CPSC 471-02 Programming Assignment Report

Members - Sara Wu, Cody, Matthew Li, Michael Yu

Methodology

For our project, we decided to code it in Python—the version of Python is Python 3.11. We decided to hand code our project and did not use any external libraries to help us such as ftplib. We did, however, utilize native Python modules such as socket, os.path, and sys to help us complete our project.

Summary

We have two channels for setting up our FTP connection, a Control channel and a Data channel. The Control Channel is set up at the start of the code on both the server and client side. Each gets its port number from the user via sys.argv[1] and sys.argv[2] respectively. We establish this connection using TCP port connections so when creating our socket, we call AF_INET and SOCK_STREAM to notify that it is a TCP socket. Then on the server side, we bind the port number with the name and the server starts listening for connections. While on the client side, we connect to the server using the same port number and name. Now that our Control Channel is established, we continuously loop to ask for user inputs, specifically one of the four commands, get, put, ls, and quit. All of these commands are sent through the Control Channel. When the server receives a command from the user, it matches the user input with the corresponding function and executes that command. For every command that is executed, except for quit - which just closes the socket connection, thus closing the server - a new channel, the Data Channel, is created. This Data Channel is what is used to send data back and forth between the server and the client.

Issues

ConnectionRefusedError

No Connection could be made because the target machine actively refused it. This error occurred when we tried to connect to the server but used the wrong port or the client/server could not find each other

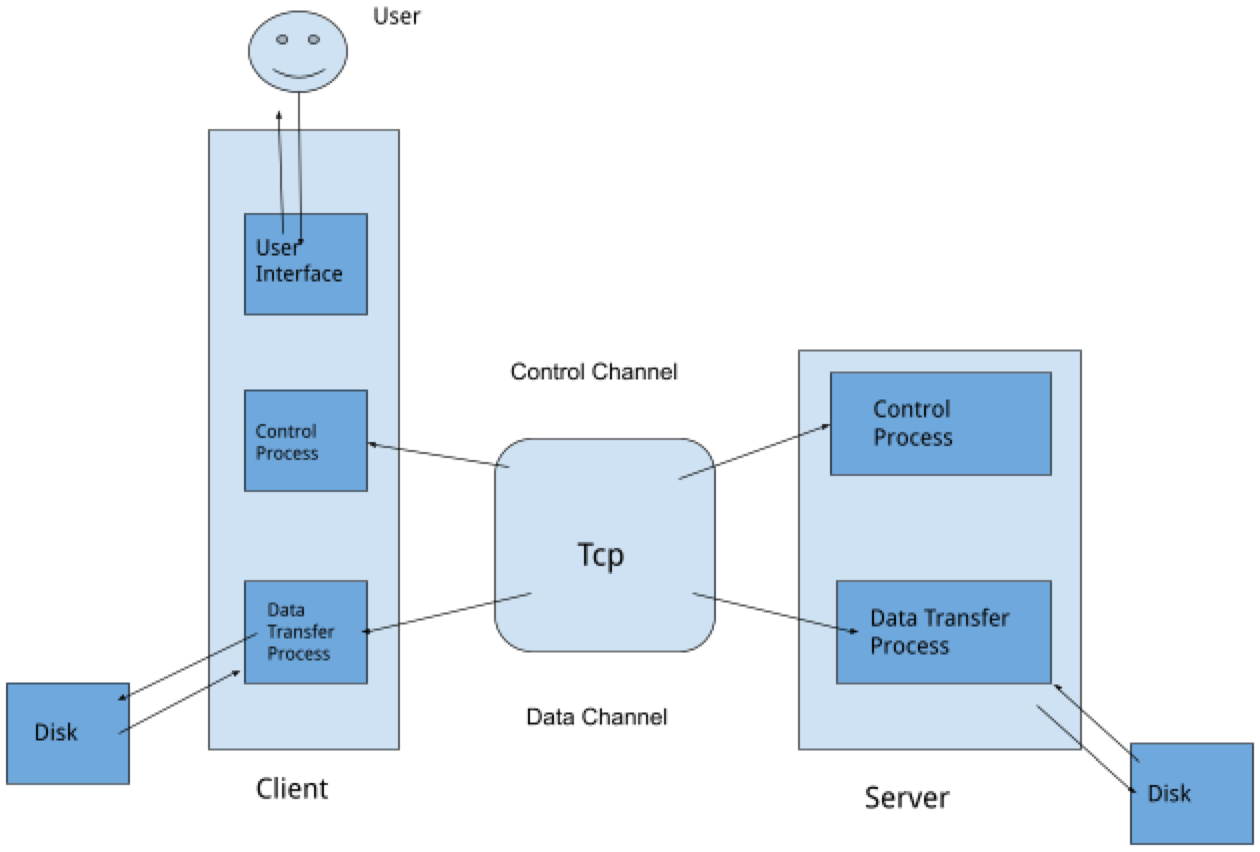
Duplicate Filenames

When uploading to the server or downloading to the client, we had to handle the edge case where the filename of the file being downloaded is already found within the directory. In this case, the server and the client will append a number to the end of the filename if it is currently found within the directory.

Bad Filenames

When attempting to download or upload a file, we had to handle the edge case where the file doesn’t exist on either the client or server. If the file doesn’t exist when attempting to upload to the server, the program will loop continuously until a valid filename is input. In the case of downloading a file not on the server, the command will be ignored and must be repeated with a valid filename.

FTP Protocol Model



Client Code

Commands:

- get <file name>: Downloads a file from the server

- put <file name>: Uploads file to the server

- ls: Lists files

- **quit**: Disconnects from the server and exits

Functions:

- **main():** Communicates with the server and handles user input for file transfer commands.

- get(filename) : downloads a specified file from the server, confirms that file name is in the directory, and utilizes the **get_unique_filename** to avoid name conflicts in the client directory.

- put(filename) : uploads a specified file to the server after confirming that file name is in the directory.

- list() : displays a list of files on the server.

Server Code

Functions:

- main() : sets the server, listens for connections, and processes client commands.

- get() : handles receiving a file from the client.

- put() : handles sending a file to the client.

- list() : sends a list of files in the servers directory to the client.