SOFTWARE SYSTEMS PROJECT 3

USER MANUAL

TEAM MEMBERS:
Anwesha Das (anwesha)
Deepti Panuganti (pdeepti)
Dhruvil Shah (dhruvils)

The project contains 3 files.

1. dchat.c – This is the code file for the client chat application. It is responsible for sending, and displaying user messages. It also handles various scenarios such as holding messages in queue to be delivered to the application, handling joining of new clients, and starting the sequencer and election algorithm threads. This file also contains the functionality to handle electing a new leader when the old one crashes or leaves.
2. seq.c – This is the code file for the sequencer function. It is responsible for assigning global sequence numbers to messages and multicasting them to all the clients. It is responsible for assigning clients with client ids, detecting when a client crashes or leaves, and ordering of the messages.
3. Makefile – This file is responsible for compiling the above two .c files.

Compiling

- Run the following command in the console to compile the code files into appropriate object files:

  make

- To remove existing object files, run the following command:

  make clean

Running the client

There are two ways to run the client:

1. Start a new chat:

   dchat USER

2. Join an existing chat:

   dchat USER ip-address port

***** Please note the port here is fixed at 1705.**

***** Please note the change in format for joining an existing chat. Instead of dchat USER ip-address:port, we decided to pass the port as another command line argument.**

***** Also, please ensure that both the object files are placed in the same location, and kindly do not rename any of the object files.**

SOFTWARE SYSTEMS PROJECT 3

FINAL REPORT

TEAM MEMBERS:
Anwesha Das (anwesha)
Deepti Panuganti (pdeepti)
Dhruvil Shah (dhruvils)

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.

ASSUMPTIONS:

1. Maximum number of participants in the chat : 15
2. Maximum size of message : 1024
3. Client Port : 1705
4. Election Port : 8174
5. Sequencer Port : 5678
6. Sequencer Heartbeat Port : 5679

DESIGN DECISIONS:

- If a sequencer does not respond to heartbeat for 2 seconds, the election algorithm (EA) pings it again and waits for another 2 seconds for a response. If it does not receive any in this time, it holds an election.
- In case the EA falsely identifies a sequencer crash, the sequencer can still send a “CANCEL” message to stop the election.
- The sequencer checks every 2 seconds whether it has received enough heartbeats from each client. If it receives less than 5 heartbeats within this time, it declares the client as crashed and removes it.

DESIGN SPECIFICATIONS:

Client:

Specification	Description
Scenario	Client creating a new chat
Actions Performed	On creating a new chat: 1. Client begins the sequencer/leader process 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#global_seq_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 message 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; int global_id; char message[MAXSIZE]; int client_id; 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. Add the message to the holdback queue (D2) 3. Check that it is the next message that needs to be displayed (global sequence id is the next one that the client is expecting) 4. Display message 5. Send an acknowledgement to the sequencer (M8)
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#lastmsgid#Ip#Port#ClientID#ClientName#lastmsgid...

Data Structures	D3:
	struct client{
	char ip[MAXSIZE];
	int port;
	int client_id;
	int last_msg_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 message queue and the holdback queue as all clients have acknowledged the receipt of the message
Actions Performed	1. Receive the removal message from the sequencer (M10) (M11) 2. Look for message in the message/holdback queue (D3) 3. Remove message from the message/holdback queue (D3)
Messages	1. M10 - SEQ#REM#Msg_id 2. M11 – SEQ#REMHB#Global_id
Data Structures	D3
Developer	Dhruvil (dhruvils)

