Group Chat

Rusty Mina

Group chat is a system wherein end-users can communicate with each other using a program called *client*. Usually, there's a program that handles clients requests and actions. Somehow analogous to post-office company in a mail system. *This* program is called *server*.

The objective of this machine problem is to create server and client programs written in C TCP. To do so, knowledge in socket programming and multithreading?(I don't know how to call it but something like that) is required. TCP is a protocol where there's always receiver for senders, unlike UDP, senders does need receivers. By multithreading, what I mean is monitoring and handling many tasks at the same time.

So how does this group chat work? First, a server is set up that handles clients' request for connection. The clients will request for connection to the server which the server will grant. Before the clients can send and receive messages, *they* are required to log-in first. If they don't have account, they can create their own accounts. After logging in, they can already exchange messages. Clients can log out and be disconnected to the server by pressing a key combination (Ctrl + C). The same way, the server could be triggered for shut down. The server will only shut down when all the clients are already disconnected.

Now for the structure of program. First, the server and client programs are largely-based on Beej's(beej@beej.us) works, specifically stream client and select server. Starting with the client program. Creating a client that connects to the server, sends and receives is pretty easy with socket programming, but, creating a client that sends to server while receiving from it is impossible with just socket programming. Note that function for storing inputs (scanf, fgets) is blocking and recv() too is blocking, meaning that you can only do either of them one at a time. One solution is to fork a child that only sends, while the parent only receives. Doing so, will still need another topic, pipes, especially in cases where sending process is largely dependent to the data received such as handling messages with protocols and determining when to mask input (password). The fork approach seems tedious and taxing. What I need is an approach that has blocking functions but are really not blocking. This is where select() comes in. So what is select()? How I understand select () is that it keeps the functions defined to *it* from blocking and changes there state to more like *waiting,* just go back to that function when it's already ready. For the benefit of this document, blocking functions that have been changed to waiting shall be called *waiting functions.* With this, the attention of the process isn't only directed to the blocking function but to other functions as well. Now, how does the process know if the waiting function is already ready? select checks the state of functions defined to *it* using FD_ISSET. Note that it's not actually the function that is being checked but the element (file descriptors) related to it. If the FD_ISSET returns positive, the functions related to being checked will be carried out. If my understanding is correct, a good example of a blocking but not blocking function are signal and sigaction, which were used in writing this program. Pressing Ctrl + C will trigger signal to send SIGINT to a user defined function which will then exit the process. . And thus, a multitasking client is created.

Much has been discussed above. The server program, in essence, is not different to the client program. It just have added/modified features such as instead of function for storing inputs, it used more and more of recv() and accept() as well. Also, it handles the create account and log in requests of the client using string compare functions. FILE programming was used as the means of storing account data. Passwords are hashed before storing and using it for compare. Basically, the server program is just a client program minus STDIN handling, plus variables(check_for_login, connections, for_logout_broadcast.. and many more) that determine the state of certain elements, more for loops, more FD_ISSET and the mentioned above. Anyway, the highlights in server program are as follows:

1. Shutting down the server is a lot more complicated than disconnect client's self to the server.

In client, pressing Ctrl+C will surely disconnect the client from the server since the next thing to *it* is exit(), but for the server, pressing Ctrl+C will result to "Interrupted System Call", EINTR. I tried many fixes like restarting interrupted system calls using SA_RESTART and many more but to no avail, they were all not working. The problem of interrupted system call was just solved by adding a continue to select() failure, while of course, commenting out exit, see line 169, 170 and 171 of server.c

2. usernames and password are referenced to the socket number where they are associated, for example user[4] will be the storage for username in socket 4

3. Linux has a weird way of handling some writing function to files.

While testing the create account feature of my server program, the file that should be written was not being written. See line 355 and 356, even if fprintf returns positive, which is a confirmation of successful writing, the file is still not updated. Then the problem was just how the operating system handles writing to file. I have to use fflush(). The same problem when printf isn't displaying text because it's waiting for newline '\n'.

4. used crypt() for hashing with salt starting with $1$, this is to indicate that the method to be used is md5. the 8 bit salt is vf3bsAyV, the salt as a whole is $1$vf3bsAyV

Overall, the program can still be improved. Use of SQL is recommended. Use of strtok instead of sscanf when processing private message is better. Password masking algorithm is yet to be found. Sending file can also be implemented. Protocol for data being sent and received is encouraged. Hierarchy of users can be implemented, especially if using SQL. Other functions of Internet Relay Chat (IRC) can be implemented.

The programs are largely based on Beej's works. beej@beej.us

And the features are largely-inspired by mIRC. www.mirc.co.uk

Programs are developed under Ubuntu 13.04 environment. crypt.h isn't available in Cygwin.