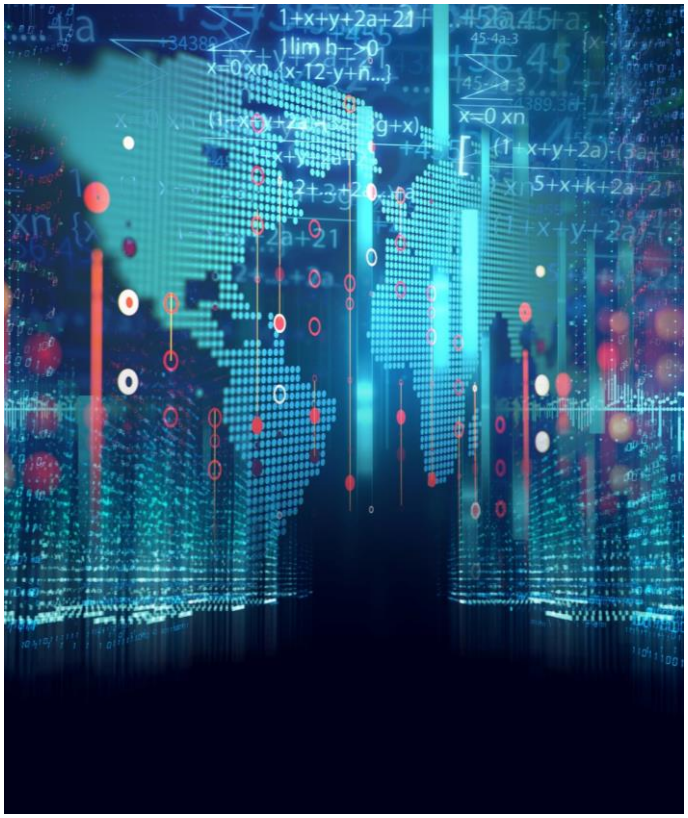


PROTOCOL DESIGN & SOCKET PROGRAMMING

CSC3002F |Networks Assignment 1

Authors

MNSTSA001	Tsatsawani Mnisi
MMKMOK001	Mokgaetsi Mmakola
KNNBON009	Bonile Kunana

INTRODUCTION

We are tasked with implementing a client-server file sharing application that will allow clients to send and receive files. The application should consist of two sockets, the client-side socket, and the server-side socket. The two sockets should be able to communicate by establishing a connection. This is accomplished when the server receives a connection request on its specific server port from the client. If any more connection requests arrive, the server accepts them in the similar way creating a new port for each new connection. Thus, at any instant, the server must be able to communicate simultaneously with many clients and to wait on the same time for incoming requests on its specific server port.

AIM

The aim of this assignment is to implement a client-server file sharing application that makes use of TCP sockets. The application must govern privacy/confidentiality.

DELIVERABLES

The application is supposed to perform the following functions:

- Server should be able to receive and send files to clients.
- Clients should be able to upload files to the server.
- Clients should be able to indicate if the file being uploaded is open or protected.
- Clients should be able to query the server of listing of available files and should be able to request files from the server.

THEORETICAL CONTRIBUTION

A connection between the client and the server is required since the application is governed by TCP; and it has error check mechanisms where the client can check if the requested file as it was sent by the source. Additionally, it can deny unauthorized personnels access to protected files. This speaks to the reliability of the application.

The server must be able to receive a connection from a client anytime and the listen () function was used to check for any requests for a connection from the client.

