Overview

For the second practical we had to write a simple client-server system that allows a user to search for a test file held on a server, the serves should then respond with the contents of the text file, which is printed out on the clients' side. It was required that the server should provide a service loop which first set up a server socket then accept a connection. Next it should read the name of a file from the client, look up that file in the same directory as the server was set-up in. If the file is there, send its contents back; if not, send back a suitable error. Finally, the sever must close the connection and go back to waiting on client connections. The client should read the name of a document and request it from the server, displaying the result or an error message.

Design

For the basic specification I have made a program consisting of five classes, which include: "Client", "ClientProtocol", "Server", "ServerProtocol" and "Serverthread". The first class, "Client", contains only one main method, which takes in the server name and port number as program arguments and uses them to create a socket. Next a new instance of the class "ClientProtocol" is created, which takes in the socket. Finally, a while loop is declared, which will keep the client connected to the server, until the client enters "exit" in the terminal. In the while loop a new DataInputStream object is created, which is used to read data sent from the server. The reason I opted to use a DataInputStream is because a DataInputStream acts like an InputStream to read data directly as primitive data types, which is all the server will send. Finally, in the while loop the response from the server is saved and interpreted.
To make the program more efficient and data less susceptible to corruption through the network, I decided to only have the server send numbers to client, which indicate what state the server is in i.e. waiting or file sent. The interpretation of the server's response is done in the "ClientProtocol" class. In here there are two methods. The first method "processInput" which takes in the numerical response from the server and translates to an appropriate message. If the server indicates the file has been found a new "DataInputStream" is created and the contents of the found file are received and printed. In order to print multiple lines from the file a while loop is declared here, which will break when the input line from the file is equal to "EOF". Finally, the "ClientProtocol" class contains a method "sendRequest" which takes in a file name from client, sets up a "DataOutStream" and send the requested file name to the server.
The "Server" class bares close resemblance to the "Client" class in the sense it has only one main method, however this time it takes in only a port number used to set up a "ServerSocket". Next a loop is defined, in which new instances of the "ServerThread" class are created on new Threads. The reason I've done this is so the server can accept and serve multiple clients simultaneously. This is possible because "ServerThread" class extends the Thread Class. In the "ServerThread" class there is the implemented method "run" which houses most of the code in this class. The "run" method is called when the thread was constructed using the "Server" class. In the run method, a new "ServerProtocol" object is constructed along with a "DataOutputStream". Initially the "ServerProtocol" object is used to process an input of null, signifying the server is waiting for some input from the client. This response is then sent to the client. Next a do while loop is defined. In here the a "DataInputStream" object is created and server listens for input from the client. Any input received from the client is then processed. The loop is terminated when the client sends the word "exit". It should be noted here if the client terminates their session without entering the word exit, the server will catch an IO exception. The final class is the "ServerProtocol" class, this class is used to handle any data sent by client. It houses two core methods, "ProcessInput" and "ContentsOfFile". The "ProcessInput" method takes in a string from the client. The method will first check if a file name on the server matches the input string, if so a 2 is sent to the client and then the "ContentsOfFile" method is called. Here an array list is created and each line from the file is added to the list. Once all the lines are added, the string "EOF" is added at the end. The array list is then handed back the "ProcessInput" method and each item in the list is sent to the client. However, if no file is found the method check if the input was "exit". Finally, if not exit or a file name a 3 is sent to the client, signifying nothing was found.

Extensions

I've implemented the following extensions:

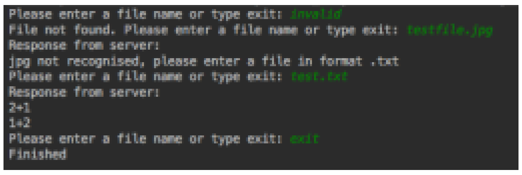
- Multi-threading
- GUI
- Ability to send files of different type through the socket
- Ability to open a downloaded file with default program

For this practical I have decided to create a GUI, this is simply because it enforces validation and verification and generally makes the program more user friendly.

