CIS 505
Project 3
Distributed Chat Server

Submitted by: Karthik Anantha Ram | akarthik
Sanjeet Phatak | sanjeet
Sarath Vadakkepat | sarathv

ACKNOWLEDGEMENT

We take this opportunity to acknowledge with deep sense of gratitude Professor Boon Thau Loo for his guidance and advice which has led to the successful completion of this project. We would also like to thank all the teaching assistants, especially Jitesh Gupta, our assigned TA, for his invaluable help.

Contents

Sr. No.	Topic	Page no.
1.	Overview	3
2.	Assumptions/Constraints	3
3.	Data Structures & Classes	4
4.	Message Format	5
5.	Components	5
6.	Code Flow	6
7.	Protocols	7
8.	Queues	8
9.	Extra Credits	8
10.	Instructions	9
11.	References	11

1. Overview

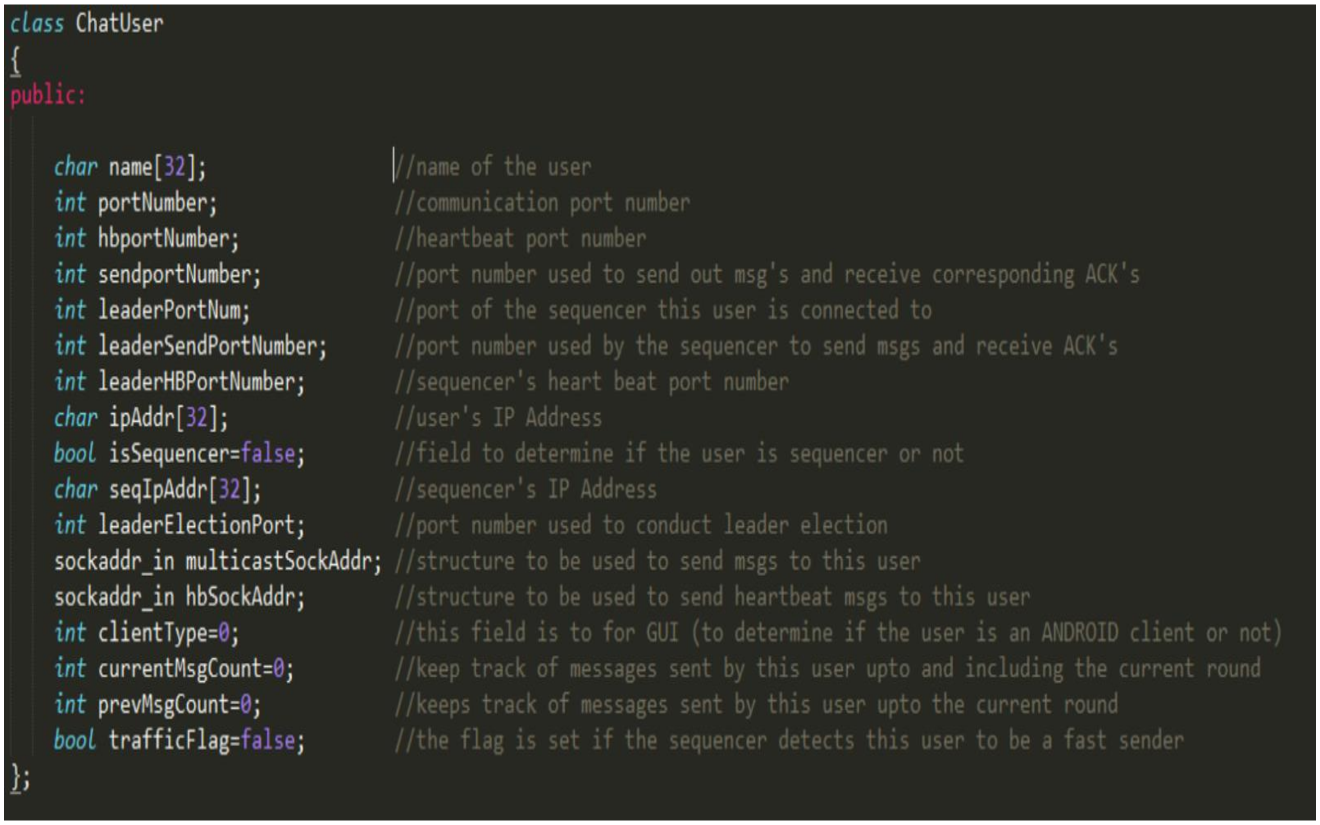
The distributed chat system was programmed using C++ and the network layer protocol used for sending messages over the network is User Datagram Protocol (UDP). This is accomplished using the socket API. The chat system allows users connected on different machines to join chat groups and communicate with each other on the respective chat group. This chat server was tested for 6-7 clients and works perfectly. Features such as leader election, total ordering, heartbeat check and acknowledgments make it a robust and reliable system though it operates on UDP where there is always a chance of losing packets (messages).

