

EE321 Lab 3

File Transfer Application

Server

Aim:

The goal for this server is to have it manage files and allow a connection from a client and receives requests from this client. The client can request to interact with these files in a number of ways. Depending on the request received the server will, list the files on the system, add a new file to the server, sends the contents of the file to the client, or simply close the connection.

Procedure:

I began by writing the code to allow the server to accept a connection from a client and to get an initial input from the client. This input will be the request from the client-side software. This was pretty simple and just required initializing a `ServerSocket` and a `PrintWriter` and a `BufferedReader`. Then initialize the client socket by accept a connection through the server socket. With this done the server can now both read requests from the client as well as write to the client.

I then went on to write a simply menu code, using if statements rather than a switch statement in order to ignore the input filenames and such.

The first function I set up was the `server_ls` function to list the files on the system. I did this by opening the file using the Java `FileReader` and printing each line out to the client using a `BufferedReader`. I also had this function return the number of lines in the file as an `int` as it may be useful later.

```
while((line = bufferedReader.readLine()) != null) {  
    tout.println(line);  
    indlength++;  
}
```

With this complete I moved on to create the disconnect function. I accomplished this by putting the server client connection code into a while loop that's variable is changed when the client makes the `server_quit` request.

The `server_get <filename>` was a bit more complex. I first searched through the `index.txt` to see if the file existed and if it does it reads the entirety of the file and prints it to the client. If the file name is not found in the `index.txt` file it prints a null string to the client.

```
if(found==true) {  
    String line2 = null;  
    FileReader fileReader = new FileReader(filename);  
    BufferedReader bufferedfileReader = new BufferedReader(fileReader);  
    //buffered reader so it reads a line at a time  
    while((line = bufferedfileReader.readLine()) != null) {  
        tout.println(line);  
    }  
    bufferedfileReader.close();  
    found = false;  
}
```

The final function to make on server is to implement the `server_put <filename>` function. I did this by taking the file name input by the client creating a writer to that filename. This created the file and

allows me to print to it. Then I read from the client the content of the file until a null character is sent. Then the file name is added to the server index and the writers are closed.

Results of this server are shown with client section.

Source code:

```
import java.io.*;
import java.net.*;

public class File_Server {

    public static void main (String[] args) throws IOException{
        try {
            ServerSocket myServerSocket = new ServerSocket(1234); //create a server side socket with port number 1234
            PrintWriter out = null;
            BufferedReader in = null;
            String request = null;
            boolean quit = false;

            while(true) { //so the server stays open

                Socket connectedClientSocket = myServerSocket.accept(); //accept a client that connects through the socket

                out = new PrintWriter(connectedClientSocket.getOutputStream(), true); //creates the output stream through
                the client socket and sets it to auto flush
                in = new BufferedReader(new InputStreamReader(connectedClientSocket.getInputStream())); //reads from
                client

                while(quit==false) {
                    request = in.readLine();

                    if((request.length()>8) && (request.equals("server_ls"))){ //the length test is to avoid a null
                    pointer crash
                        server_ls(out); //list files function
                    }
                    else if((request.length()>10) && (request.substring(0, 10).equals("server_put"))){
                        server_put(request.substring(11), in); //put file function
                    }
                    else if((request.length()>10) && (request.substring(0, 10).equals("server_get"))) {
                        server_get(request.substring(11), out); //get file function
                    }
                    else if((request.length()>10) && request.equals("server_quit")) {
                        quit = true; //so that client is disconnected
                    }
                    else out.println("invalid request");
                }

                out.close(); //closes the output stream
                connectedClientSocket.close();
                quit = false;
            }
        } catch (IOException e) { //runs if an error occurs
            System.out.println("Error:" + e.getMessage()); //prints error message
        }
    }

    private static void server_get(String filename, PrintWriter tout) throws IOException { //Get File
        String indFile = "index.txt";
        String line = null;
        boolean found = false;
        FileReader indexReader = new FileReader(indFile); //reads file
        BufferedReader bufferedindReader = new BufferedReader(indexReader); //buffered reader so it
        reads a line at a time
        while((line = bufferedindReader.readLine()) != null) { //reads through entire list
            if(line.equals(filename)) found = true;
        }
        if(found==true) {
            FileReader fileReader = new FileReader(filename); //reads file
            BufferedReader bufferedfileReader = new BufferedReader(fileReader); //buffered
            reader so it reads a line at a time
            while((line = bufferedfileReader.readLine()) != null) { //reads through entire list
                tout.println(line); //prints the lines to client
            }
            tout.println("null");
            bufferedfileReader.close();
            found = false;
        }
    }
}
```

```

        else {
            tout.println("null");
        }
        indexReader.close();
    }

    private static void server_put(String filename, BufferedReader tin) throws IOException { //Add File
        String line = null;
        FileWriter FileWriter = new FileWriter(filename);
        while(((line = tin.readLine()).equals("null")) != true){
            FileWriter.write(line + "\r\n");//writes input to file
        }
        FileWriter indexwriter = new FileWriter("index.txt",true); //the true will append the new data
        indexwriter.write("\r\n" + filename);//appends the string to the file
        indexwriter.close();
        FileWriter.close();
    }

    private static int server_ls(PrintWriter tout) throws IOException {
        String fileName = "index.txt";
        String line = null;
        int indlength = 0;
        FileReader fileReader = new FileReader(fileName);//reads file
        BufferedReader bufferedReader = new BufferedReader(fileReader); //buffered reader so it reads a
line at a time
        while((line = bufferedReader.readLine()) != null) { //reads through entire list
            tout.println(line);//prints the lines to client
            indlength++;
        }
        tout.println("null");
        bufferedReader.close();
        return indlength;//returns length of list
    }
}