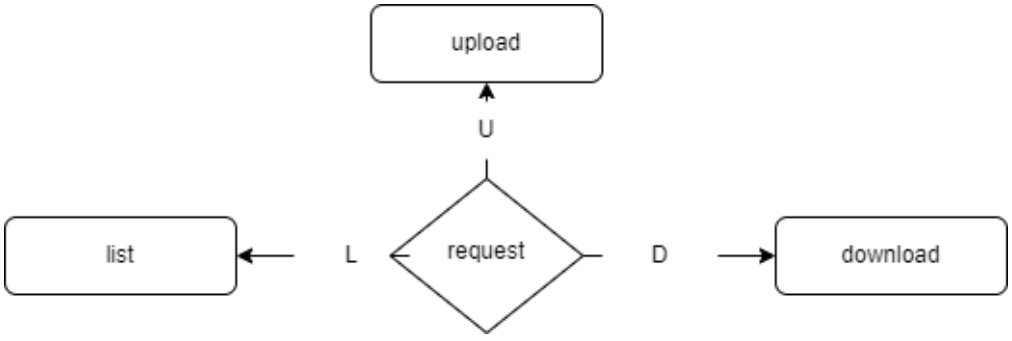
EXPERIMENTAL CONTRIBUTION

Server

A server is responsible for listening for incoming connections from clients and responding to their request. When the socket receives a connection request from the client, a connection is established. After performing the requests from user, the program prints out messages to communicate to users that a specific request has been done.

Every feature of the server program was implemented under the main function. The socket function creates a new socket, and the bind () function binds the socket to an IP address and a port number. On the while loop, the accept () method is to allow the server-side socket to accept a connection request from user. The recv () method takes in a specific number of bytes that can be accepted by the server at a time, and it decodes the information received from the client to the specified format.

Our server consists of if-else statements that control the execution of statements based on the type of requests that the user has. We have created the ***request*** variable to inform the server what operation to perform.



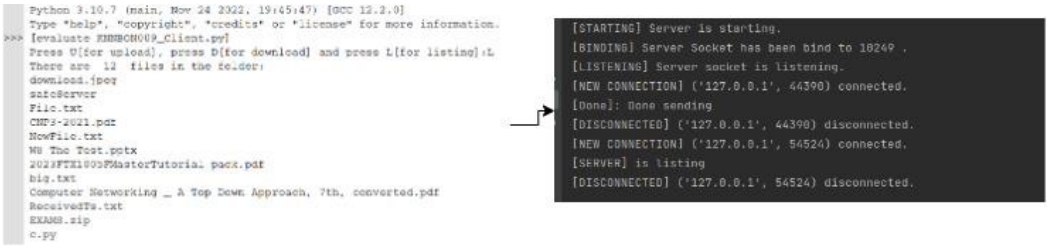
The TCP protocols communicate by sending messages. In our server class we included messages as a form of communication between the client and the server program. The messages are encoded on one end of the system and decoded in the other end. The functions

Client

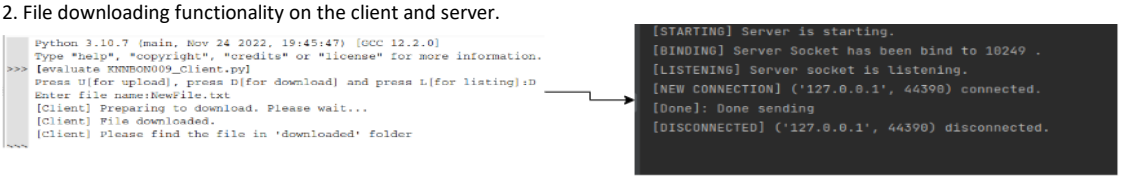
Bonile Kunana [KNNBON009]

The screenshots below illustrate the TCP connection between two programs running in different end systems. The client can send files to the server and request files from the server. The client is also capable of listing available files in the directory. When you choose U(upload), the GUI pops up, shows the file you can upload to the server, the user selects the file and click ok. For downloading the user enter D(download) and the file will be downloaded. When the user wants to list, they simply press L(listing) and the file names in the directory will be listed in the terminal. If the user enters invalid input, appropriate messages are printed.

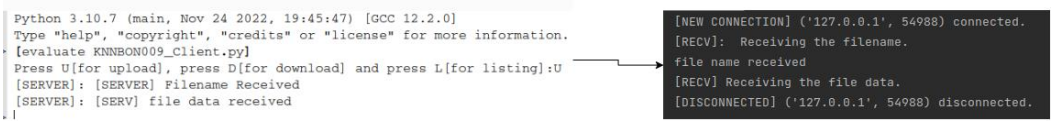
1.The server side messages confirmation of the list of files available in the directory feature/functionality.



