**Software Systems – Milestone 2**

There are three major components in the system. They are:

1. **Client** – Responsible for sending, and displaying the user messages. Also handles various scenarios such as acknowledgements, holding messages in a queue to be delivered to the application, handling joining of new clients, and starting the sequencer and election algorithm threads.
2. **Sequencer/leader** – Responsible for assigning global sequence numbers to messages and multicasting the messages to all clients. It is also responsible for ensuring ordering of messages from individual clients and on the global scale. It is also responsible for assigning client ids to new participants and maintaining information about all clients and checking whether the clients are alive or not.
3. **Election Algorithm** – Responsible for holding and electing a new leader. Also responsible for detecting the crash of the leader.

**Design Specifications**

**Client:**

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Client creating a new chat |
| **Actions**  **Performed** | On creating a new chat:   1. Client begins the sequencer/leader thread and its own election algorithm thread 2. Update isLeader flag to true 3. Waits to receive a broadcast from the sequencer announcing it is the leader, along with its IP Address and Port number 4. Update leader information (D1) 5. Send a request to sequencer, asking to update the client information that the sequencer has (M1, M2) 6. Wait for others to join |
| **Messages** | The following messages are sent/received:   1. M1 - REQUEST#my\_ip\_address#my\_port\_no 2. M2 - SUCCESS#client\_id / FAILURE |
| **Data Structures** | D1:  struct Leader{  char ip\_addr[MAXSIZE];  char port[MAXSIZE];  } |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Client joining an existing chat |
| **Actions**  **Performed** | On joining a chat:   1. Contact existing client (M3) 2. Receive leader information from client (M4) 3. Update leader information (D1) 4. Request sequencer to join chat (M1, M2) |
| **Messages** | 1. M3 – JOIN 2. M4 - JOINLEADER#Leader\_IPAddr#Leader\_Port |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Sending a message typed by user |
| **Actions**  **Performed** | To send a message:   1. Wait for user to type a message 2. Assign a Message Id to the message before sending it to the sequencer 3. Add message to end of Queue (D2) 4. Send message to sequencer (M5) 5. Wait for acknowledgement from sequencer that message is received (M6) |
| **Messages** | 1. M5 - MESSAGE#ClientID#MsgID#user\_message 2. M6 - SEQ#ACK#Msg\_id |
| **Data Structures** | D2:  struct node{  int msg\_id;  char message[MAXSIZE];  int acknowledged;  TAILQ\_ENTRY(node) entries;  }  (stored in a TAILQ) |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Handling a multicast message from the sequencer |
| **Actions**  **Performed** | To display a multicast message:   1. Receive the message from the sequencer (M7) 2. Check that it is the next message that needs to be displayed (global sequence id is the next one that the client is expecting) 3. Send an acknowledgement to the sequencer (M8)   Display message |
| **Messages** | 1. M7 - MSG#GlobalSeqID#ClientID#MsgID#Message 2. M8 - ACK#client\_id#global\_seq\_id |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Receiving an updated clients list from the sequencer. When a new client joins or an existing one leaves, the sequencer broadcasts an updated clients list to all existing clients. |
| **Actions**  **Performed** | On receiving an updated client list:   1. Receive a serialized version of the client list from the sequencer (M9) 2. Detokenize, deserialize, and re-initialize the client\_list maintained by each client (D3) |
| **Messages** | 1. M9 - SEQ#CLIENT#INFO#num\_clients#Ip#Port#ClientID#ClientName#Ip#Port#ClientID#ClientName#... |
| **Data Structures** | D3:  struct client{  char ip[MAXSIZE];  int port;  int client\_id;  char name[MAXSIZE];  }  An array of struct client (client\_list) to hold all clients’ information (Maximum of 15 clients) |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | The sequencer acknowledges the receipt of a message sent by the client to be multicast |
| **Actions**  **Performed** | On receiving an acknowledgement:   1. Mark the acknowledged field of the message in the queue to 1 |
| **Messages** | M6 |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | The sequencer notifies the client it is safe to remove a message from its sending queue as all clients have acknowledged the receipt of the message |
| **Actions**  **Performed** | 1. Receive the removal message from the sequencer (M10) 2. Look for message in the sending queue (D3) 3. Remove message from queue (D3) |
| **Messages** | 1. M10 - SEQ#REM#Msg\_id |
| **Data Structures** | D3 |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | All clients send the last message they accepted from the sequencer (due to a new sequencer coming online, it needs to know where the clients are with respect to the next message that needs to be broadcast) |
| **Actions**  **Performed** | 1. Receive a request from sequencer to send the last message the client processed from the sequencer (M11) 2. Send the message to the sequencer (M12) |
| **Messages** | 1. M11 – SEQ#SEQNO 2. M12 – SEQNO#GlobalSeqID#ClientID#MsgID#Message |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | All clients send the sequencer all messages in their message queue (once a new sequencer comes online, it builds up the message queue it had before crashing) |
| **Actions**  **Performed** | 1. Receive a request from the sequencer to send the whole message queue to the sequencer (M13) 2. Loop through the message queue, sending all messages to the sequencer in the same format as before (M5) 3. Receive acknowledgements for each message |
| **Messages** | 1. M13 – SEQ#SENDALL 2. M5 |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | The election algorithm of some client decides that there needs to be an election in place |
| **Actions**  **Performed** | 1. Receive a broadcast from some election algorithm notifying the client that an election is taking place (M14) 2. Check if you are the leader 3. If you are the leader, some client’s election algorithm made a mistake, reply to that EA with a message indicating you are still alive (M15) 4. Stop working and wait for new leader to be elected before resuming normal operations |
| **Messages** | 1. M14 – ELECTION 2. M15 – CANCEL |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | The election is cancelled because some EA made an incorrect inference that the sequencer had crashed |
| **Actions**  **Performed** | 1. Receive a broadcast from the EA that initiated the election notifying that the election has been cancelled (M16) 2. Resume normal operations |
| **Messages** | 1. M16 – ELECTIONCANCEL |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | The client is chosen to be the new leader |
| **Actions**  **Performed** | 1. Receive a message from the election algorithm notifying the client that it is the new leader (M17) 2. Start the sequencer process 3. Update the isLeader flag to true 4. Resume normal operations |
| **Messages** | 1. M17 – LEADER |
| **Developer** | Dhruvil (dhruvils) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | When some other client has been chosen leader and the election process is complete, the new leader notifies all clients that the election is complete |
| **Actions**  **Performed** | 1. Receive a broadcast from the leader about its details (M18) 2. Resume normal operations |
| **Messages** | 1. M18 – SEQ#EA#leader\_ip#leader\_port |
| **Developer** | Dhruvil (dhruvils) |

