

분산시스템 중간보고서

201912344 최동훈

구현환경

언어 : JAVA JDK20

IDE : IntelliJ

외부 라이브러리 : JAVA.net , java.io , java.util , java.swing

소켓을 이용한 네트워크 프로그래밍으로 직접 구현함.

기능별 구현 완료 여부

- 다중 클라이언트의 서버 접속/해제 기능
 - ✓ 1.1 클라이언트의 서버 접속 (또는 로그인)
 - ✓ 1.2 클라이언트 종료시 서버는 해당 클라이언트 접속 해제 처리
 - ✓ 1.3 서버는 클라이언트의 접속/해제 정보를 다른 클라이언트들에게 통보
 - ✓ 1.4 같은 클라이언트 및 여러 클라이언트들의 반복적 서버 접속/해제 테스트
- 파일 전송 기능
 - ✓ 2.1 클라이언트가 전송할 파일 하나 이상 선택
 - ✓ 2.2 클라이언트가 서버로 선택된 파일들 전송 (파일 업로드)
 - ✓ 2.3 클라이언트와 서버는 파일 전송 완료 사실 확인 (메시지 출력 등)
 - ✓ 2.4 클라이언트 접속 상태에서 파일 전송 반복 수행 테스트

전체 클래스 및 메소드 설계

개요 : 각각의 서버와 클라이언트 클래스는 start() 메소드를 통하여, 소켓통신에 필요한 Setting을 완료하고, Thread를 통하여 병렬적으로 IO 통신을 하도록 설계하였다.

MultiClient class

- void start() 메서드

기능 : 실제 Client class를 실행시키는 메서드이다. 우선적으로 서버와 연결한 소켓을 생

성한다. 이때, host 와 port 는 각각 localhost, 8000으로 한다. 그 다음 사용자로부터 로그인 할 ID를 입력 받는 UI창을 띄운다.

```
JOptionPane.showInputDialog("로그인할 ID를 입력하세요.");
```

Java의 Swing 라이브러리를 이용하였다. 그 다음 Server로부터 data를 receive 하는 기능을 가진 sendThread 클래스를 생성 및 실행한다,

```
Thread sendThread = new SendThread(socket, name);  
sendThread.start();
```

또한 socket.getInputStream을 통해 서버에 입력을 할 InputStream을 getter 함수로 받는다.

```
in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
while (in != null) {  
    String inputMsg = in.readLine();  
    if(("[" + name + "]"님이 나가셨습니다").equals(inputMsg)) break;  
    System.out.println("From:" + inputMsg);  
}
```

SendThread Class

- 우선 쓰레드를 구현하기 위하여 java.lang의 Thread 클래스를 상속한다.
- 생성자를 통해, 클라이언트 소켓과 사용자의 ID 값을 입력받는다.

```
- public SendThread(Socket socket, String name) {  
    this.socket = socket;  
    this.name = name;  
}
```

- void run() 메서드

기능: 상속받은 Thread 에 구현한 run 함수를 오버라이딩 함으로써, 실질적인 Thread의 기능을 정의한다.

최초 1회는 Client의 로그인 ID를 서버로 전송한다.

```
// 최초 1회는 client의 name을 서버에 전송  
PrintStream out = new PrintStream(socket.getOutputStream());  
out.println(name);  
out.flush();
```

- 또한 무한 루프(사용자가 터미널에 quit을 입력할때까지 지속됨.)를 통해서 서버로 전송할 파일의 이름을 입력받는다.

```
- while (true) {
    String outputMsg
        = JOptionPane.showInputDialog(name+"님 서버로 전송할 파일의
이름을 입력하세요.");
    out.println(outputMsg);
    out.flush();
    if("quit".equals(outputMsg)) break;
}
```

MultiServer Class

- void start()