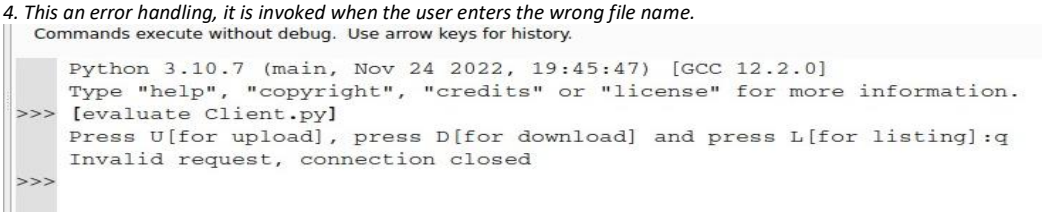
2. File downloading functionality on the client and server.



3.File uploading from the client to the server functionality



4. This an error handling, it is invoked when the user enters the wrong file name.

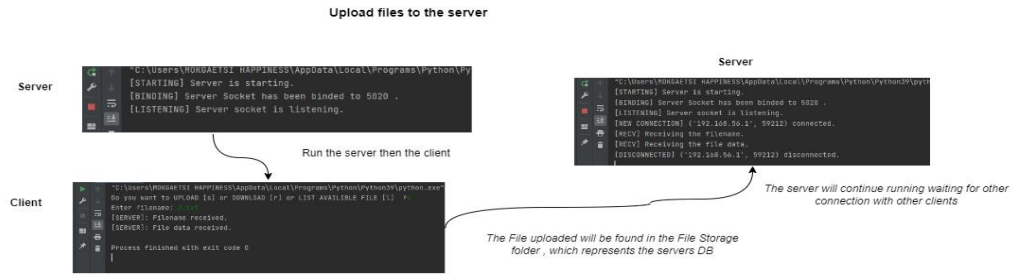


Mokgaetsi Mmakola [MMKMOK001]

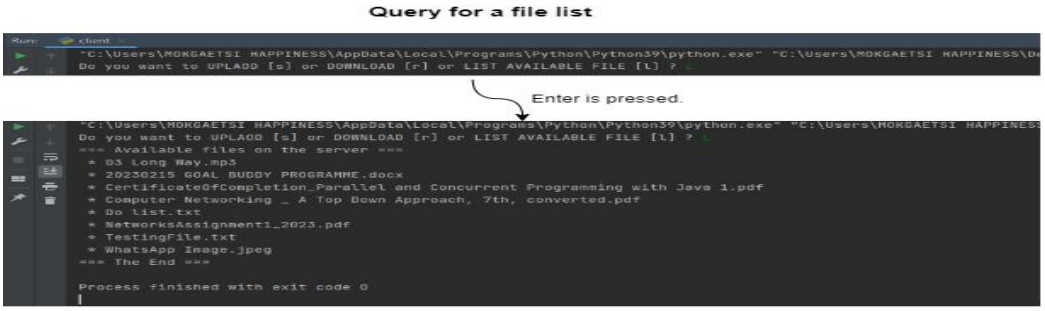
The client is responsible for connecting to the server to send requests.

Functionalities & screenshots of chat application and features

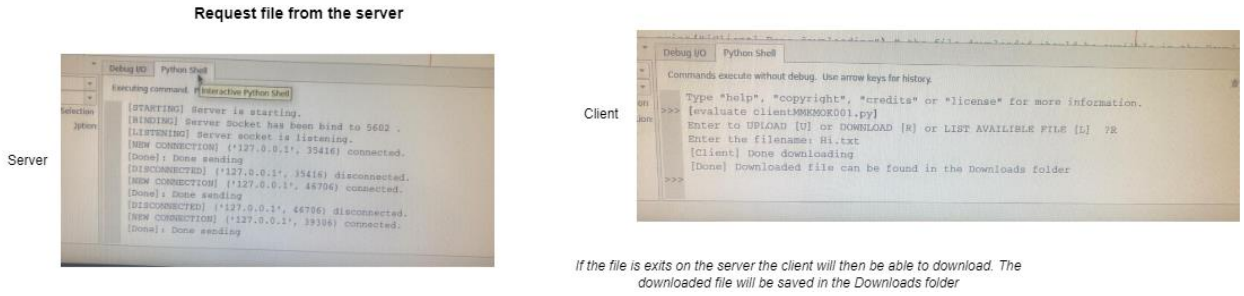
1. *Enabling the client to upload files on the server.*



