컴퓨터 네트워크 Assignment #2

2017033654 산업공학과 류동원

구현한 파일은 총 4가지 MySocketServer.java / MySocketClient.java / WritingThreadTCP.java / ListeningThreadTCP.java 이다.

MySocketServer.java 파일 (채팅 프로그램의 서버 프로그램 실행을 위한 파일)

1. Main 부분

```
public static void main(String[] args) {
    try {
        Scanner scanner = new Scanner(System.in);
        System.out.print("java Server ");
        int portNum1 = scanner.nextInt();
        int portNum2 = scanner.nextInt();
        ServerSocket welcomeSocket = new ServerSocket(portNum1);
        fileServerSocket = new ServerSocket(portNum2);  // file 전송 server socket

        // 서버 소켓이 종료될 때 까지 무한루프
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            Thread thd = new MySocketServer(connectionSocket);
            thd.start();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

파일 실행 시 2개의 인자를 입력받고, 각각 portNum1, portNum2에 저장한다. 이는 port number1과 portnumber2와 대응한다.

Client와 TCP 통신을 하기 위하여 welcomeSocket (port number1)과 fileServerSocket (port number2, 추후에 파일 송수신을 위한)생성하고 while loop 안에서 accept methods로 client가 오기를 기다린다. client가 연결을 요청하면 새로운 connectionSocket을 생성하고 connectionSocket 인스턴스를 MySocektServer의 인자로 넣어주면서 thread를 시작하고 서버 프로그램은 또다른 accept를 기다린다.

즉, client와 새로운 connection이 연결될 때 마다 새로운 server socket의 thread가 실행된다

2. 생성자, 변수

```
static ArrayList<Socket> list = new ArrayList<Socket>();
static HashMap<Socket, String> socketToChat = new HashMap<>();
static HashMap<Socket, String> socketToUser = new HashMap<>();

Socket serverSocket = null;
static ServerSocket fileServerSocket;

public MySocketServer(Socket serverSocket) {
    this.serverSocket = serverSocket;
}
```

list 는 client와 연결되어 thread가 실행중인 소켓들을 저장한다.

socketToChat은 각각의 thread를 실행중인 소켓과 참여하고 있는 채팅방 이름을 맵핑한다.

socketToUser는 각각의 thread를 실행중인 소켓과 참여한 채팅방의 user의 이름을 맵핑한다.

생성자 부분에서는 Main에서 받아온 소켓의 정보를 저장한다.

3. Run 부분

```
// client 에서 보낸 메세지 읽기
BufferedReader inFromClient = new BufferedReader(new
        InputStreamReader(serverSocket.getInputStream()));

// server 에서 client 로 메세지 보내기
OutputStream out = serverSocket.getOutputStream();
PrintWriter outToClient = new PrintWriter(out, true);
outToClient.println("서버에 연결되었습니다! 명령어를 입력해주세요.");
```

실행중인 서버 소켓의 inputstream과 outputstream을 생성해주고, printwriter을 통하여 client에게 연결되었다고 메세지를 보낸다

```
while((readValue = inFromClient.readLine()) != null) {
```

While문으로 들어가서 client가 메세지를 보낼 때마다 readValue에 저장을 하고, 알맞은 행동을 수행한다.

<명령어 부문 (#으로 시작)>

client로부터 메세지를 ‘ ‘(공백)으로 구분하여 id[ ] 배열에 저장하고 첫번째 단어가 #으로 시작하는 명령어일 경우, 각 명령어마다 일치하는 분기점 (if ~else if)에 가서 알맞은 동작을 수행한다.

A. #PUT (Client에서 Server로 파일을 송신하기 위한 명령어)

```
if(status == true && id[0].equals("#PUT")) {
```

status는 현재 채팅방에 입장을 한 상태인지 나타내주는 변수이며 채팅방에 참여하지 않았을 경우 파일을 송신할 수 없도록 한다.

```
receiveSocket = fileServerSocket.accept();
```

서버는 accept메소드를 실행하여 Client가 port number2를 사용하여 파일 송신을 위한 새로운 TCP연결을 요청하는 것을 기다린다.

