# **Assignment 3**

**Student Name:** Filipovych Marharyta

**Group:** ПКСП-1

**Github Link:** <https://github.com/VY-Assignments/client-server-assignments-3-4-6-MarharytaFilipovych>

**Task**: Transform an existing project into a structured client-server architecture. Design a simple client-server application over TCP where the server can transfer files to the client. Implement a protocol with at least five commands to facilitate communication between the client and server. Clearly separate the functionalities of the client and server components. The client should be responsible for initiating commands, and the server should handle the processing and responses. Implement at least 5 commands (GET <file\_name>, LIST <directory\_name>, DELETE <file\_name>, PUT <file\_name>, INFO <file\_name>).Develop and document a communication protocol between the client and server.

**Solution description:**

The establishment of communication itself I left as it was in both c++ program. In Java I implemented it on my own with the help of Internet resources. In C++ programs, I use a filesystem library to work with files. I use different colors for better clarity in client-server communication. Server responses are displayed in. Client requests are shown in pink or white (in Java). Error messages, like "Undefined request," in the client program are displayed in red. Files are transferred in binary mode.

**Server:**

I have defined two classes: **RequestProcessing** and **Server**.

The RequestProcessing class parses incoming requests and determines whether they are valid. It processes the user's input to extract the command and the requested file path. It has such private fields: request (stores the raw request string), command (stores the command extracted from the request), path\_to (holds the parsed file or directory path) and boolean invalid (indicates whether the parsed path is invalid due to the result of IsInvalidPath() method).

The Server class is responsible for handling client requests. It processes requests by invoking different methods for each command type, such as **GET, PUT, LIST, GetFileInfo, DeleteFileSpecified, RemoveFolder.** There are two fields: public **buffer\_** (for data transfer, when sending and receiving data) and private **client\_socket\_,** which is initialized via a constructor.

The methods **Get**(const path& file\_name) and **Put**(const path& file\_path, int size) are implemented using a chunk-based approach with a constant chunk size of 1024 bytes. The buffer\_ field also has a capacity of 1024 bytes.

For a "PUT" request, the file name is extracted from the provided path. A new path is created to store the file in the local "database" folder. The file size is passed as a parameter and used during the file transfer process.

The request is parsed in the ProcessRequest() method, and based on the extracted command, the corresponding method is invoked. RequestProcessing class is used here to parse a request and determine its validity. If the path is invalid, a response "Request denied." is sent to the client.

In the main function, I initialized a while loop that continues running as long as data is being sent from the client.

**Client c++:**

There is a class **Client** and struct **Validation**.

The Client class handles communication between the client and the server. There is a private field **buffer\_** (stores data being received or sent, capacity is 1024) and **client\_socket**\_ (initialized in the constructor).

I want to mention several methods of the Client class:

* private **Get**(path& file\_path): handles receiving a file from the server. It first checks for a "Request denied" message. If not, the data in buffer\_ is a file size converted to int and used further in a while loop logic: I receive data until the entire file is downloaded. A file is created based on the provided path (I extract the filename from a path and save it to the client's local database folder). The incoming data is processed in chunks.
* private **Put**(path& file\_name): uploads a file to the server by reading it in chunks and sending the data over the socket connection. Since the server needs to know the file size sent, I append this value to the request. Only after that is the data sent in chunks.
* public **ProcessRequest**(string& user\_input): central method of the class, responsible for processing user input. Based on the command (GET, PUT, etc.), it performs the necessary things like sending a message to the server or downloading/uploading a file. It ensures proper error handling and validation before requesting the server. However, the validity of the path is checked in the server itself (put command is an exception).

The Validation struct is used to validate commands and input from the client. There is a constant static unordered set containing all the valid possible commands. These are "GET", "LIST", "PUT", "INFO", "DELETE", and "REMOVE". The method IsIncorrectRequest(string& input) checks if a request is incorrectly formatted or if the command is not in the valid command set. It returns true if the command is not included in the set of commands or the input is incorrect (there is no info after the command itself, or it is empty).

The loop prompts the user for input until the "exit" command is entered. If the user enters an invalid command, the message "Undefined request." is displayed. If the input exceeds 1024 characters, the message "The message length exceeds 1024 bytes, which is the maximum." is shown.

