ICS451: Data Networks, Programming Assignment #1

Oscar Hernandez

June 23, 2024

1 Introduction

The codebase contains seven files:

1. a text file *server_file.txt* which is to be sent by the server
2. a server program *MyServer.java*, which is compiled by *javac MyServer.java*
3. a text file *server_log.txt* which is the log generated by executing *java MyServer.java* 3000 ¿ *server_log.txt* 2 > &1
4. a client program *MyClient.java*, compiled by running *javac MyClient.java*
5. a text file *client_log.txt* which is the log generated by executing *java MyClient.java* 3000 ¿ *client_log.txt* 2 > &1
6. a text file *client_file.txt* written by *MyClient.java*
7. this report and a document *README.md* with instructions on execution.

We describe the computer programs for the server in Section 3 and for the client in Section 2, both of which begin by importing classes for networking and for input/output.

2 Client

The client program creates a class for the *MyClient* which begins by defining variables for the hostname and port. It tries to connect to the socket, catching some common exceptions if they are thrown. If it succeeds, then it does the following:

1. creates a Buffered Reader for the input stream it gets from the socket
2. creates a Print Writer that writes to *client_file.txt*
3. for each line read from the reader until the end of the stream, it writes it to the file
4. closes the writer and prints a message indicating success.

3 Server

Like the client, the server program begins by importing *java.io.** and *java.net.**.

Then, it implements a class for the *MyServer* which begins by defining a variable for the port.

It tries to create a new server socket. If an I/O exception is thrown, then the program prints a message and the stack trace.

If the new server socket is successfully bound to the port, then the program prints a message indicating success and tries to accept an incoming request to the socket. If it succeeds, then the program proceeds to do the following:

1. prints a message indicating that the server accepted the client
2. creates a file object for *server_file.txt*
3. creates a Buffered Reader for a File Reader object for the file
4. creates a Print Writer object for writing to the Output Stream of the socket
5. creates a *line* object of class *String*
6. assigns *line* to be the value of the line read from the Buffered Reader for each line until one is null and writes each line to the socket
7. closes the Buffered Reader
8. prints a message indicating that the server successfully read the file and transmitted to the client via the socket.

4 Conclusion

We have written applications in the Java computer programming language for a simple HTTP client and a separate HTTP server. As soon as the client connects to the server socket, the server sends a simple text file *server_file.txt* which is stored by the client in a separate text file *client_file.txt*. The Java Socket class is used and the connection is closed by both applications once the file is transmitted.

The code is working as it should be with no issues, but it should be noted that multi-threading is not implemented. The code also implements print statements that show the flow through each method. Absolute paths are not used to specify the file to transfer in the program; instead, a relative path is used which expects a file called *server_file.txt* to exist within the working directory in which the application is called.

The code was written by the author who provided his name and UH ID as a comment in both programs. None of the code is copied from the Internet without proper citations in the code. The code does follow programming motifs from two Java tutorials in the Oracle documentation, and these are both cited in the server application.