2. Enabling the client to query the server for a list of available files.



3.File downloading from the server



4. Error handling when an unknown request is made.

Approached technique used behind the above-mentioned client functionalities and features is the use of TCP sockets.

1. Get the host information.
2. Create a socket.
3. Connect the socket within the host (establish connection with the server group)
4. Use functions such as *send ()* and *recv*() functions for communicate between the client and server.
   - *For the downloading functionality*, the user sends a filename that they would like to download & through implementation I used *send ()* function to send encoded filename requested by the user to the server. Then the *recv()* function is used by the server to receiver the filename then decoded so that it can be able to retrieve the file from its storage .
   - *For the sending functionality*, the server uses *recv()* to receive a filename and file data from the client. Then the *send ()* function is used by the client to receive confirmation messages that the file is uploaded on the server file storage.
   - *For the listing of files available on the server functionality*, the server uses the *send ()* function to transfer a list of files available on the server file storage. Then the client uses the *recv ()* function to receive that list then loop through the list to display them to the user.

5. Close the socket file descriptor.

Tsatsawani Mnisi [MNSTSA001]

The client code connects, communicates, and works with the server program to allows the sharing of files. To communicate, the server and the client interchange response messages as well as error messages. I have used if statements to be able to determine the sequence of statements to be executed for each unique request; in a case where the request is invalid, there is an error message to be printed out to inform user of the error. The below images show how the application responds to certain requests.

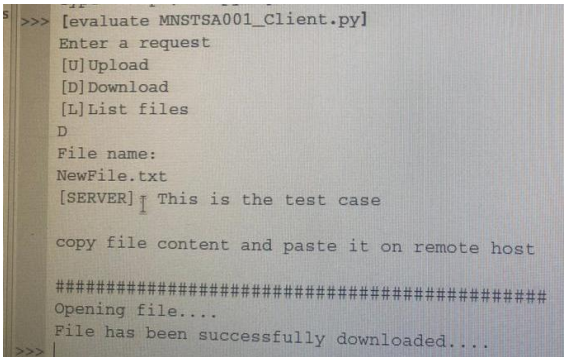
a) Error handling for invalid request

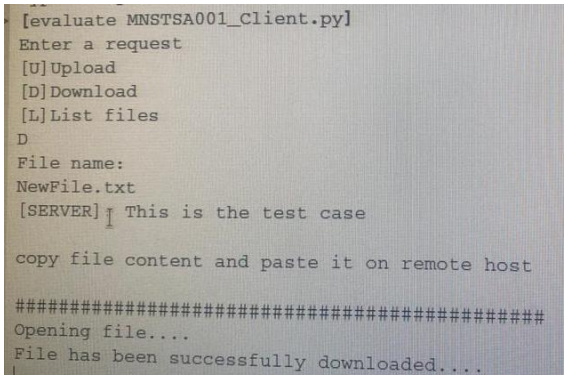
```
Enter a request
[U]Upload
[D]Download
[L]List files
R
[ERROR] Request not identified.........
Exiting gracefully...........
```

```
Enter a request
[U]Upload
[D]Download
[L]List files
U
Enter filename: File.txt
[SERVER] : [SERVER] Filename Received.
[SERVER] File data received
```

b) Response to client request to send file to server.

c) Response to client request to list files in a directory.





d) Response to client request to receive file from server

Sequence diagram

The sequence diagrams below detail the message formats and structure of how data is transferred between the client and the server using TCP sockets.

Note: the folder in context is the server's database (file storage location on the server).

