CS 408 - Computer Networks - Fall 2020

Course Project: Cloud File Storage and Publishing

This project is made of three steps; each has different deadlines as specified below.

Project Step 1 Deadline: December 2, 2020, 22:00
Project Step 1 Demo: to be announced

Project Step 2 Deadline: December 18, 2020, 22:00
Project Step 2 Demo: to be announced

Project Step 3 Deadline: January 4, 2021, 22:00
Project Step 3 Demo: to be announced

Introduction

You are going to implement a client-server application, which 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 and only that client can delete, copy or choose to publish the file so that other clients in the system can download.

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 Graphical User Interface (GUI)). Clients connect to server on corresponding port and identify themselves with their names. 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.

The abovementioned introduction is for the entire project, which is to be completed in three steps. Each step is built on the previous step and each has specific deadlines and demos.

Project Step 1 (Deadline: December 2, 2020, 22:00):

In this step, you are going to develop a server module that can accept *multiple* client connections. The connected clients should have unique names at a given time. Thus if a client wants to connect with a name that is currently connected to the server, then it should not be connected.

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. Different clients may upload files with the same filename, which causes uniqueness problem in the storage folder. In order to solve this problem once and for all, 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.

In addition, you should have a single database (DB) file to keep track of the information regarding the files (owner, filename, and other information that you think appropriate). This DB file can be line oriented text file (i.e. you do not need to use database management system).

The client module should connect to the server with a unique username as mentioned above. A connected client can upload text files of any length (the files can be very big) to the server any time after connection. Please remark that we only deal with text files in this project; that means, you do not need to consider binary files (e.g. pdf, doc, exe, mp4, etc.).

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 GUI. If a client uses a file name more than once to upload files, then an automatically incremented counter value should be appended to the end of the later filenames by the server (e.g. if the user name is "Anna" and the filename is "Karenina", the filenames stored become AnnaKarenina.txt, AnnaKarenina-01.txt, AnnaKarenina-02.txt, AnnaKarenina-03.txt, ...).

Users perform all the operations through a GUI; such as connecting to the server, entering their name, selecting files to upload, etc.

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

<u>Server Specifications:</u>

- There is only one server running on this system.
- The port number on which the server listens is <u>not to be hardcoded</u>; it should be taken from the Server GUI.
- The server will start listening on the specified port. It has to handle multiple clients simultaneously. To do so, whenever a client is connected to the listening port, the

corresponding socket should be added to a list and the server should continuously accept other client sockets while listening.

- All activities of the server should be reported using a rich text box on the Server GUI including usernames of connected clients as well as all the uploaded file information. We cannot grade your homework if we cannot follow what is going on; so, the details contained in this text box is very important.
- Server must handle multiple TCP connections. At the same time, one or more clients can send files to 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 (preferably during initial connection). The server identifies the clients using their names. If a new client comes with an existing name, server must not accept this new client.
- The storage of the files in Server is permanent. That means, unless explicitly deleted using the operating system utilities, the uploaded files should exist even after closing the Server and Client modules.
- 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.

Client specifications:

- The server IP address and the port number must not be hardcoded and must be entered via the *Client GUI*.
- There could be any number of clients in the system.
- 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 rich text box on the *Client GUI* including connection information as well as file upload details. We cannot grade your homework if we cannot follow what is going on, so the details contained in this text 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 (preferably during initial connection). The server identifies the clients using their names.
- 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 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.
- 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.
- Both connection and file transfer operations will be performed using TCP sockets.

----------------- End of Step 1 ---------------------

Project Step 2 (Deadline: December 18, 2020, 22:00):

Second step of the project is built upon the first step. In this step, you will modify client and server modules to add more functionalities. For all operations defined below, only the owner (uploader) of the file should be allowed to perform the operation.

1) A *Client* must be able to request her own **file list** from the server. The server should return a list containing *filename*, *size* and *upload time* of each file belonging to the requesting *Client*. The names of the files to be sent must **not** include the preceding user name added for uniqueness. However, if added, the counter values at the end must be included.

