Computer Networking Assignment 3 Report & User Guide

Name: Shi, Yu Student ID: 5130309117

File List and Description

The whole folder can be imported into Eclipse as a project. Also supports for running from command line, using .jar files in JarFiles folder.

There are 3 subfolders in "FileTrans" folder.

1. **FileTrans/src**

   This folder contains the source code for the project. It has two subfolders

   (1) FileTrans/src/ClientServer

   This folder contains two files, for stand alone client and server respectively.

   FileTrans/src/ClientServer/MyServer.java For stand alone server.

   FileTrans/src/ClientServer/MyClient.java For stand alone client.

   (2) FileTrans/src/P2P

   This folder contains only one source file, for file peers.

   FileTrans/src/P2P/FilePeer.java

2. **FileTrans/JarFiles**

   This folder contains the .jar files and default share/download folders for testing.

   (1) FileTrans/JarFiles/MyClient.jar For stand alone client.

   (2) FileTrans/JarFiles/MyServer.jar For stand alone server.

   (3) FileTrans/JarFiles/share This folder is the default share folder, that is, the file in this folder on server side can be accessed by the client.

FileTrans/JarFiles/share/hw3.pdf This is the file I prepared for the test.

(4) **FileTrans/JarFiles/download** An empty folder to contain the files downloaded from the server side.

3. **FileTrans/bin**

Contains some .class files produced by Eclipse when debugging. For testing tasks this folder can be ignored.

Run the application

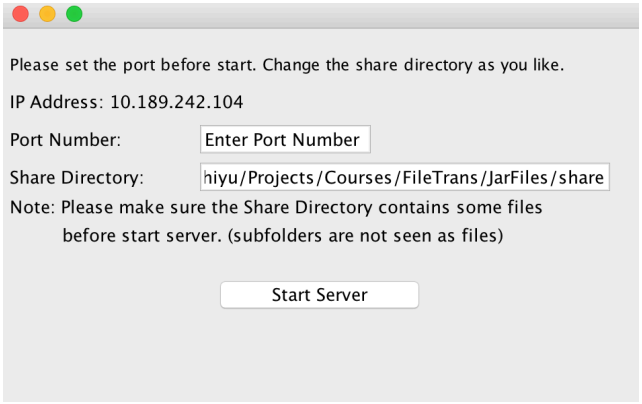
Stand alone Client-Server mode

1. **Start server**

Open the terminal, change the directory to **…/FileTrans/JarFiles**, run

```
java –jar MyServer.jar
```

This will start the server program, as this picture shows.



The server IP Address is already shown there.

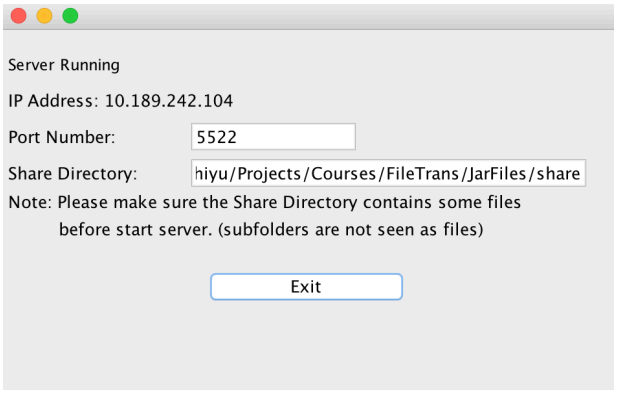
2. **Change the Share Directory if you like**

As the **Note** says, you should make sure the Share Directory you enter contains some files before start server. If the share directory contains no direct children

files, instead only subfolders, the files in the subfolder cannot be access by the client, and the server cannot provide any thing. Of course, if you use my default Share Directory, there will always be a **hw3.pdf** file waiting for you. You can change it to any directory in your computer, for example /Users/shiyu.

3. **Set the port number and click Start Server button**

   Enter a port number (> 1024) in the Port Number text field, and click Start Server button. If the port is not currently occupied by other programs, the server will start up successfully.

