**CIS 505 - Project 3 - Distributed Chat Server**

Milestone 1 : Due March 28th, 2016.

Group Name : UPenn_EMBS
Group Members :Karthik Anantha Ram, Sanjeet Phatak & Sarath Vadakkepat

**General Design Decisions :**

Language : C++
Protocol : UDP

**Program:**

"dchat.cpp" : This is the one program does both the job of server and client. It is built on UDP. We will be generating a linux command "dchat" which will be used to initiate the system.

**Approach:**

We will be creating an object for each user who is part of the distributed chat system. The object will belong to class called "ChatUser". The different fields of the class are:

- Name
- Unique Internal Resource Identifier(UIRI) [ to distinguish between same names ]
- IP Address
- Port Number
- IsSequencer : A boolean field to indicate if that user is a sequencer for that group.
- DS to store reference ID of all group the user belongs to.
- Latest reception time [ for recovery / delayed reception ]

We will also be maintaining a data structure [preferably a custom DS] that contains all the objects that belong to a given group chat.

**Message Types:**

We will be having a fixed set of messages that will be sent/received among the different users/clients that will be responsible for handling different functionalities.

- "ACK": This is an acknowledgement message sent among the sequencer and it's clients to acknowledge the fact that they have received the message from the other.
- "JOIN": When a particular client/user decides to join a chat room the client sends the message "JOIN" to the sequencer to provide him access to the chat room. There will be a timeout value (eg. 3 s) within which if he fails to receive an "ACK" message the client will terminate without connecting.
- "SEQ_ALIVE" : This message will be sent periodically by the existing clients/users to check if the sequencer is alive. If the client does not receive a response ("ACK") within a given time frame he will conclude that the sequencer is dead and will call for new leader election. Here we may modify the protocol to double check if the sequencer is really dead by pinging it twice.
- "USER_ALIVE": The sequencer will send this message to it's users/clients on a periodic basis. If the client fails to acknowledge this message in a specified time

frame then the sequencer concludes that the particular client is dead and it takes appropriate action to remove the client and inform others in the group chat about this.

- "MSG": This is the actual message that is to be sent.
- "STATUS": This is the message broadcasted by the sequencer to all it's clients/users informing each client about its peers. The updated list of objects currently in the group chat will be broadcasted each time a new user/client joins the chat. This list will be sent as a successor to this message.
- "QUIT": This is a message sent to the sequencer by a particular client who would want to exit the chat room. The sequencer acknowledges this message and disconnects that user and in the meanwhile informs other members about this update.
- "ELECT":It is the message sent out by a client to other users/clients on realizing that the sequencer is dead.
- "LEADER": It is the message sent by the newly elected leader to the other members to inform them that he is the new leader.

While initiating the chat by invoking the "dchat" program there will be three possible scenarios:

- ❖ Starting a new chat group where in that particular client/user will become the sequencer. He will be responsible in synchronizing (maintaining the order) the messages among the other members of the group, providing access to new users to join the chat, keeping track of users leaving the chat etc. He will have his own IP and will listen on a particular port number for messages (as listed above) to perform appropriate actions.
- ❖ Second case is when there is only one user connected to the chat group and another user tries to connect to the chat group. He will do this using the sequencer IP address and port number. The sequencer in the meantime will keep a track of the connected users' IP address as well as their port number since any user can connect to the chat group using any of the connected clients' IP address and port number. When a new user joins the chat, the sequencer will send a "STATUS" message to all the members informing them about the new user.
- ❖ Lastly if a client/user tries to connect by providing a IP/Port where there is no client/user listening, the program will fail and return an error message indicating to the user that he has entered an invalid IP/Port.

**Checking if Client is alive:** The sequencer maintains a queue to maintain the order of messages it broadcasts to its members. When it does this broadcast (rather multicast to its select group of member nodes) it expects an "ACK" message from the clients. If it does not receive the "ACK" message the sequencer uses the "USER_ALIVE" message to ping to that current client to check if he is alive. This way the sequencer is able to determine the liveness of its clients.