```

Client

Aim:

The aim here is to create a client-side software that communicates with the server to request the commands the server has been set up to perform. This means the client can request the server to list the files available and then have the client display the list to the user, the ability to create a new file on the server and input the data for that file in to the client, the ability to request a file from the server and save it on the client's machine, and to disconnect from the server.

Procedure:

I began by setting up a client to connect to the server. I did this by first initializing a socket that in this case connected to "localhost" and the server port number. I set up a data output stream to the server and a buffered reader on the socket input stream. I then sent a request to the server to list the files as a test to see if it connected correctly.

```
Socket myClientSocket = new Socket("localhost", 1234);
ServerOut = new DataOutputStream(myClientSocket.getOutputStream());
ServerIn = new BufferedReader(new InputStreamReader(myClientSocket.getInputStream()));

String line;
ServerOut.writeBytes("server_ls" + '\n');
while((line = ServerIn.readLine()) != "\n") {
    System.out.println(line);
}
```

With this working I knew that my client software connected to the server so I could continue implementing the client functionality. I began first by implementing a menu system for selecting the function to be performed, I did this by reading in a command from the user, then split the string (probably should have done this for the server also but forgot it existed) and then ran the specified function.

```
while(quit == false) {
    System.out.println("input command:");
    request = clientIn.readLine();
    String[] rAry = request.split(" ");

    switch(rAry[0]) {
        case "ls":
            //list files function
            System.out.println("ls");
            break;
        case "put":
            //put file function
            System.out.println("put");
            break;
        case "get":
            //get file function
            System.out.println("get");
            break;
        case "quit":
            quit = true; //so that client is disconnected
            break;
        default:
            System.out.println("invalid request");
            break;
    }}
}}
```

I then implemented the list functionality to the client. I had the client send the server the server_ls command and then display what the server sent back to the user.

```
String line;
ServerOut.writeBytes("server_ls" + '\n');
while((line = ServerIn.readLine()).equals("null") != true) {
    System.out.println(line);
}
```

When the null string is received from the server it knows that all lines have been received.

Next I added the put file functionality, I started by writing the code to find the specified file on the client system, once I had the code working to find the file I sent a command to the server server_put <filename> then I sent the contents of the file to the server and once all the lines of the file were sent I then sent a null string to notify the server the file has been sent.



```
ServerOut.writeBytes("server_put " + filename + "\n");
FileReader fileReader = new FileReader(filepath + "\\" + filename); //reads file
BufferedReader bufferedfileReader = new BufferedReader(fileReader);
while((line = bufferedfileReader.readLine()) != null) { //reads through entire file
    ServerOut.writeBytes(line + "\n"); //prints the lines to server
    System.out.println(line);
}
ServerOut.writeBytes("null\n");
```

The final function to implement is the get function to receive a file and save it to a specified location on the client system. I did this similarly to the put command by first setting up a filewriter to the file in the specified path, then I sent the get command to the server and wrote each line received back from the server to the new file.

```
ServerOut.writeBytes("server_get " + filename + "\n");
while(((line = ServerIn.readLine()).equals("null")) != true){
    filerecieve = true;
    FileWriter.write(line + "\r\n"); //writes input to file
    System.out.println(line);
}
Filewriter.close();
if(filerecieve) System.out.println("file recieved");
else {
    System.out.println("File not found");
    newfile.delete();
}
```

