Client-Server Multithreaded File Transfer Application

CSC3002F - Assignment 1

By Sabir Buxsoo

1. Introduction

In this assignment, we look at the basics of protocol design and socket programming for TCP Connections in Java. This report describes the design and functionality of the client-server application and we also take a look at privacy enforcement used in the application.

2. Design and Functionality

The application has been designed to work in the terminal with the file upload feature making use of the in-built Java Swing and AWT Libraries which provides users with the option to select file and directories from a graphical user interface. The complexity was kept to a minimum due to the nature of the assignment.

The application makes use of TCP sockets, in this case, Java Sockets, provided as part of the built-in library. Multithreading is also used to allow multiple clients to connect to the server simultaneously and interact with the server.

2.1 The project has the following structure:

2.1.1. code/Server.java - This is the Server class which handles client connections to the server. It sends and receives data to and from the client. The Server class accepts requests from the Client and executes those requests. For example, if a Client sends the "Files" request to the Server, the latter will process the `Server/Files/` directory and send the list of files on the server to the client.

2.1.2. code/User.java - This is the User class which handles user authentication. It has methods which loads a list of users from the mock database, checks if a user exists and authenticate users by doing password checks.

2.1.3. code/Client.java - This is the Client class which handles all requests from Client. It includes the following commands which the Client can input:

a. Files - Gets list of files from `Server/Files/` and returns to client

b. Upload - Opens Swing GUI to allow clients to upload files to `Server/Files`. File Size limit is set as 100mb for normal file upload.

c. Download <filename.filetype> - Allows Client to download a file from Server which moves file to `Client` folder. Opens GUI dialog to allow Client to select Destination directory.
d. SecureUpload - Allows Client to upload file to the password-protected `Server/Secured` folder. This requires Client to input a password before uploading the file. File size limit is not restricted for Secured Upload.
e. SecureDownload <filename.filetype> - Allows Client to download files from the `Server/Secured` folder. Client needs to input password to be able to download files from this folder. Opens GUI dialog to allow Client to select Destination directory.
f. SecuredFiles - Shows list of files from the `Server/Secured` folder to client. Client needs to input password for this request to be allowed.
g. Exit - This commands closes the connection between the Client and the Server.

2.1.4. Server/Files - This is the folder which contains all files available on the server which can be viewed, downloaded and also allows clients to upload (limit 100mb) without password.

2.1.5. Server/Secured - This is the folder which contains all the files available on the server which require a password from the Client in order to view the files, upload files (with no limitation on file size) and download files.

2.1.6. Client - This is the simulated Client folder with some files that users can use to test the application. Similar to dummyFiles.

2.1.7. dummyFiles - This folder contains some dummy files which can be used to perform tests on the application. Included in a 5mb file, a 100mb file and a 101mb file for boundary checks. It should be noted that the 100mb is exactly 1048576 bytes and for this application, we count each mb as a byte for exact checks.

2.2 How the application works:

For the purpose of this application, the Port used is 1999. Below are the steps on how the program works:

2.2.1. The Server is run on port 1999 by typing the command `java Server` in the terminal after navigating to the `code` folder. The Server starts running and waits for requests from the Client.

2.2.2. The Client is run by typing the command `java Client` in another terminal window. The Client makes a connection request to the Server on port 1999 and the Server asks the Client to authenticate before they can access the Server. It requires a username and a

password from the Client. Once the Client logs into the server, they get a Welcome message and can start executing commands. See Figure 1.