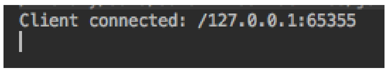
When the program is first run an instance of the GUI class initiated on a new thread and the graphical user interface is displayed. The GUI is relatively basic and only contains a few components. The first components are the “jTextFieldServerName” and the “jTextFieldPort” text fields, which take in the server name and port number. Once a connection to the server is established these are disabled and display the server name and port number respectively. The next component is the jButtonConnect which when clicked attempts to connect to the server using the server name and port number. Once a connection is made this button can then be used to disconnect from the server too. When a connection is active the “jButtonSearch” and “jTextFieldFile” are enabled and allow the user to search for a file located on the server. I’ve used a jTextAreaOutput which logs the response from the server. The next component is the “jProgressBar” which is used when a file is being transferred from the server to client and fills at the rate the file is being transferred. Finally, I’ve added a “jButtonOpen” which allows the user to open the most recently transferred file with the default program.

For an extra extension I have included functionality which allows the server to send and thus the client to receive files of any format. This is a valuable extension, because files can reside in many different formats and so a program only able to transfer txt files is very limited. To implement this when a client requests a file and server finds a file a new server socket is set up and the file is sent down the socket a stream of bytes. The client then receives this stream of bytes and reconstructs it to make the file. Because the program is also using the TCP protocol any corrupted packets will be resent.

Examples

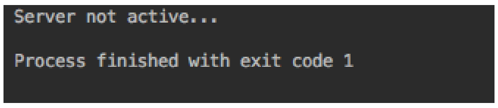


```
Please enter a file name or type exit: invalid
File not found. Please enter a file name or type exit: testfile.jpg
Response from server:
jpg not recognised, please enter a file in format .txt
Please enter a file name or type exit: test.txt
Response from server:
2+1
1+2
Please enter a file name or type exit: exit
Finished
```



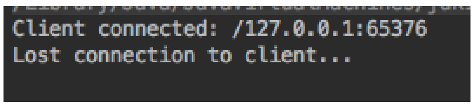
```
Client connected: /127.0.0.1:65355
```

Check connection made, file not found, file found, file wrong format and exit



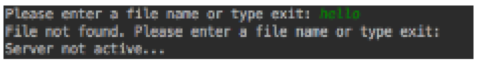
```
Server not active...

Process finished with exit code 1
```

Check connection to invalid server



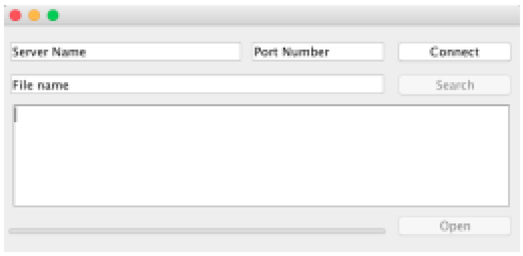
```
Client connected: /127.0.0.1:65376
Lost connection to client...
```

Check client connection lost



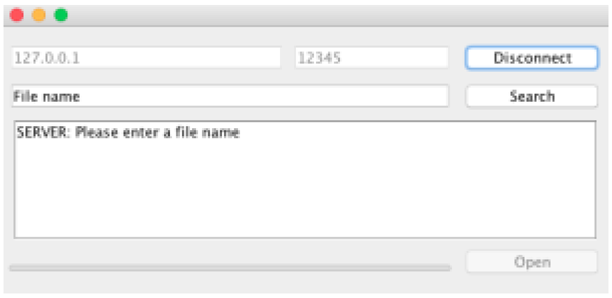
```
Please enter a file name or type exit: hello
File not found. Please enter a file name or type exit:
Server not active...
```

