CS 484 Networks I

Programming Assignment 2

Chat and FTP application

Due Date: Thursday December 8th, 2020

Overview

This project involves implementing a chat application with an option for a simple file transfer using the C/C++ Socket API. You will be required to use the UDP protocol for transport, therefore you will need to implement your own reliable data transfer mechanism as well.

Project detail

Your submission will include two programs: a server and a client (both connected through a UDP socket). The server should listen on a UDP socket and the clients should connect to the server and to each other through the server. Your clients should be able to run on different machines. Please see the requirements section for necessary error checking criteria that will be evaluated as a part of the grading.

Part 1 Chat App

Create a chat application where a user connects to the server to join the chatroom. The user must be prompted to either login with their name (chat ID) or register with a new user ID before being allowed into the chatroom ('online'). The server must keep a file to dynamically update the users who are online and the registered users with their own unique ID's. Please make sure the server authenticates the client in the case of a login attempt (you can use username and password) and/or checks to see if the user ID is unique before continuing to the next step. Next, the client is given three options from the server which the client chooses via options "0", "1", or "2". In Option 0 the user asks the server to receive the list of online users.

In Option 1, the user chooses among the list of users to chat with and the server returns an error if the username is not within the list of online users. If successful, a new terminal must be spawned while leaving the original terminal still running to be available for choosing between options (0-2). The format must be as follows for the Option 1:

"$1 < user_1 >, < user_2 >, \ldots, < user_n >$"

such that any one or more users may be selected at a time. The user must also be

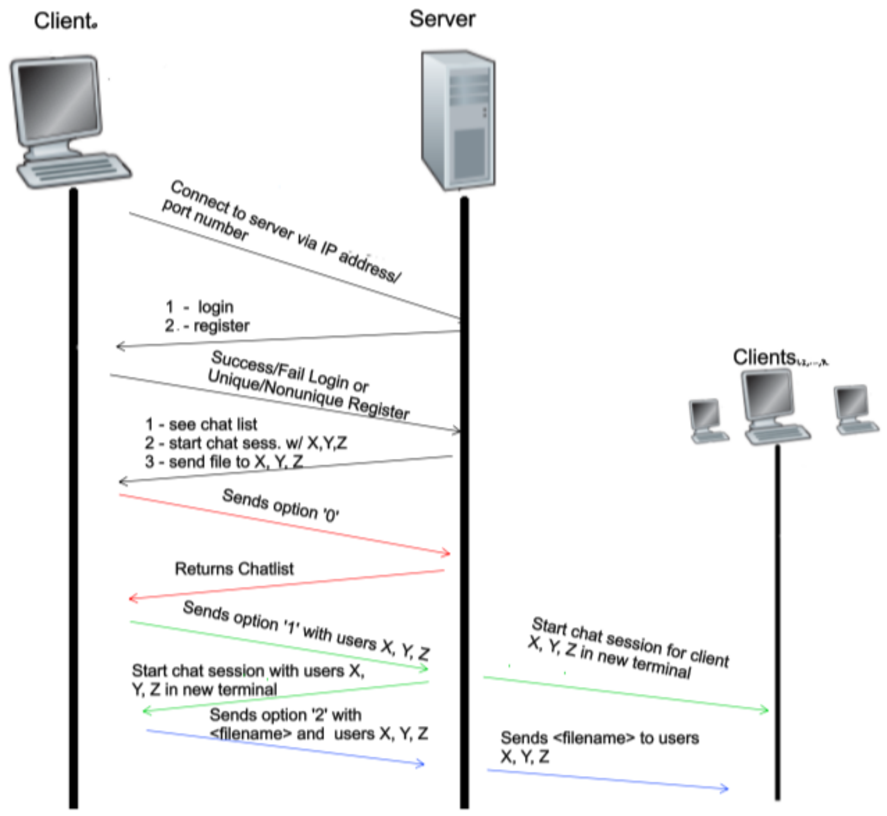
Client
Server
Connect to server via IP address/ port number
1 - login
2 - register
Success/Fail Login or Unique/Nonunique Register
1 - see chat list
2 - start chat sess. w/ X,Y,Z
3 - send file to X, Y, Z
Sends option '0'
Returns Chatlist
Sends option '1' with users X, Y, Z
Start chat session with users X, Y, Z in new terminal
Sends option '2' with <filename> and users X, Y, Z
Clients
Start chat session for client X, Y, Z in new terminal
Sends <filename> to users X, Y, Z

Figure 1: Interactions between the server and the clients

able to create more than one chat session, opening each into their own respective terminals.

