Project 2: A Messenger Application

Due: 12/03/2017

Project Objectives

1. Experiencing signal handling and thread programming
2. Mastering socket based networking application development
3. Getting familiar with application-layer network protocol design and development
4. Understanding how peer-to-peer systems such as Yahoo Messenger work

Project Description

A messenger application has two components: a messenger server and messenger clients. A messenger server is an application program that maintains the user account information and their current location, which facilitates the direct communication between peers. When a client first starts, a user will send the username and password to the messenger server (assuming the user has registered with the server). After signing into the server system, the client will also send the current location information (including both IP address and port number) to the server to which its friends can connect. The server will forward the location information to the user's friends who are currently online. Relying on this location information, two users can directly talk to each other by establishing a direct TCP connection between them.

In this project, you need to write two programs messenger_server and messenger_client, which correspond to the messenger server and messenger client, respectively. The BSD sockets are used for communications between the client and server (and between clients/peers). The functionality of each of these programs is described below.

messenger_server

usage: messenger_server user_info_file configration_file

The messenger server program takes two parameters user_info_file and configuration_file, which are both file names. The file user_info_file contains the user account information. Each user account occupies one line. Each line contains three pieces of user information: username, password, and contact list. They are separated by a vertical bar "|". Contact list may contain multiple friends (i.e.,

their usernames), which are separated by a semicolon ";". Passwords are in plain text. The following is an example line for user "user1", which has three friends in the contact list.

user1|password1|user2;user5;user6

The parameter configuration_file is a configuration file for the server program. Each configuration entry occupies one line with the following format: **keyword: value.** Currently only one keyword **port** is defined. The value of port is a nonnegative integer, which specifies the port number on which the server program should run. When the specified port number is 0, the server asks the system to assign any unused port number for the server.

After the server program starts, it will first read in the user account information and the configuration information. Then it will create a socket using the socket() system call. It binds its address to the socket using the bind() system call. The server machine's fully-qualified domain name and the (assigned) port number are displayed on the screen. Then listen() call is used to indicate that the server is now ready to receive connect requests from clients.

The server's job is to wait for either connection or disconnection requests from clients. If it receives a connect request, it accepts the connection. After the connection has been established, the client will either register with the server, or log into the server by sending the user account information and location information to the server (see below for details) if the user has registered before. When a user tries to register with the server system, the user will send a selected username and password to the server. The server will check if the username is available. If it is available, the user has successfully registered with the server system. Otherwise, the server will inform the user that the selected username has been occupied.

When a registered user tries to log into the server system, the server will first check if the username and password supplied by the client match the ones maintained by the server. If they match, the user has successfully logged into the server, and the server will display a message showing the total number of users online. If the username and password do not match, the server will inform the user that the username and password do not match and ask the user to send the username and password again. After a user (say, user1) successfully logs into the server, user1 will also send the location information to the server. The server will record the location information. Then the server will send the location information of user1's friends to user1. In addition, the server will also send the location information of user1 to user1's friends who have logged in (i.e., who are online).

If the server receives a disconnect request from user1, it closes the connection, deletes the location information, and informs the friends of user1 that user1 is offline. In addition, the total number of users online should also be updated.

messenger_client

usage: messenger_client configuration_file

The configuration_file has the same format as the configuration file of the server program. The client configuration file defines two keywords: **servhost** and **servport,** they specify the messenger server's hostname (or dotted-decimal IP address) and port number, respectively. The client first creates a socket using the socket() system call. It then connects to the server using the connect() system call. The whereabouts of the server are specified in the configuration file as discussed above.

Once the connection is established, the user can perform three operations: 1) register with the server using command "r", 2) log into the server using command "l", or 3) exit the client program using command "exit". When the user type either command r or l, the client program will ask the user to enter the username and password. The username and password will be sent to the server via the connection. The server will return status code "200" if the username is available (for registration) or the username and password match (for login). Otherwise, the server will return status code "500" and the client will inform the user the error and ask the user to enter the username and password again. This process repeats until it succeeds. After logging to the server successfully, the client will then create another socket using the socket() system. It will then bind the socket to local port number "5100" and call listen() to indicate that it is now ready to receive connect requests from other clients for chat. The client's job is to wait for user input from the keyboard, chat request from other clients, messages from other clients on established connection, or location information of friends from the server.

