Report

Application layer message format

Commands that clients can give:

- message <USER> <MESSAGE> - a message <MESSAGE> is sent to user <USER> if they are online if user has not blocked sender. Cannot send message to user that: doesn't exist or is the sender
- broadcast <MESSAGE> - send message <MESSAGE> to all online users that have not blocked the sender
- whoelse – lists all users apart from sender of the command that are online
- whoelsesince <NUMBER> - lists all users apart from sender of the command that have been online within the last <NUMBER> seconds
- block <USER> - blocks user <USER>. This means that the user blocking <USER> will not receive messages and broadcasts from <USER>. Cannot block a user that doesn't exist or is the sender or a user that is already blocked
- unblock <USER> - unblocks <USER>. The sender of the command can now receive messages and broadcasts from <USER> again. Cannot unblock someone that isn't blocked, doesn't exist or is the sender
- logout – logs user out of the server

Logging in

User is first prompted to input username and password. If correct login details are given, user is logged in. If not, the user has 2 more tries at getting the password correct. If the user fails these 2 attempts, they are banned from logging in for BLOCK_DURATION (server is run with this argument) seconds. They cannot log in (even with the right username and password) from the terminal and any terminal they open up until BLOCK_DURATION seconds have passed

Server

The server is run with: python3 server.py <PORTNUMBER> <BLOCK_DURATION> <TIMEOUT>

PORTNUMBER – publicly known port that the

BLOCK_DURATION – number of seconds user is blocked from logging in if they enter 3 incorrect passwords

TIMEOUT – number of seconds that the server doesn't receive a command before the user is logged off

The server contains 5 dictionaries and lists:

- failed – a list of usernames that are currently banned from logging in since a wrong password has been entered 3 times
- online – a dictionary of usernames:time they logged off. This is used for the whoelsesince command to keep track of who was online but logged out
- socks – a dictionary of usernames:connectionSocket. This dictionary contains all currently online users and their socket. Socks is used for message and broadcast

- blocked – a dictionary of lists. For each online user, there is a list of usernames they have blocked. This is updated by the block and unblock commands
- timers is a dictionary of timers and is used to keep track of all timers

First the server socket is set up, then for each client connection, a new thread is created and client_thread function is run. In client_thread, the user is logged in, then it continuously waits for commands from the client and acts on them. A timer is started every time the server is waiting for a command, and if no command is received before TIMEOUT seconds (an argument given to the server when it is run) then the user is logged out.

Client

Client is run with: python3 client.py localhost <SERVER PORT NUMBER>

Localhost is the IP address of the server. It is hardcoded in the client that it is using the localhost to connect to the server since instructors have said that client and server will run on the same machine

SERVER PORT NUMBER - is the publicly known port that the server is running on

The client connects to the server and prompts the user to log in. If they are successfully logged in, a new thread is made which waits for messages to print from the server and prints them. The main thread waits for commands from the user and sends them off to the server. If the client is logging off, it will let the server know, then log off.

Design

Offline messaging is not implemented because there was not enough time. I would've made a dictionary of lists for each user in credentials.txt. Users would then be able to send messages to offline users as their message would be stored in a list under the recipients name and when the offline user logs in, the server would send every element of the list (all messages sent to user while they were offline) for the client to receive and print.

Start_new_thread was one of many methods that I could have used to create new threads. This was probably not the best method but time constraints meant that I didn't have time to research better methods and implement them. Start_new_thread worked well enough but I couldn't stop a thread when I wanted (user_timer). If I had more time, the timers dictionary would probably not exist too as I don't think its needed. There was problems with all users being told multiple times that a user had logged off because they had timed out because of thread issues but I think these have been resolved. If these issues remain, presence notifications have no problem, it is the timer.