COMP2221 Networks – Coursework 2 Report

Jacob Holland – SC15J3H

Server Communication Protocol

Both the client and the server interact via 2 input and output TCP sockets via an allocated client handler for a specified client. When the server is initialised, it will create a fixed thread pool so that a maximum of 10 clients can connect to the server socket on port number 4444. For each accepted client, it will be allocated a thread with a dedicated client handler which can handle requests and process them server-side (via the use of the executor service).

Each Client is viewed with in java as an individual object with a socket which can accept incoming messages via a Scanner to view messages, and a PrintWriter to write string objects in order to talk to the server. The Client handler is passed the client's socket and provides a Scanner to process input for the server and a PrintWriter in order to process server output. It also has a logger to record any major client requests to be viewed at a later date.

The client and server interact with each other using a series of string commands in the format '$COMMAND$'. When they need to initialise a function on either side. The client has a standard CLI which presents a list of commands, where the user can interact with the program using plain English. The Client will use the system input when a line is sent via the terminal, and check if it contains any 'key words' that will require an interaction with the server. In this program, there are only 3 valid commands:

1. LIST
2. DOWNLOAD
3. EXIT

List

If the terminal successfully receives the 'list' command, then the client is requesting a list of available folders that the server can provide for it to download. This is done by initially sending the '$LIST$' Command to the output socket of the client, which is in turn received by the client handler's input socket. As the client handler has a condition for when it receives this string in this exact format, then it will proceed to call a listDirectories function, which generates an array of available files that the server provides (granted they are a directory). With this array generated, it generates a string to nicely format the contents of the array. This returned string is then sent to the Printwriter/Output socket of the server, which is then received by the Scanner/input of the client, which prints out the string in the terminal.

Download

Whilst the List command is relatively easy to implement, the Download function requires a lot more complex interaction between the client and server. The Clients CLI will detect the instance of a ('download') string, and will then ask for a folder name to be typed. The client will then send the command of '$DOWNLOAD$' followed by the name of the folder that it has requested. The client handler will then receive this information via it's input socket and will commence a check to see if that folder exists in the PublicServer folder. If it does, then a sendFile function will pass through the name of the directory and commence file transfer.

The functions to send files between the client and server are done simultaneously on the client and server side respectively (specifically the Server commands are handled via the ClientHandler). Upon request, the server will provide the directories location and will initialise a new server socket specifically for file transfer. With this new socket, it creates a data output string to send commands between sockets and a buffered output string to send segmented packets for larger items (in this instance – files). Once this has been initialised, the a '$DOWNLOAD'$ command will be sent to the client, which will trigger it to connect to the server and create a buffered input string for the files as well as a data input stream for the file information.

When the sockets have connected, the server will generate an array of files that need to be sent to the client. It will then send the size via the data output stream so the client is aware of what it is receiving, alongside the name of the directory (so the client can generate a correct folder structure on its end). The server begins to loop through the files in the array and send the appropriate information so that the client can differentiate between the files. Once the client has received this information and has generated the correct file/directory locally, the server starts sending TCP packets of the file by writing them to a File output stream to be passed to the buffered stream. The Packets are then received by the clients buffered input stream, which sends it to a local buffer to be written to the specified file. Once the client has downloaded enough packets equivalent to the file size of the intended file, then both the client and server will move to the next file and repeat the process until the array is exhausted. Once the array is exhausted, then both file transfer sockets close and regular communication can recommence.

Exit

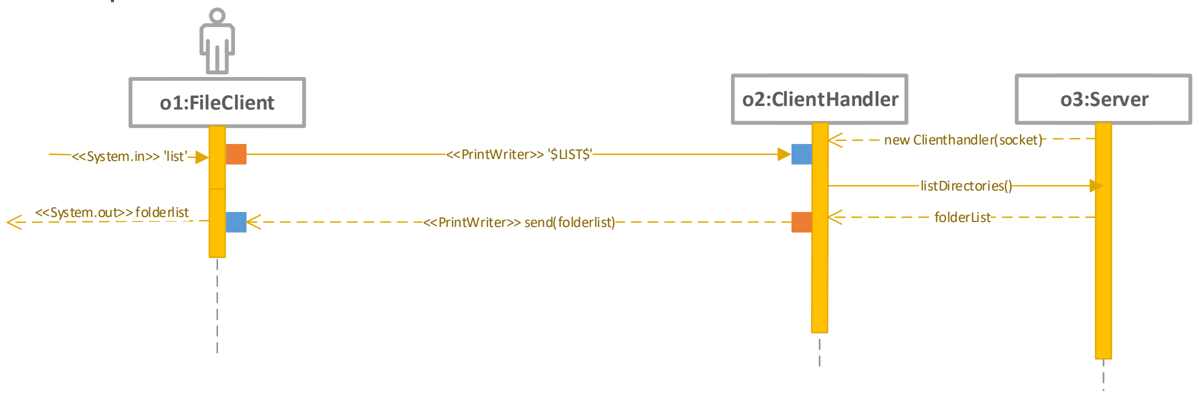
This process simply triggers a close command on all respective sockets and scanners for both the client handlers and the client sockets, which allows another client to connect to the server.