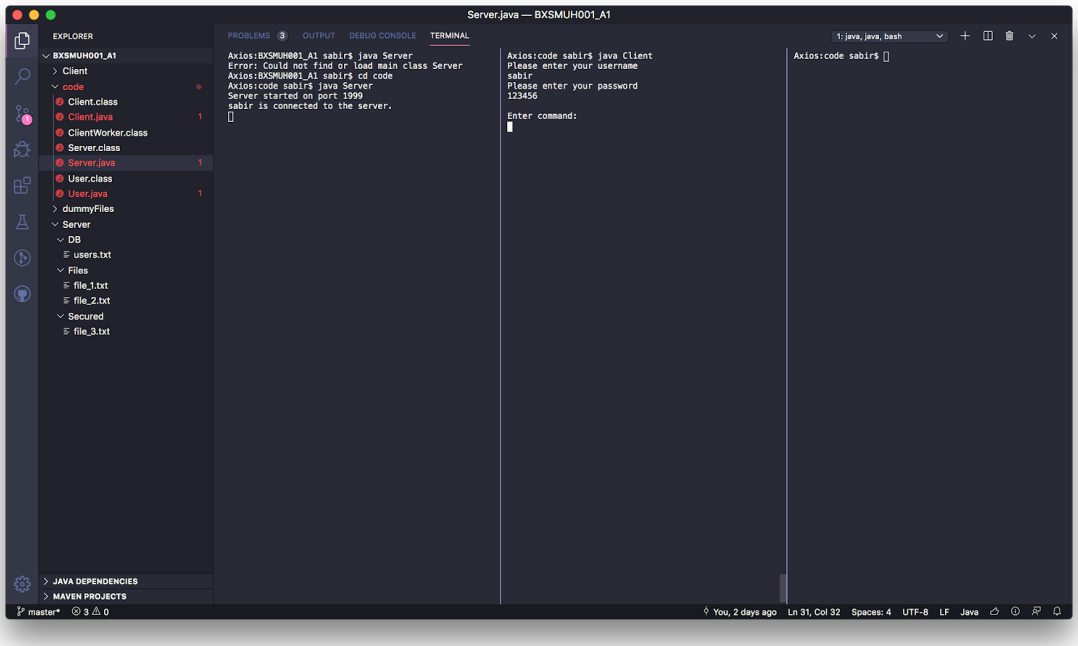

Figure 1: Client connection to Server

2.2.3. Now that a connection is opened between the Client and the Server, the Client can enter the Client can send commands to the Server. The commands are outlined in Section 2.1.3 above. For example, Figure 2 shows a `Files` request made from the Client.

