# 네트워크 – FTP Client and Server Programing

# - assignment 2 -

소프트웨어전공 2013010926 강은석

## 1. 과제 설명

: 과제 1의 구현을 thread 버전으로 구현한다. 즉, 서버는 각각의 클라이언트별로 전담 thread를 배정하여 복수의 클라이언트를 동시에 서비스 할 수 있도록 한다. 단 이때 서버에서 발생할 수 있는 쓰레드 동기화(thread synchronization) 문제도 고려되어야 한다. (쓰레드 동기화 문제는 두 클라이언트가 동시에 동일한 디렉토리에 파일을 PUT할 때 발생하는 write interference 문제를 의미한다.)

: 클라이언트가 파일 송수신을 위해서 "LIST", "GET", "PUT", "CD"와 같은 명령어를 요청할 수 있다.

- LIST: 파일 서버의 특정 디렉토리의 파일 보는 명령어
- GET: 파일 서버로부터 파일을 download 하는 명령어
- PUT: 파일 서버에 파일을 upload 하는 명령어
- CD: 파일 서버에서 현 디렉토리 위치를 변경하는 명령어

: 기본적으로 절대 경로, 상대 경로, "."(현재 위치), ".." (상위 디렉토리)를 지원해야 한다.

## 2. 프로그램 설계와 구조

: Server는 항상 Welcome Socket으로 Client의 소켓과의 연결을 기다립니다. Client에서 지정한 Ip와 Port Number로 Socket을 생성하고 서버의 Socket과 accept가 되면 Connection이 생기고 Thread를 생성하여 해당 Client에 배정합니다. 이에 Client로부터 "CD, LIST, GET, PUT"과 같은 파일 송수신을 위한 명령어로 request가 들어오면 Server는 각 명령어에 맡는 처리 작업을 진행한 뒤 response를 Client에게 전달해 줍니다. PUT의 경우 동일한 파일 이름에 대해서 동기화문제를 해결합니다. 이 후 Client로부터 "Done"이라는 명령어가 Server로 수신이 되면 Server와 Client간 connection을 끊고 다른 Client와의 연결을 기다리도록 설계하였습니다.

## 3. 동작 절차 및 구현 방법

```
Socket socket = new Socket("127.0.0.1", portNum);
String sentence, errorWrongcmd, errorNotFound;
String [] sentenceWords;
String curPath = "NoPath";
errorWrongcmd =  "Wrong cmd try again.";
errorNotFound = "Failed directory name is invalid.";
// connect to server
while (true) {
  BufferedWriter bufWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
  BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
  sentence = inFromUser.readLine() + " " + curPath;
  if ("".equals(sentence)) {
    System.out.println(errorWrongcmd);
    continue;
  }
  sentenceWords = sentence.split("\\s");
  bufWriter.write(sentence);
  bufWriter.newLine();
  bufWriter.flush();
```

**그림 1 client에서 명령어 전달**

```
    try {
      ServerSocket serverSocket = new ServerSocket(portNum);
      String initialPath =  new java.io.File("").getCanonicalPath();
      // accept to client socket
      while (true) {
        Socket clientSocket = serverSocket.accept();
        String errorResponse = "Failed directory name is invalid.";
        String errorWrongcmd = "Wrong cmd try again.";
        System.out.println("Accepted connection from" + clientSocket);
        new Thread(new Runnable() {
          public void run() {
            try {
              FtpThreadServer serve = new FtpThreadServer();
              String response, request;
              // define response about each client request.
```

**그림 2 server에서 명령어 수신 및 Thread 생성**

기본적인 CD, LIST, GET, PUT 명령어와 현재 path를 Client에서 BufferedWriter.write()로 Server로 전달해주고 Server에서는 BufferedReader.readLine() 으로 수신하고 Thread를 생성해줍니다.

