ECEN 602 Machine Problem 2
TCP Simple Broadcast Chat Server and Client

Team 11:

1. Dhanraj Murali – UIN:734003894
2. Swetha Reddy Sangem – UIN: 433003591

Implementation Roles:

1. Dhanraj – Server Implementation
2. Swetha – Client Implementation

Both were involved in the ChatGPT part of the submission. Same has been executed and tested with Hera servers.

Submission:

1. Submission 1 – Source Code/Make file written by the team.
2. Submission 2 – Source Code written by team and optimized by ChatGPT/Makefile.
3. Submission 3 – Source Code/Make file generated by ChatGPT.
4. README.pdf
5. Test case screenshots as PDF

Execution:

Run the following commands in order,

Open 2 terminals,

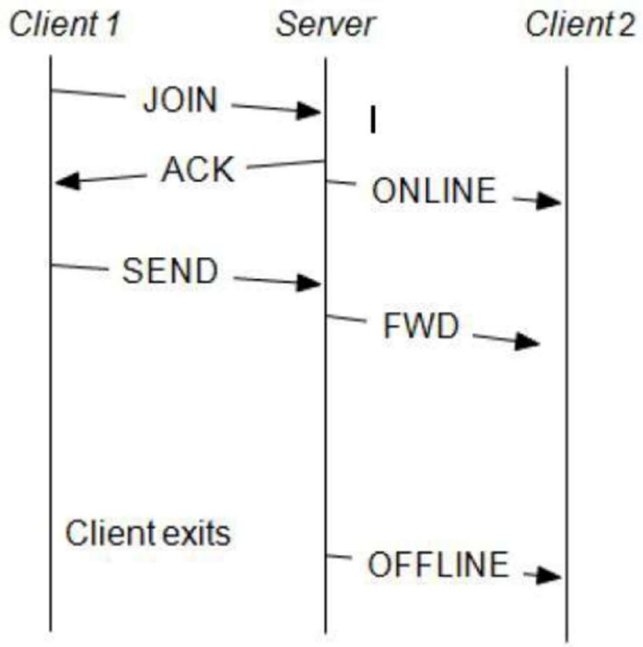
1. Terminal 1: make clean; make
2. Terminal 1: ./server <port_no> <max_clients>
3. Terminal 2: ./client <clientname> <127.0.0.1> <port_no>

Design of Simple Broadcast Chat Server and Client:

The Simple Broadcast Chat Protocol (SBCP) is a protocol that allows clients to join and leave a global chat session, view members of the session, and send and receive messages. An instance of the server provides a single chat room, which can only handle a finite number of clients.

1. The server starts listening on the port entered.
2. The Client1 sends a JOIN request to the server and server acknowledges it by sending an ACK with the client count and users currently online.
3. The server now sends ACK to the newly joined client and ONLINE upon successful join to the other clients by mentioning the username of the client currently joined.
4. The server sends an NAK with reason if the join is not successful. This happens when the username already exists or if the max number of clients is reached.

5. The client sends the message from command line using SEND by forming a message. The server takes the message from client and forwards to other clients.
6. When a client is disconnected from the chat, OFFLINE is sent from server to other clients with username.
7. If client is IDLE for 10 seconds, an IDLE is sent from the client to the server without any attributes. Server sends to the other client with username of the client who is IDLE by forming another message.



Code Description – Server/Client:

1. **write(sockfd, client_message, sizeof(struct SBCP_Message))** –
   It is used to send data from the client to the server.
   In this code, it's primarily used to send various types of messages to the server, such as JOIN, SEND, and IDLE messages.
   The sockfd is the file descriptor of the socket connected to the server.
   The client_message is a pointer to a custom structure (struct SBCP_Message) that contains the data to be sent.
   The `sizeof(struct SBCP_Message)` specifies the size of the data to send.

2. **read(sockfd, server_message, sizeof(struct SBCP_Message))**
   It is used to read data from the server.
   It reads data from the server and stores it in the server_message structure.
   The sockfd is the file descriptor of the socket connected to the server.

The server_message is a pointer to a custom structure (struct SBCP_Message) where the received data is stored.
The sizeof(struct SBCP_Message) specifies the maximum amount of data to read.

