Department of Computer Science & Software Engineering

COMP-6231 Fall 2023

Assignment 1:

Socket Programming

Problem Description

In this programming assignment, you will implement a simple version of an interactive remote terminal application with sockets in Python. The application uses the client-server architecture, where a central server listens to upcoming connections from one or more clients. A client sends the user's command to the server, which executes it and displays back the Current Working Directory (CWD) to the client with the latest changes, if any. Figure 1 shows a flow chart of both the server and client.

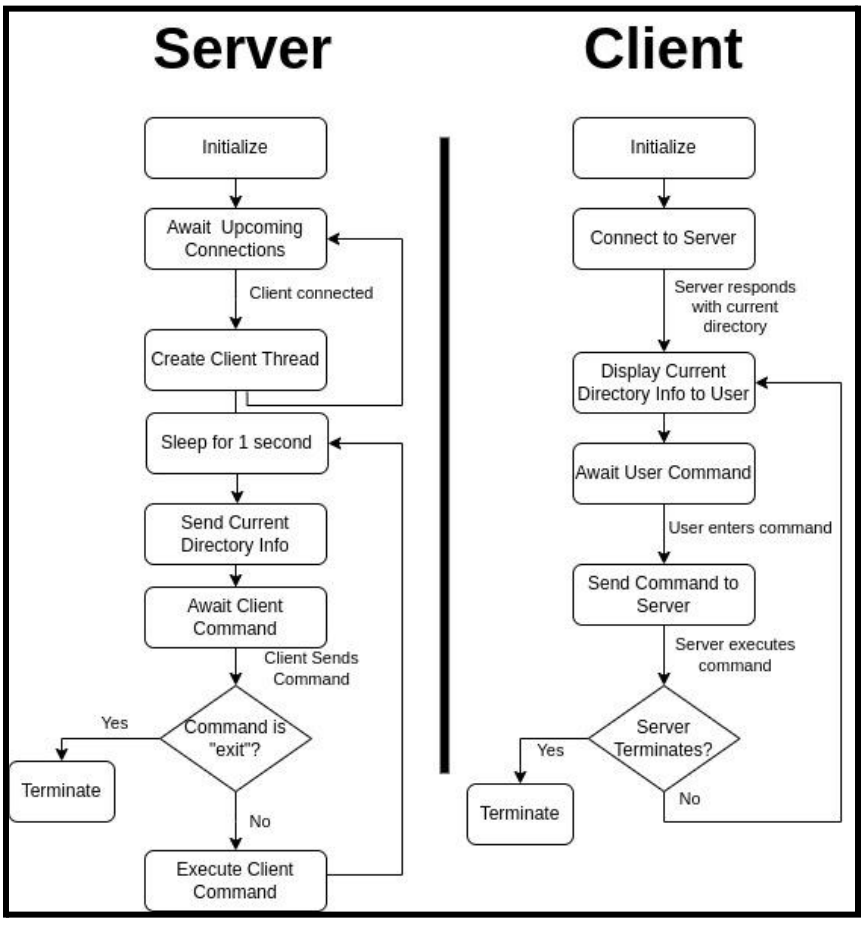


Figure 1: Flow chart of the server and client.

Server: The server initializes its socket and internal variables and awaits upcoming connections from clients. When a client connects to the designated socket (pre-defined), the server handles the connection in a new thread and awaits connections from other clients. In the client thread, the server and client interact with each other to execute the upcoming client commands until the client terminates. When initializing the connection, the server sends the client a random token **of size 10 bytes**, which both the client and server will use to indicate the end of their messages (EOF). The server sends the CWD info to the client before receiving each command. To support multiple clients, the server maintains a `cwd` variable per client.

