Project3

Team Bloopers

Harsh Verma(hverma)
Prahalad Venkataramanan(prahalad)
Rony Edde(redde)

Dchat is a messaging system based on UDP. The spec relies on strict total ordering of messages received and message queues for storing sent messages and holding back messages that were unsuccessful or not in the expected order of delivery.

Main Behavior:
The first instance of dchat is started as server with the following command "./dchat bob", where bob is the name of the user and bob is a server. The output seen by the user is the following:
This user listens to port number '44900'
This indicates that the server port is 44900. Subsequent clients requesting a connection to the server will need to specify this port along with the ip of the server. A client requesting to join a local chat for example will have to run the following command: "./dchat steve 127.0.0.1:44900" at which point the server displays "**steve joined the chat on 127.0.0.1:46577, listening on 127.0.0.1:44900**" and will broadcast the following to all clients and itself:
bob 127.0.0.1:44900 (Leader)
steve 127.0.0.1:46577

Connection:
A client can connect to the server directly by specifying the server ip and port. The client sends a request connect to the server, once the server received the request, it adds the client to the client list and sends a connection established acknowledgement to the client confirming that the client is now connected to the server.
A client can also connect through another client by sending a request connection to another client. Upon receipt of the request, the receiving client, not being a server will send a server info message which contains the server ip and port information. Once the requesting client receives the server info message, it will then connect directly to the server using the info sent from the other client.

Messaging:
When a client sends a chat message, it adds it to its send buffer and sends it to the server.. The server then sends an ack acknowledge message back to the client. Once the client receives this message, it takes it out of the send buffer otherwise if it didn't receive an ack from the server it tries to resend the message to the server.
The server, upon receiving the message will add it to the broadcast queue and increment its sequence number. The server will send the messages as MSG type in sequence to the clients and will maintain the same order while sending to all the clients, this is achieved using the HoldBackMap at the clients. Once the message is sent, it's taken out of the broadcast queue and added to the broadcast buffer.
Lost messages can be retrieved when a client received a sequence number higher than 2 compared to the expected sequence number, it will then send a retrieve message to the server before displaying the messages with higher message ids. This ensures total ordering.

Disconnection:
- When a client gets disconnected, the server will know about this using a heartbeat. The hearbeat is sent every 3 seconds to all clients by the server to check the status of all the clients. If a client fails to respond, the server removes the client from the client list and will broadcast a message to all clients including itself about the disconnection. For example if Steve disconnects, the following output is seen on all clients and server: Also if the client disconnects normally, the disconnection message is displayed instantly to all the clients and the server.
NOTICE: steve left the chat or crashed
Upon receiving this message, the clients will also update their client list by removing steve from the list.
- When a server disconnects, all clients will detect this by their own server heartbeat which checks for the server's status every 3 seconds. Once the server is declared dead, a leader election takes place and the process with the highest port is declared leader.

Leader Election:
We are performing exact Bully Algorithm to determine the next leader of the chat system.
Once the server dies, all clients will know about it from the 3 second heartbeat. Once the server is declared dead, every client will check all other client's port numbers and will bully the clients which have a port id less than their own. Every client will send a request leader election message to all clients with a process ID higher than their own. Then every client will wait for 3 seconds to hear back from other clients. If no new bully message is received of type **STOP_LEADER_ELECTION**, it means that the receiving client is either the one with the highest process id or that the clients with higher process ids have crashed and did not send a bully message. At this point, if no bully message is received, the client declares itself the new leader and becomes the server and will send an elected leader message to all clients in this format: "**Sending new leader elected msg to 34040**". The clients receiving the messages will then declare the new leader a server and will display a message in this format: "**Recd new leader**

elected msg, port:40942”. They will now listen to the new leader as a server and will receive all messages from the new server.

Encryption(Extra Credit):
The encryption algorithm is based on an encryption key string and it is divided into 2 steps for encoding and decoding.