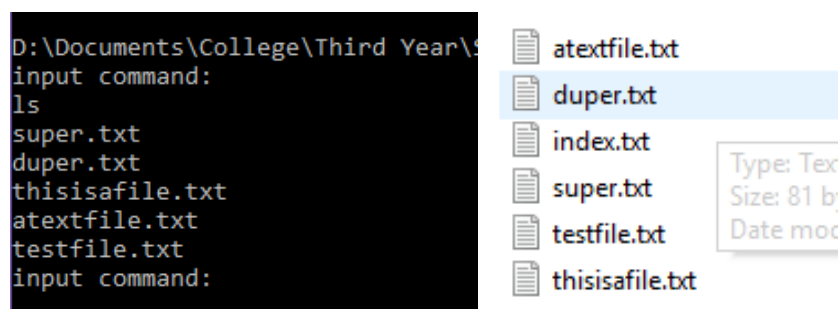
Results

With both the client and the server finally complete it was time to test them both. I exported them as jar files for testing. I simply had to run the server by double clicking as it needs no interface. The client had to be run from the windows command line.

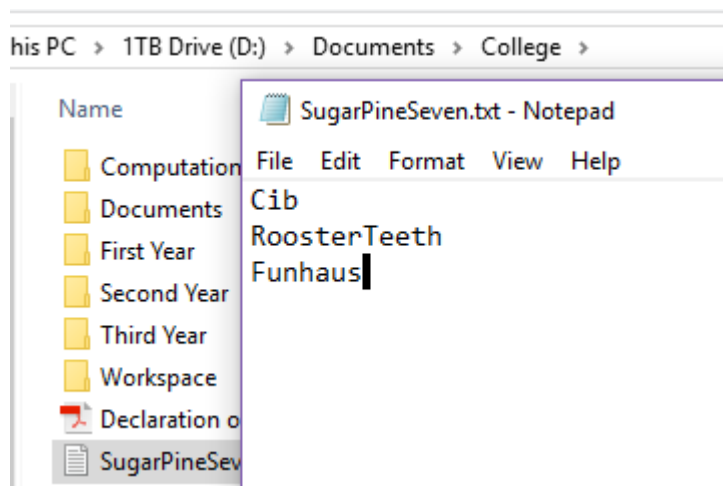
 Client.jar	04/03/2018 16:32	Executable Jar File	12 KB
 Server.jar	04/03/2018 16:24	Executable Jar File	12 KB

```
D:\Documents\College\Third Year\Semester 2\Datacomms and Networks\Assignments\Assignment3>java -jar Client.jar
```

I first tested the ls command to see if it would list the files in the server directory and it did.



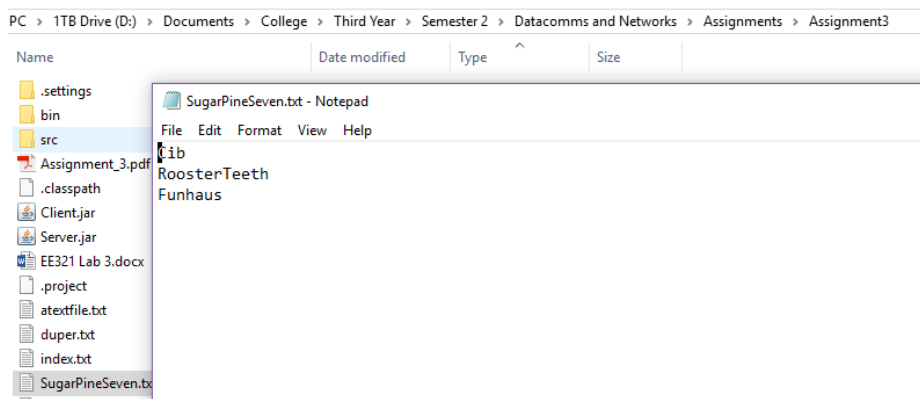
I then decided to make a new file and put it on the server using the put command. I first had to make the file on the client machine. I named this file SugarPineSeven.txt and placed it in D:\Documents\College



I then entered the put command into the command line (the file line readouts were for testing and can be disabled)

```
input command:
put SugarPineSeven.txt D:\Documents\College
Cib
RoosterTeeth
Funhaus
file sent
input command:
```