**Election Algorithm:**

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Checking if the sequencer is active. |
| **Actions**  **Performed** | Once the election algorithm starts running:   1. Sends a heartbeat message to the sequencer (EM1). 2. Waits for a reply from the sequencer until timeout (EM2) |
| **Messages** | 1. EM1 - “PING#client\_id” 2. EM2 - “I AM ALIVE” |
| **Data Structures** | D3:  struct client{  char ip[MAXSIZE];  int port;  int client\_id;  char name[MAXSIZE];  }  An array of struct client (client\_list) to hold all clients’ information (Maximum of 15 clients) |
| **Developer** | Deepti (pdeepti) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | On reaching timeout when waiting for response from sequencer |
| **Actions**  **Performed** | If it doesn’t receive a response from the sequencer within timeout (EM2):   1. It sends another message to the sequencer to confirm that it is in fact, inactive (EM3). 2. Waits for a response from the sequencer until timeout (EM2). 3. If timeout is reached again it holds an election. |
| **Messages** | 1. EM3 - “ARE YOU ALIVE?” |
| **Developer** | Deepti (pdeepti) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Starting an election when a timeout is reached waiting for sequencer’s response. |
| **Actions**  **Performed** | If a timeout is reached while waiting for a response from sequencer:   1. Sends a message to all clients to wait for the new sequencer (EM4). 2. Sends a message to all election algorithms to tell them to stop pinging the client and participate in the election (EM4). 3. Sends a message to all election algorithms which have a higher ID than it (EM5). 4. Waits for their response until timeout. (EM6). |
| **Messages** | 1. EM4 - “ELECTION” 2. EM5 - “CLIENT\_ID#client\_id” 3. EM6 - “OK” |
| **Data Structures** | D3 |
| **Developer** | Deepti (pdeepti) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Election algorithms receive message telling them to stop pinging the sequencer and participate in election (EM4). |
| **Actions**  **Performed** | On receiving the message EM4 the election algorithm:   1. Stops pinging the sequencer and waits for a message from an election algorithm requesting to be the new leader (EM5) or message telling it the election is complete (EM7). 2. Sends a message to the new leader client, informing it that it has won the election (EM8) |
| **Messages** | 1. EM7 - “I AM LEADER#client\_id” 2. EM8 - “LEADER” |
| **Data Structures** | D3 |
| **Developer** | Deepti (pdeepti) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | An election algorithm receives a request from another election to be the leader. |
| **Actions**  **Performed** | On receiving request (EM5):   1. Sends a response back so it can hold its own election (EM6). 2. Sends all election algorithms with a higher id that itself, a request to be leader (Just as the previous election algorithm had done). 3. Waits for response (EM6) |
| **Messages** | 1. EM5 2. EM6 |
| **Data Structures** | D3 |
| **Developer** | Deepti (pdeepti) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | When a timeout is reached while waiting for responses from the other election algorithms (with a higher id) during an election, the current election algorithm then makes its client the new leader. |
| **Actions**  **Performed** | If a timeout is reached while waiting for the response:   1. Sends a message to all the election algorithms broadcasting its client as the new leader (EM7). 2. Send a message to the new sequencer informing it that it has won the election (EM9). |
| **Messages** | 1. EM7 - “I AM LEADER#client\_id” 2. EM9 - “LEADER” |
| **Data Structures** | D3 |
| **Developer** | Deepti (pdeepti) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Receives a message from the sequencer requesting its status. |
| **Actions**  **Performed** | On receiving request message from the sequencer (EM10):   1. Sends a message to the sequencer telling it that it is still alive (EM2). |
| **Messages** | 1. EM10 - “SEQ#PING#STATUS” 2. EM2 |
| **Developer** | Deepti (pdeepti) |