Specification	Description
Scenario	All clients send their holdback queues to 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). The holdback queue contains messages that the clients have displayed, thus accepted, but haven't been acknowledged yet by all the clients.
Actions Performed	1. Receive a request from sequencer to send the holdback queue (M12) 2. Send the holdback queue to the sequencer (M13)
Messages	1. M12 – SEQ#HB 2. M13 – HB#ClientID#NextGlobalSeqID#TotalMsgs#GlobalSeqID#ClientID#MsgID#Msg#...
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 (M14) 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. M14 – SEQ#SENDALL 2. M5 - MESSAGE#ClientID#MsgID#user_message
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 (M15) 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 (M16) 4. Stop working and wait for new leader to be elected before resuming normal operations
Messages	1. M15 – ELECTION 2. M16 – 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 (M17) 2. Resume normal operations
Messages	3. M17 – 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 (M18) 2. Start the sequencer process 3. Update the isLeader flag to true 4. Wait to send sequencer the contents of the holdback queue (M12)(M13) 5. Wait to send the sequencer the contents of the message queue (M14)(M5) 6. Resume normal operations
Messages	1. M18 – LEADER 2. M12, M13, M14, M5
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 (M20) 2. Resume normal operations
Messages	1. M19 – SEQ#EA#leader_ip#leader_port
Developer	Dhruvil (dhruvils)

Election Algorithm:

Specification	Description
Scenario	Checking if the sequencer is active (This is also used by the sequencer to detect the crash of a client).
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 to hold all clients' information (Max 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, in-active (EM2). 2. Waits for a response from the sequencer until timeout (EM2). 3. If timeout is reached again it holds an election.
Messages	1. EM1
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 which have a higher ID than it (EM5). 3. Waits for their response until timeout. (EM6).
Messages	5. EM4 - "ELECTION" 6. EM5 - "CLIENT_ID#client_id" 7. EM6 - "OK"
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 (EM6)back so it can hold its own election if it hasn't already detected the crash of the sequencer. Otherwise, it simply waits for its own responses (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 winning client 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	After an election is completed, waits for the new sequencer to come up before resuming pinging.
Actions Performed	On receiving the new end of election message from the winning election algorithm (EM7), 1. The elections then wait until the new sequencer is up. 2. When the new sequencer is up, they receive a message (EM10), with the sequencer’s IP Address. Then, using this IP Address the elections resume pinging the sequencer as before (EM1).
Messages	1. EM10 - “SEQ#EA#Leader IP#Leader Port” 2. EM7 3. EM1
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)

Specification	Description
Scenario	Canceling an election when an election algorithm that started the election cancels.
Actions Performed	If the election receives a message from the election algorithm that detected the crash (EM12): 1. Stops the election and goes back to pinging the sequencer.
Messages	3. EM12 - “ELECTIONCANCEL”
Developer	Deepti (pdeepti)

Sequencer/leader:

Specification	Description
Scenario	New client requesting to join the chat
Actions Performed	Receive message (SM1) with the IP and port number of the new client. If the max. limit of participants is not reached, assign the new client an unique client id by which it will be identified. Update client information in the client list which is of the type client structure (SD1). Send back to the requesting client a success or failure message depending on whether it was assigned a unique id or not. (SM2) Multicast the updated client list to all the existing clients (SM3).
Messages	The following messages are sent/received: SM1 - REQUEST#my_ip_address#my_port_no SM2 - SUCCESS#client_id / FAILURE SM3 - SEQ#CLIENT#INFO#num_clients#Ip#Port#ClientID#ClientName#Ip#Port#ClientID#ClientName#LastMsgID...

Data Structures	SD1:
	struct client{
	char ip[BUFLEN];
	char name[BUFLEN];
	int port;
	int client_id;
	int last_msg_id;
	int leader;
	int counter;
	double time_of_join;
	};

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. Remove the client record from the client list. (SD1) 2. Multicast the updated client list to all the existing clients (SM3).
Messages	The following messages are sent/received: 1. SM4 - PING#Client_id 2. SM3 - SEQ#CLIENT#INFO#num_clients#Ip#Port#ClientID#ClientName#Ip#Port#ClientID#ClientName#LastMsgID#...
Data Structures	SD1
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. Store it in the message queue (SD2) 2. 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 2. SM10 – SEQ#ACK#MsgID

Data Structures	SD2:
	struct message{
	int seq_id;
	int client_id;
	int msg_id;
	int msg[BUFLEN];
	int counter;
	int sent;
	};
	
	This is stored as a TAILQ linked list.