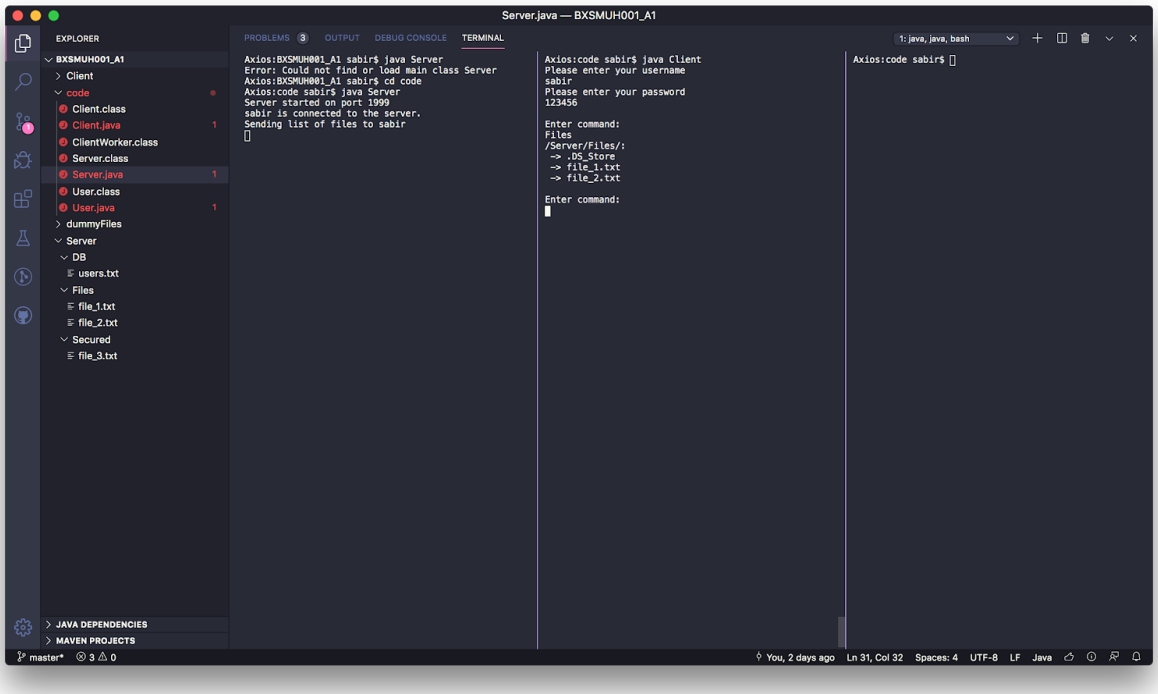

Figure 2: Client requests file list from server

2.2.4. If the Client wants to access the Secured Files, they have to input a password. The password is `**password123**`, which is stored on the Server and is checked before allowing the Client to make requests.

2.2.5. The Server keeps the connection alive between the Server and the Client until the Client decides to exit the Server, at which point the thread is released. This ensures that the application is robust. This was done on purpose to allow meaningful testing of the application. The Client can exit the Server at any time, by using the `Exit` command.