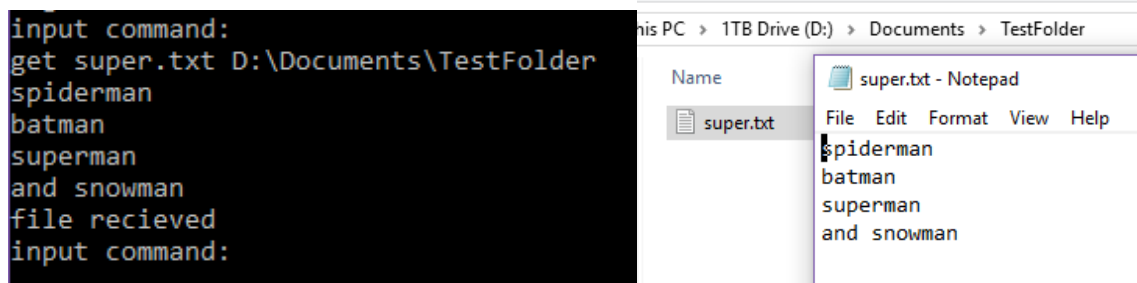
The file successfully was saved into the server's directory by the server:



And then when using the ls command again we can see it had been added to the index file:

```
input command:
ls
super.txt
duper.txt
thisisafire.txt
atextfile.txt
testfile.txt
SugarPineSeven.txt
input command:
```

Happy with the fact that the put functionality is working it was time to test the get command, I input the get command to save the super.txt file to D:\Documents\TestFolder



Again this worked. I am very happy with how this Server and client worked and I learnt a lot about writing code for client-side software in java.

Client Source code:

```
import java.io.*;
import java.net.*;

public class File_Client {

    static BufferedReader clientIn = null;
    static DataOutputStream ServerOut = null;
    static BufferedReader ServerIn = null;
    static final String host = "localhost";
    static final int port = 1234;

    public static void main (String[] args) throws IOException{
        try {
            String request = null;
            boolean quit = false;
            clientIn = new BufferedReader(new InputStreamReader(System.in));

            Socket myClientSocket = new Socket(host, port);//creates the client socket
            ServerOut = new DataOutputStream(myClientSocket.getOutputStream());//creates output to server
            ServerIn = new BufferedReader(new InputStreamReader(myClientSocket.getInputStream()));//in
            //from server

            while(quit == false) { //for closing client
                System.out.println("input command:");
                request = clientIn.readLine();//reads command from user
                String[] rAry = request.split(" "); //splits command into string array

                switch(rAry[0]) { //sees what first string in command is
                    case "ls":
                        ls(); //list files function

                    break;
                    case "put":
                        if(rAry.length==3) put(rAry[1], rAry[2]);//send file to server
                        else System.out.println("invalid request");

                    break;
                    case "get":
                        if(rAry.length==3) get(rAry[1], rAry[2]);//get file from server
                        else System.out.println("invalid request");

                    break;
                    case "quit":
                        quit = true;//so that client is disconnected

                    break;
                    default:
                        System.out.println("invalid request");//if a bad request

                    break;
                }
            }

            ServerOut.writeBytes("server_quit");//disconnects from server
            ServerOut.close();
            ServerIn.close(); //closes everything
            myClientSocket.close();
        }

        catch(FileNotFoundException ex) {
            System.out.println(ex);
        }
    }

    private static void get(String filename, String filepath) throws IOException {
        String line = null;
        File newfile = new File(filepath + "\\\" + filename);//creates file
        FileWriter FileWriter = new FileWriter(newfile);//to write to file
        Boolean filerecieve=false; //if file is recieved
        ServerOut.writeBytes("server_get " + filename + "\n");//sends get command to server
        while(((line = ServerIn.readLine()).equals("null")) != true){ //reads through entire file
```



```

        filereceive = true; // indicates a file was received
        FileWriter.write(line + "\r\n"); // writes input to file
    }
    FileWriter.close();
    if(filereceive) System.out.println("file received");
    else {
        System.out.println("File not found");
        newfile.delete(); // deletes empty file
    }
}

private static void put(String filename, String filepath) throws IOException {
    String line = null;
    ServerOut.writeBytes("server_put " + filename + "\n"); // sends put command
    FileReader fileReader = new FileReader(filepath + "\\ " + filename); // reads file
    BufferedReader bufferedfileReader = new BufferedReader(fileReader);
    while((line = bufferedfileReader.readLine()) != null) { // reads through file
        ServerOut.writeBytes(line + "\n"); // prints the lines to server
    }
    ServerOut.writeBytes("null\n"); // to indicate file sent
    System.out.println("file sent");
    bufferedfileReader.close();
}

static void ls() throws IOException {
    String line;
    ServerOut.writeBytes("server_ls" + '\n'); // sends list command
    while((line = ServerIn.readLine()).equals("null") != true) { // receives line from server
        System.out.println(line); // prints line to user
    }
}
}

```