**Client java:**

The overall idea of implementing the Java client is the same as that of the C++ client, just adapted to the Java style. I decided to define a **Request** enum instead of a Validation struct since Java enums offer more capabilities. To handle file operations, I used classes like **BufferedInputStream**, **BufferedOutputStream**, and **FileInputStream**. Just like in the C++ version, the files are read and written in chunks. There is also a database folder for GET requests.

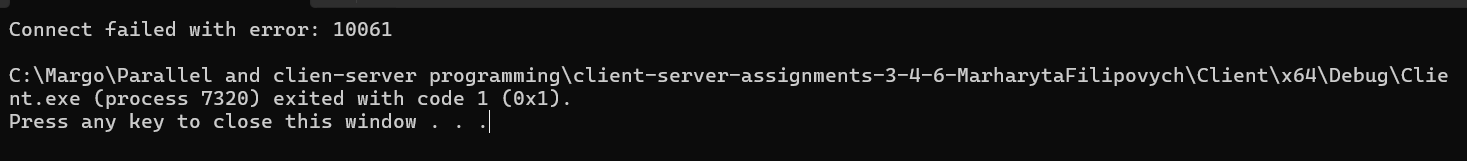
**Testing:** I have 3 separate programs: a server (in C++) and 2 clients (c++ and Java versions). To conduct the test, I launched the Server program, then the C++ client. In the client, I entered various requests and observed the results. I repeated the same steps using the Java client.

P.S. No files from the clients or server are present in the database folders, as I deleted them to ensure the repository remains clean and free of extra files.