2.2.6. The application also allows multiple Clients to connect to the Server simultaneously as shown in Figure 3.

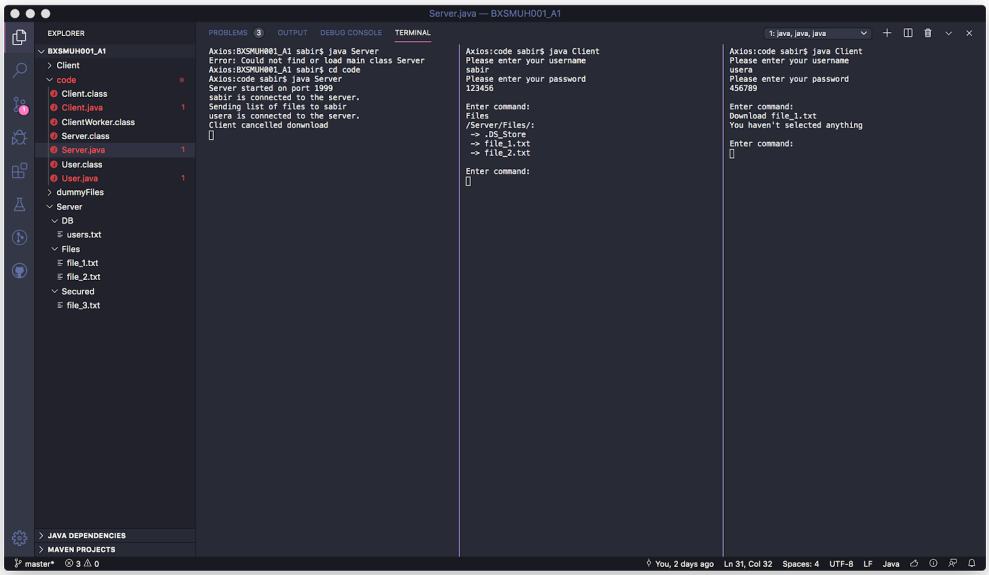

Figure 3: Multiple Client connections to Server

2.2.7. If a Client wants to upload a file to the Server, a GUI is provided as shown in Figure 4.

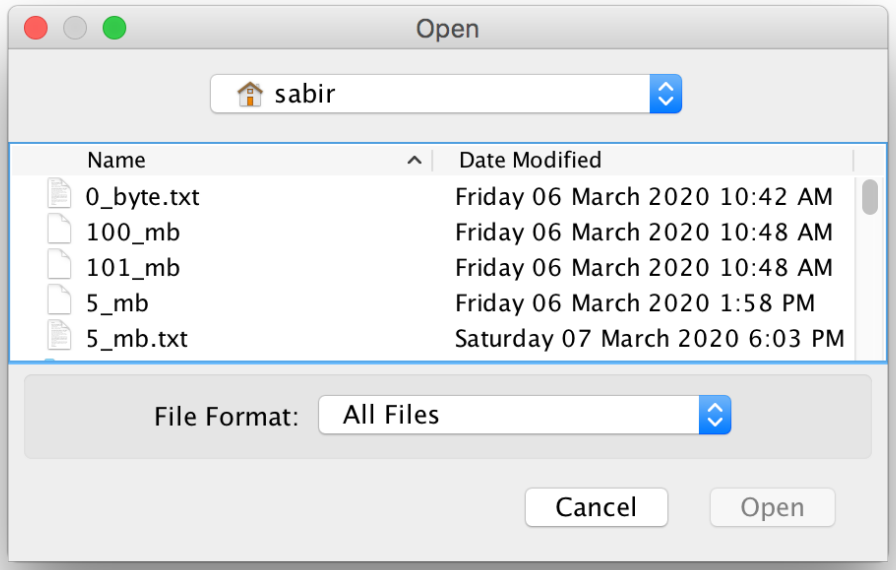

Figure 4: Upload GUI

2.2.8. Once the Client uploads a file, the file moves to the appropriate folder on the Server. For unsecured upload with file size limit of 100mb (`Upload` command), the file moves to `Server/Files` and for Secure Upload (`SecureUpload` command) with no file size limit, the file moves to the `Server/Secured/ folder. An example is shown in Figure 5, where the 100_mb file has been uploaded to `Server/Files`.

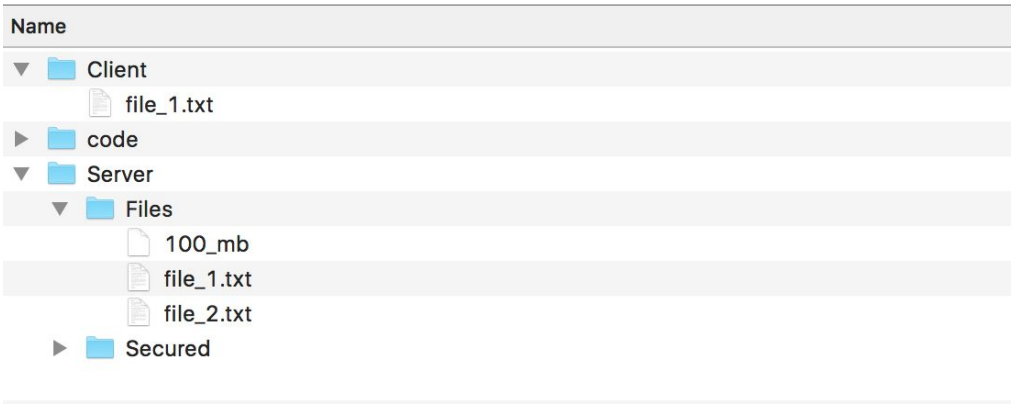

Figure 5: 100_mb File Uploaded

2.2.9. If a Client wants to download a file, a GUI is provided for them to select the Destination directory to save their Download as shown in Figure 6 below.