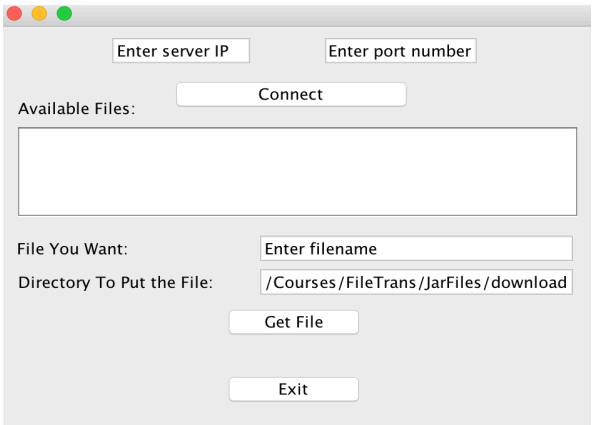


4. **Start up the client**

   Open another terminal, change the directory to **⋯/FileTrans/JarFiles**, run

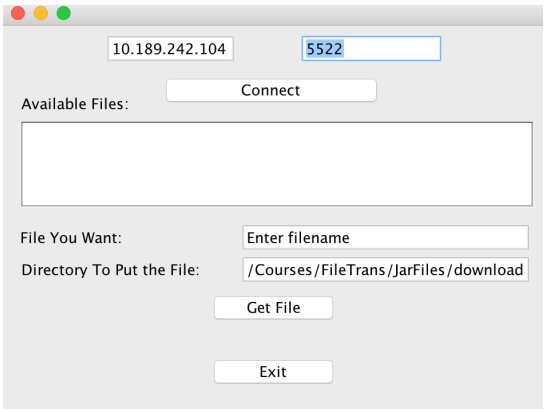
   `java -jar MyClient.jar`

   This will start up the client program.

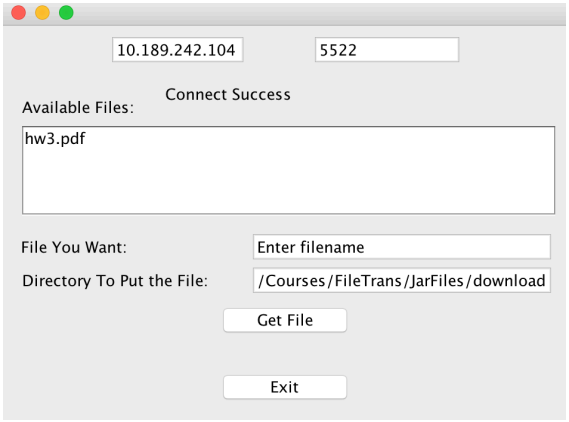


5. **Enter the server IP address and port number, then click Connect button**

   The server IP address and port number can be seen from the Server program UI in Step 3. We enter them into the corresponding text fields in client, like



   Then click **Connect**. And the server will transmit all the file names under its share directory to the client, as this picture shows



   In this case, only one hw3.pdf file is in the default share directory of server.

6. **Set the directory where you want to put the downloaded files**

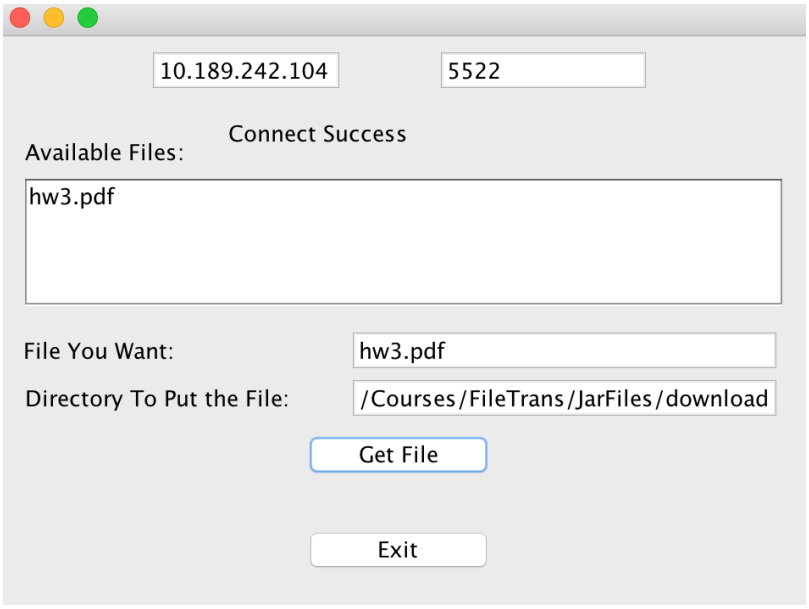
   Similar to the Share Directory in server side, you can change this directory to anywhere you like. Of course, you may just use the default one.