서버 소켓을 생성한다 포트 8000

또한 무한 roop 를 통하여 Client가 접속할때마다 새로운 쓰레드를 생성한다. 즉, 서버 클래스는 “다중” 쓰레드 환경으로, 한 process 내에서 무한히, 루프를 돌면서, 클라이언트와 서버의 소켓연결하는 쓰레드와, Client로부터 Stream을 통하여 I/O를 수행하는 쓰레드가 병렬적으로 동작하도록 설계하였다.

```
serverSocket = new ServerSocket(8000);
while (true) {
    System.out.println("[클라이언트 연결대기중]");
    socket = serverSocket.accept();

    // client가 접속할때마다 새로운 스레드 생성
    ReceiveThread receiveThread = new ReceiveThread(socket);
    receiveThread.start();
}
```

ReceiveThread Class

중요 트러블 슈팅

발생된 문제상황 **쓰레드의 동시성 Issue** : 우선적으로 각각의 생성된 클라이언트 쓰레드에게 보낼 data를 List 라는 자료구조에 담아서 전송 할려고 하였다. 그런데, List라는 자료구조는 모든 쓰레드가 공유하기에, 원래 다른 쓰레드에게 보낼 data를 미리 List를 선점한 쓰레드가 변경해 버리는 공유자원 접근문제 Race Condition이 발생하였다. 이를 해결하기위해서 JAVA의 Collection라이브러리에서 제공하는 공유자원의 Atomicity(원자성)을 보장해주는 synchronized 함수를 사용하여, 공유자원 동시접근 문제를 해결하였다.

```
static List<PrintWriter> list =
    Collections.synchronizedList(new ArrayList<PrintWriter>());
```

-public ReceiveThread(Socket socket) 생성자

기능 : 클라이언트가 접속할때마다, 해당 클라이언트와 연결할 소켓과 inputStream,outputstream을 추가해주고, 해당 output 스트림을 공유자원인 List에 추가함.

```
static List<PrintWriter> list =  
    Collections.synchronizedList(new ArrayList<PrintWriter>());  
  
Socket socket = null;  
BufferedReader in = null;  
PrintWriter out = null;  
FileOutputStream fout=null;  
  
public ReceiveThread (Socket socket) {  
    this.socket = socket;  
    try {  
        out = new PrintWriter(socket.getOutputStream());  
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
        list.add(out);  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

-public void run() 메서드

기능 : 실질적으로 Client로부터 파일의 정보를 받아와서 서버의 저장소에 파일을 새롭게 생성한 뒤에, Client에서 전송한 파일의 내용을 서버에 새롭게 생성한 파일에 Write 해 준다. 또한, 최초 1회는 Client 의 로그인 정보를 수신한 뒤, 서버에 접속된 다른 클라이언트들에게 void sendAll(String s) 함수를 통해 접속/해제 정보를 통보한다.

소켓에서 받은 file 정보로 file 입력받는다.

```
FileInputStream fis = new FileInputStream(filename);
```

파일의 내용을 byte 단위로 읽어온 다음 저장할 버퍼 byte 배열 생성한다.

```
byte[] byteBuff = new byte[9999];
```

파일을 읽고 읽은 크기를 nRlen에 저장한다.

```
int nRlen = fis.read(byteBuff);
```

출력을 위하여 byte 배열을 문자열로 변환한다.

```
String strBuff = new String(byteBuff, 0, nRlen);
```

client로부터 받은 파일을 읽은 정보를 출력한다.

```
System.out.printf("읽은 바이트수[%d] : \n 읽은 내용 : \n%s \n", nRlen, strBuff);
```

중요(내가 구현한 클라이언트의 파일을 서버의 저장소에 새롭게 생성해주는 부분)

서버 영역에 "ServerRepository" 라는 새로운 파일을(FileOutputStream을 통해) 생성한뒤, strBuff의 byte 버퍼에 담겨져 있는 Client로부터 전송받은 파일의 내용을 Write 해준다. 즉, 요약하자면, Client로부터 파일을 전송받아 서버의 저장소에 새롭게 파일을 생성한뒤, 전송받은 파일의 내용을 그대로 복사/저장한 것이다.

```
// 기존의 파일이 없으면 만들어지고 있으면 덮어쓰게 되어 기존 파일내용이 지워진다.
FileOutputStream fos = new FileOutputStream( name: "ServerRepository.txt");
```

```
// 파일에 저장할 내용
String strText = strBuff;
// 문자열을 바이트배열로 변환해서 새롭게 생성한 서버의 파일에 저장한다.
fos.write(strText.getBytes());
```

또, 파일전송이 정상적으로 일어났을시에, 서버는 sendAll() 메서드를 통하여 저장 완료 사실을 서버에 접속된 다른 클라이언트들에게 통보한다.