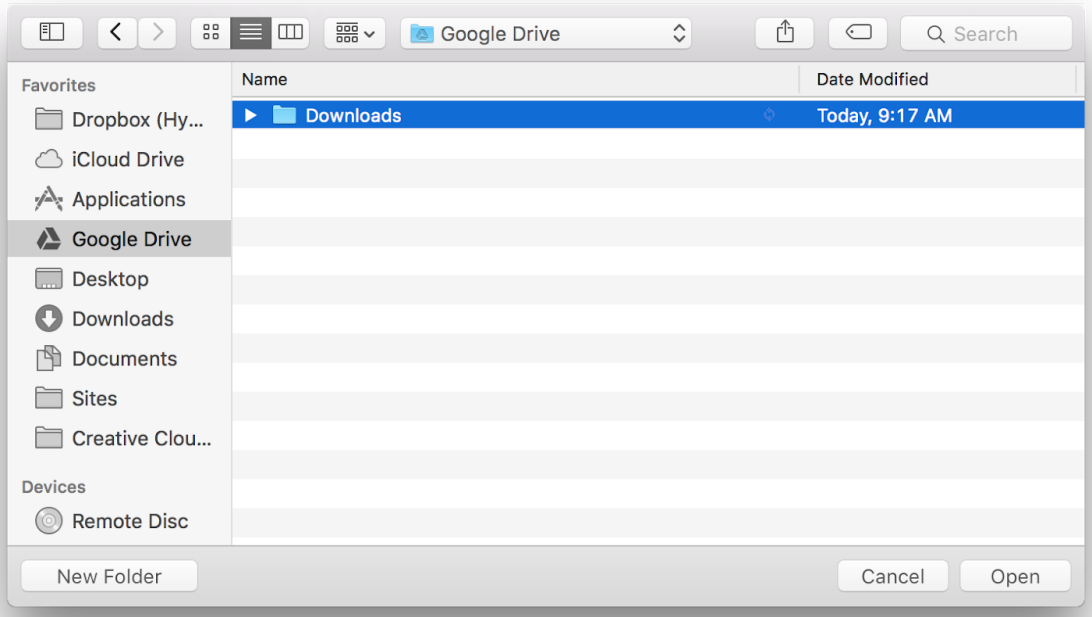

Figure 6: GUI to select Download destination directory

3. Robustness of the Application

Due to the nature of this assignment, the robustness of the application was tested with multiple file sizes. We protected a folder to allow Clients with password-protected access to upload files without size limit as a way to demonstrate a different user access level with special permissions. This models a real-life situation where, for example, a premium user can upload large files and a free user can upload files with a maximum size of 100 mb.

Several checks were also added to make the application more robust. In this particular application, the 404 (not found) HTTP status , the 403 (access forbidden) HTTP status and the 499 Client Closed Request HTTP status. This ensured that the application can simulate an actual request. For example, requesting a file without authentication(authorization) will not allow a user to download a file in a real-life application.

The 404 status occurs when a Client tries to download a file which does not exist on the Server. This sends back an error to the Client. The 403 status occurs when the Client enters the Wrong password and thus denied permission to access the files from the Secured folder. This also sends back a message to the Client.

The 499 status code is also used for when a Client hits the Cancel button on the Download GUI Dialog. This sends a status code to the Server telling it the Client cancelled the Download request.

4. Privacy Enforcement

As a means of privacy enforcement, an authentication method was implemented, which requires the Client to login to the Server before the connection between the Client and the Server is opened. This comes in the form of a login and password which the Client has to input to authenticate to the Server.

A simple mock database is used in the form of a text-file which contains the details of the users. Once the Client tries to connect to the Server, their login details are verified and if they are found in the mock

database and they have the correct password, they are logged into the Server and can start executing their commands.

A second part of the requirements, privacy enforcement was made by password protecting access to the SecuredFolder. This ensures that only people with the password can view, upload and download files from the `Server/Secured` folder. However, all normal users who do not have a password (or have a password) can access the public folder `Server/Files`. This demonstrates how privacy can be taken into consideration when building such protocol. Due to the nature of this assignment, the complexity was kept to a minimum. There are many ways of authenticating users where a login and password could be used to ensure several layers of access. This however, requires extensive back-end programming and a database to store necessary user authentication information, file locations, among others. In this dummy application, we make use of a single private variable on the server to store the password which is verified before the Client is allowed to access the `Server/Secured` folder contents for their requests.

5. Limitations

For the purpose of this assignment, the goal was to demonstrate a simple client-server application using multithreading and demonstrate the basic concepts of sockets and TCP. Some of the limitations are:

5.1 The mock database is being used to simulate a real-database. This means that information is stored on a simple text file which in real-life would be replaced with an actual database (SQL, for example). Due to the nature of this project, this was kept very simple to stay in scope.

5.2 The user passwords are not encrypted. In a real-life situation, passwords would be encrypted using a hashing algorithm (Example, PBKDF2) and stored on the server. This is extremely important to prevent accounts from being compromised in case of attacks. In this project, passwords were not encrypted due to the complexity of the encryption algorithm and difficulty to implement. The goal here was to demonstrate a simple authentication system and how it works.

5.3 Java GUI has some limitations on OS X (El Capitan) which was noted during testing. Sometimes the dialog hangs on OS X (El Capitan).

6. Sequence Diagrams

The general sequence diagram is shown in Figure 7 for unsecured commands from clients with access to the public folder `Server/Files`. It works for the Files, Download and Upload and Exit commands.

