Due: 2017/April/05 24:00

Assignment 3 Socket Programming

Student ID: 5140219073 Name: Cao Ruisheng Email: 211314@sjtu.edu.cn

1 Client/Server Model

1.1 Procedures

(a) Server listens to a given port, we use 2680 in this assignment.

(b) The client initiates a TCP connection to the server via IP address or hostname.

(c) If the TCP connection is established, the client send the absolute file path and filename to the server.

(d) The server checks whether the file exist, if it exists, the server sends the size of the file to the client first and starts to transmit the whole file afterwards. Otherwise, it sends *long* 0 to the client, and waiting to accept a valid filename.

(e) The client receives a *long* number first, if it equals to 0, it tells the user that the desired file doesn't exist and asks the client to input another valid filename. Otherwise, it creates a local file to save the file.

(f) After the file transfer completes, the server waits for another filepath and name, the client command line asks the client whether to continue.

(g) If the user types in N/n, the client tears down the connection and exits. Otherwise, the user can type in another filename and repeat the process.

1.2 Core codes

For complete code, refer to the project source code.

Client Side

```
ClientSide client = new ClientSide();
// Instantiate our created client socket class
clientSocket = new Socket(hostname, port);
// Create the input and output stream or channel
out = new DataOutputStream(clientSocket.getOutputStream());
in = new DataInputStream(
        new BufferedInputStream(
                        clientSocket.getInputStream()));
// Read one long number from the socket
long length=in.readLong();
// Writes a string to the socket using UTF-8 encoding
out.writeUTF(filepath);
// Create a file output stream to write bytes into the file
fos = new FileOutputStream(file);
// Create a byte array acting as a buffer
byte[] bs=new byte[Data_Size];
// Read bytes with fixed size from the socket
size=in.read(bs); //if EOF is read, size=-1
// Write the byte array into file with index from 0 to size
fos.write(bs,0,size);
// Close the socket and tear down theconnection
in.close();
out.close();
clientSocket.close();
```

Client Side GUI

The basic idea is similar, except that we import javax.swing.* and java.awt.* to create a JFrame window. Since it is not the main task in this assignment, we leave out the details of implementation.

Server Side

Read and Write methods with respect to the input and output stream of socket is similar to that of client side. We omit here and only list the different core codes.

```
//Create a server socket to listen to connection requests
ServerSocket server = new ServerSocket();
InetSocketAddress address=new InetSocketAddress(hostname,port);
//Bind the server socket with default hostname and port number
server.bind(address);
//Create a new socket to answer the client request
Socket connection = server.accept();
//Using multithreads to provide service
Thread task = new FileThread(connection);
//Enable the thread, we need to override the run() method
task.start();
```

1.3 Runnable Files

Export the source code to runnable jar files, to use these executable programs.

Server Side

Two methods, without argument or with argument, illustration see Figure 1.

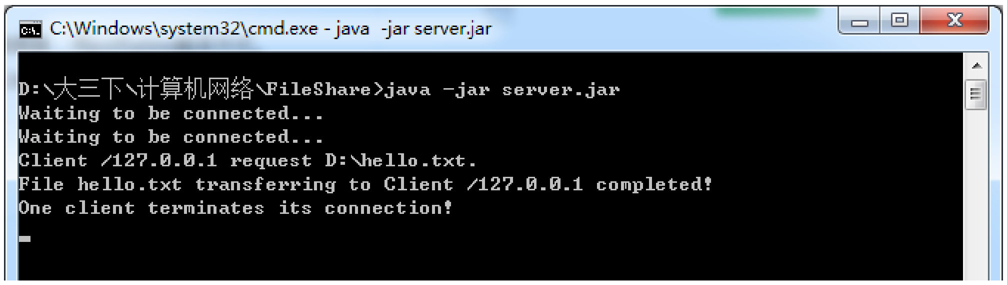
(a) Open the cmd and enter the location where server.jar file resides. Input

$$java\ -jar\ server.jar$$

(b) Open the cmd and enter the location where server.jar file resides. Input

$$java\ -jar\ server.jar\ 127.0.0.1$$

127.0.0.1 is replaced by the hostname or ip address that the server wants to bind.