2) A *Client* must be able to **download** her own files from the server. To do so, the *Client* should enter the name of the file to be downloaded via the GUI. Other clients' files must not be able to be downloaded. 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.

3) A *Client* must be able to **create a copy** of her own files on the server without resending. To do so, the *Client* should enter the name of the file that is going to be copied. The server should apply automatic naming procedure when creating the copy. The filename of the new copy should be the filename of the copied file and an incremental counter appended to it. For example, if the original file name is Pan.txt and the user name is Peter, when this file is replicated at the server, its name at the server folder is PeterPan-01.txt.

4) A *Client* must be able to **delete** her own files on the server. To do so, the *Client* should enter the name of the file to be deleted via the GUI. Other clients' files must not be able to be deleted.

In download, create copy and delete functions described above, the client should enter a file name using the GUI. This file name must not include the preceding user name used for uniqueness; but, if exists, counter values at the end will be used.

As in the step 1, all operations must be clearly shown on the client and server GUIs. For all file operations, successful and unsuccessful completion messages should be shown in detail at the GUIs. Error handling (e.g. non existence of files) should be performed properly.

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

----------------- end of Step 2 ---------------------

Project Step 3 (Deadline: January 4, 2021, 22:00):

Third and the final step of the project is built on the first and the second steps. In this step of the project, the clients will be able to make their files public to all clients. All files uploaded are private (i.e. only the owner can access) by default. To make a file public, the owning client will enter the file name to be made public using the GUI and send it to the Server. After a file is made public, all clients get access to download it, but only the owner can copy and delete it.

If a file is copied using the "create a copy" function that you implemented in the second step of the project, then the access right (public/private) of the original file should also be passed to the new copy.

In addition to the previous **file list** operation, a *client* must be able to request **public files list** from the server. The server should return a list containing *owner*, *filename*, *size* and *upload time* of each file that were made public.

You can keep the public/private information of a file in the DB file so that access rights are kept permanently. In this way, if the Clients and Server close and come back later, information about all public files is restored.

----------------- End of Step 3 ---------------------

Group Work

- You can work in groups of 4-5 people (not less than that except really exceptional cases). No group changes are allowed after they are decided.
- Equal distribution of the work among the group members is essential. All members of the group should submit all the codes for both client and server. All members should be present during the demos.
- Your TA Amir Sepahi (amirsepahi@sabanciuniv.edu) is responsible for grouping and group management. He will send an announcement to the class about how the group information will be collected, how underpopulated groups will be merged and how people without groups will be grouped.

Programming Rules

- Preferred languages are C#, Java and Python, but C# is recommended.
- Your application should have a graphical user interface (GUI). **It is not a console application!**
- You must use pure TCP sockets as mentioned in the socket lab sessions. You are not allowed to use any other socket classes.
- In your application when a window is closed, **all threads related to this window should be terminated.**
- Your code should be clearly commented. This affects up to 10% 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+. Each step will be submitted and graded separately.
- Delete the content of debug folders in your project directory before submission.
- Create a folder named **Server** and put your server related codes and other files here.
- Create a folder named **Client** and put your client related codes and other files here.
- Create a folder named **XXXX_Surname_Name_StepY**, where XXXX is your SUNet ID and Y is the project step (1, 2 or 3) (e.g. fkerem_Ors_FaikKerem_Step1). Put your Server and Client folders into this folder.
    - Compress your XXXX_Surname_Name_StepY folder **using ZIP or RAR**.
- 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 SUCourse+ Social Forum module for the general questions related to the project. For personal questions and support, you can send email to course TAs.

Good luck!

Albert Levi
Faik Kerem Örs - fkerem@sabanciuniv.edu
Mustafa Aydın - mustafaaydin@sabanciuniv.edu
Mehmed Yuşa Ergüven - merguven@sabanciuniv.edu
Amir Sepahi - amirsepahi@sabanciuniv.edu