CS349 : Networks Lab
Assignment 2: Socket Programming
Problem No. 4 : Instant Messenger

Submitted By :-

Name: Kashish Babbar
Roll No. : 10010178

Date of Submission: 27th February,2013

This README contains the following sections:
1. Contents of Folder
2. Steps for creating the executable and running the Instant Messenger
3. Sample Input and Output
4. Assumptions and Details regarding Implementation of Instant Messenger.

- ## Contents

The complete Instant Messaging program has been compressed as **10010178.tar.gz**. Extract the folder.
(Folder can be be extracted by using the command “tar -zxvf 10010178.tar.gz”)

The Extracted Folder “10010178_InstantMessenger” contains the following:
1. Directory Server (Containing server.cpp and message.h)
2. Directory Client (Containing client.cpp and message.h)
3. Makefile (for creating the executables “ims” in Server folder and “imc” in Client folder)
4. README

The Server and the Client directories contain the server and client code files respectively.
Code for the server is present in Server directory as server.cpp and code for the client is present in Client directory as client.cpp . Also both the directories contain message.h file which is the header file containing the information about the message type and the structure of the im_message.

- ## Steps for creating the executables and running the Instant Messenger :

1. Go to the 10010178_InstantMessenger directory.
    1.1 It contains a Makefile for compiling the codes and creating the executables.
        First run '**make clean**' to delete any un-necessary files.
    1.2 Run the Makefile by using the command '**make**'
    1.3 After running the Makefile, the following executables are created:
    i) **ims** in the Server directory
    ii) **imc** in the Client directory

2. Now, in order to start the server, go to the Server Directory where you'll find an executable by the name '**ims**'.
The instant messaging server takes no command line arguments can simply be run using the command:
./ims
This will start the instant messaging server on your machine (localhost) listening on **port 8888** .
The server will display a message “Server is up and running on port 8888” which means the socket creation and binding has been done properly.

3. Now next we want to run our client.
For this, open a new terminal or a new tab and move to the Client directory where you'll find an executable by the name '**imc**'.

The instant messaging client takes three command line parameters when it's running.
The format is :

./imc USER_NAME HOSTNAME PORT

where USER_NAME is the name of the user, HOSTNAME is the host on which the server is running, and PORT is the port number on that host where the server is listening on.

Here is an example:

./imc kashish 127.0.0.1 8888

user_name is kashish
host is 127.0.0.1 if the server is running on the same machine.
Port is 8888 and is fixed.

In order to run another client, open a new terminal or tab and run the same command but with a different user_name. This will create two create clients who can now communicate.
One can create as many clients as one wants in a similar manner, all of which can now communicate with each other.

When the client starts, it sends a REGISTRATION MESSAGE to the server.
On doing this, the user will receive a message from the server saying the user has been registered successfully if the user didn't exist earlier.
OR if the user already existed the user sends a message saying the user has been already registered and updates the address information of the user.

To exit the client, simply type exit as the client prompt. Before exiting the client first sends a DEREGISTRATION MESSAGE to the server.

The server is intended to run indefinitely, and thus has no explicit means to shut is down. It will simply loop forever, continuously waiting for and processing client requests.
To stop the server, you need to kill it using Ctrl-C.

Now we give a sample input and output run of the messenger and following that provide the details regarding implementation.

- ## Sample Input and Output

Here are series of steps to be followed. Need 3 terminals windows(1 for server and 2 for clients)

Terminal 1 → Server
Terminal 2 → User peter
Terminal 3 → User anna

Firstly, we need to start the server and the clients:

Terminal 1. GOTO Server folder, run the server :

./ims

Terminal 2. GOTO Client folder, run the client named 'peter' as: (assuming you are running the client on same host and if you are not then give the IP of the machine on which server is running)

./imc peter localhost 8888

Terminal 3. GOTO Client folder, run the client named 'anna' as: (localhost → 127.0.0.1)

./imc anna 127.0.0.1 8888

This is what the server shows and what the two clients see:
Terminal 1(i.e. server):

Registered user peter

Registered user anna

Terminal 2 (i.e. User peter):

imc >
ims:You have been Registered Successfully.
Welcome..!!

Terminal 3 (i.e. user anna):

imc >
ims:You have been Registered Successfully.
Welcome..!!

Now, let us send a message from 'peter' to 'anna':

Message from 'peter' to 'anna'

On Terminal 2 (i.e. user peter) type :

imc > anna:how are you ?

On Terminal 3 (i.e. user anna) you'll see 'anna' received a message from 'peter' :

imc >
peter:how are you ?