1) CD

```
// request about "CD" and get response
if ("CD".equals(sentenceWords[0])) {
    InputStream in = socket.getInputStream();
    DataInputStream din = new DataInputStream(in);
    int statusCode = din.readInt();
    int responseLength = din.readInt();
    if (statusCode == -1) {
      System.out.println(din.readUTF());
    }else {
      curPath = din.readUTF();
      System.out.println(curPath);
    }
}
// request about "LIST" and get response
```

**그림 3 client에서 CD 명령어 response 수신**

```
/* responses about CD request
 * response return values are status code, response length, response value
*/
else if("CD".equals(requestWords[0])) {
  returnClass result = new returnClass();
  OutputStream out = clientSocket.getOutputStream();
  DataOutputStream dout = new DataOutputStream(out);
  result = serve.moveWorkDictionary(requestWords, errorResponse);
  dout.writeInt(result.status);
  dout.writeInt(result.value.length());
  dout.writeUTF(result.value);
  out.flush();
}
```

**그림 4 Server에서 CD명령어에 대한 response, moveWorkDictionary함수 실행**

```
public returnClass moveWorkDictionary(String[] requestWords, String errorResponse)
  FtpThreadServer serve = new FtpThreadServer();
  returnClass result = new returnClass();
  String curPath="";
  if(requestWords.length == 2) {    // response about "CD" request
    if("NoPath".equals(requestWords[1])) {
      result.value = System.getProperty("user.dir");
    }else {
      result.value = requestWords[1];
    }
    return result;
  }
  else {  // response about "CD path" request
    if ("NoPath".equals(requestWords[2])) {
      curPath = new java.io.File(requestWords[1]).getCanonicalPath();
    }
    else {
      File f = new File(requestWords[2]+"\\"+requestWords[1]);
      curPath = f.getCanonicalPath();
    }
    File dir = new File(curPath);
    if (dir.isDirectory()) { // response about "CD relative path"
      result.value = curPath;
      return result;
    }
    else {  // response about "CD absolute path"
      dir = new File(requestWords[1]);
      if (dir.isDirectory()) {
        result.value = requestWords[1];
        return result;
      }
```

**그림 5 moveWorkDictionary 함수**

우선 그림4을 보면, Server에서 CD요청에 따라 DataOutputStream을 이용해서 statusCode, responseLength, response를 Client에 보내주고 그림 3에서 보면 Client는 DataInputStream를 통해서 각 response를 수신합니다. 또한 그림 4에서 moveWorkDictionary함수를 호출하여 그림 5과같이 동작합니다. 기존 assignment1에서는 setProperty로 위킹디렉터리를 변경했는데 이는 client1과 client2의 워킹디렉터리를 전부 변경시켜서 현재 path를 받고 CD명령어에 맞게 이동하고 넘겨주는 방식으로 변경하였습니다.

2) LIST

```
// request about "LIST" and get response
else if("LIST".equals(sentenceWords[0]) ) {
  InputStream in = socket.getInputStream();
  DataInputStream din = new DataInputStream(in);
  int statusCode = din.readInt();
  int responseLength = din.readInt();
  String response = din.readUTF();
  String responses [] = response.split("\t");
  for(int i=0; i < responses.length;i++) {
    System.out.println(responses[i]);
  }
}
// request about "GET" and get response
```

**그림 6 client에서 LIST 명령어 response 수신**

```
/* responses about LIST request
 * response return values are status code, response length, response value
*/
else if("LIST".equals(requestWords[0])) {
  returnClass result = new returnClass();
  OutputStream out = clientSocket.getOutputStream();
  DataOutputStream dout = new DataOutputStream(out);
  result = serve.showList(requestWords, errorResponse);
  dout.writeInt(result.status);
  dout.writeInt(result.value.length());
  dout.writeUTF(result.value);
  out.flush();
}
/* responses about GET request
```

**그림 7 Server에서 LIST명령어에 대한 response, showList 함수 실행**

```
// response about "LIST relative path" and "LIST absolute path" request
else {
  if ("NoPath".equals(requestWords[2])) {
    curPath = new java.io.File(requestWords[1]).getCanonicalPath();
  }
  else {
    File f = new File(requestWords[2]+"\\"+requestWords[1]);
    curPath = f.getCanonicalPath();
  }
  File dir = new File(curPath);
  if (dir.isDirectory()) { // about relative path
    result.value = serve.findLISTandResponse(dir);
    return result;
  }
  else { // about absolute path
    File dir1 = new File(requestWords[1]);
    if (dir1.isDirectory()) {
      result.value = serve.findLISTandResponse(dir1);
      return result;
    }
```

**그림 8 showList 함수 내부**

우선 그림7을 보면, Server에서 LIST요청에 따라 DataOutputStream을 이용해서 statusCode, responseLength, response를 Client에 보내주고 그림 6에서 보면 Client는 DataInputStream를 통해서 각 response를 수신합니다. 또한 그림 7에서 showList함수를 호출하여 그림 8과같이 Client에서 보낸 path정보와 LIST명령어의 인자값을 이용해서 해당 경로에 있는 파일과 디렉터리 목록을 client에게 보내줍니다.