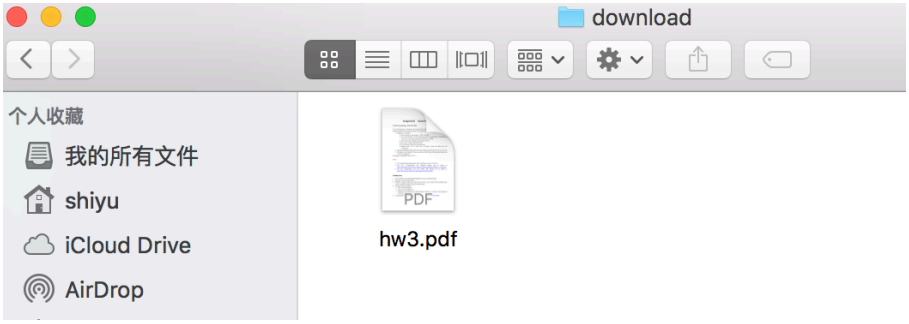
7. **Download the file you want**

   Select one file from the ‘Available Files’ list. Enter its name to the ‘File You

Want' text field.

Then click **Get File**. Then the file will be downloaded to the destination folder.





Peer-to-peer mode

P2P application is just a combination of Client and Server applications. I will describe how to run them briefly.

1. **Open two terminals and run two FilePeer applications**

   Change directory to **…/FileTrans/JarFiles**, run

   `java -jar FilePeer.jar`

for each terminal.

2. **Set the server parts for two FilePeer applications**

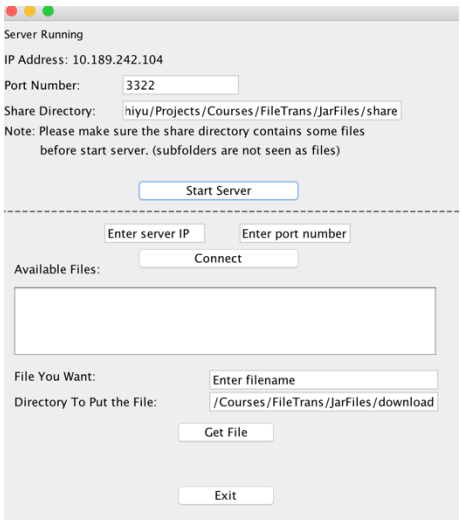
   The rules are just the same as in stand alone mode, just remember to set

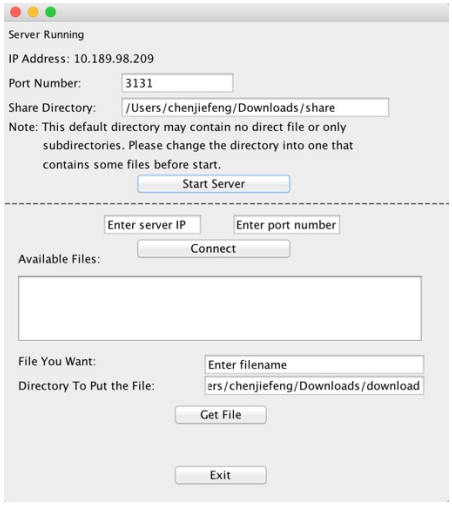
   (a) Port Number

   (b) Share Directory, if you want. It is not a must to change to default one.

   in two FilePeer applications. Then click **Start Server** buttons.