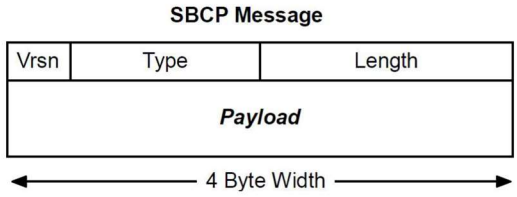
3. **select(sockfd + 1, &readfds, NULL, NULL, &timeout)**
   It is used to monitor multiple file descriptors for read readiness and to introduce a timeout for the client's interactions.
   sockfd + 1 specifies the highest file descriptor number to check (plus one).
   readfds is a set of file descriptors to monitor for read operations (in this case, the socket descriptor).
   timeout is a struct timeval that sets a timeout for the `select()` operation.
   It is used to wait for activity on the socket for a specified time.
   If select() returns 0, it means a timeout occurred (indicating client IDLE state).

4. **struct timeval** is a structure in C that represents a time interval. It has two fields:
   tv_sec: This field represents the number of seconds.
   tv_usec: This field represents the number of microseconds (1/1,000,000th of a second).

   The timeout structure is then passed as an argument to the select() function:

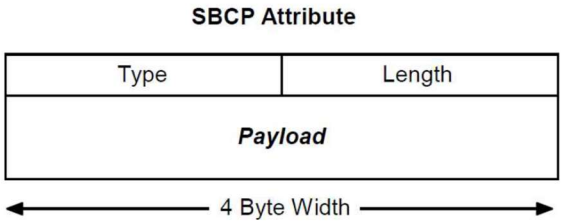
Structure of SBCP Message:

The structure of SBCP Message is implemented using struct in the sbcp.h file as show in figure.



Structure of SBCP Attribute:

The structure of SBCP attribute is implemented using struct in the sbcp.h file as show in figure.

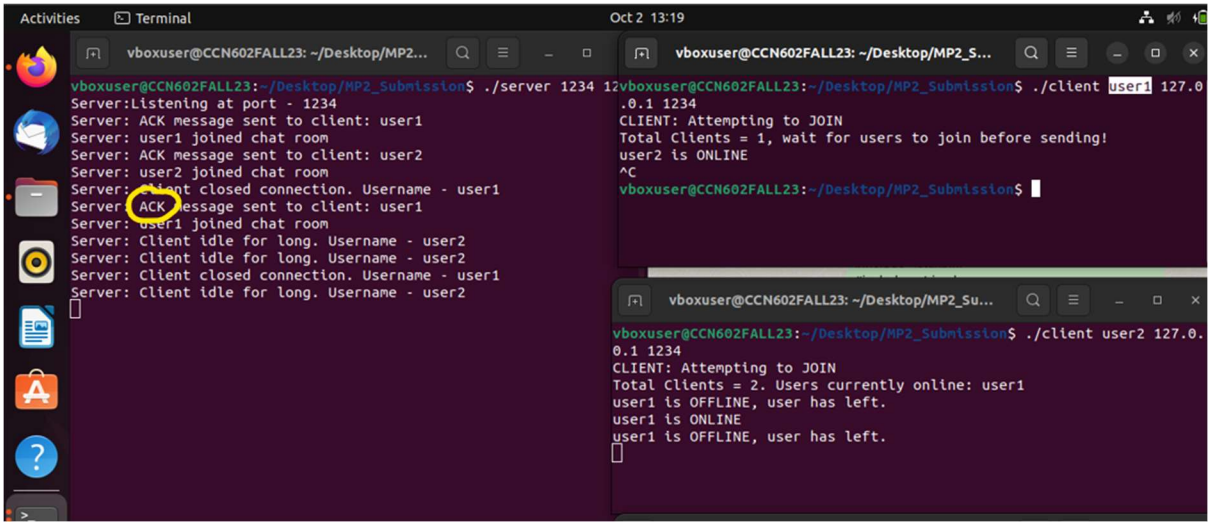


Header types for Bonus features:

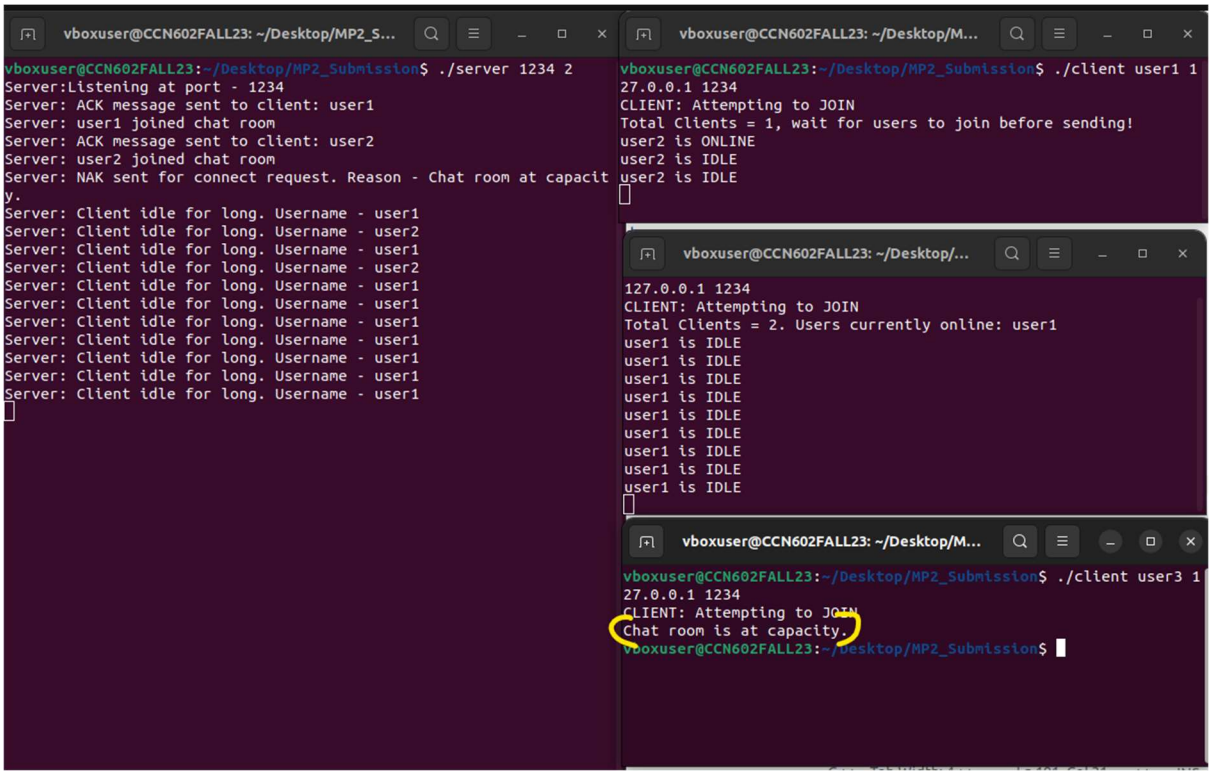
NAME	Type	Description
ACK	7	server sends to client to confirm the JOIN request
NAK	5	server sends to client to reject a request (JOIN, etc)
ONLINE	8	server sends to client indicating arrival of a chat participant
OFFLINE	6	server sends to client indicating departure of a chat participant
IDLE	9	Client sends to server indicating that it has been idle. Server sends to clients indicating the username which is idle.

Bonus Features:

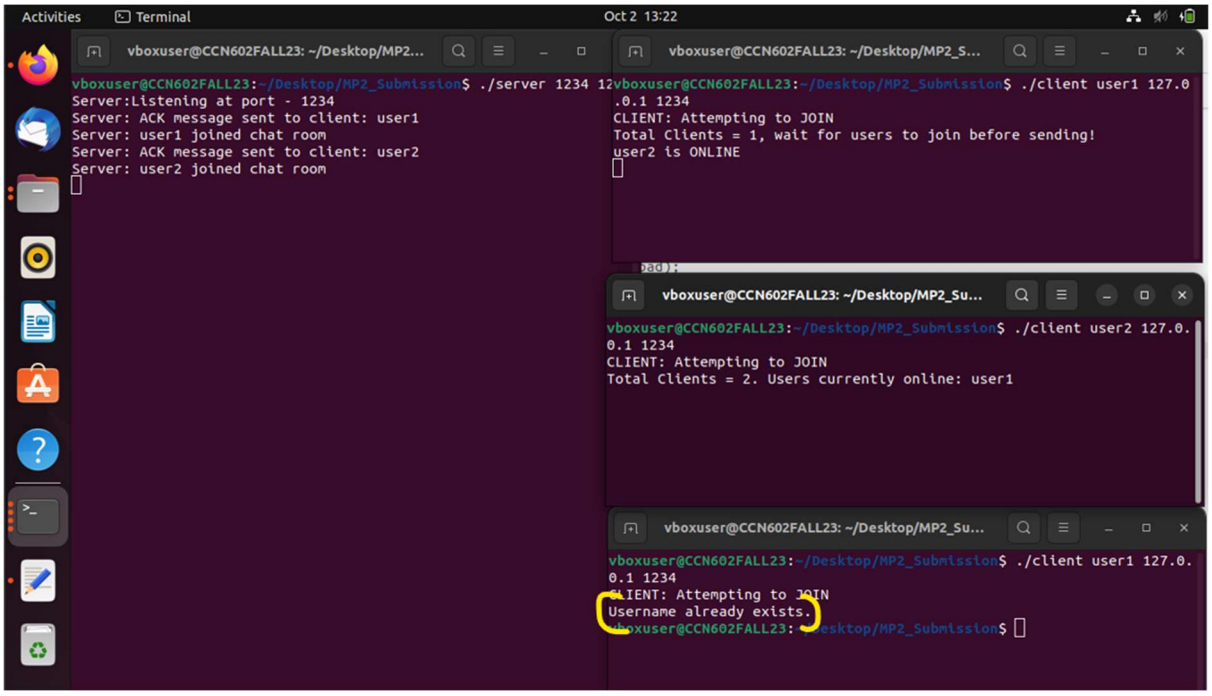
1. ACK



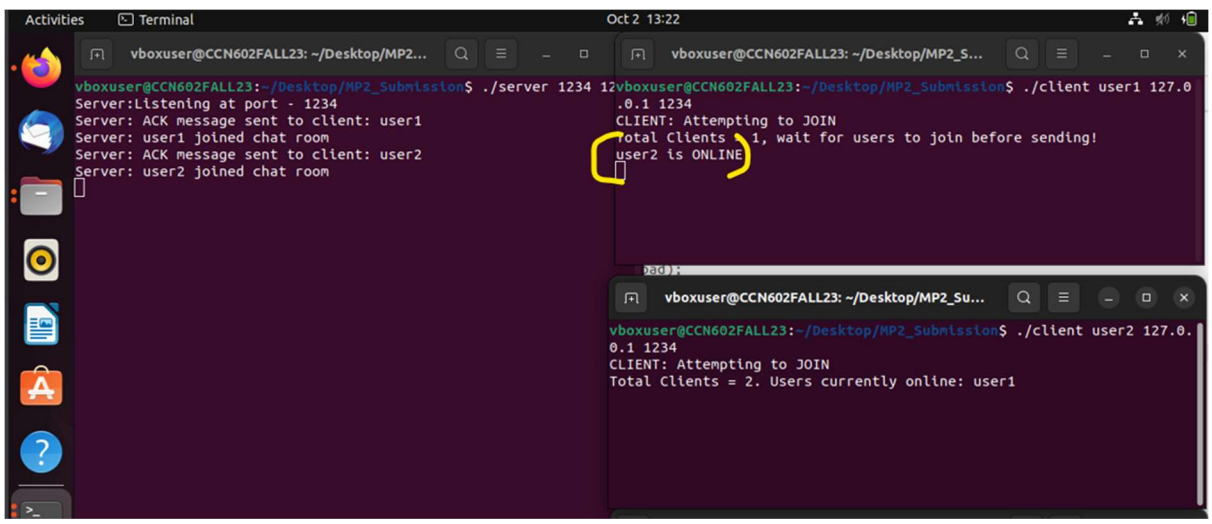
2. NAK
    a. Capacity Limit