```
byte[] buffer = new byte[4096];
int readBytes;

BufferedOutputStream fos = new BufferedOutputStream(new
FileOutputStream("output.txt"));
BufferedInputStream is = new BufferedInputStream(
        new DataInputStream(receiveSocket.getInputStream()));

while((readBytes = is.read(buffer)) != -1) {
    fos.write(buffer, 0, readBytes);
}
```

연결이 되었으면 연결 완료 확인 메세지를 창에 띄우고, socket의 inputstream으로 들어오는 데이터를 수신하여 파일을 전달받고 output.txt 파일로 저장한다.

```
for(int i=0; i<list.size(); i++) {
    if (status == true && (chatRoom.equals(socketToChat.get(list.get(i))))) {
        if (list.get(i) == serverSocket) {
            continue;
        }
        out = list.get(i).getOutputStream();
        outToClient = new PrintWriter(out, true);
        outToClient.println("채팅방에 파일이 업로드되었습니다. 업로드된 파일의 이름은
[output.txt] 입니다.");
    }
}
```

이후, 채팅룸 참여자들에게만 채팅방에 output.txt라는 파일이 수신되었다고 알려준다.

이후 파일 전송을 위하여 연결되었던 socket과 입출력 스트림을 close한다.

B. #GET (Client가 Server로부터 파일을 수신 위한 명령어)

```
else if (status == true && id[0].equals("#GET")) {
    sendSocket = fileServerSocket.accept();
```

#PUT 명령어와 동일하게 status를 확인하고 accept메소드를 통하여 client와 port number2로 연결되어 지기를 기다린다.

```
DataOutputStream os = new DataOutputStream(sendSocket.getOutputStream());
FileInputStream fis = new FileInputStream("output.txt");
byte[] buffer = new byte[4096];
int readBytes = 0;
while ((readBytes = fis.read(buffer)) != -1) {
    os.write(buffer, 0, readBytes);
    sendedBytes = sendedBytes + readBytes;
    if ((sendedBytes / 64000) == 1) {
        outToClient.println("#");
        sendedBytes = sendedBytes - 64000;
    }
}
```

연결이 완료되면 socket의 outputstream으로 서버가 저장했던 "output.txt"파일을 전송한다. 이때 전송되는 바이트를 sendedBytes변수에 저장하고 64Kbyte가 넘을 때 마다 client에게 "#" 진척도를 표시해 준다. buffer의 크기는 임의로 설정해 주었고 파일전송이 끝난 후 파일전송을 위한 소켓과 입출력 스트림을 close한다.

C. #CREATE (채팅방 생성을 위한 명령어)

```
else if (id[0].equals("#CREATE")) {
    if(socketToChat.containsValue(id[1])) {
        outToClient.println("[실패] 이미 존재하고 있는 채팅방입니다.");
        continue;
    }
    chatRoom = id[1];
    userName = id[2];
    socketToChat.put(serverSocket, chatRoom);
    socketToUser.put(serverSocket, userName);
    list.add(serverSocket);
    outToClient.println("[성공] 채팅방을 생성하였습니다 : " + chatRoom);
    status = true
```

#CREATE (채팅방이름) (유저이름) 을 각각 id[] 배열에 저장한다. socketToChat의 containsValue 메소드를 통하여 채팅방이름이 이미 기존에 존재하고 있다면 실패 메세지를 보낸다.

채팅방 생성이 가능하다면 socketToChat, socketToUser 각각의 해쉬맵에 서버소켓과 채팅방, 서버소켓과 유저이름 정보를 맵핑하고, 채팅에 참여하고 있으므로 list에 서버소캣정보를 넣는다. 이후, status = true로 변경하여 채팅에 참여중인 상태로 변경한다.

D. #JOIN(채팅방 입장을 위한 명령어