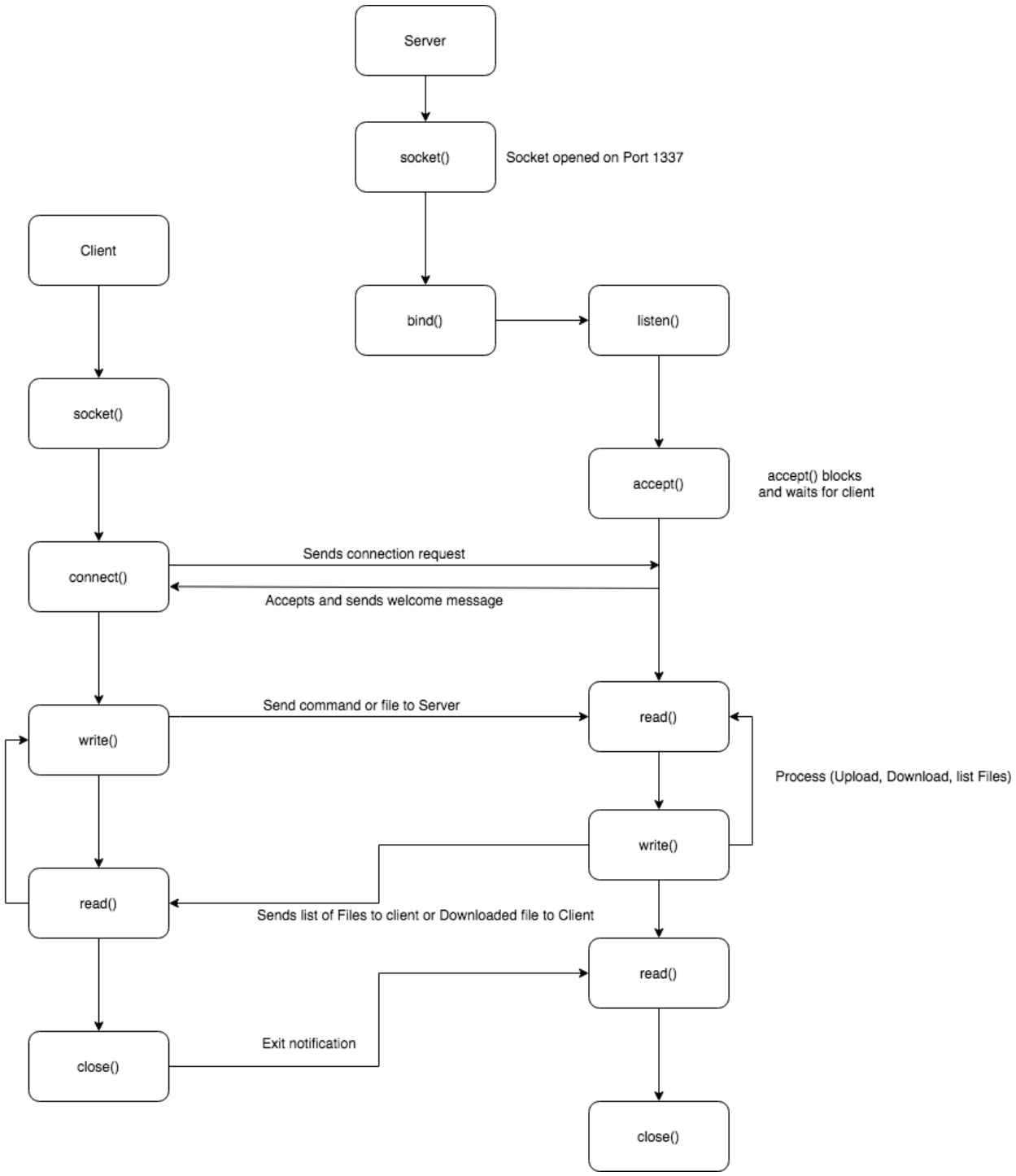

Figure 7: Client connection to Server for Files, Download, Upload, Exit commands

The sequence diagram for a password access to folder `Server/Secured` is shown in Figure 8. It works for SecuredFiles, SecureUpload, SecureDownload commands.