**Screenshots** confirming the tests:

**With C++ client:**

***Server unavailable:***

****

***Everything okay:***

*A screenshot of a computer program

Description automatically generatedClient perspective:*

*A computer screen shot of a program

Description automatically generatedServer perspective:*

*A screenshot of a computer

Description automatically generatedDatabase client:*

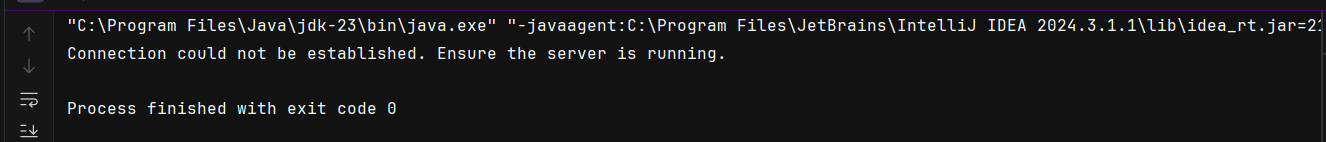
*Get command (updated):*

*A screenshot of a computer

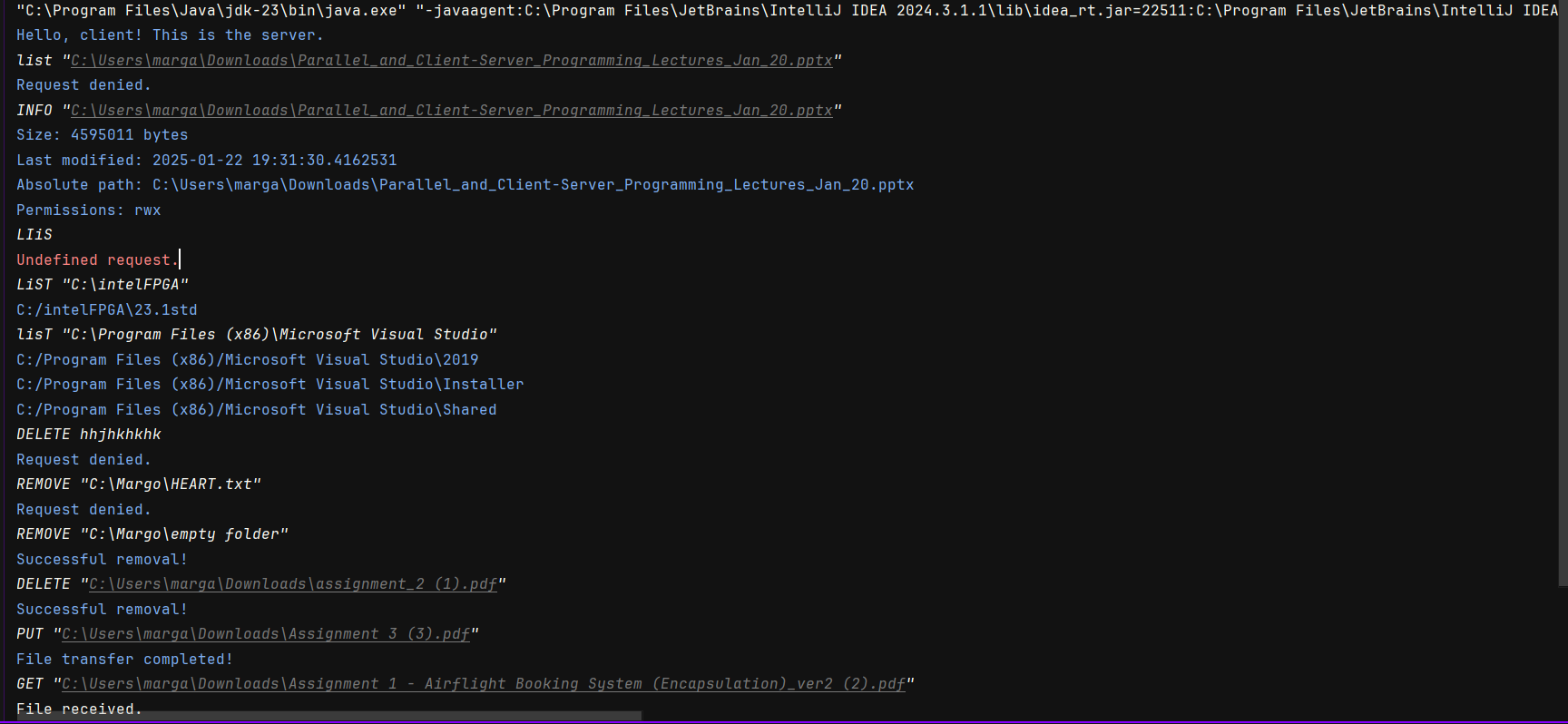
Description automatically generated*

**With Java client:**

***Server unavailable:***

****

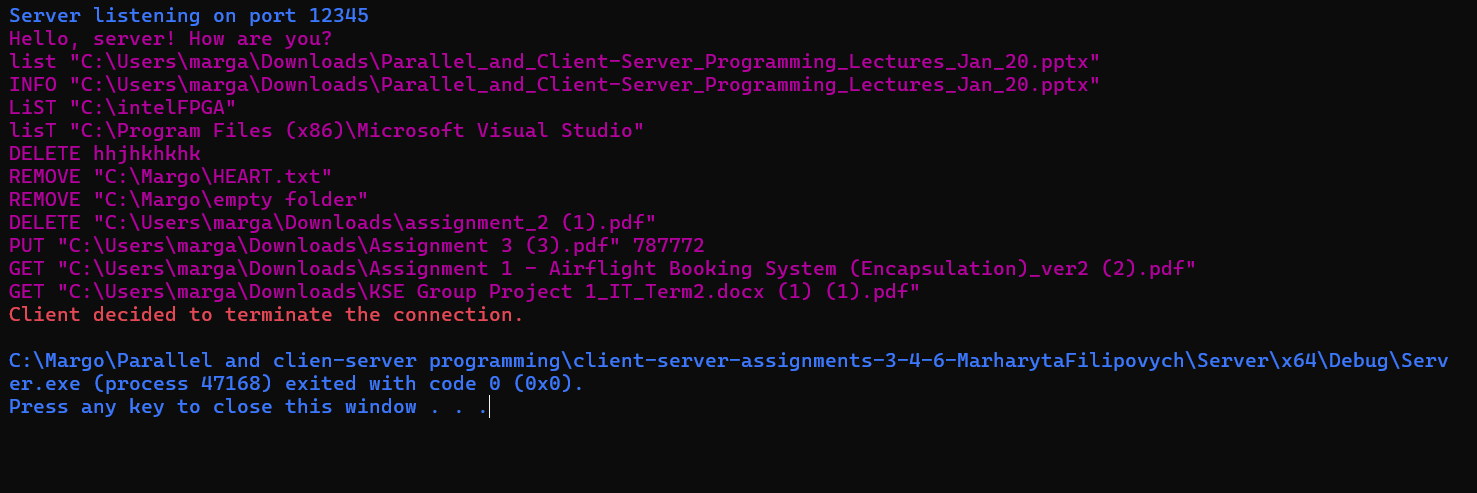
***Everything okay:***

*******Client perspective:*

**A black screen with white text

Description automatically generated**

*Sever perspective:*

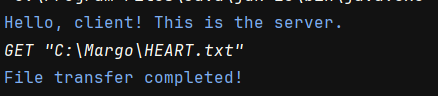
**

*Client database:*

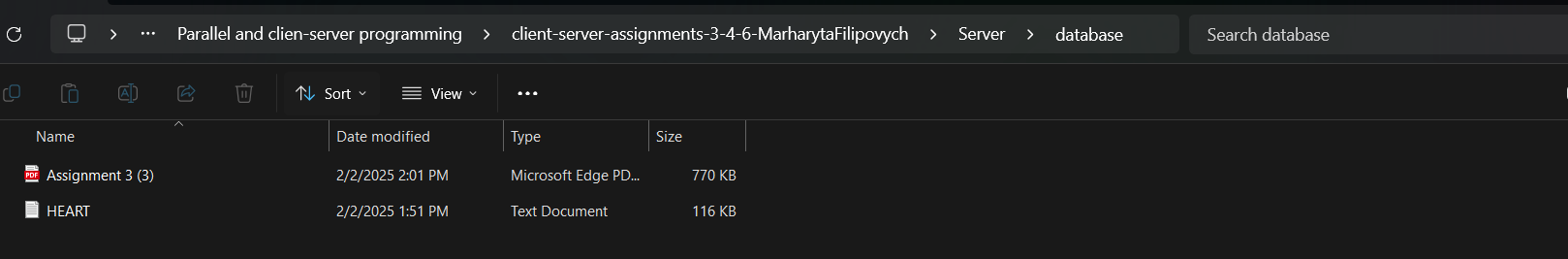
*A screenshot of a computer

Description automatically generated*

