

Networks Report

Introduction

The purpose of this project is to create a network application using protocol design and TCP(Transmission Control Protocol) connections to transfer files between the client and a shared server similar to Google Drive. An application useful for storing and sharing files on a network. This is accomplished by using a socket. A socket is the one end-point of a two-way communication link between two programs(client and server) running over the network. The server executes a service(e.g. download) and the client connects to the server to get served.

OO Design

The application consists of 4 classes, a server and clients class with their respective main classes. We run the main class e.g. ServerMain, first which calls Server.java. We do the same for the server side. To establish the connection we run the Servermain class and create the server object with its port number. The Server class or specifically the Listen() method manages the parsing of messages between itself and the client with the help of DataInputStream and calls the necessary method e.g. download, view etc. The ClientMain class contains all the prompts for the user, these messages are sent to the server via the DataOutPutStreams.

Server data is stored on a Server folder, each client has its own folder and a private folder. When permissions are changed(by calling permissions method) for a file it is moved to the private folder. After a download, the client data is stored in the client folder. Login information is stored in userInfo.txt. This document has the username and password of the clients.

Functionality

Once the server and client classes have been run, the client is prompted to supply a username and password. If the username is unknown, they are prompted to try again. If the password does not correspond to the server records they are prompted to try again. Once login is complete, the client is prompted to pick between the below options. Once picked the prompts loop to see what else the client wishes to do, until they press "Q" to quit.

Upload and Download:

The application allows the user to download files to the server and upload them to the server. The client supplies their username to the server which is also the folder name of where all their data is stored on the server. The client can either choose to download or upload a file. Once they have specified which file they want to download from the server, the application searches for the file in the folder and downloads it to the client folder. If they wish to download a private file, they have to supply a password so they can have admin access.

Permissions:

The application allows users with admin access/owner to make their files private to others. Allowing them to be able to share files in the public 'space' but have certain files in a private folder, that only they can view or download. A file selected to be private is moved to this private folder. Owner can also move a file from the private folder to the public folder.

View:

This allows the client to view their uploaded files on the server, if they are the owner they can view private files. This was added so clients can know what files are available on the server.

If a private or public folder is empty, the program returns empty square brackets "[]".

Protocols

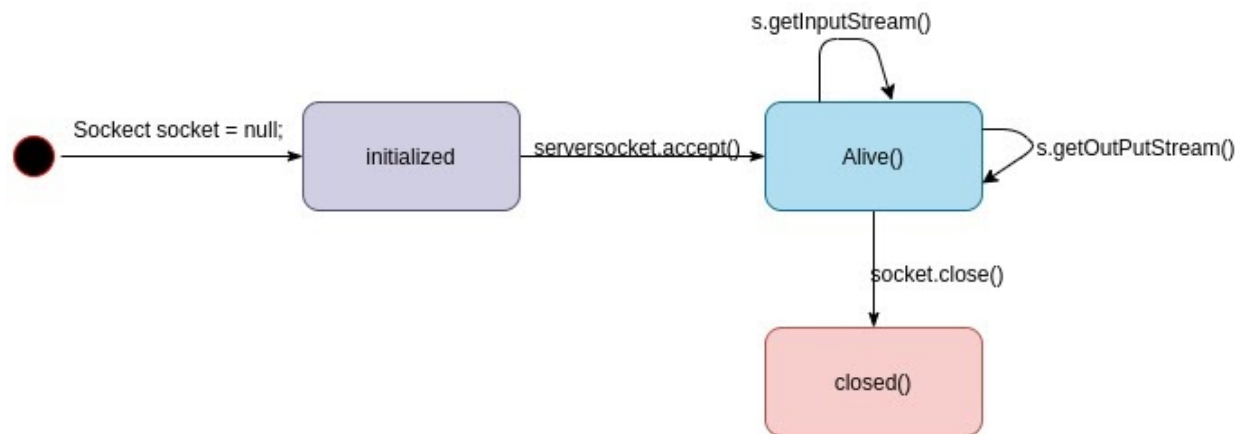
Login protocol:

The user is prompted by the client to supply a username, if its incorrect, which we know by parsing it to the user() method which searches for the username in the server records. They are prompted to try again until they input a known username. After that they are prompted to specify an access level, if public they are finished with the login process and can proceed to the

application. If they pick admin access they must supply a password, which is parsed to the password() method that checks if username and password exist in the records. If the password is incorrect, the user is prompted to try again until they get it right.