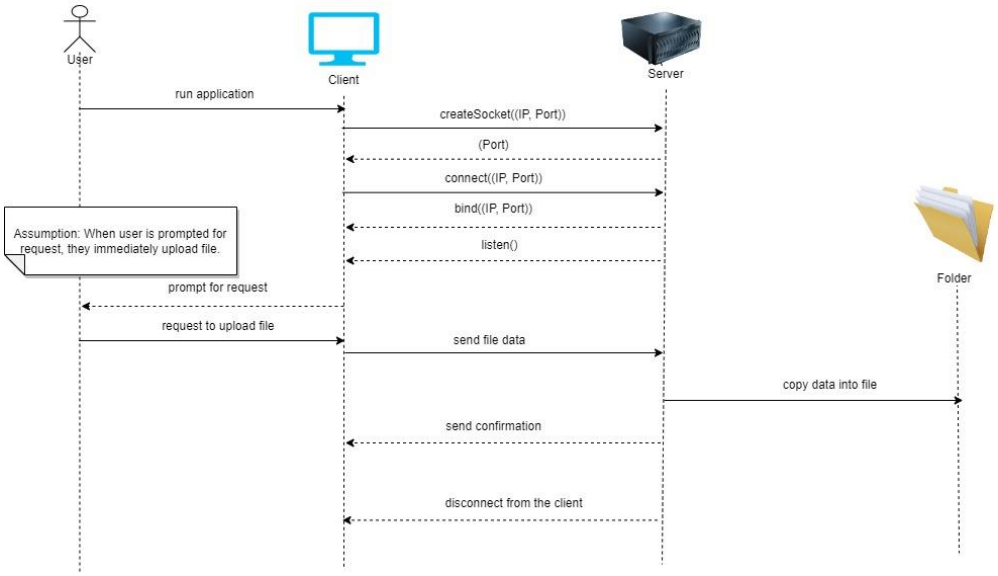


Diagram 1: Sequence diagram showing the process of uploading any file type from the client to the server.

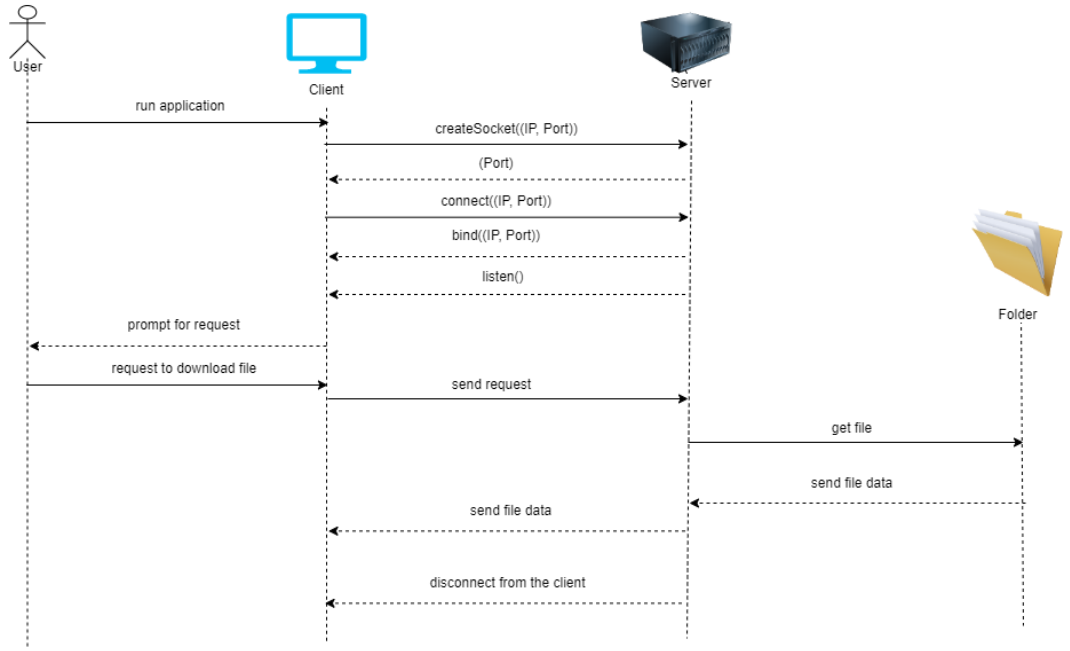


Diagram 2: Sequence diagram showing the process of downloading files of any file type on the server.

