CS 408 - Computer Networks - Fall 2016

Course Project: Cloud File Storage and Sharing

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

Project Step 1 Deadline: November 23, 2016, 22:00
Project Step 1 Demo: to be announced

Project Step 2 Deadline: December 14, 2016, 22:00
Project Step 2 Demo: to be announced

Project Step 3 Deadline: December 28, 2016, 22:00
Project Step 3 Demo: December 30, 2016

Introduction

You are going to implement a client-server application which will operate as a cloud file storage and sharing 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 uploaded it and only that client can delete, rename or choose to share the file with other clients in the system.

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: November 23, 2016, 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 directory in 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 that happen to have the same filename and in this case the server should be able to store both files and distinguish which file is owned by which client. There are various approaches to this issue; one suggestion might be that the server creates a sub-folder for each client under the main folder and store files owned by the same client in this sub-folder.

The client module should connect to the server with a unique username as mentioned above. A connected client can upload any type of file (text, executables, pdf, word, video, audio, etc.) of any length (the files can be very big) to the server anytime after connection. 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 GUI. If a client uploads a file that she uploaded previously, then the previously uploaded file on the server should be replaced with the new one.

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.

Server Specifications:

- There is only one server running on this system.
- The port number on which the server listens is not to be hardcoded; it should be taken from the Server GUI.
- The server will start listening on the specified port. It must handle multiple clients simultaneously. To do so, whenever a client is connected to the listening sockets, it should immediately be switched to another socket.
- 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 any type of file in any size. Be careful while handling big files.
- Server must save received files in a folder with their original filenames.

- 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.
- The storage of the files in Server is permanent. That means, unless explicitly deleted via the operating system, the uploaded files should exist even after closing the Server and Client modules.
- When the server application is closed (even abruptly), nothing should crash!

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 must have a unique username.
- Each client can upload any number of and type of file 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.
- Both connection and file transfer operations will be performed using TCP sockets.

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

Project Step 2 (Deadline: December 14, 2016, 22:00):

Second step of the project is built on the first step. In this step, you will modify previous 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 *name*, *size* and *upload time* of each file belonging to the requesting *Client*.

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 **rename** her own files on the server. To do so, the *Client* should enter old and new file names via the GUI. If the old file (the one to be renamed) does not exist, then an error message should be returned. If the *Client* already has a file on the server with the new file name, then the operation should be aborted and the *Client* should be informed. Other clients' files must not be able to be renamed.

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.

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: December 28, 2016, 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 share files with other clients. A file can be shared with multiple other clients as exemplified in Figure 1.

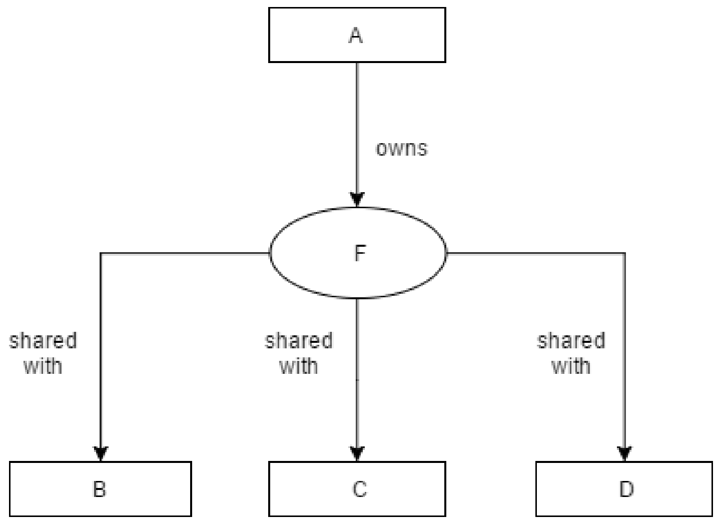

Figure 1. An example file sharing

When a file is shared, the client that is on the receiving end of the process (clients *B*, *C* and *D* in the above example) should be notified that client *A* shares file *F* with them. The shared clients get access to download the file, but they cannot rename, delete, re-upload or share it with other clients.

The owner of the file may choose to discontinue to share a file with a client. In the case of such a revocation, the revoked client should get a notification and should no longer be able to download the file.

If a shared file is renamed, deleted or re-uploaded, then all sharing access on that file should be automatically revoked. All the clients, who had access to this file previously, should receive an explanatory notification explaining the revocation and its reason. For example, let us assume that client *A* owns *recipe.txt* and currently shares it with clients *B*, *C* and *D*. Then, client *A* renames *recipe.txt* to *formula.txt*. In such a case, clients *B*, *C* and *D* should no longer have access on *formula.txt*. As another example, if client *A* uploads a new file with name *recipe.txt* again, it should be treated as a new file and should now be unshared (of course it can be shared later on if *Client* wants to do so). These are just some examples of revocations; there can be other cases as well.

It is your design decision how to keep the sharing information at client and server sides. The only requirement here is that sharing information must be kept permanently so that if the Clients and Server close and come back later, all sharing info is restored.

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

Group Work

- You can work in groups of **three** people. You may change groups between steps 1 and 2; however, no group changes are allowed between steps 2 and 3.
- 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.

Programming Rules

- Preferred languages are C# and Java, 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.**
- 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 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.
- Provide a README file explaining your implementation, detailing any issues and/or other implementation details. If your program is lacking in some requested functionalities, clearly explain them. This file should also serve as a user's manual.
- 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 (e.g. mcandan_Candan_OmerMert) and Y is the project step (1, 2 or 3). Put your Server and Client folders into this folder.
- Compress your XXXX_Surname_Name folder **using ZIP**. Please **do not use** other experimental compression utilities like ICE, bz2. etc.
    - You can download winzip here: www.winzip.com Trial version is free!
- 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 Discussion 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
Ömer Mert Candan - mcandan@sabanciuniv.edu
Emir Artar - eartar@sabanciuniv.edu
Halit Alptekin - halitalptekin@sabanciuniv.edu