Terminal 1 (i.e. server) shows the follwing:

Instant_Message From 'peter' to 'anna' Message: how are you ?

Now peter and anna can easily communicate by sending the messages to each other.
For eg:

anna replies:
On Terminal 3 (i.e. user anna) type:

imc > peter:i am good.

peter receives:
On terminal 2(i.e. User peter) you'll see:

imc >
anna:i am good.

Thus both of them can communicate in this manner with each other.
If there are other registered users, they can send messages to each other as well.

Now let us try to send a message to a user who is not registered (eg. kim)

On Terminal 2 (i.e. user peter) type :

imc > kim:hello

On Terminal 2 (i.e. the same user peter) will receive the following:

imc >
ims:User 'kim' not registered.

Now, suppose 'anna' wants to exit i.e. logout

anna types:
On Terminal 3(i.e. User anna) type:

imc > exit

On terminal 3(i.e. User anna receives) you'll see:

imc >
You have Logged Out Successfully..!!
Bbye..!! :)

and client 'anna' exits normally.

Terminal 1(i.e. Server) shows the following:

User anna Deregistered.(Entry deleted from map)

Now if user peter wants to send a message to user anna, he'll receive a message saying User 'anna' is not registered.
Now user peter can exit in a similar way, by typing exit at the prompt.
After that, we can stop the server by killing it using Ctrl-C.

- **Details and Assumptions regarding implementation of Instant Messenger**

The client communicates with the server using UDP packets, and not TCP.

The server is running on the port 8888

The data structure used for storing the records of registered user is “map” for which the #include<map> library has been included.

map<string,struct sockaddr_in> user_table;

The map is between **user_name (string)** and his/her **address information (stored in struct sockaddr_in).**
Initially as the server is started the map is empty, whenever a new user registers an entry is added for that user and his/her address info in the map and an acknowledgement is sent to the user.
If the user already exists, his/her address info is updated and the user is notified.

When a users exits (i.e. sends a *DEREGISTRATION MESSAGE* to the server) the corresponding entry for the user is deleted and the user is notified.

Program Sequence

- When the client starts, it sends a registration message to the server. In this case, it sets the ***type*** field in the message structure sent to the server to be ***REGISTRATION_MESSAGE*** and sets the ***from*** field to be the user name supplied as a command line parameter. The ***to*** and ***message*** fields in the structure are left blank. The address and port number remain packed in the address structure it came from. The server records the client's name as well as the client's address and port number (in the form of struct sockaddr_in) retrieved when receiving the client's registration message with recvfrom() by adding an entry into the map. If an entry already already exists for that user, it's address info is updated.

- Instant messages are input at the client through its standard input. To prompt users for input, the client displays a prompt like ***imc*** >, after which users can type their instant messages to send to other users. To input an instant message, the user inputs the recipient's user name, followed by a colon, followed by the message to send, all on one line. It is assumed that the messages are single line only (no multi-line messages). Also, the messages will be text only, with no file attachments, images, and so on that you might find in a modern messaging application.
  The format is:
  imc > **user_name:Message**
  eg: Suppose I want a send a message 'hello' to kash, this is how it should be done:
  imc > kash:hello

- Periodically, the server will send instant messages to the client when it receives them for delivery from other clients.
- To keep the display nice, an extra blank line is printed before the message to skip past the current client prompt.

- The select() function has been used to wait for input from multiple sources at once, i.e. input from standard input (message to be sent) or input from socket (message received from other client through server or message from server). The select functions allows to process the input which arrives early.

- It is also possible that the user to whom the message is sent does not exist. This happens when the server doesn't contain an entry for the user to whom the message was sent.
  In such a case, the ims sends an error message back to user saying that the user [name] is not registered. (name → the user to whom the message was sent).
  Eg: if user bob was not registered and a registered user kash sends a message as :
  imc > bob:how are you?
  The user kash receives a message from the server saying user bob not registered.
  ims: User bob not registered.

- The client will loop forever, waiting for input from the user or instant messages, processing whichever input arrives, and going back and waiting again. It does not simply send or receive one message and exit.To exit the client, the user simply types ***exit*** at the client prompt.Before exiting, however, the client first sends a deregistration message to the server. The server then looks for the entry of the corresponding user and deletes it from the registered users map. Also the server sends an acknowledgement saying that the user has logged out successfully after which the client exits normally.
  imc > exit
  Here is the received message:
  You have Logged Out Successfully..!!
  Bbye..!! :)