```
if("quit".equals(inputMsg)) break;
sendAll(name + "님께서 " + inputMsg+" 파일을 ServerRepository로 전송완료하였습니다.");
System.out.println("서버 저장소에 저장 완료.");
```

만약 연결 중 클라이언트와 의 연결이 끊어진다면, 해당 사실을 sendAll() 메서드를 통해 다른 클라이언트들에게도 통보하고, list에서 outputstream을 제거한다.

```
} catch (IOException e) {
    System.out.println "[" + name + " 접속끊김"];
} finally {
    sendAll "[" + name + "]"님이 서버에서 로그아웃 하셨습니다);
    list.remove(out);
```

- void sendAll(String s) 메서드

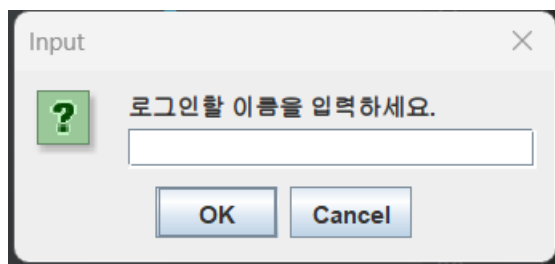
기능 : 공유자원인 list에 해당 client 별로 outputstream을 저장한 뒤(즉, 클라이언트가 새로 연결될때마다 해당 클라이언트와 연결되는 outputstream) 이 list에 저장된 outputstream을 이용하여 전체 client들에게 서버가 data를 전송 한다.

```
private void sendAll (String s) {
    for (PrintWriter out: list) {
        out.println(s);
        out.flush();
    }
}
```

클라이언트와 서버 동작 프로토콜

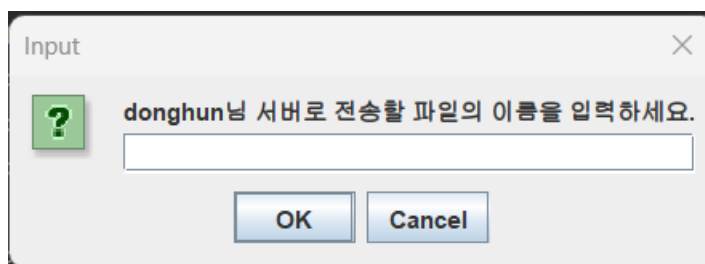
1. 우선 클라이언트가 서버로 접속하기 위해서는 서버의 쓰레드가 동작하고 있어야 한다. 또한 client는 서버 소켓 port 8000 번으로 연결을 시도한다.
2. 클라이언트의 접속이 성공하면, 소켓이 연결될때마다 새로운 쓰레드(ReceiveThread)가 생성된다.
3. Client의 기능 항목별 메시지 전송 포맷 및 수신동작

1)클라이언트가 접속 성공시 , 로그인 ID 입력받는다.



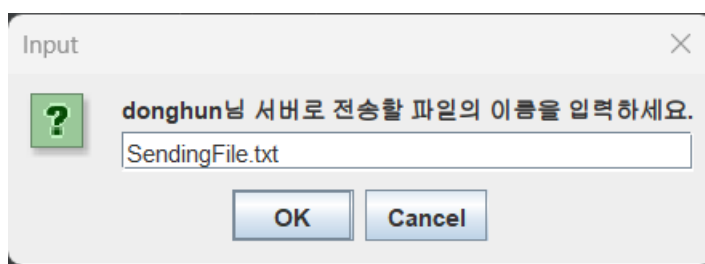
Input dialog box with a green question mark icon. The text says "로그인할 이름을 입력하세요." (Enter the name you want to log in with). There is an empty text input field and two buttons: "OK" and "Cancel".

2) 정상적으로 연결 시 서버로 전송할 파일의 이름을 입력받는다.



Input dialog box with a green question mark icon. The text says "donghun님 서버로 전송할 파일의 이름을 입력하세요." (Enter the name of the file you want to send to the donghun server). There is an empty text input field and two buttons: "OK" and "Cancel".

3) 여기서 파일 이름의 형식은 아래와 같이 입력한다.



Input dialog box with a green question mark icon. The text says "donghun님 서버로 전송할 파일의 이름을 입력하세요." (Enter the name of the file you want to send to the donghun server). The text input field contains "SendingFile.txt". There are two buttons: "OK" and "Cancel".

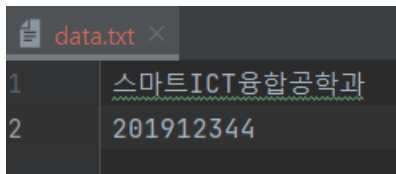
4) 파일 전송이 정상적으로 완료되면 해당 사실을 List의 output 스트림을 통하여 다른 클

라이언트 들에게 알린다.(아래의 예시 : donghun이 우선 접속 한뒤 파일 전송 후, 최동훈이 접속 하여 다른 파일을 서버로 전송된 상황)