*Get command (updated):*

**

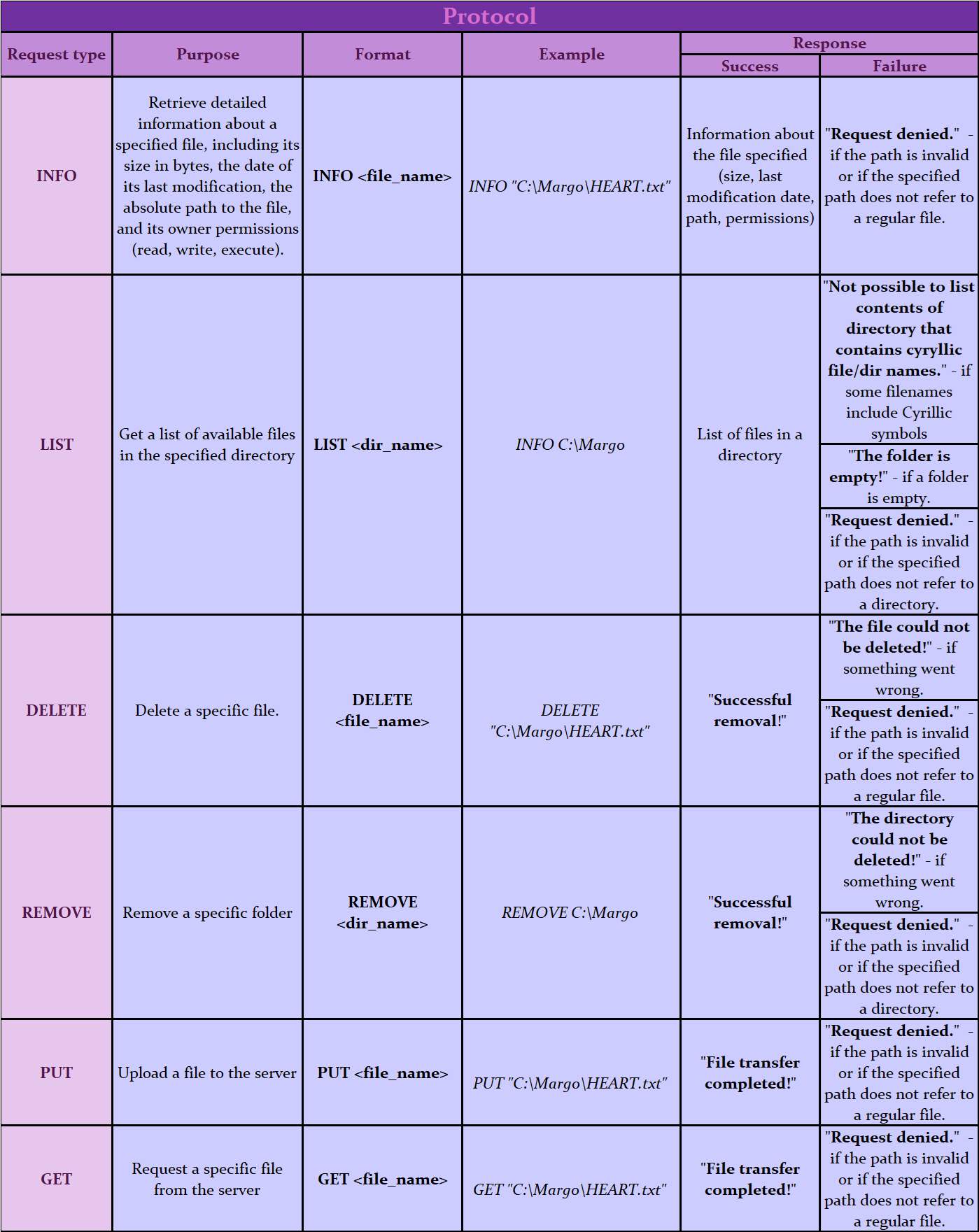
*Server database after both clients testing:*

**

**Protocol:**

The communication begins with the client sending the message **"Hello, server! How are you?"** to the server. In response, the server sends the message **"Hello, client! This is the server."** back to the client. This exchange serves as the acknowledgment stage of the communication.

It is **important** to note that file names can be written with or without double quotation marks. Both formats are valid. Additionally, commands are case-insensitive and can be written in upper, lower, or mixed case. The maximum length of a request is **1024** bytes.

**6 types** of requests are possible and valid:

**Appendices:**

* <https://attacomsian.com/blog/java-check-if-enum-exists#:~:text=There%20are%20multiple%20ways%20to,Otherwise%2C%20it%20throws%20an%20exception>.
* <https://www.geeksforgeeks.org/stream-nonematch-method-java-examples/>
* <https://www.baeldung.com/java-file-directory-exists>
* https://www.geeksforgeeks.org/establishing-the-two-way-communication-between-server-and-client-in-java/?ref=ml\_lbp
* <https://www.geeksforgeeks.org/how-to-extract-file-name-and-extension-from-a-path-in-cpp/>
* <https://www.geeksforgeeks.org/how-to-extract-file-name-and-extension-from-a-path-in-cpp/>
* <https://en.cppreference.com/w/cpp/filesystem/path/operator_slash>
* <https://www.cppstories.com/2024/common-filesystem-cpp20/#20-checking-file-permissions>
* <https://docs1.w3cub.com/cpp/filesystem/permissions/>
* <https://en.cppreference.com/w/cpp/filesystem>
* [Path resolve() method in Java with Examples - GeeksforGeeks](https://www.geeksforgeeks.org/path-resolve-method-in-java-with-examples/)
* <https://www.w3schools.com/java/java_files.asp>
* <https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html>
* Other resources whose links I successfully lost
* Sometimes gpt for mistakes searching (when the situation seemed to be dreadful)