Check server connection lost



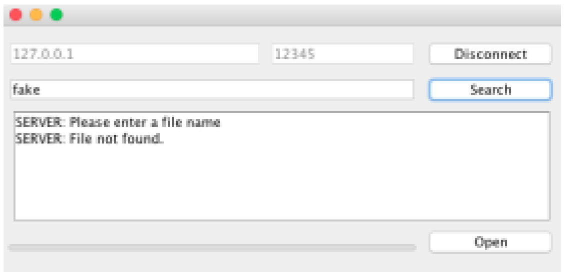
Starting interface

After connection has been made



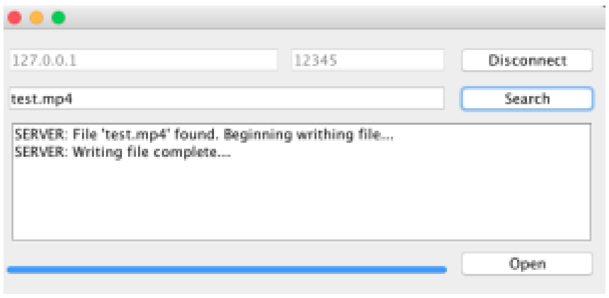
```
SERVER: Please enter a file name
```

Searching for a file not on the server



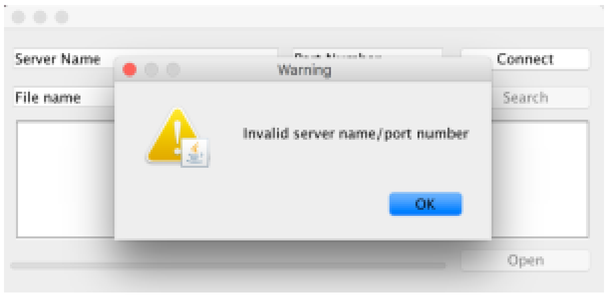
```
SERVER: Please enter a file name
SERVER: File not found.
```

Searching for a file on the server



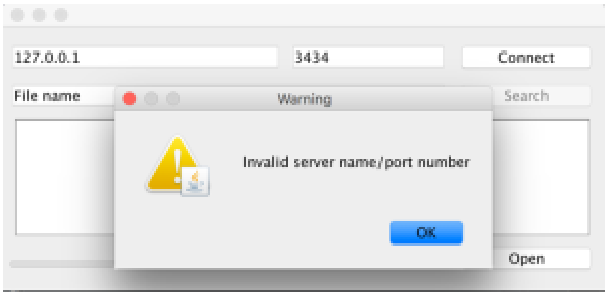
```
SERVER: File 'test.mp4' found. Beginning writhing file...
SERVER: Writing file complete...
```

Entering an invalid sever name and/or port number



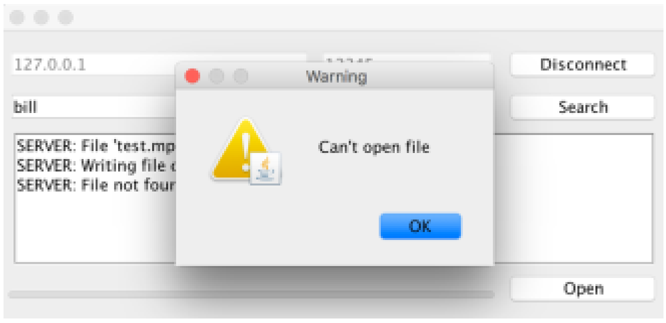
Invalid server name/port number

Entering reserved port number



Invalid server name/port number

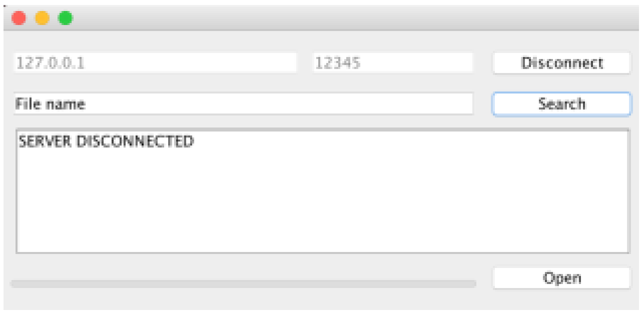
Trying to open an invalid file



```
SERVER: File 'test.mp
SERVER: Writing file c
SERVER: File not four
```