```
> Task :MultiClient.main()
[서버와 연결되었습니다]
From:[donghun]님이 서버에 로그인하였습니다.
From:donghun님께서 SendingFile.txt 파일을 ServerRepository로 전송완료하였습니다.
From:[최동훈]님이 서버에 로그인하였습니다.
From:최동훈님께서 SendingFile.txt 파일을 ServerRepository로 전송완료하였습니다.
From:최동훈님께서 data.txt 파일을 ServerRepository로 전송완료하였습니다.
```

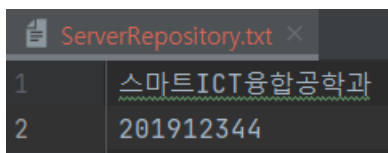
5) 파일 전송 결과

(1) 전송하려는 파일 (data.txt)



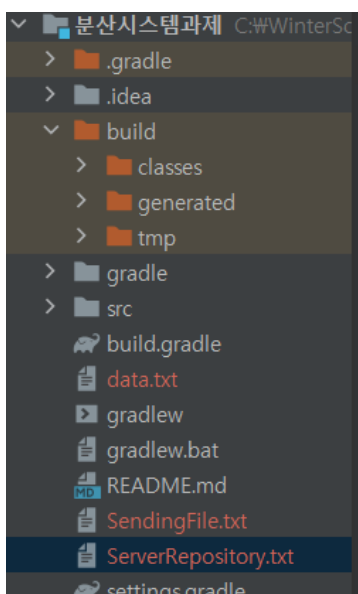
Line	Content
1	스마트ICT융합공학과
2	201912344

(2) 위 4) 단계에서 최동훈 사용자가 서버로 전송 완료후 ServerRepository에 전송된 모습.



Line	Content
1	스마트ICT융합공학과
2	201912344

(3) 서버 저장소에 새롭게 파일이 생성된 모습.



- 분산시스템과제 C:\WinterSc
- > .gradle
- > .idea
- ▼ build
 - > classes
 - > generated
 - > tmp
- > gradle
- > src
- build.gradle
- data.txt
- gradlew
- gradlew.bat
- README.md
- SendingFile.txt
- ServerRepository.txt**
- settings.gradle

6) 클라이언트 접속 해제시, 서버가 다른 클라이언트들에게 통보

