(object sender, EventArgs e)
{
    if (!connect)
    {
        tcpClient = new TcpClient();
        tcpClient.Connect("147.46.240.42", 20385);
        stream = tcpClient.GetStream();

        thread = new Thread(new ThreadStart(read));
        thread.Start();
        threading = true;
        connect = true;
    }
    string message = "[Ask]";
    byte[] buf = Encoding.ASCII.GetBytes(message + this.roomTextBox.Text);
    stream.Write(buf, 0, buf.Length);
    askButton.Enabled = false;
}
```

Ask or Enter connects with the server. (Ask and Enter looks quite similar, nearly same. Only difference is that Ask sends Ask msg, Enter sends Enter) Then sends messages. If client is in the room, the Ready button is enabled.

The boardPicture_MouseDown is called when the user made a move.

```
참조 1개
private void boardPicture_MouseDown(object sender, MouseEventArgs e)
{
    if (!playing)...
    if (!nowTurn) return;
    Graphics g = this.boardPicture.CreateGraphics();
    int x = e.X / rectSize;
    int y = e.Y / rectSize;
    int play = 0;
    if (currPlayer == Side.WHITE) play = 1;

    /* 범위 벗어났는지 체크 */
    if (x < 0 || y < 0 || x >= 11 || y >= 11)...
    if (board[x, y] != Side.none) return;
    board[x, y] = currPlayer;

    /* 돌 칠하기 */
    if (currPlayer == Side.BLACK)...
    else...
    /* 놓은 돌의 위치 전송 */
    string message = "[Put]" + roomTextBox.Text + "," + x + "," + y+","+play;
    byte[] buf = Encoding.ASCII.GetBytes(message);
    stream.Write(buf, 0, buf.Length);

    /* 상대방의 차례 */
    status.Text = "상대방이 둘 차례입니다.";
    nowTurn = false;
}
```

Normally it paints the stone and sends the data to server. On special case, when the user clicks the out of range twice, the function notifies the server.

There is the timer function that calculates and shows time. If more than 60 seconds passes, it notifies the info to the server. The server than decides the result and send the result back to clients.

```
private void read()
{
    while(true)
    {
        byte[] buf = new byte[1024];
        int bufBytes = stream.Read(buf, 0, buf.Length);
        string message = Encoding.ASCII.GetString(buf, 0, bufBytes);
        /* 있는 방 묻기 */
        if (message.Contains("[Ask]"))...
        /* 접속 성공  */
        if (message.Contains("[Enter]"))...
        /* 상대방이 나감 */
        if (message.Contains("[Exit]"))...
        /* 방 이미 가득 */
        if (message.Contains("[Full]"))...
        /* 게임 시작 처리 */
        if (message.Contains("[Play]"))...
        /* 상태 변화 */
        if (message.Contains("[Put]"))...
    }
}
```

The read function is called when the TCPclient receives the data from the server.

It handles the data according to its header.

When the data is Ask, it shows the room info received by using Textbox.

When Enter, is enables Ready button and disables Ask and Enter button.

When Exit, it shows in the text box that opponent ended the connection.

When Play, it starts the game locally. The timer is then Enabled.

When Put, if the data is the result of the game, it displays it in the text box. Else, the data is painting the board (opponent move). It updates the local board on behalf of the data.

3) User Manual(client)

1. start the game by clicking the exe.

2. click the '함께 하기' button. (I didn't made single play)

3. on the game interface you can either Ask("ASK" button) the server about the room, or can access the room directly(by putting the number into the input box then clicking "접속하기")

*** Warning: please don't click "접속하기" on empty input box...**

4. if you logged in to the room, you can click Ready button if you are ready to play.

5. if your opponent is ready too, the game starts. Enjoy!

3. Results

Please look at the video.

4. Discussion

Gomoku project was hard enough. There were some number of specifications which was not easy to implement (especially Timer). Also making the UI was painful, too...

Ironically the main protocol part was easier. There were many sources regarding the socket programming and TCP protocol. Also, most of languages supported kind headers with implemented functions. I bit regretted of starting with C since Python was much easier. But C# was kind enough in case of Server-Client connection.

Threading was bit hard too. Making the thread was not that hard but handling them was bit uneasy. Especially the timer part. When using thread it was hard to calculate and handle Time. I've used 4 different ways to make the timer all didn't worked. Lastly I went back to

the most oldest function that gets Date information, and calculated the passed time manually.

Even though there were some hard part, it was a meaningful time learning the Socket Programming, Protocols and UI? Parts. If there's more time, I would love to develop this game more. Thank you.