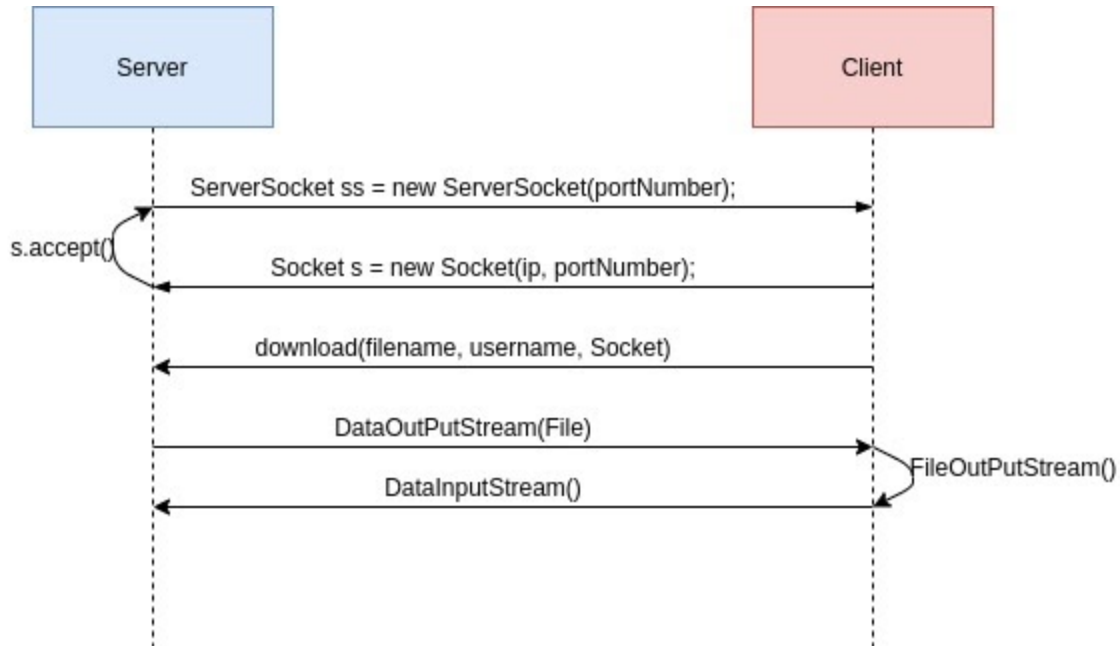
State Machine Diagram for the Socket:

We first initialize the socket to null on the server side, then use the ServerSocket object to listen to and accept connection from client. After accepting a request from a client the server creates a client socket to communicate (to send/receive data) with the client. The last stage of its life cycle is when we close it.



Sequence diagram:



When the socket has been accepted by the server, the download command parses the filename, username(which is also the folder name where the file is located on the server) and the socket to the server. During the download process the client and server communicate with the output and input streams.







Screenshots



```
Run: ServerMain x ClientMain x
/usr/lib/jvm/java-13-openjdk-amd64/bin/java -Didea.launcher.port=38531 -Didea.launcher.bin.path=/usr/local/intellij/
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m
Enter your username
user1
Please enter your access level<Public/Admin>
Admin
Please enter in your password
123
Select an action:
<View> - To view files on server
<Download> - To download a file
<Upload> - To upload a file
<Permission> Set Permission
<Stop> to end the connection between server and client
View
[./Server/user1/Private/vid.mp4, ./Server/user1/Private/notforyoureyes]
[./Server/user1/vid.mp4, ./Server/user1/plzwork]
Select an action:
<View> - To view files on server
<Download> - To download a file
<Upload> - To upload a file
<Permission> Set Permission
<Stop> to end the connection between server and client
```



Run: ServerMain x ClientMain x














```
/usr/lib/jvm/java-13-openjdk-amd64/bin/java -Didea.launcher.port=38735 -Didea.  
Picked up _JAVA_OPTIONS: -Xms512m -Xmx4096m  
Socket initialized  
Listening  
false  
Connection from 127.0.0.1 detected  
Sending prompts...  
upload to Server started
```

4: Run

6: TODO

Terminal

9: Version Control