```
> Task :MultiClient.main()
[서버와 연결되었습니다]
From:[최동훈]님이 서버에 로그인하였습니다.
From:최동훈님께서 SendingFile.txt 파일을 ServerRepository로 전송완료하였습니다.
From:최동훈님께서 data.txt 파일을 ServerRepository로 전송완료하였습니다.
From:[donghun]님이 서버에서 로그아웃 하셨습니다
```

컴파일 및 실행방법

1. 우선 MultiServer.Class 를 컴파일 및 실행

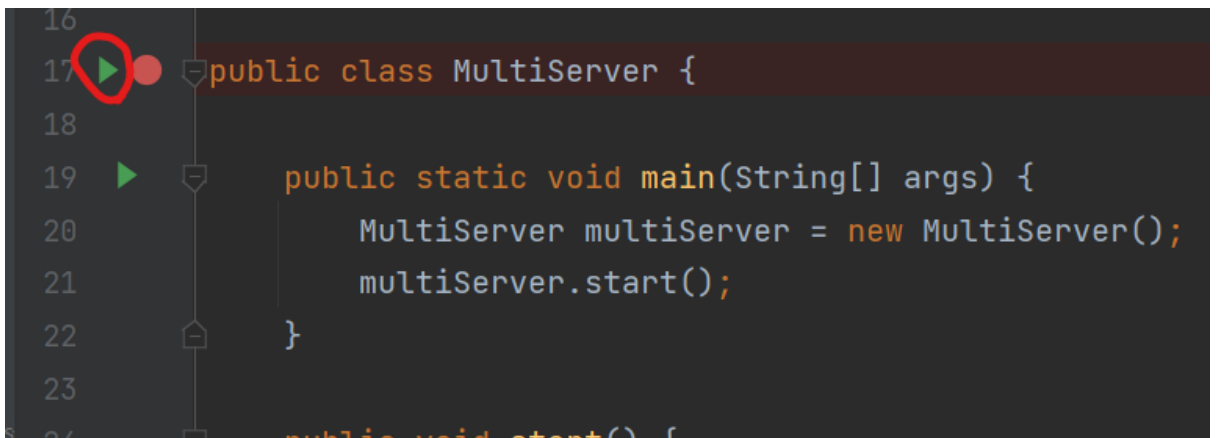
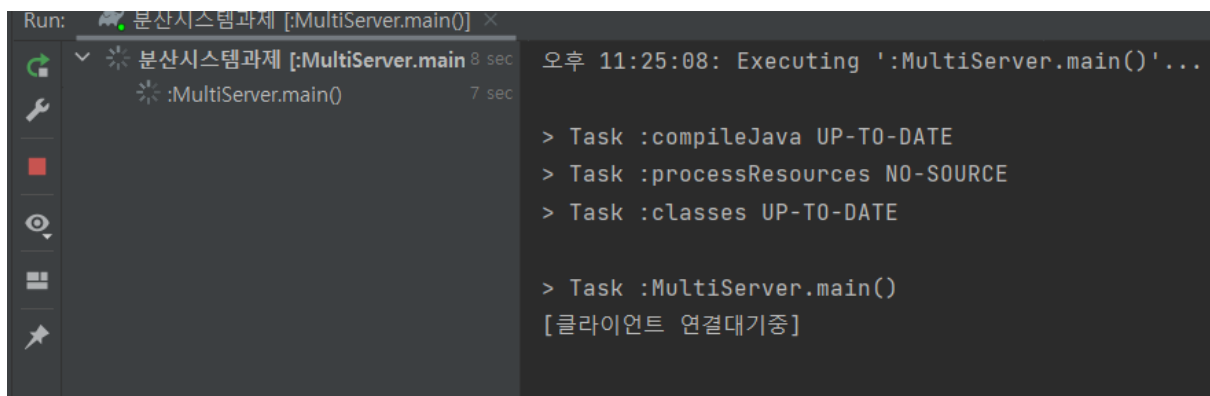


Figure 1 위 표시한 파란색 버튼 누르면 컴파일+실행 가능.

2. 서버가 정상적으로 실행됨을 확인



3. 그 다음 클라이언트를 실행해야 하는데 여기서 **주의점**. 다중 클라이언트 를 실행해야 하므로, IntelliJ에서 하나의 클래스로 여러 인스턴스를 실행 가능하도록 설정을 변경해 주어야 함.

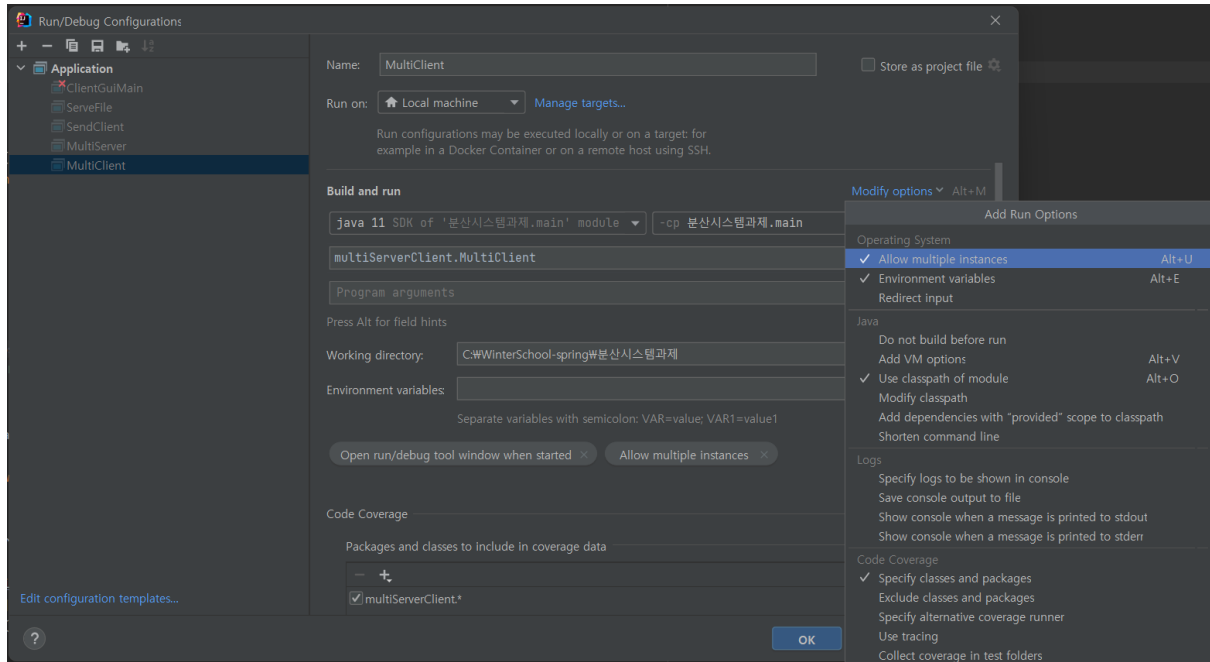