**Ordering of messages by the sequencer:**
The order of messages is decided by the sequencer. So based on which message arrives first at the sequencer that will be sent out first and the clients will receive it in that order. In order to ensure that the order of messages is maintained each message that arrives at the

sequencer will be given a sequence number. This sequence number will determine the order in which messages will be received at the other clients/users. This is done because in UDP packets might get lost. So if a packet gets dropped, the messages should still be displayed in order.

**Exiting the Chat:**
In order to leave the chat the user/client needs to send the appropriate “LEAVE” message to the sequencer. This sequencer will take appropriate action and terminate that user/client. Incase the client sends a “LEAVE” message and it does not receive an “ACK” for it. The sequencer might have ceased functionality then the client will wait for a specified timeout period after which it will automatically terminate itself. The new leader will take note of this.

**Leader Election:** When a user/client transmits a message, it will receive an “ACK” from the sequencer to indicate that the sequencer has received its message. The client wait for the “ACK” to be returned to it from the sequencer. Apart from this message the sequencer will wait for the transmitted message to come back to itself within a specific time frame. The user will wait for a timeout period for the sent message. If the user/client does not receive the message within the timeout period, it will check the status of the sequencer using “SEQ_ALIVE”. If both of these methods fail it realizes that the sequencer is dead, then this particular client will start an election.

The election procedure is as follows:

- The particular client (who realizes the sequencer is dead) will first remove the dead sequencer from the list of objects.
- This particular client will send the “ELECT” message to the rest of the clients connected in that group
- Each client then realizes that the sequencer is dead and it’s time for a new election.
- Each client is aware of all the clients connected in that group (as the data structure containing all objects in a group chat) are shared with all users.
- Each client will compare its port number with the port numbers of the rest of the members and the client having the highest port number decides to be the leader.
- This new leader then transmits “LEADER” message to the rest of the clients and waits for an ACK from them.

**Recovery:**
For the purpose of recovery, we plan to maintain a file in the server side which logs all activities. Each line of the file will contain one unique resource identifier which can be looked up to to find out the exact group the chat belongs and timestamp of the entry and the message itself. When a client crashes or delays reception, then when the client comes online for recovery or after some finite time, the last online timestamp is checked and all the unreceived messages from the file is sent to client.

Periodically as part of maintenance, all the lines in the recovery file which has timestamps earlier than the earliest last available timestamp of any client is deleted and refreshed periodically.

**Message Structure**:
While joining the group, the name of the client, IP address and port number will be included in the message first, to get him/her registered in the group. The sequencer will acknowledge by displaying “Welcome to the group (name)”. Internally, the user will be allocated a reference ID (UIRI) and the user will be updated to the database with all the fields. The format of a message will be an instance of a struct that will have the following fields

- Message Type [ New joining/Leaving, / Acknowledgement / Checking if Alive / Election protocol related ]
- Message Specific Data depending on message type like Name, UIRI etc.
- Content of the message
- Time stamp of the message as sent from client side, using NTP.
- Once received on the server side, the receiving time stamp using NTP will be updated to the message and sent for further processing.

**Max size of the group**:
The max size size of the chat group, by consensus will be set to be 20 members.

**Thread Pool :**
As we mentioned above, the number of members that can be accommodated in each group is 20 members, We plan to use the concepts of thread pool where the number of threads available is 20, which will be allocated one each to each client that joins the group.

**Data structures**:
- A queue at the sequencer to maintain the order of messages to be delivered to the clients
- A list of objects (users/clients) maintained by each sequencer which contains the users present in that group chat.
- The reception and sending of messages will be via a struct.

**File :**
For the purpose of recovery, the transcript of the chats will be continuously written to a file for recovery purpose, the file maintenance will happen periodically, by clearing the old messages by checking timestamps of the messages vs last available timestamps of the clients.

**Multithreaded server:**
The server we are expecting to be a multithreaded server running one thread each for each group that is created on the server. Each group will have its own sequencer.