C:\Windows\system32\cmd.exe - java -jar server.jar

```
D:\大三下\计算机网络\FileShare>java -jar server.jar
Waiting to be connected...
Waiting to be connected...
Client /127.0.0.1 request D:\hello.txt.
File hello.txt transferring to Client /127.0.0.1 completed!
One client terminates its connection!
```

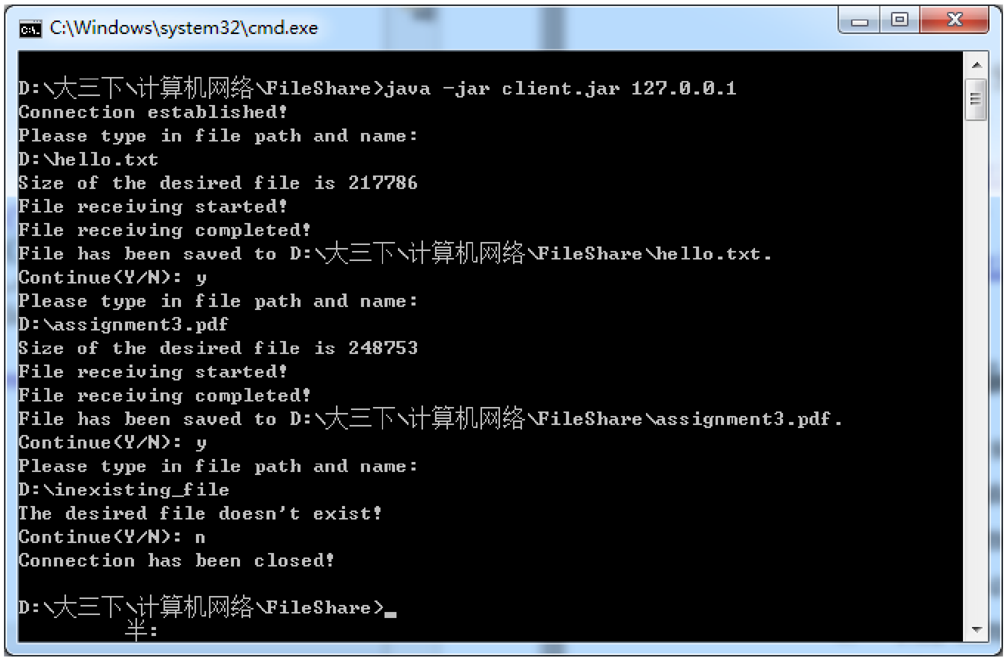
Figure 1: server.jar

Client Side

In client side, we must specify the destination hostname or ip address in the cmd mode. Illustration see Figure 2. Type in

$$java\ -jar\ client.jar\ 127.0.0.1$$

127.0.0.1 is replaced by the hostname or ip address of the server. After that, input the file path and name to be download according to instruction. The file will be saved to the directory with default filename where client.jar resides by default. Attention: the server program must be running on the server host!



```
D:\大三下\计算机网络\FileShare>java -jar client.jar 127.0.0.1
Connection established!
Please type in file path and name:
D:\hello.txt
Size of the desired file is 217786
File receiving started!
File receiving completed!
File has been saved to D:\大三下\计算机网络\FileShare\hello.txt.
Continue(Y/N): y
Please type in file path and name:
D:\assignment3.pdf
Size of the desired file is 248753
File receiving started!
File receiving completed!
File has been saved to D:\大三下\计算机网络\FileShare\assignment3.pdf.
Continue(Y/N): y
Please type in file path and name:
D:\inexisting_file
The desired file doesn't exist!
Continue(Y/N): n
Connection has been closed!

D:\大三下\计算机网络\FileShare>_
         半:
```

Figure 2: client.jar

Client GUI

The improvements are:

(a) User can connect and disconnect the server with freedom.

(b) User can specify the download path and filename to be saved, the program will check whether the directory is valid and whether the file already exists.