3) GET

```
// request about "GET" and get response
else if("GET".equals(sentenceWords[0])) {
  int len, data,statusCode;
  byte[] buffer = new byte[1024];
  InputStream in = socket.getInputStream();
  DataInputStream din = new DataInputStream(in);
  statusCode = din.readInt();
  data = din.readInt();              // get file size
  String filename = din.readUTF();   // get file name
  if (statusCode == -1) {            // error response
    System.out.println(filename);
    continue;
  }
  FileOutputStream out = new FileOutputStream(filename);
  for(int i = data; i > 0; i--){    // save file to Client
    len = in.read(buffer);
    out.write(buffer,0,len);
  }
```

**그림 9 client에서 GET 명령어 response 수신, 파일 저장**

```
else if("GET".equals(requestWords[0])) {
  byte[] buffer = new byte[1024];
  returnClass1 result = new returnClass1();
  OutputStream out = clientSocket.getOutputStream();
  DataOutputStream dout = new DataOutputStream(out);
  result = serve.getfile(requestWords, errorResponse, errorWrongcmd);
  dout.writeInt(result.status);
  dout.writeInt(result.dataSize);
  dout.writeUTF(result.fileName);
  if (result.status != -1) {
    int len = 0;
    // send file to client
    FileInputStream fin = new FileInputStream(result.path);
    for(int i = result.dataSize; i > 0; i--){
      len = fin.read(buffer);
      out.write(buffer,0,len);
    }
```

**그림 10 Server에서 GET명령어에 대한 response, getfile 함수 실행**

```
byte[] buffer = new byte[1024];
int len;
File file = new File(currentPath + "\\" +requestWords[1]);
if (file.isFile()) {
  result.path = file.getCanonicalPath();
  FileInputStream fin = new FileInputStream(result.path);
  String [] parseFileName = requestWords[1].split("\\\\|/");
  result.fileName = parseFileName[parseFileName.length-1];
  System.out.println(result.fileName);
  while((len = fin.read(buffer))>0){  // calculate file size
    result.dataSize++;
  }
  fin.close();
  return result;
}else {
    file = new File(requestWords[1]);
    if (file.isFile()) {
    // get file name from file path
      result.path = requestWords[1];
      FileInputStream fin = new FileInputStream(requestWords[1]);
      String [] parseFileName = requestWords[1].split("\\\\|/");
      result.fileName = parseFileName[parseFileName.length-1];
      while((len = fin.read(buffer))>0){  // calculate file size
        result.dataSize++;
      }
      fin.close();
      result.status = 2;
      return result;
    }
```

**그림 11 getfile 함수 내부**

우선 그림10을 보면, Server에서 GET요청에 따라 DataOutputStream과 FileInputStream을 이용해서 statusCode, fileSize, filename, fileData를 Client에 보내주고 그림 9에서 보면 Client는 DataInputStream과 FileOutputStream를 통해서 각 response를 수신하고 파일을 저장합니다. 그림 10에 Server에서 buffer크기를 할당해서 파일을 읽어서 file의 크기를 파악하고 file을 Client로 보내는데 buffersize를 1024로 정한 이유를 os에서 파일 크기를 보여줄 때 kbyte 기준으로 보여주기 때문에 1024로 정했습니다.