2. Assumptions/Constraints

1. This chat server was tested for 6-7 clients in the group.
2. The maximum length of the message that can be sent over the network is 256 bytes.
3. The name of a person can be up to 32 bytes long.

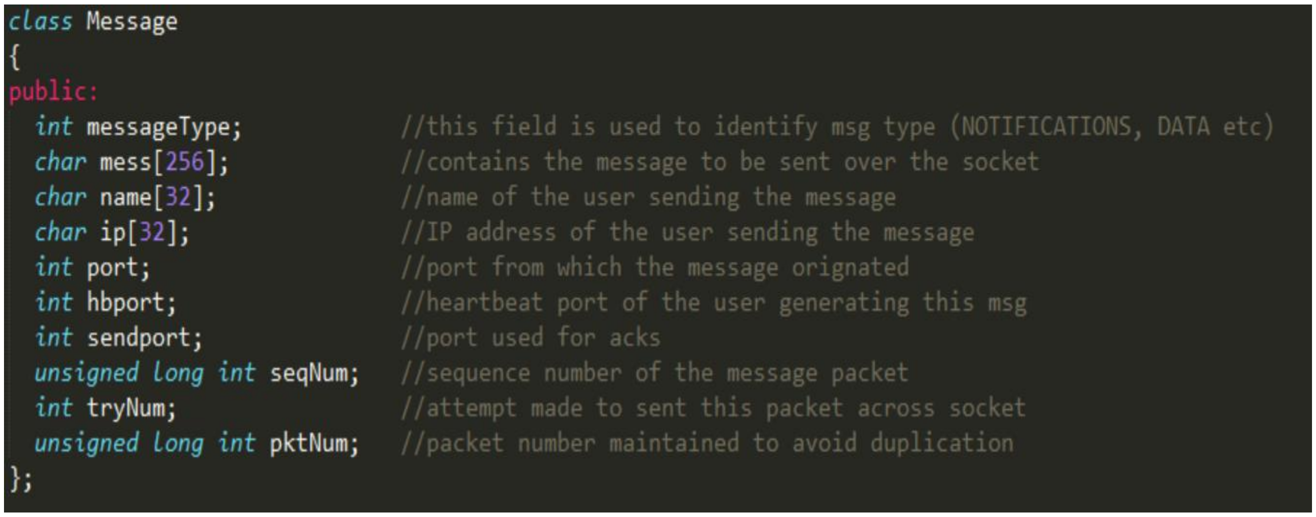
3. Data Structures

The following classes/data structures were used to implement the system.
a. Chat User Class



