COVID Safe Network App by Edward Saucis z5015224

Programming language:

The language used was python3. No additional libraries are required for download.

Program design:

The program consists of four files.
server.py
ServerFunctions.py
client.py
ClientCommands.py

server.py contains the base code for setting up the server through a TCP connection, threading for each client connection, and then reacting to client commands. The IP address is assumed to be the address of the current machine, but it can be edited by changing the value of server_host in server.py

It does so with functions that are stored in ServerFunctions.py, such as authenticate(username, password), which checks the credentials.txt if username and password are correct, newTempID() which generates and returns a random tempID, and getUserDetails(client_temp_ID) which returns the username (phonenumber) of the given tempID.

client.py contains the base code for setting up both the TCP connection with the server, and setting up a UDP socket for sending and threading a receiving UDP socket too. After running, client.py takes in a command and sends these commands to the server and waits for responses.

It does so with functions that are from ClientCommands.py, such as loginDetails() which takes the input of the login, and check_tempID(tempID) which checks if a tempID is valid (not older than 15 minutes)

All messages sent between server and client, and client and client, are encoded with utf-8.

Running the program

1. Start up the server:
   python3 server.py <server_port > <block_duration>
   - server_port: the port number of the server being set up
   - block_duration: the amount of time that a user will be locked out of the program after failing login more than three times.
2. Start up a client (can be done multiple at the same time):
   python3 client.py <server_ip> <server_port> <client_udp_port>
   - server_ip: the ip address of the server being connected to
   - server_port: the port number of the server being connected to

- client_udp_port: the port number of the UDP being set up

3. Login the user. A list of usernames and passwords are stored in credentials.txt. If a username and password are valid, the program will be entered.
4. Run client commands
    a. "Download_tempID": generates and downloads a new tempID to the client, adding it to the list of tempIDs.txt. (no tempID is set when logging in, so this needs to be run first for a successful beacon to be sent)
    b. "Upload_contact_log": sends the current contact log to the server where it prints all the phone numbers of contacts
    c. "Beacon <ip_address> <udp_port>": sends a beacon to a client through a UDP with IP address and port number.
    d. "logout": logs out the client. Client must be logged out before server can shut with Keyboard Interrupt

Assumptions

Contact log:
As per the specs of the assignment, the contact log is shared between all of the clients. This means that when uploading a contact log, it doesn't matter who the command is sent from as it will always be the same. In an ideal program, each client would have its own contact log (titled as username_contact_log.txt), and therefore a more accurate contact list could be made.

Logging out:
Logging out removes the current tempID from the contact log. This is to ensure that the tempID doesn't remain in the contact log when logging out before the 3 minute expiry time. Due to the nature of the project, all clients share one contact log. In an ideal program, after the client logs out, the 3 minute timer would continue outside of the program, and be able to delete the contact log entry after termination of the connection.

TempID:
TempID has a lifetime of 15 minutes. This means that a tempID becomes invalid after 15 minutes. However, this does not mean that it needs to be deleted from the client or the log of tempIDs.txt. A check is done in the beacon stage for if the tempID is valid or not, and is only appended to the contact log if it is valid.

Designs of note:

Upload contact log:
When uploading the contact log, I made it so that the client sends each line of the contact log at a time, and as the server receives each line, it will check for its user and print the phone number. This will produce duplicates if a user with multiple different tempIDs sends a

beacon within the expiry time of the first beacon. This could have been changed so that stores it as a list before then printing the list to remove those duplicates, but the number of duplications can indicate the level of contact a user has with another user.

Also, the sending and receiving of messages through the socket was originally handled so that the server would constantly wait for each line, without sending any signal back, but due to the differing speeds between sends by the client to the receives by the server, it resulted in messages being combined into one. The new design was that as each line is sent, the server will send back a “next” message to ask for the next line, and then an “end” will be sent by the client when the end of the file is reached.

Resources:

Basic socket design was based off the provided code on webcms3 by the lecturer for both TCP and UDP, and adapted to the given design.