```
if(!socketToChat.containsValue(id[1])) {
        outToClient.println("[실패] 존재하지 않는 채팅방입니다.");
        continue;
    }
    chatRoom = id[1];
    userName = id[2];
    socketToChat.put(serverSocket, chatRoom);
    socketToUser.put(serverSocket, userName);
    list.add(serverSocket);
    outToClient.println("[성공] 채팅방에 참여하였습니다 : " + chatRoom);
    status = true;
```

#JOIN (채팅방이름) (유저이름)을 각각 id[ ] 배열에 저장한다. socketToChat의 containsValue 메소드를 이용하여 채팅방 이름이 기존에 존재하지 않는다면 실패 메세지를 전송한다.
채팅방 참여가 가능하다면 #CREATE와 유사하게 socketToChat, socketToUser 각각의 해쉬맵에 서버소켓과 채팅방, 서버소켓과 유저이름 정보를 맵핑하고, 채팅에 참여하고 있으므로 list에 서버소캣정보를 넣는다.
이후, status = true로 변경하여 채팅에 참여중인 상태로 변경한다.

E. #EXIT (채팅방 퇴장을 위한 명령어)

우선 채팅방에 참여중인지 status를 통하여 확인한 후, 참여중이라면 socketToChat, socketToUser, list에 해당 소켓 정보를 remove하고 status를 false상태로 변경한다.

F. #STATUS (채팅 구성원 확인을 위한 명령어)

```
else if (status == true && id[0].equals("#STATUS")) {

    outToClient.println("현재 참여하신 채팅방은 [" + chatRoom + "] 입니다.");
    for(int i=0; i<list.size(); i++) {
        if(chatRoom.equals( socketToChat.get(list.get(i)) )) {
            outToClient.print("(" + socketToUser.get(list.get(i)) + ") ");
        }
    }
    outToClient.println("이 대화에 참여중입니다.");
}
```

채팅방에 참여중인지 status를 통하여 확인한 후, 참여중이라면 참여한 채팅방을 알려주고 해당 채팅방에 어떤 user가 참여중인지 확인하기 위하여, socketToChat 테이블을 이용하여 list들중 채팅방 이름이 자신과 동일한 socket을 찾고, socketToUser 테이블을 이용하여 해당 socket의 유저 정보를 알려준다.

G. 명령어 (“#”으로 시작하는 단어)가 아닐 때 -> client끼리의 대화

```
for(int i=0; i<list.size(); i++) {
    if( status == true && (chatRoom.equals( socketToChat.get(list.get(i))) ) {
        if (list.get(i) == serverSocket) {
            continue;
        }
        out = list.get(i).getOutputStream();
        outToClient = new PrintWriter(out, true);
        outToClient.println("FROM " + userName + " : " + readValue);
    }
}
```

status 정보를 확인하여 채팅방에 참여중이라면 #STATUS와 마찬가지로 list들 중 동일한 채팅방 이름을 가지고 있는 socket들을 찾고, 해당하는 socket들에게 outputstream을 생성하여 메세지를 전달한다.

MySocketClient.java 파일 (채팅프로그램 client 프로그램 실행을 위한 파일)

1. MAIN 부문

```
public class MySocketClient {
    public static void main(String[] args) {
        try {
            Socket clientSocket = null;

            Scanner scanner = new Scanner(System.in);
            System.out.print("java Client ");
            String hostName = scanner.next();
            int portNum1 = scanner.nextInt();
            int portNum2 = scanner.nextInt();

            clientSocket = new Socket(hostName, portNum1);

            ListeningThreadTCP t1 = new ListeningThreadTCP(clientSocket);
            WritingThreadTCP t2 = new WritingThreadTCP(clientSocket,
hostName, portNum2);

            t1.start();
            t2.start();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Main 메소드로만 구성되어 있으며 실행을 하면 (host name) (port number1) (port number2) 세개의 인자를 입력받아 각각의 변수에 저장한다.

host name과 port number1을 이용하여 client socket을 생성하면 server와 연결되며 ListeningThread와 WritingThread 두개의 쓰레드를 실행시키며 소켓의 정보를 인자로 넘겨준다.