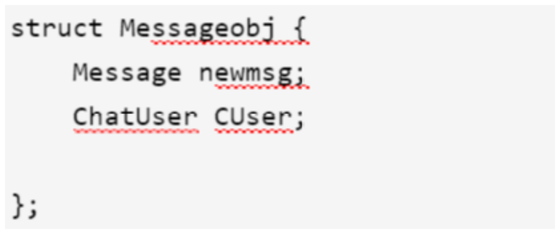
```
class ChatUser
{
public:

    char name[32];                  //name of the user
    int portNumber;                 //communication port number
    int hbportNumber;               //heartbeat port number
    int sendportNumber;             //port number used to send out msg's and receive corresponding ACK's
    int leaderPortNum;              //port of the sequencer this user is connected to
    int leaderSendPortNumber;       //port number used by the sequencer to send msgs and receive ACK's
    int leaderHBPortNumber;         //sequencer's heart beat port number
    char ipAddr[32];                //user's IP Address
    bool isSequencer=false;         //field to determine if the user is sequencer or not
    char seqIpAddr[32];             //sequencer's IP Address
    int leaderElectionPort;         //port number used to conduct leader election
    sockaddr_in multicastSockAddr;  //structure to be used to send msgs to this user
    sockaddr_in hbSockAddr;         //structure to be used to send heartbeat msgs to this user
    int clientType=0;               //this field is to for GUI (to determine if the user is an ANDROID client or not)
    int currentMsgCount=0;          //keep track of messages sent by this user upto and including the current round
    int prevMsgCount=0;             //keeps track of messages sent by this user upto the current round
    bool trafficFlag=false;         //the flag is set if the sequencer detects this user to be a fast sender
};
```

b. Class Message



```
class Message
{
public:
  int messageType;            //this field is used to identify msg type (NOTIFICATIONS, DATA etc)
  char mess[256];             //contains the message to be sent over the socket
  char name[32];              //name of the user sending the message
  char ip[32];                //IP address of the user sending the message
  int port;                   //port from which the message orignated
  int hbport;                 //heartbeat port of the user generating this msg
  int sendport;               //port used for acks
  unsigned long int seqNum;   //sequence number of the message packet
  int tryNum;                 //attempt made to sent this packet across socket
  unsigned long int pktNum;   //packet number maintained to avoid duplication
};
```

c. Structure Messageobj: We employed this structure to send objects over sockets.



```
struct Messageobj {
    Message newmsg;
    ChatUser CUser;

};
```

4. Message format

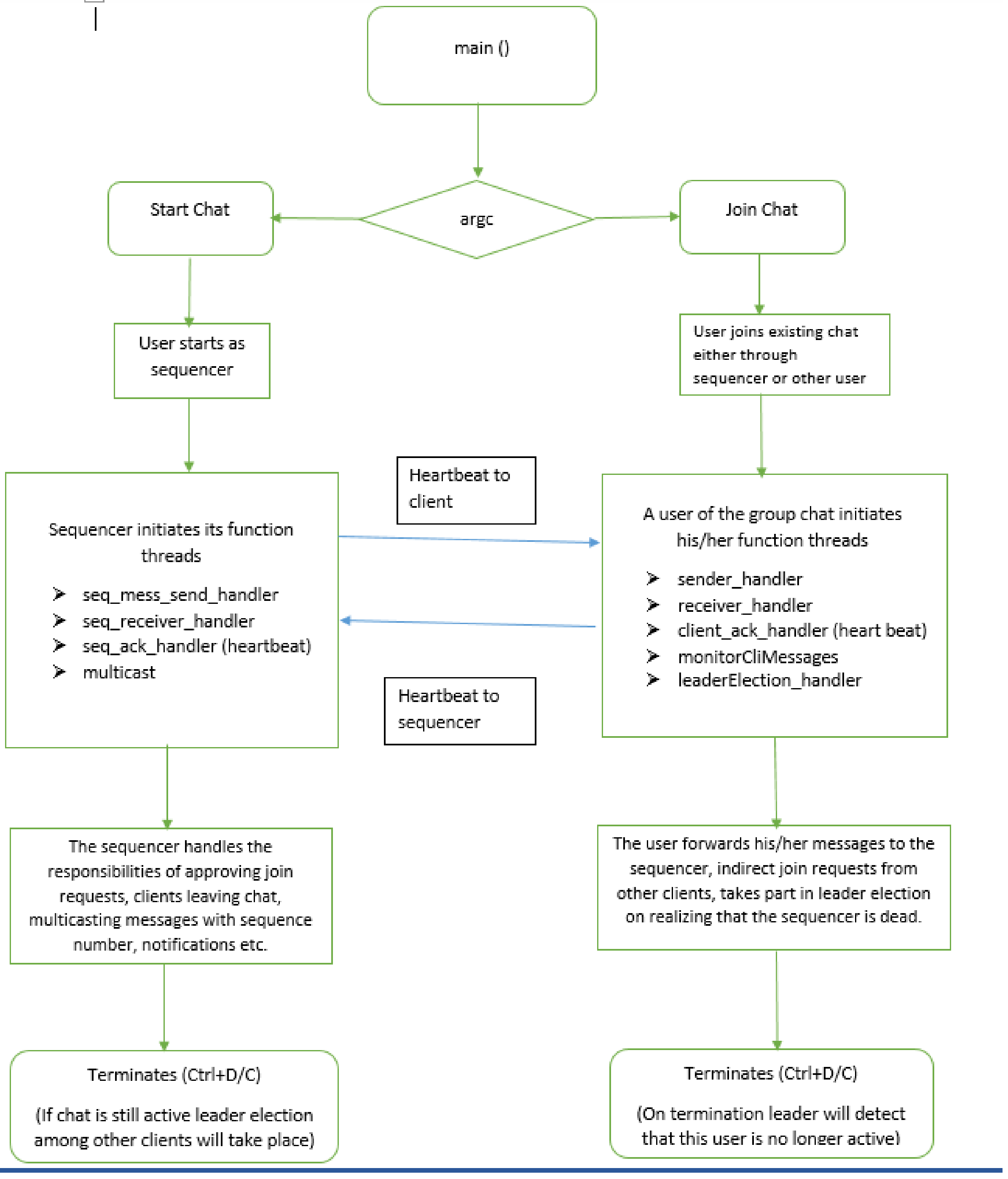
Message is sent as a struct of type “Messageobj” (shown above) which contains an object of the Message class as well as an object of the ChatUser class. The corresponding fields of both the classes are inserted before the packet leaves a given client.