The first step consists of a series of additions and subtraction operations.
These are alternating operations for the first half of the key where he char value
is divided by 2 and added or divided by 3 then subtracted.

For the second half of the key, these operations are swapped but the division operations are ignored changing the values of the string based only on the current key char value.

The decoding applies the same operations in reverse order, thus recovering the original string.

the encoding itself takes a user defined string length for extended encryption. For example a string such as "hello" might reside in memory as a char[512]. After encryption all 512 characters will be scrambled making the string unreadable and having a non defined length. Only after encryption will it be possible to determine the string's original length.

Example:
Using the following key "TEAM BLOOPERS ROCKS YOUR SOCK TO THE MOON AND BACK!", The resulting encryption of a 448 char wide string with the first entries equal to "hello\0" result in the following encrypted string:

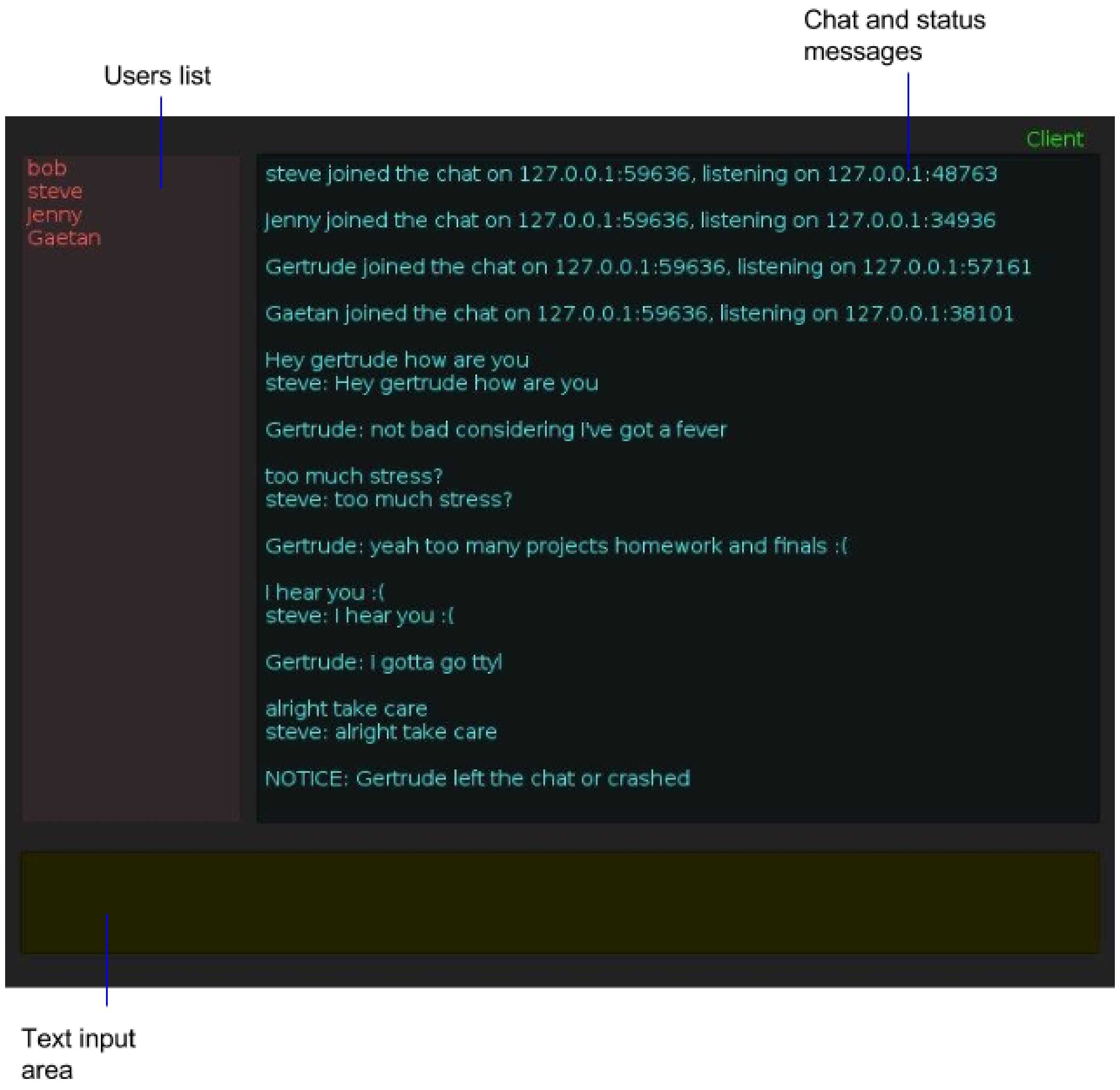
```
�����||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

However there are exponentially as many combinations for the same message provided the sting contained previous values prior to setting the first null terminated characters. As another example, using the same string that contained previous values, the same "hello\0" encryption will result in the following string:

```
�����||▒l`gk]▒o`g▒dac]\▒hYkkaf_▒]n]jql`af_▒af▒Y▒\]]h▒hdYmkaZd]▒l]je▒g^▒fgf]k]fk]▒o`alg
ml▒Yfq▒hjagj▒cfgod]\_]▒g^▒l`gk]▒o`g▒\gfl▒][Yj]▒YZgml▒)*+▒Yf\▒./01,+*▒oadd▒^af\▒8▒mfV▒
k\^▒cb`_▒Yf\▒fgl▒o`g]n]j▒ak▒ZdY`▒ZdYZZ]jaf_▒l`]▒ZdYZZ]j▒l`af_▒YZgml▒fgf]▒ZdYZZ]j▒^mf
_mk▒^dYZZ]jaf_▒l`]▒_YdZdY\ja^dYZZaf_▒_dgZ���������������������������
��������������������������������������������������������
��������������������������������������������������������
������������
```

the decrypted string will still contain the correct leading value "hello\0" but to an outsider these strings are different.

Interface(Extra Credit):
The GUI provides a user friendly interactive user experience with the same backend code.
The window title displays the name of the user, the top right side shows the status of the user client or server. The left side is a list of all users starting with the server, the right side of the window is the message display where all sent received and status messages can be seen. The bottom text edit is where users can type their message. Hitting enter will send the message.



```
steve joined the chat on 127.0.0.1:59636, listening on 127.0.0.1:48763

Jenny joined the chat on 127.0.0.1:59636, listening on 127.0.0.1:34936

Gertrude joined the chat on 127.0.0.1:59636, listening on 127.0.0.1:57161

Gaetan joined the chat on 127.0.0.1:59636, listening on 127.0.0.1:38101

Hey gertrude how are you
steve: Hey gertrude how are you

Gertrude: not bad considering I've got a fever

too much stress?
steve: too much stress?