|  |  |
| --- | --- |
| **Specification** | **Description** |
| **Scenario** | Canceling an election in case it was initiated due to lost messages and not the actual crash of the sequencer. |
| **Actions**  **Performed** | If the election receives a message from the current leader client telling it stop the election (EM11):   1. Stops the election and goes back to pinging the sequencer. 2. Sends message to all the other election algorithms also to stop the election and go back to pinging the sequencer (EM12). 3. Sends a message to the leader client telling it that the election has been canceled (EM12). |
| **Messages** | 1. EM11 - “CANCEL” 2. EM12 - “ELECTIONCANCEL” |
| **Developer** | Deepti (pdeepti) |

**Sequencer/leader:**

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | New client requesting to join the chat |
| Actions Performed | 1. Receive message (SM1) with the IP and port number of the new client. 2. If the max. limit of participants is not reached, assign the new client an unique client id by which it will be identified. 3. Update client information in the client list which is of the type client structure (SD1). 4. Send back to the requesting client a success or failure message depending on whether it was assigned a unique id or not. (SM2) 5. Multicast the updated client list to all the existing clients (SM3). |
| Messages | The following messages are sent/received:   1. SM1 - REQUEST#my\_ip\_address#my\_port\_no 2. SM2 - SUCCESS#client\_id / FAILURE 3. SM3 - SEQ#CLIENT#INFO#num\_clients#Ip#Port#ClientID#ClientName#Ip#Port#ClientID#ClientName#... |
| Data Structures | SD1:  struct client{  char ip[BUFLEN];  char name[BUFLEN];  int port;  int client\_id;  int last\_msg\_id;  int leader;  }; |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | Client leaves the chat/Client crashes |
| Actions Performed | On not receiving a “Ping” (SM4) from the Election Algorithm run by a client within the timeout period-   1. Send a message to the EA requesting for its status. (SM5) 2. If client does not reply with an alive status (SM6), remove the client record from the client list. (SD1) 3. Multicast the updated client list to all the existing clients (SM3). |
| Messages | The following messages are sent/received:   1. SM4 - PING#Client\_id 2. SM5 - SEQ#PING#STATUS 3. SM6 - I AM ALIVE 4. SM3 - SEQ#CLIENT#INFO#num\_clients#Ip#Port#ClientID#ClientName#Ip#Port#ClientID#ClientName#... |
| Data Structures | SD1:  struct client{  char ip[BUFLEN];  char name[BUFLEN];  int port;  int client\_id;  int last\_msg\_id;  int leader;  }; |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | The Election Algorithm of each client pings the sequencer |
| Action Performed | On receiving a ping from the Election Algorithm of a client (SM4):   1. Send back message saying the sequencer is still alive. (SM7) |
| Messages | 1. SM4 - PING#Client\_id 2. SM7 - I AM ALIVE |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | A new sequencer comes online (First client to start the chat/Winning election after the old sequencer crashes). |
| Action Performed | 1. Broadcasts its own IP address and Port Number to all the existing clients in the chat (SM8) |
| Messages | 1. SM8 – SEQ#EA#Leader\_IP#Leader\_Port |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | Sequencer receives message typed by a client |
| Actions Performed | On receiving message typed by the client (SM9)-   1. Assign it a global sequence ID. 2. Store it in the message queue (SD2) 3. Sends back an acknowledgement to the client about the receipt of the message (SM10) |
| Messages | The following messages are sent/received:  1. SM9 - MESSAGE#ClientID#MsgID#user\_message   1. SM10 – SEQ#ACK#MsgID |
| Data Structures | SD2:  struct message{  int seq\_id;  int client\_id;  int msg\_id;  int msg[BUFLEN];  int acknowledgement\_vector[MAX];    };  This is stored as a TAILQ. |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | Sequencer multicasts the message from its message queue to each client |