ListeningThreadTCP.java 파일 (client가 server로부터 메세지를 읽는 thread)

```
Socket socket = null;
public ListeningThreadTCP(Socket socket) {
    this.socket = socket;
}

public void run() {
    try {
        BufferedReader inFromServer = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
        while(true) {
            System.out.println( inFromServer.readLine() );
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

생성자에서 client소켓정보를 받아온 후, run 메소드를 통하여 쓰레드를 실행시킨다. 소켓의 InputStream을 생성한 후, while loop안에서 server가 보내주는 data를 수신하고 출력한다.

WritingThreadTCP.java 파일 (client가 server로 메세지를 보내는 thread)

1. 생성자, writing 부분

```
public WritingThreadTCP(Socket socket, String hostName, int portNum2) {
    this.socket = socket;
    this.hostName = hostName;
    this.portNum2 = portNum2;
}

public void run() {
    try {
        BufferedReader inFromUser = new BufferedReader(new
                InputStreamReader(System.in));
        OutputStream out = socket.getOutputStream();
        PrintWriter outToServer = new PrintWriter(out, true);

        while(true) {

            sentence = inFromUser.readLine();
            outToServer.println(sentence);
            String[] id = sentence.split(" ");
```

Thread가 호출될 때 서버의 client의 socket정보, server의 host name정보, port number2를 전달받고 각각의 변수에 저장한다.

socket에 output stream을 생성하여 PrintWriter을 통하여 System.in으로 입력되는 정보를 server로 전송한다.

2. #PUT이 입력될 때 (서버로 파일 송신)

```
if(id[0].equals("#PUT")) {

    sendSocket = new Socket(hostName, portNum2); // server 에게 파일전송 소켓
연결 요청


    String fileName = id[1];

    DataOutputStream os = new
DataOutputStream(sendSocket.getOutputStream());
    FileInputStream fis = new FileInputStream(fileName);
    byte[] buffer = new byte[4096];
    int readBytes = 0;
    int sendedBytes = 0;


    while ((readBytes= fis.read(buffer)) != -1) {
        os.write(buffer, 0, readBytes);
        sendedBytes = sendedBytes + readBytes;
        if ((sendedBytes / 64000) == 1) {
            System.out.println("#");
            sendedBytes = sendedBytes - 64000;
        }
    }
    os.flush();
    os.close();
    fis.close();
    sendSocket.close();
```

입력이 #PUT (file name)인 경우, 파일을 전송하는 tcp connection을 생성하기 위해 port number2를 이용해서 소켓을 생성하고 server에게 tcp연결을 요청한다.

client와 같은 디렉토리에 있는 file을 서버에게 송신하기 위하여 data output stream과 file input stream을 생성하고 파일의 데이터를 서버로 전송한다.

이때 전송되는 데이터가 64Kbyte를 초과시마다 진척도 표시를 위하여 "#"을 화면에 띄어준다.

이후, output stream, file input stream, 파일 송수신을 위한 tcp socket을 close 한다.

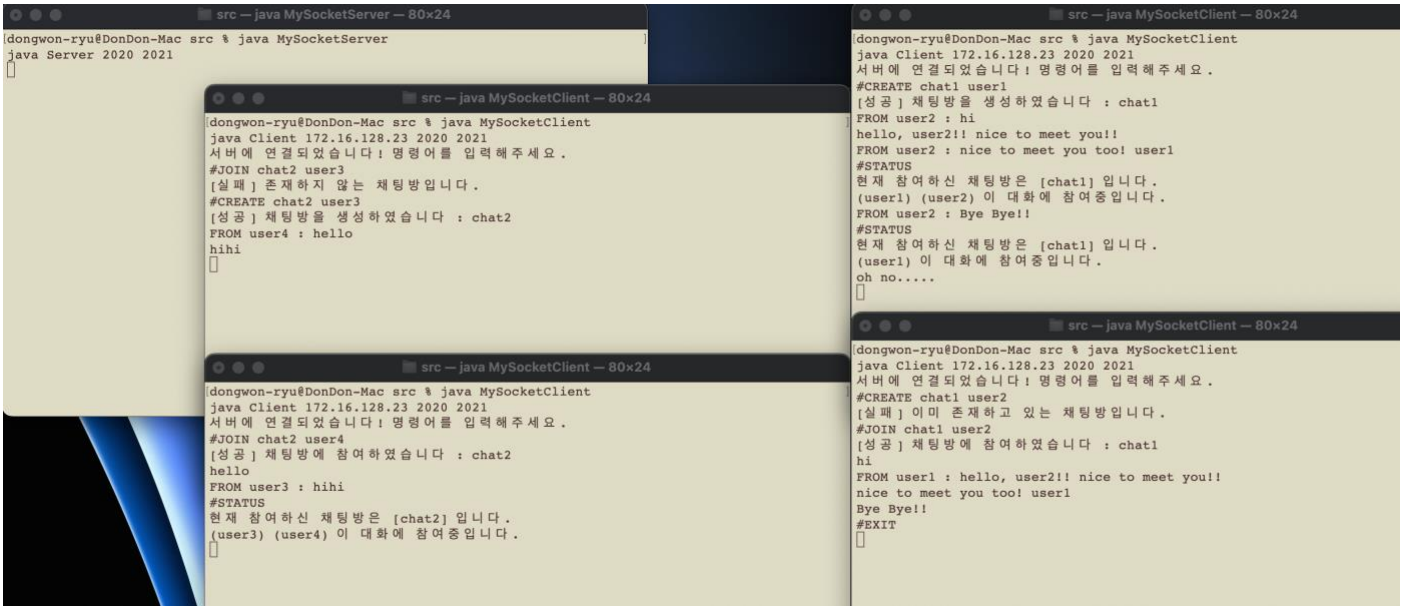
3. #GET이 입력될 때 (서버에서 파일 다운로드)