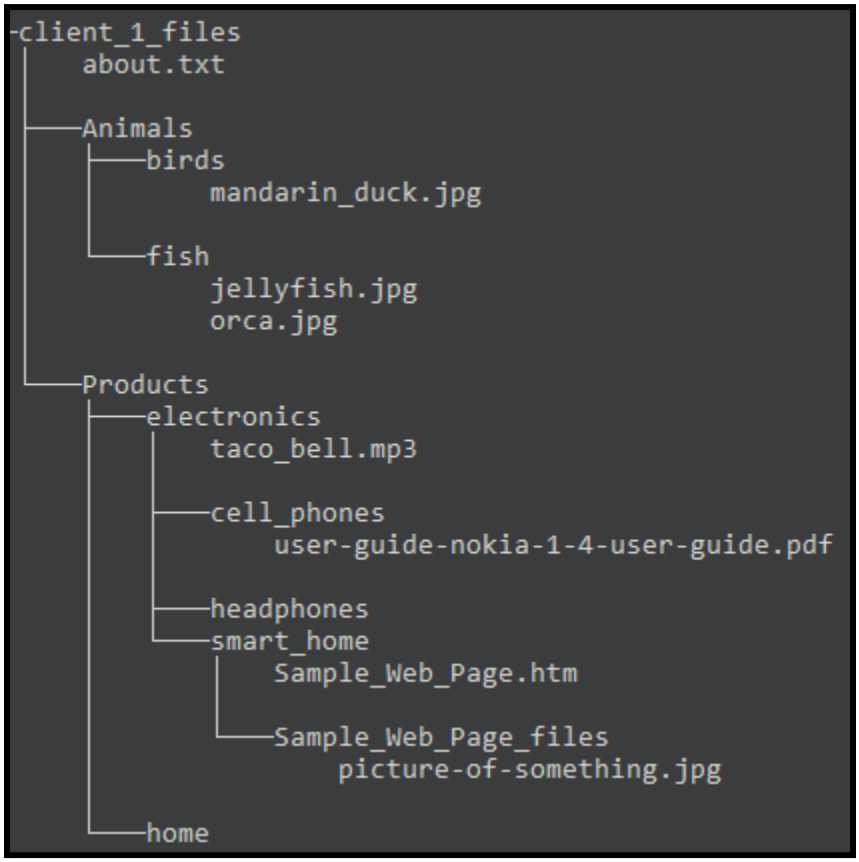
Client: The client initializes its internal variables, establishes a connection to the designated server socket, gets the random EOF token, and awaits the user's command. Before each command, the client displays the received CWD info from the server to the user. After the server has executed the command and sent back the latest directory info, the client displays it to the user and awaits the next command. If the user enters the `exit` command, the client terminates the connection exits gracefully. For simplicity of our application, we assume a fixed current working directory **on the client**, i.e. the user might upload/download files to/from the CWD on the server (mutable), but these files are uploaded/downloaded from/to the directory where the client script is executed (immutable).

In this assignment, you will implement the following functionalities on both the client and server:

Command	Functionality	Example
`cd`	Changes the current working directory on the server to a parent or child directory.	`cd products` `cd ..`
`mkdir`	Creates a new directory on the server inside the current working directory.	`mkdir client_1_files`
`rm`	Removes a file or directory from the current working directory on the server.	`rm about.txt` `rm animals`
`mv`	Relocates an existing **file** on the server to a different location, AND it can also be used to rename files.	`mv about.txt info/` `mv about.txt my.txt`
`dl`	Downloads a **file** from the current working directory on the server to the client.	`dl about.txt`
`ul`	Uploads a **file** from client to the current working directory on the server	`ul orca.jpg`
`info`	Reads **file** size from the server and sends it to the client	`info image.jpg`
`exit`	Exits the applications.	`exit`

You are given a client template and a server template, your task is to fill-in the missing code indicated by a raised `NotImplementedError`. Along with the client, you are given sample

files. After implementing the missing functionalities, your code should be able to create the following directory tree at the server (next to `server.py`):



For all messages between the server and client, use a buffer (packet) size of 1024 bytes. Start by implementing the helper methods in both the client and server. Then implement the main methods that contain the logics of the client and server, including multi-threading the server. Finally, implement the different methods corresponding to each command on both the client and server.

You've been given a `test.py` file to validate your code's fundamental structure and functionality. It's crucial that after finishing your code, you execute this script and ensure it terminates smoothly. Successfully running this script will contribute to a portion of your overall grade, but the entire assignment is not graded solely based on the outcome of this test script.

Grading Distribution

- Initialization, handshake, and helper methods: 30 points
- Main logics of client and server: 10 points
- `cd` implementation: 10 points
- `mkdir` implementation: 10 points
- `mv` implementation: 10 points
- `ul` implementation: 10 points
- `dl` implementation: 10 points
- `rm` implementation: 5 points
- `info` implementation: 5 points

Submission Instructions

- The assignment is due at 11:59PM on Monday October 09, 2023.
- Your code must be in Python within the given templates. Any modifications to the template will incur a 10% penalty.
- Your submission should be a zip file containing two scripts: `server.py` and `client.py`. The zip file name should have the format: `<first_name>_<last_name>_<ID>_A1.zip` (e.g. `john_doe_11111111_A1.zip`).
- If you need clarification about an unclear part in the assignment, send an email to waleed.afandi@mail.concordia.ca.
- If you require help in programming, please schedule a POD session with your respective tutor and prepare your questions. The tutors may assist you with the programming and API's but are not able to provide solutions to the assignment.
- This is an **individual** assignment. You are not allowed to copy/share your solutions with your colleagues. Doing so is considered cheating that disqualifies both submissions (0%) and may be reported to the department.

Late Policy

- 0-24 hours late = 25% penalty.
- 24-48 hours late = 50% penalty.
- More than 48 hours late = you lose all the points for this assignment.
- Submissions of corrupted files, blank files, or the assignment templates will be considered late submissions.