Can't open file

If the server disconnects



```
SERVER DISCONNECTED
```

Testing - Basic

Test Description	Test Data	Expected Output	Actual Output	Comment
Searching for valid file	Test.txt	The contents of the file should be printed on the client's terminal	The contents of the file were printed on the client's terminal	Works as expected
Entering "exit"	"exit"	The client's terminal should graceful exit and a message should be displayed on the server terminal	The client's terminal gracefully exited and a message was displayed on the server terminal	Works as expected
Search for file that exists but isn't a text file	N/A	The server should send a message notifying the client it can only handle text files	The server sent a message notifying the client it can only handle text files	Works as expected
Incorrect number of program arguments	N/A	An error message should be displayed notifying the client or server the incorrect number of program arguments have been entered	An error message was displayed notifying the client or server the incorrect number of program arguments had been entered	Works as expected

Testing – Basic & Extension

Test Description	Test Data	Expected Output	Actual Output	Comment
Connecting client to server	Port number:12345 Server name:127.0.0.1	A message will be displayed on the client side, asking to input a file and a message will be displayed on the server side saying a client has connected and the clients IP	A message was displayed on the client side, asking to input a file and a message was be displayed on the server side saying a client has connected and the clients IP	Works as expected
Searching for invalid file	Invalid	A message telling the client the file hasn't been found should be printed on the client's terminal	A message telling the client the file hasn't been found was printed on the client's terminal	Works as expected
Manually terminating the client	N/A	A message should be displayed on the server terminal saying connection to a client has been lost	A message was displayed on the server terminal saying connection to a client had been lost	Works as expected
Manually terminating the server	N/A	A message should be displayed on the client's side saying connection to the server has been lost	A message was displayed on the client's side saying connection to the server had been lost	Works as expected
Attempt to connect to server with wrong server name	test	A message should be displayed to client saying they have	A message was displayed to client saying they had	Works as expected

		entered an invalid server name/ port number	entered an invalid server name/ port number	
Attempt to connect to server with wrong port number	-1	A message should be displayed to client saying they have entered an invalid server name/ port number	A message was displayed to client saying they had entered an invalid server name/ port number	Works as expected
Start the client then the server	N/A	The server should start normally and a message should be displayed to the client notifying them the server cannot be reached	The server did start normally and a message was displayed to the client notifying them the server couldn't be reached	Works as expected

<u>Testing – Extension</u>

Test Description	Test Data	Expected Output	Actual Output	Comment
Try to transfer a file	test.mp4	A message should first be printed notifying the client the file has been found. Next the transfer should begin and the progress bar should fill up appropriately. Finally, the file should have been copied and the open button enabled. A message should then be displayed notifying the user the file transfer is complete	A message was first printed notifying the client the file had been found. Next the transfer begun and the progress bar did fill up appropriately. Finally, the file had been copied and the open button enabled. A message was then displayed notifying the user the file transfer was completed	Works as expected
Try to open a file	Test.mp4	The program should attempt to open the file with the computers default program	The program did open the file with the computers default program	Works as expected
Have multiple clients connect to one server	N/A	The server should be able to accept and serve multiple clients simultaneously	The server was able to accept and serve multiple clients simultaneously	Works as expected
Use the disconnect button	N/A	The client should disconnect from the server; a message should be printed on the server's terminal	The client was disconnected from the server; a message was printed on the server's terminal	Works as expected

Open an invalid file	Invalid	An error message should be displayed on client's side and the program should continue	An error message was displayed on client's side and the program did continue	Works as expected

Evaluation
After testing my program, I pleased to say it works as expected and is able to gracefully transfer files data through java sockets. Moreover, I am also pleased the GUI I created is clean and user friendly. It also enforces a lot of validation and verification by enabling and disabling components. The ability to transfer files and open them provides a lot of helpful functionality to the program too.

Conclusion
In conclusion I'm satisfied with end program, it is able to carry out the required tasks with ease and is able to handle erroneous data without crashing. The added extensions massively increase the overall functionality.