```
else if (id[0].equals("#GET")) {

    receiveSocket = new Socket(hostName, portNum2); // server 에게 파일수신
소켓 연결 요청
    String fileName = id[1];
    File file = new File(fileName);
    byte[] buffer = new byte[4096];
    int readBytes;

    BufferedOutputStream fos = new BufferedOutputStream(new
FileOutputStream("receive.txt"));
    BufferedInputStream is = new BufferedInputStream(
            new DataInputStream(receiveSocket.getInputStream()));
    while((readBytes = is.read(buffer)) != -1) {
        fos.write(buffer, 0, readBytes);
    }
    System.out.println("다운로드 하신 파일 이름은 receive.txt 입니다.");
    fos.flush();
    fos.close();
    is.close();
    receiveSocket.close();
}
```

입력이 #GET (file name)인 경우 #PUT과 유사하게 port number2를 통하여 파일 수신을 위한 Tcp connection socket을 생성한다. server의 데이터를 수신하기위한 socket의 input stream과 파일을 생성하기 위한 file output stream을 생성한 후, socket의 input stream으로 입력되는 data를 받아서 파일을 생성한다. 이때 다운로드 가능한 파일 이름은 서버에서 미리 알려주게 된다 (ex. “output.txt”). 다운로드 완료한 파일은 (“receive.txt”)라는 파일 이름으로 저장되며, file outputstream과 socket inputstream, 파일 수신 socket을 close 한다.

