CS 408 - Computer Networks - Fall 2024

Course Project: Cloud File *SU*torage and Publishing

<u>Deadline</u>: December 5, 2024, Thursday, 22:00 (be aware that the deadline is <u>not</u> midnight)

Project Demo: week of December 9 (Schedule to be announced)

Second Chance Demos will be done within the last week of the classes (week of December 23, 2024).

Group Work

You can work in groups of **<u>two</u>** people. If you cannot find a group mate, you can do the entire project alone. We will not group people and will not facilitate groupings. However, you can use the WhatsApp group to find a group member. Your TA Alaa Almouradi will collect group information (please wait for his announcement).

Equal distribution of the work among the group members is essential. Both members of the group should submit all the codes for both the client and the server. Both members should be present during the demos. In case of any dispute within the group, please do not allow the problematic group member to submit the code, so that they will not be graded. However, if a group member submits the same code, then the other member automatically accepts their contribution.

Second chance demo rules

After the announcement of the project grades, you will have three days to correct the problematic parts to make a second demo. However, you will be able to claim only <u>half</u> of the deducted points in the second demo. No group changes are possible during the second demo.

Second chance demo is only for the groups who did regular submission and demo; we do **<u>not</u>** allow using this right for unofficial extension for the entire project with 50 points penalty.

Introduction

In this project, you are going to implement a client-server application using TCP sockets. This application will operate as a cloud file storage and publishing system. In the application, there will be a *Server* module which stores and manages files uploaded by *Client* modules. Each file on the server belongs to the client who uploaded it, but other clients in the system are able to download. In other words, the uploaded files are automatically shared. Each client can delete or update its own uploaded files. Every client can view what files have been uploaded and by whom.

Connections and Operations

The server listens on a predefined port and accepts incoming client connections. There may be one or more clients connected to the server at the same time. Each client knows the **IP address** and the listening **TCP port** of the server (to be entered through the client's Graphical User Interface (GUI)). Clients connect to the server on corresponding port and identify themselves with their names. The connected clients should have unique names at a given time. Server needs to keep the names of currently connected clients to avoid the same name to be connected more than once at a given time. If a client tries to connect with a name that is already currently connected to the server, the client should receive and display an error and fail to connect. **This error, along with all errors and operations, should be displayed on the Server GUI as well.**

Connected clients should be able to upload files to the server. Uploaded files should be stored in a predetermined folder at the server side. This folder path should be set via the server's GUI by browsing the file system before the server starts listening for incoming connections. The Server should keep a list of filenames and corresponding (owner) Client names for each uploaded file. This list of files and owners can be requested by Clients at any moment and should always be up-to-date.

Different clients may upload files with the same filename. In order not to cause confusion of having multiple files with the same filename, the server should employ a mechanism to distinguish which file is owned by which client. There are various approaches to this issue; one suggestion might be that the server appends the name of the owner before the filename so that the filenames are guaranteed to be unique. If you want to use another mechanism for this purpose, it is also OK.

A connected client can upload text files of any length (the files can be very big) to the server any time after connection. Please note that we only deal with text files with ASCII characters in this project; that means, you do not need to consider binary files (e.g. pdf, doc, exe, mp4, etc.) or Unicode characters in files and in filenames. The file format is assumed to be correct. The details and the outcome of this operation should be displayed at both GUIs.

Of course, a connected client can upload several files one by one during the connection. Clients should be able to upload files by browsing the filesystem with the help of the Client GUI. If a client tries to upload a file with a filename that currently exists in the server under that client's ownership, the existing file should be overwritten. The client should be notified about this and the details of this operation should also be shown in the Server GUI.

Connected clients should be able to view files available to download with filenames and the names of clients who uploaded the files, by requesting the list from the server via the Client GUI. There can be multiple files with the same name as long as they were uploaded by different clients. Please remark that the clients do not keep track of existing files at the server; whenever they need a list, a request is sent to the server via the Client GUI.

A client can delete a file that it has uploaded at the Server by sending a *delete file* request to the server via Client GUI. This request should also include to filename to delete. If a client tries to delete a file that it hasn't uploaded or tries to delete a file that doesn't exist, it should get an error. All of these operations and outcomes must be displayed at both GUIs.

Connected clients should be able to download files (of any client) from the server. For this purpose, the client should send a download request to the server along with the filename and the client name who uploaded it. If such a file does not exist, the server returns an appropriate error to the requesting client; otherwise, sends the file. At the Client side, the downloaded files should be saved in a folder, which is to be determined at the client GUI by browsing the file system. When a file is downloaded, the client who uploaded the file should be notified about the details of download, if currently connected (if not, you do not need to remember this for future notifications). All of these operations and outcomes must be displayed at respective GUIs.

In addition, all reasonable/unreasonable user actions must be handled properly and rationally. We cannot list all of them here; use your commonsense for them! Moreover, client and server GUIs must reflect every detail (even if it is not written explicitly here).