3) PUT

```
if (file.isFile()) {  // find relative path file
  FileInputStream fin = new FileInputStream(currentPath + "\\" +sentenceWords[1]);
  String fileName = sentenceWords[1];
  while((len = fin.read(buffer))>0){   // calculate upload file size
      data++;
  }
  fin.close();
  fin = new FileInputStream(sentenceWords[1]);
  dout.writeInt(data);
  dout.writeUTF(fileName);
  len = 0;
  for(int i = data; i>0; i--){       // upload file to server
    len = fin.read(buffer);
    out.write(buffer,0,len);
  }
```

**그림 12 client에서 PUT 명령어 response 수신, 파일 upload**

```
/* responses about PUT request
 * response return values are "whether the file transfer is okay" response value
 */
else if("PUT".equals(requestWords[0])) {
  OutputStream out = clientSocket.getOutputStream();
  DataOutputStream dout = new DataOutputStream(out);
  returnClass result = new returnClass();

  result = serve.putFile(clientSocket, requestWords, errorWrongcmd);
  dout.writeInt(result.status);
  dout.writeUTF(result.value);
  out.flush();
}
```

**그림 13 Server에서 PUT명령어에 대한 response, putfile 함수 실행**

```
else {  // get file into server
  File file = new File(requestWords[1]);
  synchronized(requestWords[1].intern()) {
    int len, data;
    String filename;
    byte[] buffer = new byte[1024];
    InputStream in = client.getInputStream();
    DataInputStream din = new DataInputStream(in);
    data = din.readInt();
    if (data == -1) {
      result.status = -1;
      result.value = "No such file exist";
      return result;
    }
    else {
      System.out.println(client.getPort());
      filename = din.readUTF();
      FileOutputStream fout = new FileOutputStream(filename);
      // get file from client
      for(int i = data; i > 0; i--){
        len = in.read(buffer);
        fout.write(buffer,0,len);
      }
```

**그림 14 putfile 함수 내부**

우선 그림13을 보면, Server에서 PUT요청에 따라 DataOutputStream을 이용해서 status response message를 Client에 보내주고 그림 12에서 보면 Client는 DataInputStream과 FileInputStream를 통해서 서버에 파일을 저장합니다. 그림 14에서 synchronize를 requestWords[1](파일 이름)걸어서 동시에 put을 하는 경우의 동기화 문제를 해결했습니다.

## 4. 컴파일 방법, 실행방법

```
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpThreadServer/src$ ls
FtpThreadServer.java
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpThreadServer/src$ javac FtpThreadServer.java
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpThreadServer/src$ ls
'FtpThreadServer$1.class'   FtpThreadServer.class   FtpThreadServer.java   returnClass.class   returnClass1.class
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpThreadServer/src$
```

**그림 15  Compile FtpThreadServer.java**

```
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpClient/src$ ls
FtpClient.java
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpClient/src$ javac FtpClient.java
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpClient/src$ ls
FtpClient.class  FtpClient.java
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpClient/src$
```

**그림 16 Compile FtpClient**

위의 그림과 같이 "FtpThreadServer/src" 경로에 들어가서 명령어 "javac FtpThreadServer.java"를 입력하고 "FtpClient/src" 경로에서 명령어 "javac FtpClient.java"를 입력하면 각 FtpThreadServer.class, FtpClient.class 가 생기는 것을 확인 할 수 있습니다.

```
'FtpThreadServer$1.class'   FtpThreadServer.class   FtpThreadServer.java   returnClass.class   returnClass1.class
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpThreadServer/src$ java FtpThreadServer
Accepted connection fromSocket[addr=/127.0.0.1,port=57562,localport=2020]
```

**그림 17 run FtpThreadServer**

```
FtpClient.class  FtpClient.java
smkoy@LAPTOP-BS4SAUVL:/mnt/c/Users/leade/eclipse-workspace/FtpClient/src$ java FtpClient
```

**그림 18 run Ftpclient**

서버와 클라이언트 경로에서 java <class 이름> <port number>를 입력하면 실행이 됩니다.

이후 각 CD , LIST, GET, PUT에 대한 설명을 이어가겠습니다.

1) CD


**그림 19 CD 명령어 실행**

CD 명령어 예시를 보면 "CD", "CD ."에 대해서 현재 working directory path를 보여줍니다.

"CD .."은 상대경로를 지원하며 현재 working directory의 부모 path로 이동합니다.

"CD Path"는 절대 경로를 지원하며 Path에 해당하는 path로 이동합니다.