Client-Server Interaction Diagrams

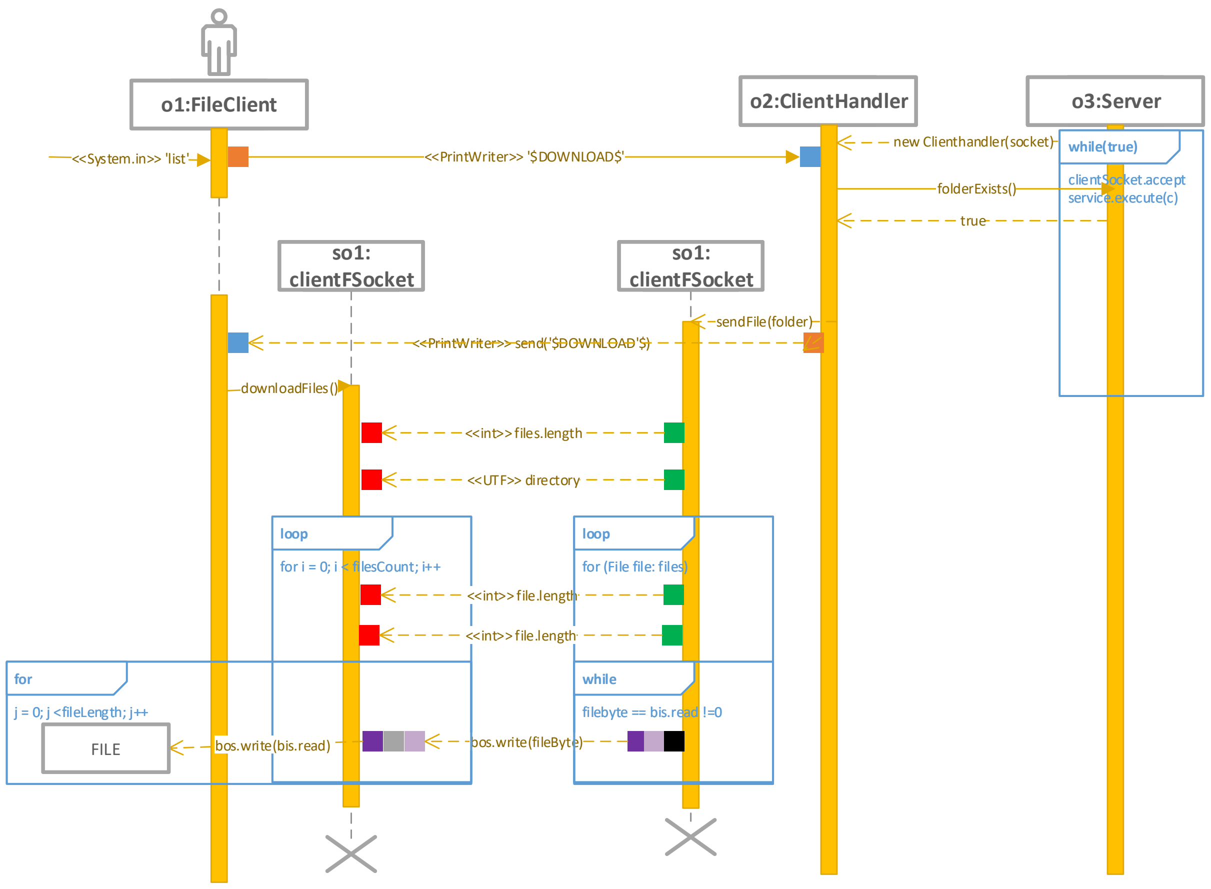
Key (Sockets) *Colour to the left of the text represents their socket*:

Scanner	FileInStream	DataInStream	BufferedInputStream
PrintWriter	FileOutStream	DataOutStream	BufferedOutPutStream

List Request



Folder Request



Reference List

Mark Walkey – ChatServer, ChatClient & ClientHandler from Lecture 13 of Networks. This has provided the foundation for the client-server model and was modified to accept specific commands and run on the executor thread pool.

Logging: "Java Logging Explained in 3 minutes" – https://www.youtube.com/watch?v=4Bpg5i4tUFg
File handling implementation for the class was used to make logging to a file convenient.