   For example





3. **Set the client parts for two FilePeer applications**

   Again, the rules are identical to stand alone mode, just remember to set

   (a) Server IP address

   (b) Server port number

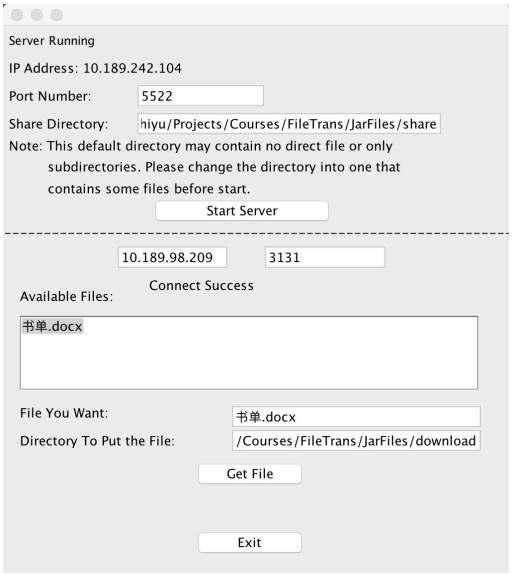
   Then click **Connect**. And set

   (c) Destination directory

   (d) The file you want

   Finally click **Get File**.

For example



For all the applications above you can press ESC or click **Exit** button to exit.

Problem and experiences

1. Troubles with buffered read of socket

Because the **read()** method of socket is blocking. Even if the file has already been read, the client still waits for more bytes from the socket, and blocked at that **read()** function.

A simple but dirty way to solve this problem is check whether there are still some bytes available in the socket input stream. However, this is unreliable. It is possible client checks before the server is able to put next bytes into the stream. In this case client will stop reading directly, leaving the rest part of the file in the socket stream. I have tested this method and find that the problem

occurs at a very high frequency.

So, I choose to first send the length of the file to the client. After receiving the length, client is aware of how many bytes it should read, and it can just read enough then break. Here is the code

```
String s = fromServerInfo.readLine();
try {
    expectedSize = Integer.parseInt(s);
}catch(Exception e) {
    getFile.delete();
    System.out.println("Exception in requestFile(): " + e.getMessage());
    e.printStackTrace();
    Warn.setText("No such file: " + fileName);
}
if(expectedSize >= 0) {
    outToServer.write("get size\n".getBytes());
    downloadedFile = new BufferedOutputStream(new FileOutputStream(directory + fileName));
    while((len = fromServerContent.read(buffer)) != -1) {
        downloadedFile.write(buffer, 0, len);
        downloadedFile.flush();
        sum += len;
        if(sum == expectedSize)
            break;
    }
    downloadedFile.close();
}
```

2. Send the file list to client

Sending file list to client so that it can choose the wanted file is convenient. To provide this service,

(1) In server side, I get the file names of all files under the share directory. Send them to client.

```
int fileNum = 0;
//Calculate number of available files in current shared directory.
for(int i = 0; i < list.length; ++i) {
    if(list[i].isFile() && !list[i].getName().startsWith("."))
        ++fileNum;
}
//toClientInfo.writeBytes(fileNum + "\n");
toClientInfo.write((fileNum + "\n").getBytes());
fromClient.readLine();
//The server first transfers names of all the available files to the client.
for(int i = 0; i < list.length; ++i) {
    if(list[i].isFile() && !list[i].getName().startsWith(".")) {
        //toClientInfo.writeBytes(list[i].getName() + "\n");
        toClientInfo.write((list[i].getName() + "\n").getBytes());
        fromClient.readLine();
    }
}
```

(2) In client side, wait for the file name list right after get connected to the server. And display them in the 'Available Files' list.

```
int fileNum = Integer.parseInt(fromServerInfo.readLine());
outToServer.write("get\n".getBytes());
for(int i = 0; i < fileNum; ++i) {
    filename = fromServerInfo.readLine();
    fileList += filename + '\n';
    outToServer.write("get\n".getBytes());
}
txtrAvailableFiles.setText(fileList);
```

Acknowledgement

Thanks to Prof. Shen and TA for providing me this interesting assignment and assessing my applications and report!