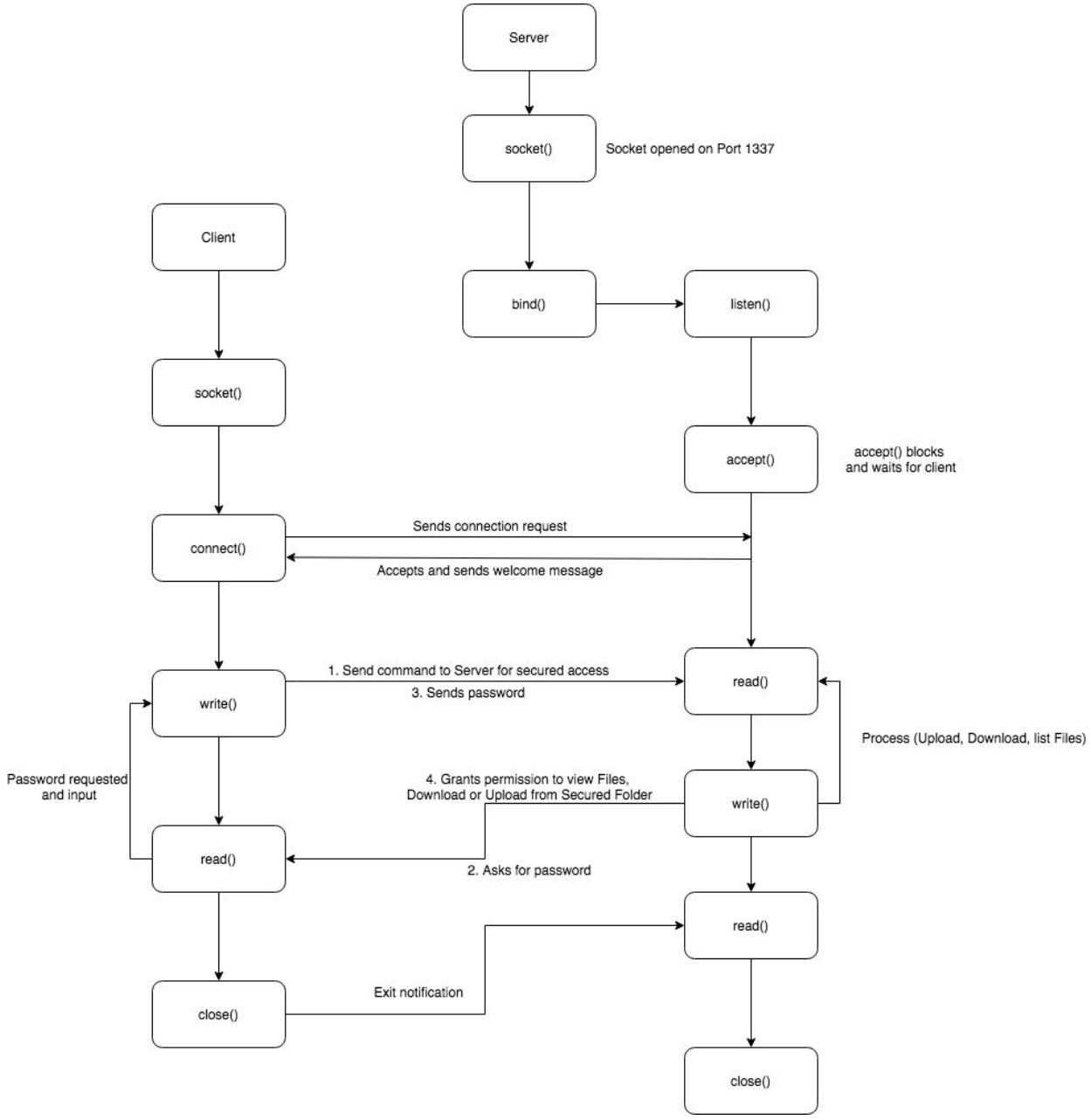

Figure 8: Client connection to Server for SecuredFiles, SecureDownload,Secure Upload commands