| Actions Performed | 1. The sequencer checks the message on top of its message queue (SD2). 2. If the top message’s MsgID is the next in line to be sent  * multicast this message to all the clients (SM11) * update that client’s last\_msg\_id to the message id of the message just multicasted (SD1)  1. If the message is not next in line  * Traverse through the message queue and see if the message to be sent next exists in the queue. If it does, multicast this message to all the clients and update the last\_msg\_id of that client to the message id of the message just multicasted (SD1). * Push the message on top of the queue to the bottom. |
| Messages | The following messages are sent/received:  1. SM11 - MSG#GlobalSeqID#ClientID#MsgID#Message |
| Data Structures | SD1:  struct client{  char ip[BUFLEN];  char name[BUFLEN];  int port;  int client\_id;  int last\_msg\_id;  int leader;  };  SD2:  struct message{  int seq\_id;  int client\_id;  int msg\_id;  int msg[BUFLEN];  int acknowledgement\_vector[MAX];    };  This is stored as a TAILQ. |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | Sequencer receives acknowledgement of receipt of message from client |
| Actions Performed | On receiving acknowledgment of receipt of message from a client (SM12):   1. Update the acknowledgement vector of that message to reflect receipt by that client. |
| Messages | The following messages are sent/received:  1. SM12 - ACK#client\_id#global\_seq\_id |
| Data Structures | SD2:  struct message{  int seq\_id;  int client\_id;  int msg\_id;  int msg[BUFLEN];  int acknowledgement\_vector[MAX];    };  This is stored as a TAILQ. |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | Sequencer removes a message from its message queue |
| Actions Performed | Once a message has been acknowledged as received by all clients   1. Send a message to the client that sent this particular message asking it to remove it from its local queue. (SM13) 2. Remove this message from the message queue. (SD2) |
| Messages | The following messages are sent/received:  1. SM13 - SEQ#REM#Msg\_id |
| Data Structures | SD2:  struct message{  int seq\_id;  int client\_id;  int msg\_id;  int msg[BUFLEN];  int acknowledgement\_vector[MAX];    };  This is stored as a TAILQ. |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | On election as new sequencer, request clients to send the last message received by them. |
| Actions Performed | 1. Send request to each client to send the last message received by them. (SM14) 2. On receipt of the last message (SM15) from each client, check if the global sequence ID of each of them.  * If they are the same, set the next global sequence ID to be assigned to the global sequence ID just received from the clients. * If not, send the message with the higher global sequence ID to all the clients that haven’t received it yet. Put this message on the message queue (SD2). Set the next global sequence ID to be assigned to the global sequence ID of the message just put on the queue. |
| Messages | The following messages are sent/received:   1. SM14 - SEQ#SEQNO 2. SM15 – SEQNO#GlobalSeqID#ClientID#MsgID#Message |
| Data Structures | SD2:  struct message{  int seq\_id;  int client\_id;  int msg\_id;  int msg[BUFLEN];  int acknowledgement\_vector[MAX];    };  This is stored as a TAILQ. |
| Developer | Anwesha (anwesha) |

|  |  |
| --- | --- |
| Specification | Description |
| Scenario | On election as new sequencer, request clients to send all the messages in their local message queue. |
| Actions Performed | 1. Send request to each client to send all the messages from their local message queue. (SM16) 2. Store the messages received to the message queue. (SD2) 3. Sends back an acknowledgement to the client about the receipt of the message (SM10) |
| Messages | The following messages are sent/received:   1. SM16 - SEQ#SENDALL 2. SM9 - MESSAGE#ClientID#MsgID#user\_message 3. SM10 – SEQ#ACK#MsgID |
| Data Structures | SD2:  struct message{  int seq\_id;  int client\_id;  int msg\_id;  int msg[BUFLEN];  int acknowledgement\_vector[MAX];    };  This is stored as a TAILQ. |
| Developer | Anwesha (anwesha) |