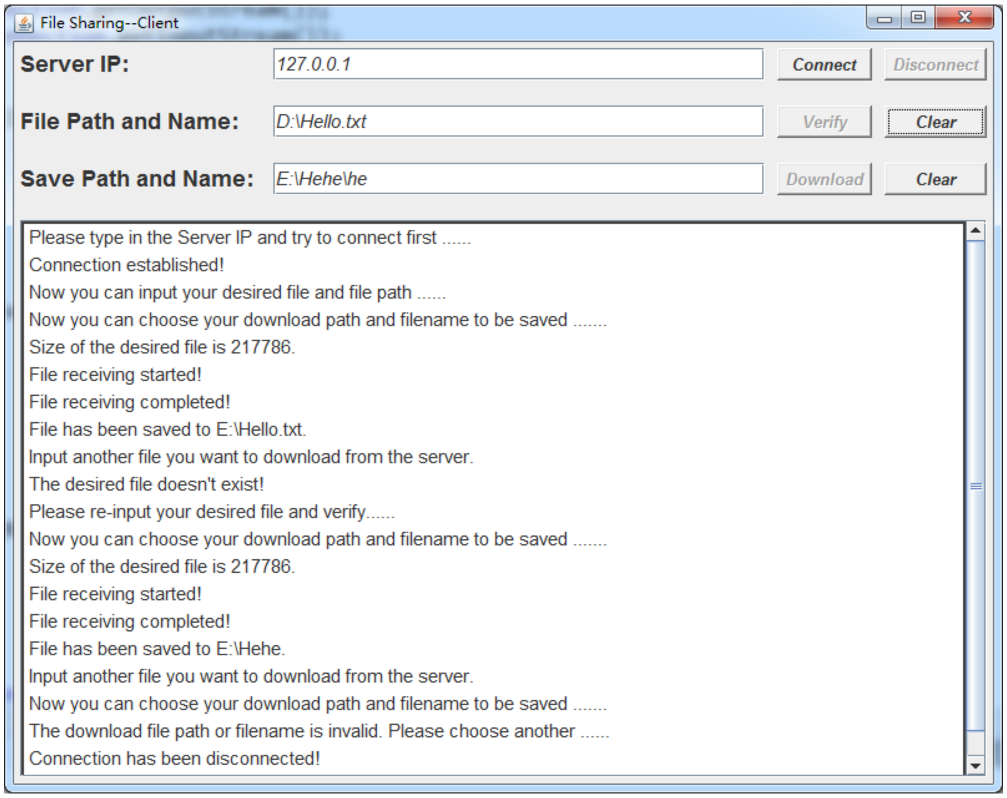
(c) Press Esc, user will exit.

(d) A GUI mode is more user-friendly.

First type in

$$java\ -jar\ clientgui.jar$$

in the cmd window. A GUI window will pop up. The remainging detailed operation is obvious from the GUI window, user only need to follow the instruction in the text area below. For illustration, see Figure 3.



Server IP: 127.0.0.1

File Path and Name: D:\Hello.txt

Save Path and Name: E:\Hehe\he

```
Please type in the Server IP and try to connect first ......
Connection established!
Now you can input your desired file and file path ......
Now you can choose your download path and filename to be saved .......
Size of the desired file is 217786.
File receiving started!
File receiving completed!
File has been saved to E:\Hello.txt.
Input another file you want to download from the server.
The desired file doesn't exist!
Please re-input your desired file and verify......
Now you can choose your download path and filename to be saved .......
Size of the desired file is 217786.
File receiving started!
File receiving completed!
File has been saved to E:\Hehe.
Input another file you want to download from the server.
Now you can choose your download path and filename to be saved .......
The download file path or filename is invalid. Please choose another ......
Connection has been disconnected!
```