After a user successfully log into the server system, the user can issue a few commands to perform the desired operations.

1. chat: a user uses the command "m" to send a message to another friend with the following format: m friend_username whatever_message. If a connection has not been established between the user and friend_username, the client program should first establish a TCP connection to the client program of friend_username (based on the location information sent from the server).
2. invitation: a user can invite a friend by issuing the command "i"

with the following format: i potential_friend_username [whatever_message]. The invitation message is sent to the server, which then forward the invitation to the invited user.

3. accept invitation: after seeing an invitation message, the user can use the command "ia" to accept the invitation: ia inviter_username [whatever_message]. The invitation acceptance message is sent to the server, which in turn forwards the message to the initial inviter. The server will also update the friend list of both users.
4. log out: a user can log out from the server by issuing the following command: "logout". After the user logs out from the server, the user will have the same interface as when the client program first starts. That is, the user can issue three commands "r" to register, "l" to login again, or "exit" to quit the client program.

When user1 receives chat request from a friend, it will accept the connection request. When user1 receives a message from friend on an established connection, it will display the message in the following format: friend_username >> whatever_message

When user1 receives location information from the messenger server, it will record this information locally and display all the online friends (including the new one) on the terminal.

Application layer protocol

In this project, you have the freedom to design your own application layer protocol, in particular, the message format exchanged between client and server, and among clients (peers). However, you should follow the above mentioned requirements and the well-established principles as illustrated in the design of HTTP and SMTP.

Mandatory implementation requirements

- I/O multiplexing on the server and client programs must be implemented using either POSIC threads or select(), and they must be implemented using different approaches. For example, if the I/O multiplexing in the client program is implemented using POSIX threads, the I/O multiplexing in the server program must be implemented using select() system call. You need to document in the README file which technique is used in which program.
- signal process. Your programs must handle the signal SIGINT, generated when a user types Ctrl-C. In particular, your programs must close the opened socket connections before they exit. In addition, the server should save the user accounts into the user_info_file (if it has not done so).
- properly closing opened socket connections: when a user issues

the command "logout", all the opened socket connections should be properly closed before logging the user out from the server, including the chat connections with other friends.

Optional implementation recommendations

- After a user registers with the server, you have the flexibility to automatically log the user into the server, or the user has to issue the command "l" to log in the server himself after a successful registration.
- You can keep an encrypted password for a user in the user_info_file instead of plaintext.
- You are encouraged to use C++ STL (or TR1) containers and algorithms. You should consider to use C++ STL containers like list, vector, map, or unordered_map (in TR1) to maintain user information, for example, online friends list at the client side and the online users at the server side. TR1 library is installed on linprog machines. You are discouraged from developing home-made containers to manage user information. In particular, you should not statically allocate an array with a certain size to hold the user information.

Grading Policy

A program with compiling errors will get 0 point (we must be able to compile and run your program on linprog). A program that does not implement any above requirements will get 0 point. A program that does not implement I/O multiplexing according to the requirements will get 0 point.

1. proper README file and makefile(5)
2. server initialization with configuration file(5)
3. user registration "r" (5)
4. login "l" (15)
5. quit the program "exit" (5)
6. logout from the server "logout" (15)
7. client chat command "m" (15)
8. client invitation command "i" (15)
9. client invitation acceptance command "ia" (15)
10. handling SIGINT signal (15)

Extra Points

There will be 20 extra points if you implement a GUI on linprog. If you implement a GUI for this project, you do not need to follow the commands we specified above, but you must have the same functionalities (you can certainly add more functionalities). The popular choices of GUI packages on Linux include Qt, wxWidgets, and GTKmm.

Deliverables Requirement

Tar all the source code files including the header files and the makefile in a single tar file and submit via the the submission page on Canvas. Here is a simple example on how to use the Unix command tar (assuming all your files related to this projects are under one directory, say project2). To tar all the files:

tar -cvf proj2.tar *

To check the contents in the tar file:

tar -tvf proj2.tar

To untar a tar to get all files in the tar file (to avoid potential overwriting problems, it is better to do this under a different directory instead of the original project2 directory):

tar -xvf proj2.tar