5. Components

a. Sequencer - The user who starts the chat becomes the sequencer by default. All the message exchange among the clients takes place through the sequencer. The sequencer also maintains a record of all the clients in the form of vector containing objects of the ChatUser class, one object per client. It is also the role of the sequencer to detect a crash or graceful exit of any client and remove its object or record from the structure it maintains.

b. Client - Here client refers to users who join an existing chat. The clients send all the messages to the sequencer who then multicasts these messages to everyone in the chat group. It is also the responsibility of the clients to track the sequencer activity to determine if he is alive or not. If he is not alive, it is his job to alert all the other clients and perform leader election (this is mentioned later in the report) to elect a new sequencer (leader).

6. Code Flow


main ()
argc
Start Chat
Join Chat
User starts as sequencer
User joins existing chat either through sequencer or other user
Sequencer initiates its function threads
seq_mess_send_handler
seq_receiver_handler
seq_ack_handler (heartbeat)
multicast
Heartbeat to client
Heartbeat to sequencer
A user of the group chat initiates his/her function threads
sender_handler
receiver_handler
client_ack_handler (heart beat)
monitorCliMessages
leaderElection_handler
The sequencer handles the responsibilities of approving join requests, clients leaving chat, multicasting messages with sequence number, notifications etc.
The user forwards his/her messages to the sequencer, indirect join requests from other clients, takes part in leader election on realizing that the sequencer is dead.
Terminates (Ctrl+D/C)
(If chat is still active leader election among other clients will take place)
Terminates (Ctrl+D/C)
(On termination leader will detect that this user is no longer active)

7. Protocols

a. Leader Election - This protocol comes into picture when the sequencer crashes. The clients, upon detecting that the sequencer has crashed, multicast their port number to all the existing clients in the chat. If the port number received by the client is higher than its own port number, it sends a message "LESSER" and if it is higher than the received port number, it sends a message "HIGHER" to where it came from. This is in line with the bully algorithm.
Note**:** Only port number is used for comparison as the randomizer logic used to generate the port number ensures that no repeated port numbers are allotted. This drastically reduces the probability of two clients having the same port number and hence, not leading to any conflict. In case the port numbers are the same, the last octet of the IP is taken into consideration.
If the potential leader crashes before it becomes the sequencer, the clients, who are waiting for a message from the new leader indicating that he is the new sequencer, will timeout and assume that the new leader has also crashed. Thus, they will reinitiate the leader election process to elect the new leader. This is a recursive process till a new leader is elected.