Figure 3: clientgui.jar

2 P2P Model

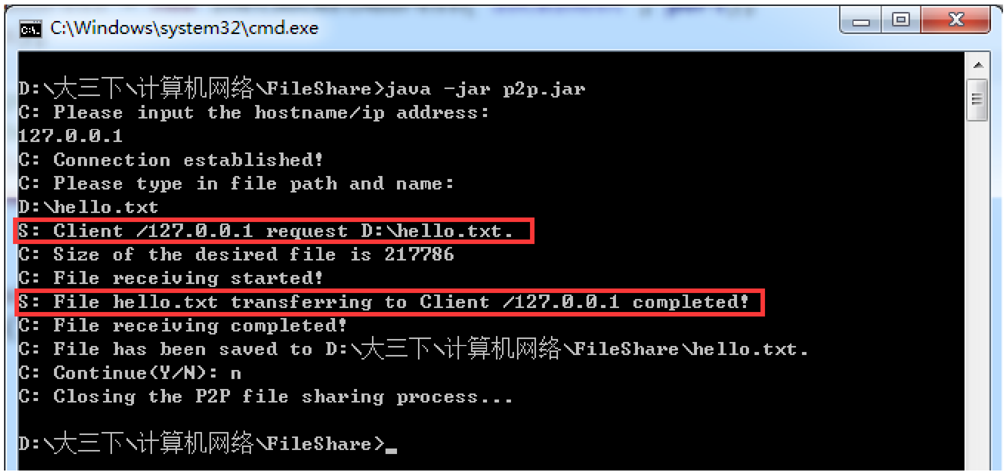
With the experience from C/S Model, all we need to do is create a thread to run the server side program, another thread or the current process to run the client program. To avoid redundant explanantion, we only provide the results and show how to run.

CMD Mode

Type in cmd (Illustration Figure 4)

$$java\ -jar\ p2p.jar$$

From Figure 4 we can find two lines surrounded by red square indicating that a server thread is running at the same time (since in our demo we are trying to connect to ourself).



```
C:\Windows\system32\cmd.exe

D:\大三下\计算机网络\FileShare>java -jar p2p.jar
C: Please input the hostname/ip address:
127.0.0.1
C: Connection established!
C: Please type in file path and name:
D:\hello.txt
S: Client /127.0.0.1 request D:\hello.txt.
C: Size of the desired file is 217786
C: File receiving started!
S: File hello.txt transferring to Client /127.0.0.1 completed!
C: File receiving completed!
C: File has been saved to D:\大三下\计算机网络\FileShare\hello.txt.
C: Continue(Y/N): n
C: Closing the P2P file sharing process...

D:\大三下\计算机网络\FileShare>_
```

Figure 4: p2p.jar

GUI Mode

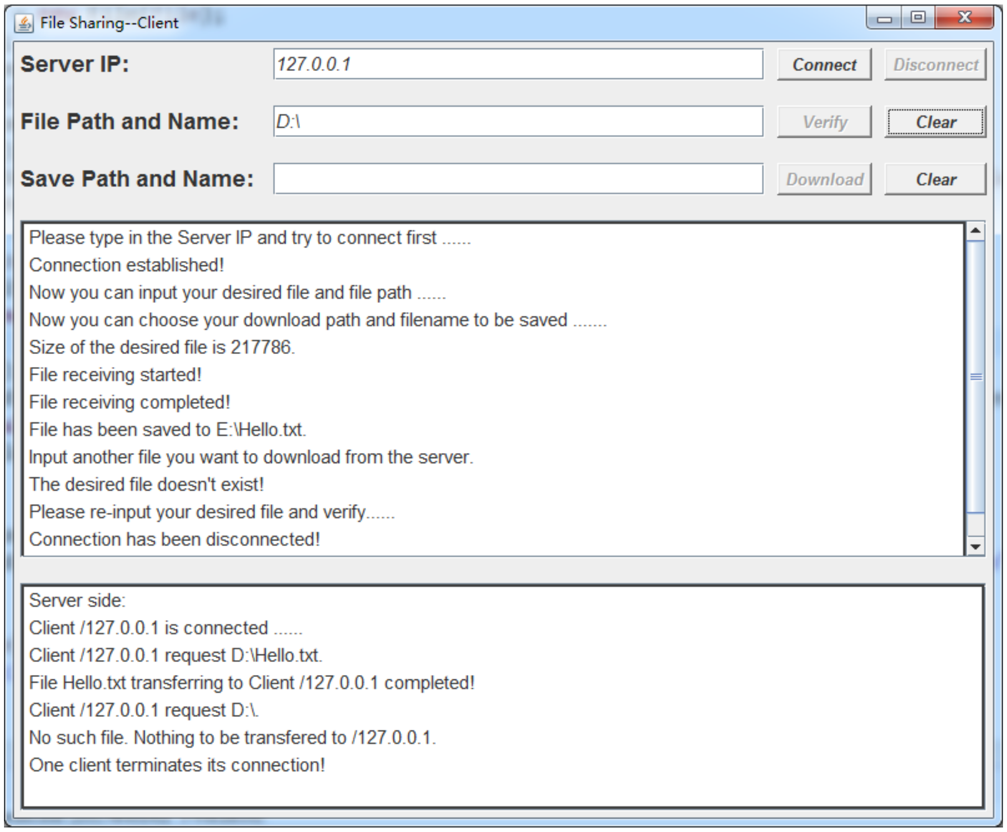
Codes to implementing the Esc exit function.

```java
JFrame client=new ClientGUI();
//Press Esc to exit the program
final Toolkit toolkit = Toolkit.getDefaultToolkit();
toolkit.addAWTEventListener(new AWTEventListener(){
public void eventDispatched(AWTEvent e){
if (e.getID() == KeyEvent.KEY_PRESSED) {
        KeyEvent evt = (KeyEvent) e;
        if (evt.getKeyCode() == KeyEvent.VK_ESCAPE) {
                client.dispose();
                System.exit(0);
        }
    }
 }
},AWTEvent.KEY_EVENT_MASK);
```