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 that message has already been sent out, keep looking through the queue until the next message hasn't been marked as sent. 3. If the top message's MsgID is the next in line to be sent • Assign this message the next global sequence id • 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) • mark this message as sent 4. 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, assign it the next global sequence id, 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). Also mark this message as sent. • Push the message on top of the queue to the bottom if it hasn't been marked as sent.
Messages	The following messages are sent/received: 1. SM11 - MSG#GlobalSeqID#ClientID#MsgID#Message
Data Structures	SD1 SD2
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 counter for each message by decrementing it.
Messages	The following messages are sent/received: 1. SM12 - ACK#client_id#global_seq_id
Data Structures	SD2
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. Multicast a message to all the clients to remove this message from the holdback queue. (SM14) 3. Remove this message from the message queue. (SD2)
Messages	The following messages are sent/received: 1. SM13 - SEQ#REM#Msg_id 2. SM14 – SEQ#REMHB#GlobalSeqID
Data Structures	SD2
Developer	Anwesha (anwesha)

Specification	Description
Scenario	On election as new sequencer, request clients to send their last message hold-back queue.
Actions Performed	1. Send request to each client to send their holdback queue. (SM15) 2. On receipt of the holdback queues (SM16) from each client: • Update the global sequence ID to the next global sequence ID the client expects • Builds up its own message queue by the messages in the hold-back queue. Each message sent in the queue contains the global sequence ID which it was assigned before, the Client ID of the client that originally sent it, the message ID of the message, and the message.
Messages	The following messages are sent/received: 1. SM15 - SEQ#HB 2. SM16 – HB#ClientID#NextGlobalSeqID#TotalMsgs#GlobalSeqID#ClientID#MsgID# Msg#..
Data Structures	SD2
Developer	Anwesha (anwesha)

Specification	Description
Scenario	Once the holdback queues have been processed and all messages have been sent out appropriately, request clients to start sending messages in their local message queues.
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
Developer	Anwesha (anwesha)

Specification	Description
Scenario	On receipt of lost message request.
Actions Performed	1. Loop through the message queue (SD2) and find the message being requested. (SM17) 2. Send back this message to the client that requested it. (SM11)
Messages	The following messages are sent/received: 4. SM11- MSG#GlobalSeqID#ClientID#MsgID#Message 5. SM17 – LOST#GlobalSeqID
Data Structures	SD2
Developer	Anwesha (anwesha)

PROTOCOL ANALYSIS

Since our system is built over UDP, we dealt with the possibility of loss, reordering and duplication of messages.

LOST MESSAGES

1. Every time the client receives a message –
    - It places it in the holdback queue.
    - Next it checks if the next message to be delivered is in the holdback queue. If not, it asks the sequencer for the lost message.

DUPLICATE MESSAGES

2. The client only delivers a message from the holdback queue if its **global sequence ID** is the next one to be delivered. Even if it ever receives a message twice, it will never deliver it again.
3. If the sequencer receives a message twice, it will never multicast it twice. This is because each client assigns a message sequence ID to its own messages and the sequencer only multicasts a message if it is the next one to be sent from that client. Thus the **client's message sequence ID** helps in this scenario.

TOTAL ORDERING & FIFO ORDERING

4. We implement **total ordering** by using global sequence IDs. The sequencer is responsible for assigning each message that it receives, a global sequence ID in strictly increasing order.
5. Even if a sequencer crashes, the new sequencer will start multicasting messages using the next global sequence ID that every client is expecting.
6. This ensures total ordering because all clients will display messages in the same order (increasing global sequence ID) as every other client.
7. We further implement **FIFO ordering** by assigning a local sequence ID to every message from a particular client in strictly increasing order.
8. This ensures that the first message sent by a particular client is the first message received by every client.

HANDLING SYSTEM CRASHES

9. We implemented a heartbeat pinging system that detects crashes.
10. Client crashes are detected by the Sequencer.
11. Sequencer crashes are detected by the Election Algorithm, and then a new Leader (Sequencer) is elected.

LIMITATIONS

12. Users cannot join or leave while an election is being held to select the new leader/sequencer.

13. Since the port number for the client is fixed as per our assumption, there can only be one client per system.

14. Since we are assigning global sequence IDs and local message IDs in a strictly increasing order, there is a ceiling on the number of messages that can be sent in the chat system (MAXINT value of the system).