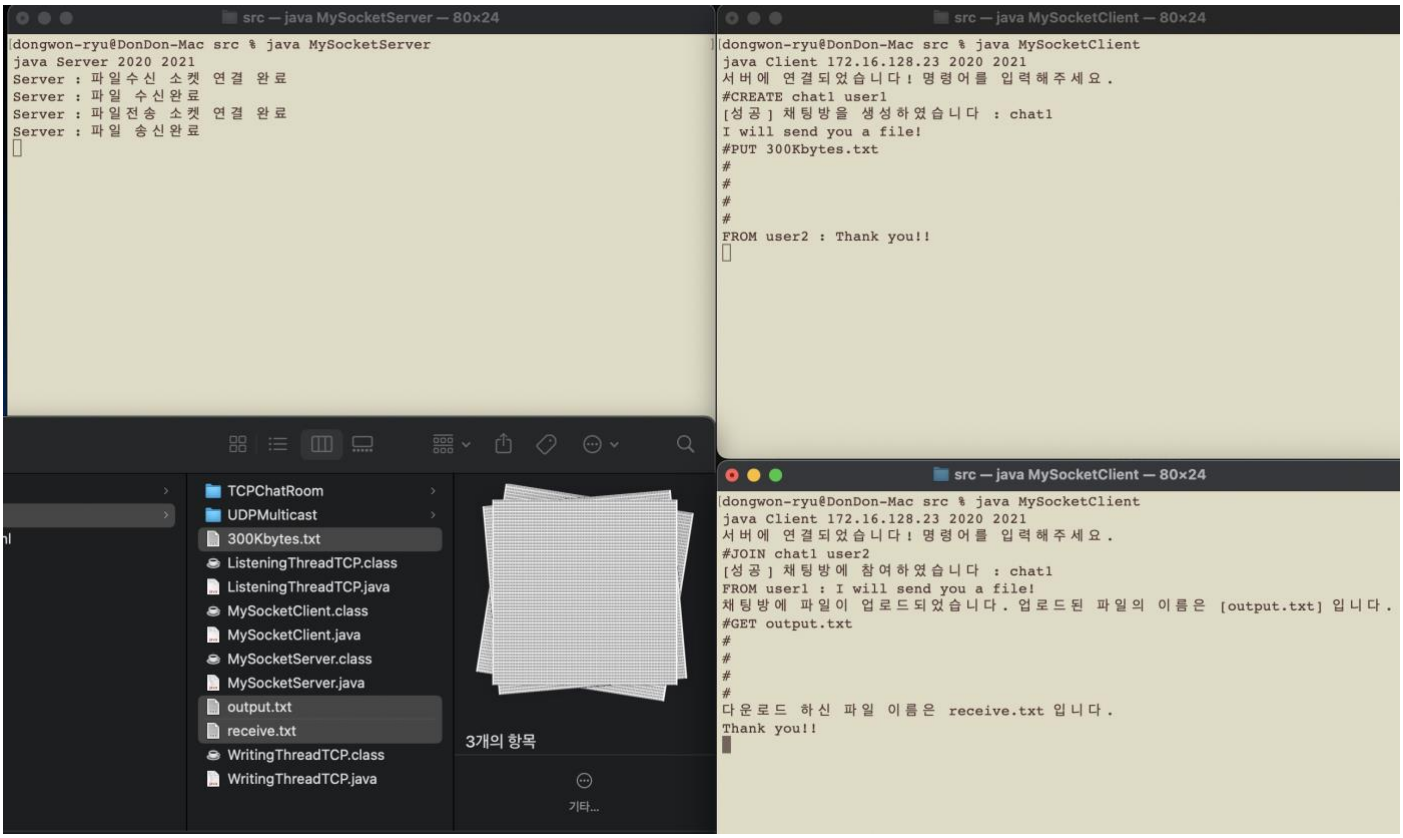
<실행 결과>

1. 작동방법 및 실행결과 (대화)
    1. 서버는 port number1과 port number 2를 입력하여 실행한다. (공백으로 구분)
    2. client는 서버의 IP주소와 서버와 동일한 port number 2개를 입력한다. (공백으로 구분)
    3. client는 " #CREATE chat1 user1 " 과 같이 (명령어) (채팅방이름) (유저이름)으로 채팅방을 생성한다.
        i. 기존에 존재하는 채팅방 이름을 생성할 시에는 실패 메시지 출력
    4. client는 " #JOIN chat1 user2 " 와 같이 (명령어) (채팅방이름) (유저이름)으로 채팅방에 참여한다.
        i. 기존에 존재하지 않는 채팅방에 참여하고자 할 경우 실패 메시지 출력
    5. client는 " #STATUS " 를 입력하여 현재 채팅방의 이름과 구성원들을 확인
    6. client는 " #EXIT " 를 입력하여 채팅방에서 퇴장한다. 더이상 기존의 채팅방과 대화를 주고받을 수 없다.



7. 왼쪽 위 : 서버 프로그램
8. 오른쪽 위, 오른쪽 아래 : client 프로그램 (채팅방 : chat1)
9. 중간 위, 중간 아래 : client 프로그램 (채팅방 : chat2)
10. 채팅방 chat1과 채팅방 chat2가 동시에 작동한다.

2. 작동방법 및 실행결과 (파일 송수신)



1. 왼쪽 위 : server program
2. 오른쪽 위, 오른쪽 아래 : client program
3. 우선, 파일 송/수신을 위하여 서버와 클라이언트를 실행시키고, 두 클라이언트가 같은 채팅방에 입장한다.
4. 파일을 가진 client가 #PUT (파일 이름)을 입력하여 같은 디렉토리에 있는 파일을 서버에 송신한다.
5. 수신자 client는 서버로부터 파일이름을 전달받고, #GET (파일 이름)을 입력하여 파일을 다운로드(수신)한다.
6. 수신과 송신 둘 다 64Kbytes마다 #으로 진척도를 표시해준다.