For programming rules and submission specifications, please read the corresponding sections at the end of this document.

Below is a summary of the issues that you need to be careful about. However, this list is not comprehensive; please also consider the detailed explanations given above.

Server:

- There is only one server running in this system.
- The port number on which the server listens is not to be hardcoded; it should be taken from the Server GUI.
- The directory in which the files are to be stored must be browsed via the Server GUI (before the server starts listening).
- The server will start listening on the specified port. It has to handle multiple clients simultaneously.
- All activities of the server should be reported using a list box on the Server GUI. We cannot grade your homework if we cannot follow what is going on; so, the details contained in this list box are very important.
- Server must handle multiple TCP connections. At the same time, one or more clients can upload/download files to/from the server.
- Server must accept text files in any size. Be careful while handling big files.
- Each client must have a unique name. This name must be entered using the client GUI. This name is also sent to the server (during initial connection). The server identifies the clients using their names. If a new client comes with a name already in-use, server must not accept this new client.
- Every operation and error should be displayed on the Server GUI in detail and in understandable way.
- The storage of the files in Server is permanent. That means, unless explicitly deleted by the uploading client (in your application) or manually by using the operating system utilities, the uploaded files should exist even after closing the Server and/or Client modules.

- The list of currently uploaded files and the names of the owners might be kept in a text file in Server. You can also find another way to keep them at the server side, but this list should be permanent even after Server closes.
- When the server application is closed (even abruptly), nothing should crash! Also, the process in the operating system regarding the server should be terminated properly; this normally should happen automatically but you are recommended to check the process table of your operating system to make sure.

Client:

- The server IP address and the port number must not be hardcoded and must be entered via the *Client GUI*.
- If the server or the client application closes, the other party should understand disconnection and act accordingly. Your program must not crash!
- All activities of the client should be reported using a list box on the *Client GUI*. We cannot grade your homework if we cannot follow what is going on, so the details contained in this list box is very important.
- Each client must have a username. This name must be entered using the *Client GUI*. This name is also sent to the server (during initial connection).
- Each client can upload any number of text files in any size.
- Client must choose a file to upload by browsing the file system.
- Each client must have on-demand access to the list containing file names and the names of the clients who uploaded each file. For this, clients should be able to request this list from the Server after a successful connection.
- A Client must be able to update or delete files that it has uploaded.
- Different clients must be able to upload files with the same filename, but when a client uploads a file with a filename it has uploaded (and not deleted) before, the previous file should be overwritten.
- When a file is downloaded, the uploader client must be notified, if it is currently connected.
- Each client can disconnect from the system at any time. Disconnection can be done by pressing a button on *Client GUI* or just closing the client window.
- A disconnected client can reconnect at a later time.
- If the client application is closed (even abruptly), nothing should crash! Also, the process in the operating system regarding the client should be terminated properly; this normally should happen automatically but you are recommended to check the process table of your operating system to make sure.
- Both connection and file transfer operations will be performed using TCP sockets.

Programming Rules

- Preferred languages are Python, C# and Java, but Python is recommended.
- Your application should have a graphical user interface (GUI). **It is not a console application!**
- You must use **TCP sockets** as mentioned in the socket lab session. You are not allowed to use any other type of socket. Especially, please do **not** write us to ask for permission to use Websocket or any API for communication.
- Client and server programs should be working when they are run on at least two separate computers. So please test your programs accordingly.
- Your code should be clearly commented. This affects up to 5% of your grade.
- Your program should be portable. It should not require any dependencies specific to your computer. We will download, compile and run it. If it does not run, it means that your program is not running. So do test your program before submission.

Submission

- Submit your work to SUCourse+. **Both group members must submit.**
- Create a folder named **Server** and put your server related codes and, if any, other files here.
- Create a folder named **Client** and put your client related codes and, if any, other files here.
- Create a folder named **XXXX_Surname_OtherNames**, where XXXX is your SUNet ID (e.g. alizesevgi_Yalcinkaya_AlizeSevgi). Put your Server and Client folders into this folder.
    - Compress your XXXX_Surname_OtherNames folder **using ZIP or RAR**.
    - Make sure the name of the zip/rar file contains XXXX_Surname_OtherNames
    - Submit this XXXX_Surname_OtherNames.zip/.rar file.
- You will be invited for a demonstration of your work. Date and other details about the demo will be announced later.
- 24 hours late submission is possible with 10 points penalty (out of 100).

We encourage the use of our WhatsApp group for general questions related to the project. For personal questions and support, you can send email to course TAs.

Please check out syllabus of the course for plagiarism and other general rules.

Good luck!

Alaa Almouradi - alaaalmouradi@sabanciuniv.edu
Saleh Alshurafa - saleh@sabanciuniv.edu
Alize Sevgi Yalçınkaya - alizesevgi@sabanciuniv.edu
Recep Yıldırım - recep.yildirim@sabanciuniv.edu
Albert Levi