Figure 2 MulltiClient.class 의 RunConfiguration 옵션에서 Allow multiple Instances 옵션 허용.

4. 위의 설정 완료후, MultiClient.class를 중복실행.

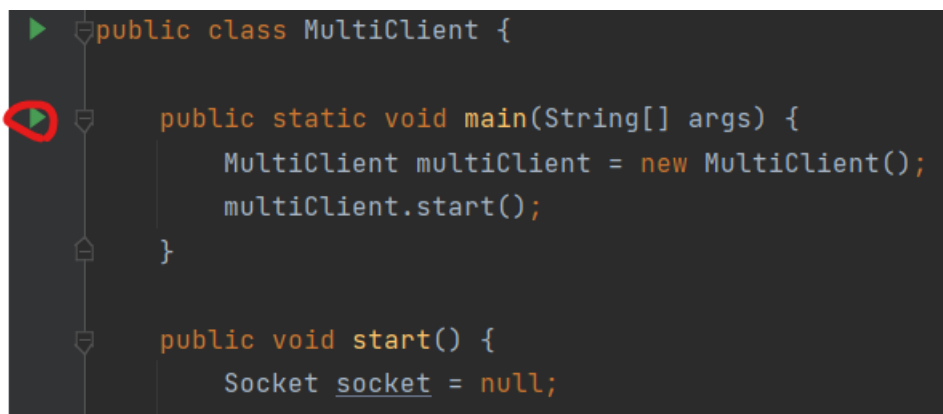


Figure 3 해당 버튼을 여러 번 클릭해도 여러 개의 MultiClient가 실행됨.

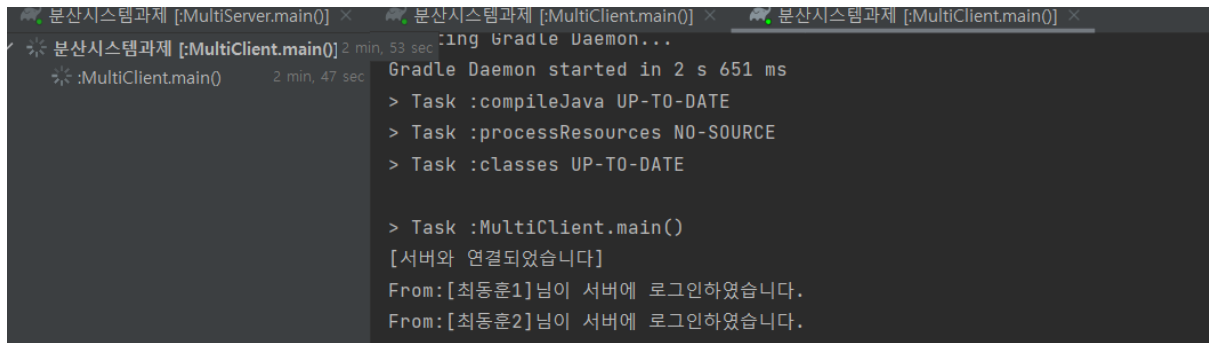


Figure 4 다중 클라이언트가 실행되어서 서버에 접속된 모습.

5. 메시지 포맷에 맞게 파일 이름 입력시, 서버로 파일 전송 가능.

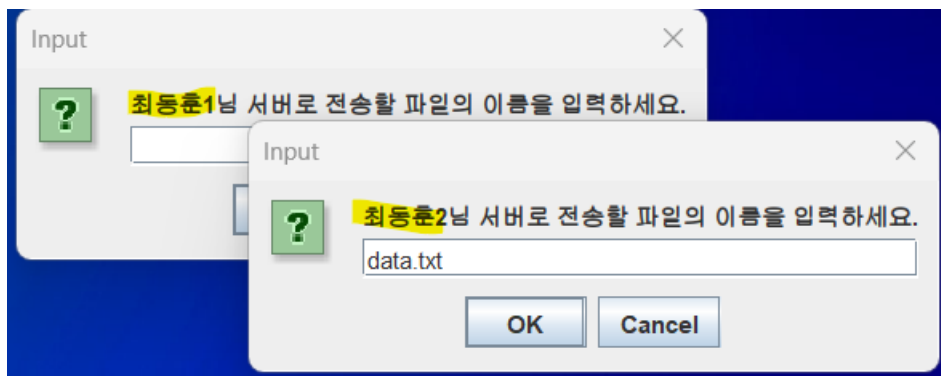


Figure 5 서로 다른 두 Client 가 서버로 File을 전송 가능함.

소스코드 포함 프로젝트 Github URL

<https://github.com/ulsandonghun/Distributed-System>

기능별 시연 영상 youtube 링크

다중 클라이언트의 서버 접속/해제 기능

1.1 클라이언트의 서버 접속 (또는 로그인)

[클라이언트의 서버 접속\(또는 로그인\) - YouTube](#)