Gertrude: yeah too many projects homework and finals :(

I hear you :(
steve: I hear you :(

Gertrude: I gotta go ttyl

alright take care
steve: alright take care

NOTICE: Gertrude left the chat or crashed
```

Decentralized Total Ordering(Extra Credit):

This feature has been implemented using the Algorithm for Decentralized Total Ordering as given in the Lecture Slides. Every process maintains two values P and A which are the highest sequence number proposed by the process and the highest sequence number that has been observed by the process in the chat respectively. When someone wants to send something to all the other clients, they just broadcast it without a valid sequence number. Now every client on receiving this message will push it into their holdback queue and they will propose a sequence number for this message using a message type of PROPOSE.The proposed sequence number wlll be taken as the max of the value of P and V maintained at the process.Now the sending process will compare the sequence number received by all the processses for a particular message that it sent and will send a final Timestamp for that message to all the processes. Now the clients will assign this Timestamp to their message in the holdback queue, they will now reorder the HoldBack Queue in the order of the timestamps and if the front of the Queue is a occupied by a deliverable Message having a valid Timestamp, it will deliver it. So the total ordering is maintained.

Fair Queueing(Extra Credit):
For implementing this feature round robin scheduling is used, the server maintains a queue for each of the chat clients and puts the chat message from a particular chat client to this queue map. The map has Client Port Number as the key and the Queue of Chat messages as the value. While broadcasting, the Broadcast thread runs through all the clients' queues and checks if the queue is not empty. If so the thread will broadcast one message and pop it from the queue and go to the next client's queue. So in this way all the clients will equally share the Broadcaster Thread and the Bandwidth of the server.

Message Priority(Extra Credit):
For this feature two queues and two message listeners are maintained(Both the message listeners listen on two different ports) and whenever a message is received of higher priority it is put in the higher priority queue and the lower ones are put in the other queue. Messages, Retrieve Message, MSG_NOT_FOUND etc. are messages of lower priority and Leader Election, Connection Requests, Disconnection Messages have a higher priority. So now the broadcaster thread will check the higher priority queue and if that is empty it will go to the lower priority queue.Otherwise it will totally flush the higher priority queue before going to the lower priority queue. Thus the message priority is maintained for the messages of the two priorities. We have implemented this feature for the normal credits also as our base code is also listening on two ports and pushing the messages in to the two queues depending on the type and importance of the messages.

Assumptions:

1) Please ensure that the username you use is alphanumeric. The maximum length of the username can be 50 characters.
2) Considering the limitations of UDP payload size, the maximum allowed chat data can be a maximum of 420 characters

Payload architecture:

The payload that's being transferred as UDP packets constitute the following parts:

a) MSG_TYPE:
   This parameter defines the type of message that's transferred. Following are the message types defined in our dchat system:
   1) CHAT: Contains the data that the client wishes to broadcast to all the members
   2) MSG: Contains data that the sequencer broadcasts upon receiving a CHAT request from the client
   3) ACK: Server's acknowledgement upon receiving a CHAT message
   4) REQ_CONNECTION: This is the message a new client sends in order to join an existing chat
   5) CONNECTION_ESTABLISHED: Server's positive response to a REQ_CONNECTION request
   6) SERVER_HEARTBEAT: Heartbeat signals that clients ping to the server to check if it is alive or not
   7) CLIENT_HEARTBEAT: Heartbeat signals that the server pings to all the clients to check if they are alive or not
   8) REQ_LEADER_ELECTION: If a client observes that the server is not alive, it sends this message to all members listening on a higher port requesting them for a new leader election
   9) STOP_LEADER_ELECTION: If a member with a higher listening port receives a leader election request from a lower member, it responds to the request with this message
   10) NEW_LEADER_ELECTED: After leader election process, the leader sends this message to all the members in the dchat system
   11) RETRIEVE_MSG: If a client is expecting a MSG of a particular sequence number but receives a MSG with seq number that is greater than the expected sequence number by two, this message is sent to the server along with the expected sequence number requesting it for a retransmission
   12) MSG_NOT_FOUND: If the server doesn't have the requested sequence number (using RETRIEVE_MSG) in its file, it sends this message to the client asking it to move on with the next seq number

13) SERVER_INFO: If a new client tries to connect to a chat system through one of its clients, the existing client sends the server's information to the new client in this message asking it to reconnect through the server.
14) NEW_CLIENT_INFO: This message is sent to all the clients by the server whenever there's a new notification with regard to new connection/client deletion.
15) CLIENT_LIST: This message carries the information of all the clients in the chat system
16) CLIENT_EXITED: This message is sent by the client to the server to notify that it has exited the chat.

b) SEQ_NUM: Carries the sequence number of the message.
c) MSG_ID: Carries the message ID of the CHAT message that the client sends to the server for broadcast.
d) NAME: Carries the name of the sender
e) SENDER_LISTENING_PORT: Carries the port of the sender on which it could possibly be awaiting response.
f) DATA: Contains DATA that's to be transmitted.

Running the app:
Type make dchat to generate the executable and run the application as mentioned above and the project spec.