Per chat session there will be multiple inputs into the stdout buffer. To make sure that messages don't get overwritten, design a mechanism such that either 1) the buffer of incoming messages is released only after a user finished typing their message and 2) the user's message gets sent after the other users' message.

Additionally, for Option 2 the chat application will handle sending files like explained in Part 2 below.

Part 2 FTP

There will be two ways to send a file. First, a file may be sent via option "2" which must be in the format:

“$2 < filename > < user_1 > < user_2 > ... < user_n >$”

The second option while in a chat session is via hot-key of the form “Ctrl (or Command)+ F” where pressing these two keys at the same time will prompt the user for the name of the file and then sends it to all users in the chat. For both options, the “receiving” client(s) must be notified and prompted to accept the file before receiving it.

You must implement mechanisms to (a) allow multiple clients to download simultaneously (multithreaded) and, because UDP is connection-less and unreliable, (b) recover from lost or rearranged chunks. It is up to you how you implement these features. Moreover, you must design some mechanism which will allow the server and client to communicate and determine when the end-of-file is reached. Some ideas are given in the Hints section below. The receiving end of the file transfers must be saved by each of the $clients_i$ in the format : $< filename > sender's\_name$. E. g., if a user Casey receives a file named file1 from the user Ecem, then his client will save that file as “file1_Ecem”.

Additional Requirements

- Use either C or C++ for this assignment.
- Your code must compile and run on the machines in the CS department labs.
- Code should be well-documented and modular, and must follow the .h /.c standards and naming conventions discussed in class.
- Include a Readme which describes how to use your program, and a Makefile to compile your code.
- Test your code thoroughly. It is also a good idea to run it under valgrind to ensure it is free of memory leaks.
- Make sure to handle all possible error conditions in your code. For example: file not found, no permission, port in use, host not found... You may find these in the codes in ‘Unix Network Programming Vol. 1’
- You’ll be required to do this assignment in groups of not more than 2 individuals. If this is the case, the tarball (.tar, .tgz)(submitted should include the full names of each group member separated by underscore with the suffix _Assignment2)

- Your program when compiled will result in a server and client executable of the following form:

  $./server < port\_numToListenOn >$ and $./client < serverIp\_address > < port\_numOfServer >$

- Sending messages have to be in a format such as $\$username :< message >$
- Signal Handling: The $< enter >$ key will send the user's message and transfer the message to the server's buffer to be sent to the other client. Make sure the clients don't over write each other.
- $< Ctrl + Q >$ will exit the chat-box. $< Ctrl + F >$ will start the file transfer. (You can use other combinations, but put them in the README.
- You are to submit a tarball (.tar, .tgz) to Canvas including your code, the Readme, and the Makefile. Name the file using full names of the both group members and the suffix _Assignment2 (e.g. JayMisra Assignment2.tgz).
- Do not include the executable in your submission.

Hints

- To easily implement reliability and sessioning, you may want to make your file transfer protocol receiver-driven. In a sender-driven protocol, the sender bursts data to the receiver in sequence, then the receiver acknowledges the packets it receives. In a receiver-driven protocol, the receiver requests particular data chunks from the sender, then the sender replies with the chunks requested (think how TCP works). The advantage of a receiver-driven protocol is that instead of maintaining state for several clients at the server, you can instead maintain all state within the client.
- The easiest way to implement reliability is with a stop-and-wait protocol. However, this will result in much slower downloads than TCP. Though it is not required, you may try to implement some sort of pipelining.
- You do not need to implement hash-checking or checksums in your program. Though UDP is an unreliable protocol, it still uses checksums on each packet. Therefore, mangled packets will not be delivered to your application. You only need to ensure that you get all of the chunks, and that they are in the right order.
- You can test your program using a utility like md5sum. Transfer a file to yourself, then check if the MD5 hash of the copy matches that of the original.
- You may want to add some kind of debug mode in your server program, which

will simulate packet loss and reordering to facilitate testing your protocols reliability. For example, debug mode may cause the server to randomly drop packets before sending them, or randomly choose to send chunks in the wrong order. This will allow you to check your reliability implementation without waiting for an error to occur on the network itself.

- You can also fake network unreliability without implementing a debug mode. The following page shows some ways to configure the Linux network stack to force delays and loss:

  http://stackoverflow.com/questions/614795/. Unfortunately, you cannot use these programs on the lab machines, so you will need to have your own Linux instillation to use them. Sufficient experience in Linux administration is recommended if you want to use this method.

- One way to handle the users entering signals such as Ctrl+F or Ctrl+D is to catch and handle the signals. Check out how to use signal.h.