b. Heart Beat Protocol - This protocol is used to check whether any of the clients or sequencer have crashed or deliberately left the chat. In this protocol, every 3 seconds the sequencer sends a message "ALIVE?" to each client and the client, upon receiving the message sends a message "I AM ALIVE!" back to the sequencer. The sequencer waits for this message type from the client it recently sent the heart beat message to, then moves on to the next client and so on. There is a timeout on the sequencer and clients' recvfrom and if that fails, it assumes the corresponding client or sequencer has crashed or failed. The heartbeat thread is killed whenever there is a leader election.

c. Handling loss of messages (packets) - Since the chat server runs on UDP (User Datagram Protocol) which is not a completely reliable protocol, there is a chance that a message can get lost. To avoid this, whenever a message is sent, it will wait for an ACK message back from the recipient of that message. If it doesn't receive in the defined timeout, it sends the message again to the recipient. This process continues up to three times.

d. Preventing duplication of messages - Whenever an ACK messages fails to deliver, there will be duplication of the messages on the receiver side. To prevent this from happening, there is an array of recently arrived 10 messages maintained at the recipient's side. If the receiver is a client - A field in the Message object "tryNum" gets updated whenever there is a resend of a message. By default, it is zero. If a tryNum of the received message is greater than zero, it checks the array of previously received messages and compares the seqNum to the seqNum of the arrived message. If it already exists, it discards the message. If the receiver is the sequencer it works similar to the above description. The only difference is it checks for the pktNum field of the sender of the latest message and the pktNum fields in all the message objects in the array of 10 previously received messages. The pktNum field is a combination of the port number and the packet count maintained by the sender. The packet count is incremented by 1 as and when it sends a new message. This ensures a unique pktNum to every message. If a match is found then the message will be discarded.

e. Total Ordering – The sequencer allots each message a sequencer number (each time it increments it by 1). All the clients receiving the message check if the sequence number is as expected using the seqCheck variable. If it is greater than the sequence number expected, it puts it into the holdback queue till all the received messages with a lower sequencer number arrive.

8. Queues

The following queues have been maintained to prevent any malfunction –

a. Holdback Queue – This queue is used at the client side. When the sequencer sends a message to the client, if the sequence number the client is expecting is greater than the sequencer number of the received message, it will place the message received in the hold back queue and when all the messages meant to be received before that message are received, it will be dequed and then displayed on the client screen. A Vector data structure is used for this purpose.
b. CliMsgQueue – This queue is created to cover the case when the sequencer crashes midway between one of the clients sending messages. Since, ack will not be received by the client, it is not delivered to the sequencer. Hence, a queue is needed which keeps these undelivered messages and deliver them when a new sequencer is elected.
c. Priority Queue – The priority queue is implemented to ensure that the NOTICE messages get printed first. The NOTICE messages contain information about a client joining the chat or a client crashing/leaving the chat or the sequencer crashing/leaving the chat. Whenever the messages get displayed each client checks the queue for any notice messages. If there is any message in the queue, then it will get displayed first, then the rest of the messages.

9. Extra Credits

a. Traffic Control – Whenever a client is sending messages at a high rate, the sequencer informs the client to reduce the rate of sending. Upon receiving this message from the sequencer, the client dequeues the messages from the CliMsgQueue at a slower rate. When the rate becomes normal, the sequencer again sends a message to the client to dequeuer the messages at a normal rate.
b. Fair Queuing – When the traffic at the sequencer side is normal, the messages are multicast to all the users in a FIFO order. When the traffic increases, a time slice is allotted to each client and the messages are multicast in that manner till the traffic eases.
c. Message Priority – A priority queue is implemented which has been discussed above.
d. Encrypted Chat Messages – An RSA encryption scheme has been implemented. The encryption key is public and the decryption key used to decode the message is different from the publ ic key and is kept secret.
e. Decentralized total ordering – In decentralized total ordering, there is no concept of sequencer. When a client wants to send a message to the chat group, it sends the message to

all the clients first. Upon receiving the message, each client computes the maximum agreed sequence number it has received till the current received message, it increments it by 1 sends the proposed sequence number back to the sender. Once all clients' proposed sequence number is received by the sender, it computes the maximum of the proposed sequence number and sends it back to the clients as the agreed sequence number along with the message again. The clients, upon receiving the message display it with the agreed sequence number.

f. GUI – An Android application is created which serves the same functionality as the normal chat server. A user on an android phone can participate in the chat, normally send and receive messages using his phone.