1.2 클라이언트 종료시 서버는 해당 클라이언트 접속 해제 처리

[클라이언트 종료 시 서버는 해당 클라이언트 접속 해제 처리 - YouTube](#)

1.3 서버는 클라이언트의 접속/해제 정보를 다른 클라이언트들에게 통보

[클라이언트의 접속해제 정보를 다른 클라이언트 들에게 통보 - YouTube](#)

1.4 같은 클라이언트 및 여러 클라이언트들의 반복적 서버 접속/해제 테스트

[같은 클라이언트 및 여러 클라이언트들의 반복적 서버 접속/해제 테스트 - YouTube](#)

파일 전송 기능

2.1 클라이언트가 전송할 파일 하나 이상 선택

[클라이언트가 전송할 파일 하나 이상 선택 - YouTube](#)

2.2 클라이언트가 서버로 선택된 파일들 전송 (파일 업로드)

[클라이언트가 서버로 선택된 파일들 전송\(파일 업로드\) - YouTube](#)

2.3 클라이언트와 서버는 파일 전송 완료 사실 확인 (메시지 출력 등)

[클라이언트와 서버는 파일 전송 완료 사실 확인 \(메시지 출력 등\) - YouTube](#)

2.4 클라이언트 접속 상태에서 파일 전송 반복 수행 테스트

[클라이언트 접속 상태에서 파일 전송 반복 수행 테스트 - YouTube](#)