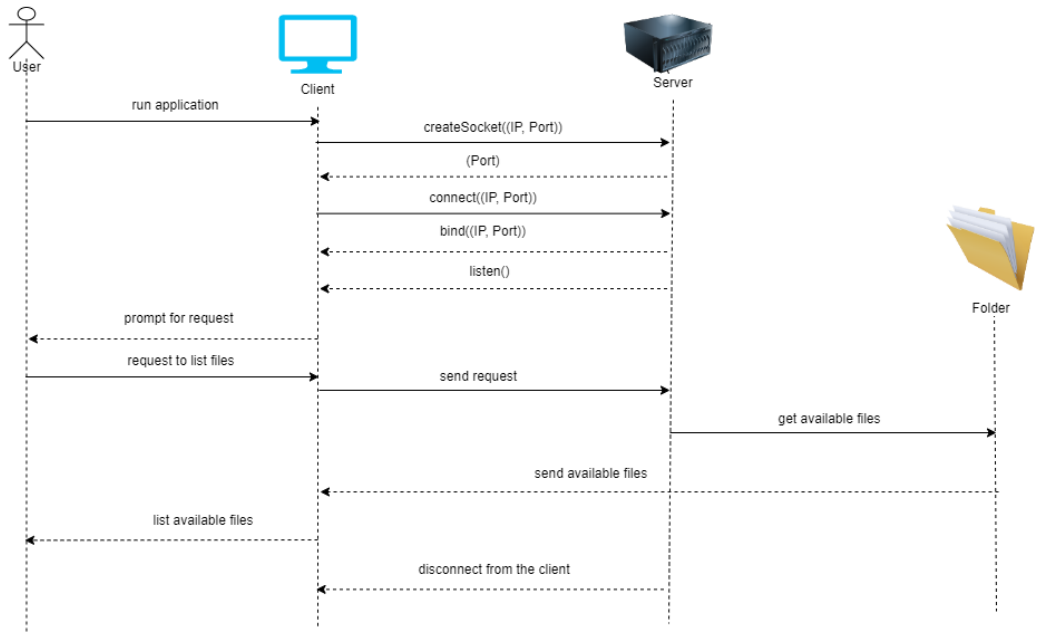


Diagram 3: Sequence diagram showing the process of listing all the files in a folder.

DISCUSSION

After having tested our application on different machines, it has been observed that Linux has better performance. Due to the different size of files being shared and more than one client able to initiate connection with the server, then a powerful machine is necessary to handle the load, hence Linux machine gave us a better performance compared to windows machine.

This made us to realize that the number of resources required to establish and maintain a TCP socket depends on the complexity of our program's applications, the number of parallel connections (number of clients connecting to the server simultaneously) and frequency of data transmissions between the server and the client.

DIFFICULTIES

We faced problems running the application on some of the windows machines, we kept on getting the <<builtins.ConnectionRefusedError: [WinError 10061] No connection could be made because the target machine actively refused it>> error for some time. We also had struggles enabling the large files to be uploaded by the user. Additionally, the validation feature is also one of the things we struggled with adding to the application. Additionally (TSATSAWANI MNISI) had problems running our clients request for listing request.

CONCLUSIONS

When designing and implementing the client-server file sharing application using TCP sockets it is important to define a protocol such as message formats and types which will be used for communication between the client and the server.

In conclusion, the TCP file sharing application is a convenient tool for file transferring/sharing over a network. Its key features include fast transferring speed/ efficient transmission of files and secure protocol.

Overall, the TCP file sharing application provides end-to-communication, flow control and congestion control which makes the transmission speed fast and most importantly the benefits it provides makes the tool valuable for file sharing, and from our group we highly recommend TCP file sharing protocol.