Type in cmd

$$java\ -jar\ p2pgui.jar$$

A GUI which contains two text areas will show up, one is for client side, the other is for server side.(Illustration in Figure 5)



127.0.0.1

D:\

```
Please type in the Server IP and try to connect first ......
Connection established!
Now you can input your desired file and file path ......
Now you can choose your download path and filename to be saved .......
Size of the desired file is 217786.
File receiving started!
File receiving completed!
File has been saved to E:\Hello.txt.
Input another file you want to download from the server.
The desired file doesn't exist!
Please re-input your desired file and verify......
Connection has been disconnected!
```

```
Server side:
Client /127.0.0.1 is connected ......
Client /127.0.0.1 request D:\Hello.txt.
File Hello.txt transferring to Client /127.0.0.1 completed!
Client /127.0.0.1 request D:\.
No such file. Nothing to be transfered to /127.0.0.1.
One client terminates its connection!
```

Figure 5: p2pgui.jar

3 Problems and Experience

Dertermine whether a file exists in the server

The client and server negotiate first, that is to say, in our application protocol, the server will send a *long* number denotes whether the file exists(equals to 0). However, there is little flaw that, if the naughty user wants to download a file with size 0, and there happends to be the exact empty file resides in the server. The server will send 0 to the client first denoting the size of the file. However, the client will consider it as an indication that the desired file doesn't exist. Since such circumstance seldom occurs, we just omit it.

Determine whether the file transfer is finished

I search the Internet, $size = in.read(bs)$ will return -1 if an EOF is detected. However, in my simulation, it seems that while the server has finshed its transfer, the client doesn't detect the EOF and is still waiting for bytes from the virtual pipe. To solve this problem, I reuse the variable *length* to determine whether the transfer is finished. Variable *current* is used to accumulate the size of file which has been transfered. If $current >= length$, it means that the transfer is finshed and the client can turn to doing something else.

About the accessibility of the server

Since nowadays most computers connect to the Internet via WIFI through routers. NAT technique is widely used. Many computers hide behind the NATs and it's difficult for other computers to establish a TCP connection unless TCP NAT-Traversal. To simplify our simulation and demonstration, I use the local host to be both client and server.

4 Suggestions

To ensure that every student really understand application level protocols, the task can be more specific, for example, before server transmitting the file, it has to send the student ID or some info special first, and the client has to display this information on the cmd or GUI window. In this way, the task for each student is different and avoid copying.