**Readme**

**Objective:**

In this assignment, we designed a simple broadcast chat protocol server and client.

**Architecture:**

*Client*

* The Client program starts by accepting three arguments, i.e. Username, IP address of the server and port number.
* Socket is created by defining the domain, type and protocol.
* Different parameters of “sockaddr\_in” (socket address), like Family, Port address & IP Address is populated.
* Then the created socket is connected to the server using the address in “sockaddr\_in” structure (we use connect() function).
* Next a JOIN Simple Broadcast Chat Protocol (SBCP) message is sent using write() to server by preparing the content of the message structure with username.
* After that, an ACK is received from server indicating that the connection is successful.
* Next, both the terminal and socket file descriptors are added into select() Read\_FDs set.
* Use select() to check if there is something to write to socket or read from server.
* If there is something to read from terminal, prepare the content of the message structure with the chat message and write the message to server.
* If there is something to read from server socket file descriptor, receive message from server and DISPLAY FOR THE USER
* Also, client keeps track of the time for which the user is IDLE. If user doesn’t type anything for more than 10 seconds, client is said to go into IDLE mode.

*Server*

* The server program starts by accepting three arguments, i.e. IP Address, port number and maximum number of clients.
* After that a socket is created by defining the domain, type and protocol.
* Next, different parameters of “sockaddr\_in” (socket address), like Family, Port address & IP Address (server program accepts any client IP address) are populated.
* Then the created server socket is bound to the address and port number in “sockaddr\_in” structure (we use bind() function).
* Use listen() to see if there is someone who wants to connect to the server from the Socket File Descriptor.
* Every time there is a new client connection, the client socket file descriptor is added into select() Read\_FDs set and an ACK message is sent to client. Also, an ONLINE message is sent to all other clients.
* Select() is used to check if there is something to read and something to write to all other clients.
* New client is connected using accept(), if max\_clients is not reached and new File Descriptor is assigned. Also, the user name is reserved in the usernames array.
* If max\_clients is reached or an already assigned username is used then a NAK is sent. Subsequently, the connection is terminated.
* Message from existing client is received using read() or else if the message has chat content, the message is FWD to the clients.
* If the message shows that a client exited, the OFFLINE information is sent to clients.
* An IDLE message is sent to the clients if a client is idle for a specific amount of time. (This happens upon receiving IDLE message from the particular client)

**Usage:**

* We are using the below mentioned files as source code.
  + makefile
  + Client.c
  + Server.c
  + Utils.h
* The Utils.h file contains SBCP message structure.
* Keep all the source code (makefile, Client.c, Server.c, utils.h) in one directory.
* Do “make clean” to remove all previously created object files.
* Do “make server” to compile server codes, which will create below mentioned object file.
  + server.o for server program.
* Do “make client” to compile client codes, which will create below mentioned object file.
  + client.o for client program.
* Open two terminals. One for server and one for client.
* First run the below mentioned command in server terminal.
  + ./server <SERVER\_IP> <SERVER\_PORT> <MAX\_CLIENTS>
  + SERVER\_IP is the IP address of server, SERVER\_PORT is the port of server,
  + MAX\_CLIENTS is the max number of clients that can connected to the server.
* Always server should be run first.
* Next, in client terminal run the below command.
  + ./client <USERNAME> <SERVER\_IP> <SERVER\_PORT>.
  + USERNAME is the username the client chooses, SERVER\_IP is the IP address of server.
  + SERVER\_PORT is the port of server.
* We will run multiple client by following the above same steps.
* After server is running, number of max\_clients can connect to the server and enter the chat room.
* If more than the number of MAX\_CLIENTS clients want to use the server, it has to wait for an empty slot.
* If the client provides a username which is already in use the client will be prompted use new username and forced out.
* Client, after successfully entering the chat room, will receive a list of clients present in the chatroom (username of other clients).
* Next enter message using keyboard and the message can be sent out, when typed “enter”, to other clients in the chatroom.
* Clients will receive the username and message as "<username>: message"
* Client can exit at any time during the chat and server will send other clients an OFFLINE message.

**Contribution:**

* The Client program was written and compiled by Sabyasachi Gupta ([the.saby\_grad17@tamu.edu)](mailto:the.saby_grad17@tamu.edu))
* The Server program was written and compiled by Chiranjeev Ghosh ([chiranjeev.ghosh@tamu.edu)](mailto:chiranjeev.ghosh@tamu.edu))

Each of us developed the relevant module. Next, we tried to run the code in parallel and we found below issues:

* The SBCP message protocol structure was not working properly.
* The select() function was at first unable to handle multiple clients.
* There was compiler error due to few missing header files.

We collaboratively debugged each of the issues.