2) LIST

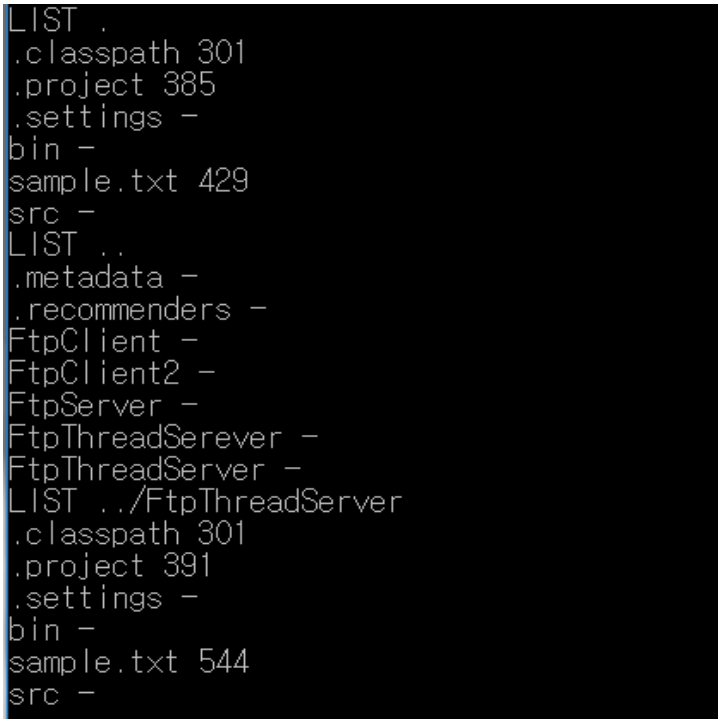


**그림 20 LIST 명령어 실행**

LIST 명령어 예시를 보면 "LIST ."에 대해서 현재 working directory path의 파일 목록을 보여줍니다.

"LIST .."은 상대경로를 지원하며 working directory의 부모의 파일 목록을 보여줍니다.

"LIST Path"는 절대 경로를 지원하며 Path에 해당하는 path로 이동합니다.

위의 그림에서 파일은 옆에 파일 크기가 나오며, directory인 경우는 "-"를 보여줍니다.

3) GET


**그림 21 GET 실행**

그림 21에서 "FtpServer/src" 폴더에서 상대 경로로 GET "test.png", "AliceInWonderland.txt"를 할 수 있고 절대 경로로 "GET /mnt/c/Users/leade/Desktop/test/sample.txt"를 하고 "FtpClient/src" 폴더를 LIST로 확인하면 GET으로 받아온 파일들이 있는 것을 확인 할 수 있습니다. 위에서 보시는 것과 같이 GET을 상대 경로로 받아올 수 있는 폴더는 서버가 있는 "FtpServer/src"폴더가 기준입니다. 그리고 Client에서 GET으로 받아온 파일들은 "FtpClient/src"에 저장된다는 것을 확인 할 수 있습니다.

