Design Doc:

Server

To get the server running, 3 arguments must be passed down in the python script:

- The IP address of the server
- The Port of the server
- The folder in which the files of the server are kept

Therefore, the script carries a quick checking for the number of parameters to keep running.

Then, the server is configured by creating its socket, binding it with the IP and Port chosen and start listening for possible connections. To be able to accept multiple connections, a forever loop is implemented to accept every new connection and manage it with a thread. This thread gets the basic information of the connection: its socket and address.

The thread runs a function with another loop which starts by a synchronous receive from the client. This means that until the client does not send a command to the server, this thread will be waiting.

If the data received is empty, the server handles the issue and prints out that the client has disconnected. In case it is not empty, it gets decoded to get a string and split by empty spaces to be able to handle all the possible input commands.

If the command is get_files_list, the server reads the list of files in the folder from the arguments and sends it back to the client.

If the command is add, the server calls the function add with the name of the file as a parameter. This function receives the file specification from the client, sends a confirmation message and the receives all the data from the file to save it in its folder.

If the command is modify, the server calls the function modify with the name of the file and the new name as parameters. This function verifies if the file that is going to be modified exists and, in case it does, the file is renamed.

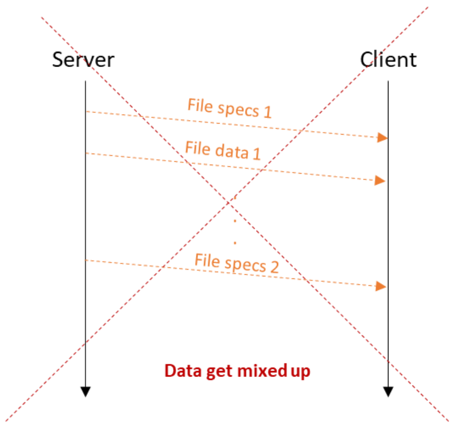
If the command is delete, the server calls the function delete with the name of the file as a parameter. As with modify, this function verifies if the file that is going to be deleted exists and, in case it does, the file is removed.

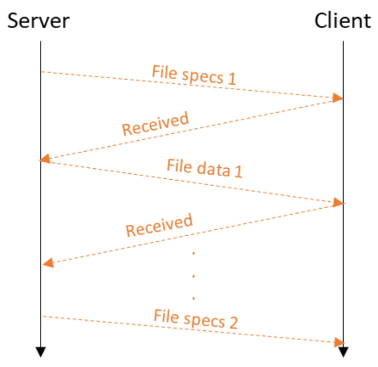
Finally, when the command is download, it can be either serial or parallel. Both call the same function download, however:

- serial downloads one file after another one. To do so, a for loop is generated with the same length as the total number of files that want to be downloaded. For each iteration, the download function is called.
- for parallel downloads, the client will create one connection with one request for every different file, so the server just calls the download function once.

The download function gets the file name as a parameter. This function verifies if the file that the client wants to be downloaded exists. In case it does, the server sends the client the file specifications (specifically, the file size and its hash) and waits for a confirmation. Then, the file data is sent to the client and, again, waits for a confirmation message.

There have been several problems implementing this function. Specifically, when it came to having multiple consecutive sends or receives. Therefore, the solution has been to implement a confirmation message after each receive so that the information does not mixed between messages.





Finally, the server apart from printing the information also records it in a file call log.log.

Client

To get every client running, the same 3 arguments than to the server must be passed down in the python script:

- The IP address of the server
- The Port of the server
- The folder in which the files of the **client** are kept

As with the server, the script carries a quick checking for the number of parameters to keep running. Then, the client is configured by creating its socket and connecting it to the server IP and Port.

A loop that runs without stop is created to be able to call one command after another until the client wants. Hence, the first thing the client does is read the input of the shell for new commands and when one is written, split it up by blank spaces.

For every command except download, add and close, the client sends the input to the server, which processes it as it has been described before, and waits for an answer.

For the command close, the client connection is closed.

For the command add, the client sends the command request to the server and call the function send with the file as a parameter. This function checks if the file exists in the client's folder and sends its specifications. Then, it waits for a confirmation to send all the data, and after sending the data from the file, it waits for a final confirmation that the file has been received.

When the command is download, it can be either serial or parallel:

- Serial download: First, the command request is sent and then, a loop is created to go over all the files that want to be downloaded (one after another), calling the function receive_files. This function is pretty much the same function than the download function of the server changing the sends for receives and vice versa. Hence, it receives the file specifications (if the file does not exist, it receives an error message and sends back a confirmation of the error received), sends a confirmation message, receives the data of the file, saves the file, checks if the file is the same than the original one (to notify the client if it is corrupted) and sends a confirmation message to the server.
- Parallel download: With a loop that goes through all the files that want to be downloaded, for every file a thread is created and the function parallel_receiving called. This function creates a new socket, connects the socket to the server, sends the command request, calls the function receive_files (the same than serial download) with just one file and closes the connection when the file has been received. Also, to be able to get the total download time, an empty list (files_received) is created and passed down as a parameter to the threads. When the threads finish downloading their respective files, they append the name of their file in the list. Meanwhile, after launching all the threads, a while loop is

running until the condition that the files_received list has the same length than the total number of files that are wanted to be downloaded is met.

Finally, it is important to note that, both for the client and server two functions have been implemented: send_all and recv_all.

Every time that something wants to be sent or received, these functions are called and ensure that all the data is correctly sent and received.

The send_all function implements a loop that goes through the data and sends it in buffer (1024 bytes) size chunks until all the data has been sent. (Note that the last chunk can be less than buffer)

The recv_all function also implements a loop to make sure that it returns the data when all of it has been received. At the same time, in case that the length of the data is unknown but less than the buffer size, the loop is terminated after the first iteration.

Some of the possible improvements and extensions could be:

- Make the command *add* work with more than one file. It is the same mechanism as the download command but the other way around (server receives instead of sending).
- Instead of modifying the names of the files, modify their content by sending another file or something written in the command line. It could be done by using delete of the file and add of the new one.
- Instead of just printing to the console everything, implement a log for each client to store the operations carried out. A way to do it would be to name the log with the port the client uses to connect to the server.