11. Instructions

Commands to run the chat server –

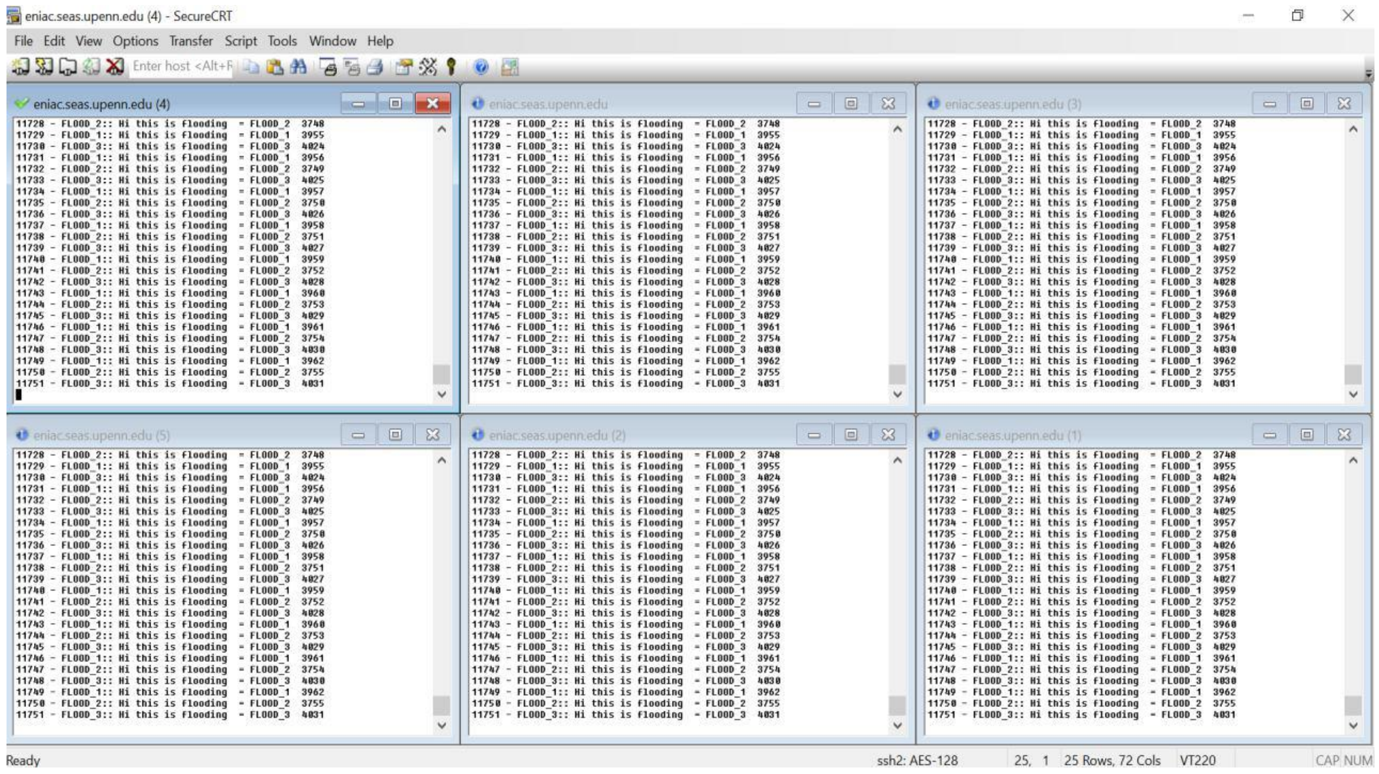
Type the following in the command window:

1) make to compile all the files. It will compile in the following way
2) ./dchat to start the server.
3) ./dchat <client_name> <IP:Port_number> to start a client. Eg- ./dchat client_1 127.0.0.1:5000