4) PUT

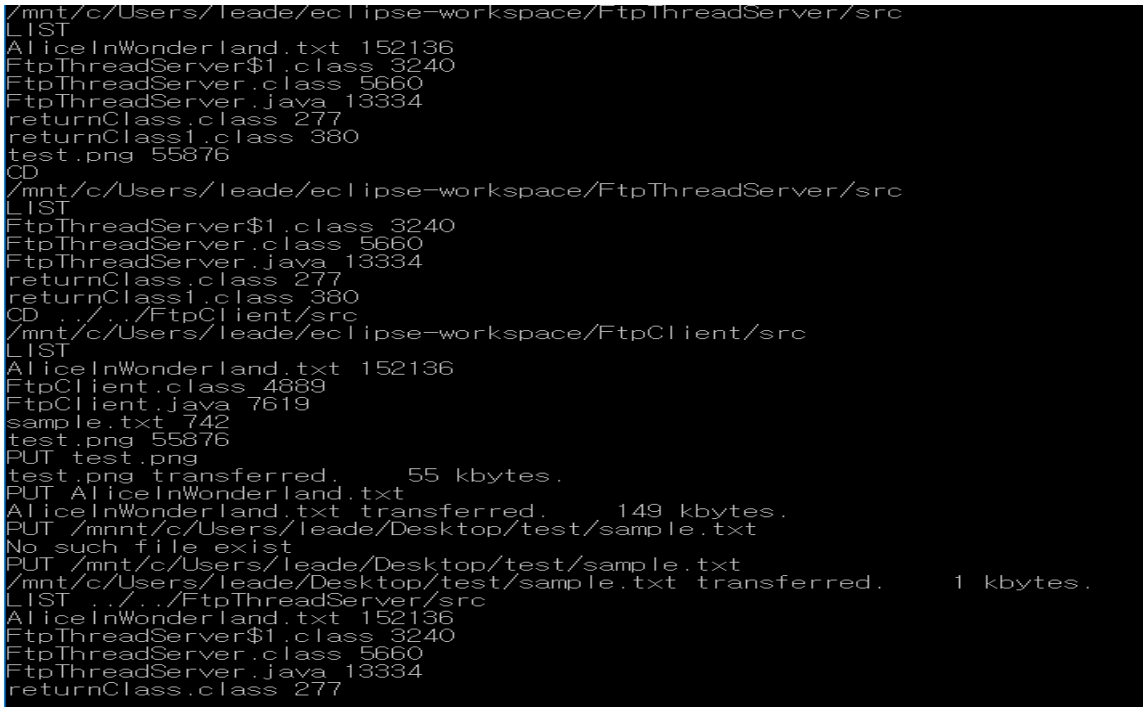


**그림 22 PUT 실행**

그림 6에서 "FtpClient/src" 폴더에서 상대 경로로 PUT "test.png", "AliceInWonderland.txt"를 할 수 있고 절대 경로로 "PUT /mnt/c/Users/leade/Desktop/test/sample.txt"를 하고 "FtpThreadServer/src" 폴더를 LIST로 확인하면 PUT으로 받아온 파일들이 있는 것을 확인 할 수 있습니다. 위에서 보시는 것과 같이 PUT으로 파일을 서버로 upload 하는 폴더는 클라이언트가 있는"FtpClient/src" 폴더가 기준입니다. 그리고 Client에서 PUT으로 upload한 파일들은 "FtpThreadServer/src"에 저장된다는 것을 확인 할 수 있습니다.

## 5. Thread 배정 복수 클라이언트 동시 서비스

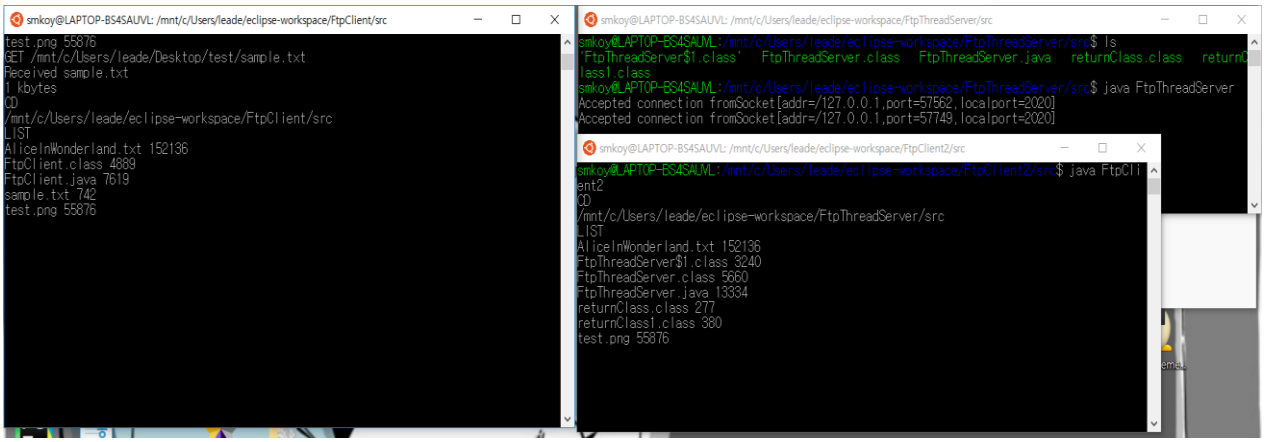


**그림 23 우측 상단: server, 우측 하단: client2 좌측: client1**

그림23에서는 FtpClient와 FtpClient2를 동시에 FtpThreadServer에 접속하여 명령어를 요청하고 response를 받고 있습니다. CD, PUT, GET의 경우 쓰레드 동기화 문제는 없다고 판단하고 PUT의 경우 파일의 이름에 synchronized를 걸어서 쓰레드 동기화 문제를 해결하였습니다.