Commands to run the clients which test the extra credits –

1) Traffic Control – Make ExCredit_1 true in the code
2) Fair Queuing – Make ExCredit_2 true in the code
3) Message Priority – Make ExCredit_3 true in the code. Run clients named DATA_C1, DATA_C2, DATA_C3, JOIN_C1, JOIN_C2 and JOIN_C3. The JOIN clients will join the chat; DATA clients will flood messages. The Notice messages will get printed before the data messages.
4) Encrypted Chat Messages – Make ExCredit_4 true in the code
5) Run as **./dchat_dec <client name>** to start the chat and **./dchat_dec <client name> <IP:PORT>** to join another existing chat.
6) GUI – Make ExCredit_6 true in the code

Note: - In order to test special scenarios, the client names while joining the chat should be FLOOD_1, FLOOD_2 and FLOOD_3. These clients flood 7000 messages to the server as shown below.



eniac.seas.upenn.edu (4)

```
11728 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3748
11729 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3955
11730 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4024
11731 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3956
11732 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3749
11733 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4025
11734 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3957
11735 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3750
11736 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4026
11737 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3958
11738 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3751
11739 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4027
11740 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3959
11741 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3752
11742 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4028
11743 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3960
11744 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3753
11745 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4029
11746 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3961
11747 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3754
11748 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4030
11749 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3962
11750 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3755
11751 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4031
```

eniac.seas.upenn.edu

```
11728 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3748
11729 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3955
11730 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4024
11731 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3956
11732 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3749
11733 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4025
11734 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3957
11735 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3750
11736 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4026
11737 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3958
11738 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3751
11739 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4027
11740 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3959
11741 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3752
11742 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4028
11743 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3960
11744 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3753
11745 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4029
11746 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3961
11747 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3754
11748 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4030
11749 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3962
11750 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3755
11751 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4031
```

eniac.seas.upenn.edu (3)

```
11728 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3748
11729 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3955
11730 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4024
11731 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3956
11732 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3749
11733 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4025
11734 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3957
11735 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3750
11736 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4026
11737 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3958
11738 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3751
11739 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4027
11740 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3959
11741 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3752
11742 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4028
11743 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3960
11744 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3753
11745 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4029
11746 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3961
11747 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3754
11748 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4030
11749 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3962
11750 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3755
11751 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4031
```

eniac.seas.upenn.edu (5)

```
11728 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3748
11729 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3955
11730 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4024
11731 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3956
11732 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3749
11733 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4025
11734 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3957
11735 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3750
11736 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4026
11737 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3958
11738 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3751
11739 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4027
11740 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3959
11741 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3752
11742 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4028
11743 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3960
11744 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3753
11745 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4029
11746 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3961
11747 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3754
11748 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4030
11749 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3962
11750 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3755
11751 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4031
```

eniac.seas.upenn.edu (2)

```
11728 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3748
11729 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3955
11730 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4024
11731 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3956
11732 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3749
11733 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4025
11734 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3957
11735 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3750
11736 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4026
11737 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3958
11738 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3751
11739 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4027
11740 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3959
11741 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3752
11742 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4028
11743 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3960
11744 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3753
11745 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4029
11746 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3961
11747 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3754
11748 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4030
11749 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3962
11750 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3755
11751 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4031
```

eniac.seas.upenn.edu (1)

```
11728 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3748
11729 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3955
11730 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4024
11731 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3956
11732 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3749
11733 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4025
11734 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3957
11735 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3750
11736 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4026
11737 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3958
11738 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3751
11739 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4027
11740 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3959
11741 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3752
11742 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4028
11743 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3960
11744 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3753
11745 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4029
11746 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3961
11747 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3754
11748 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4030
11749 - FLOOD_1:: Hi this is flooding  = FLOOD_1  3962
11750 - FLOOD_2:: Hi this is flooding  = FLOOD_2  3755
11751 - FLOOD_3:: Hi this is flooding  = FLOOD_3  4031
```

12. References

1) Lecture Notes and recordings

2) Beej Guide for Networking

3) https://en.wikipedia.org/wiki/Bully_algorithm

4) https://en.wikipedia.org/wiki/Round-robin_scheduling