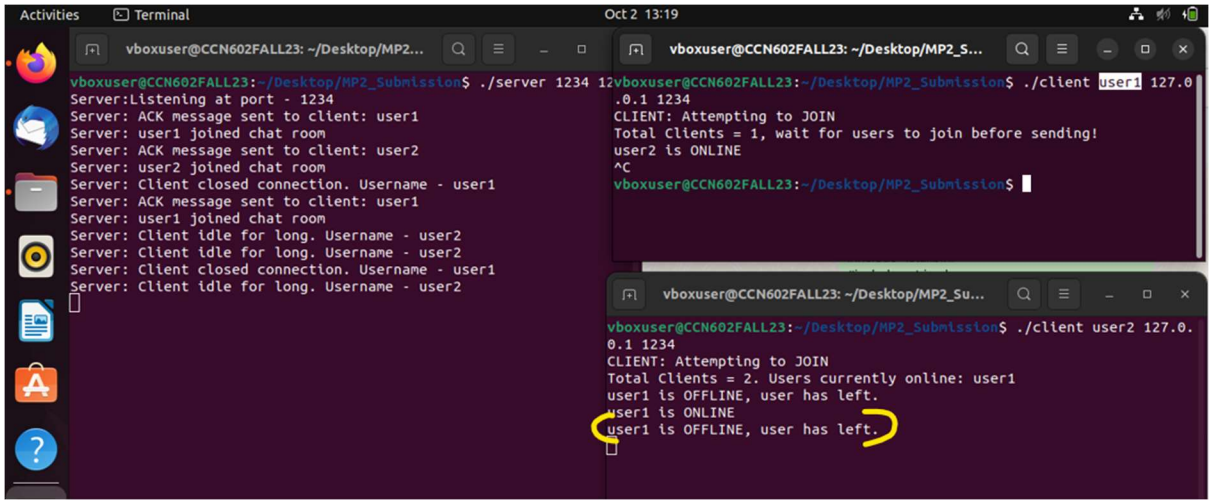
b. Username exists



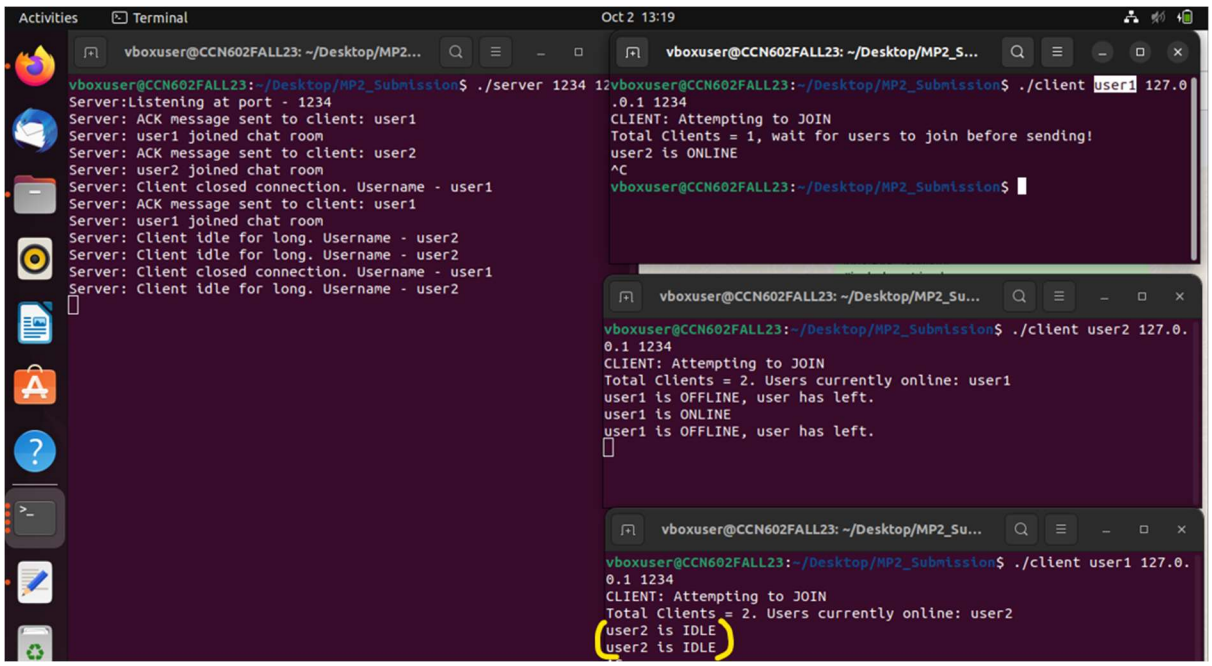
3. Online



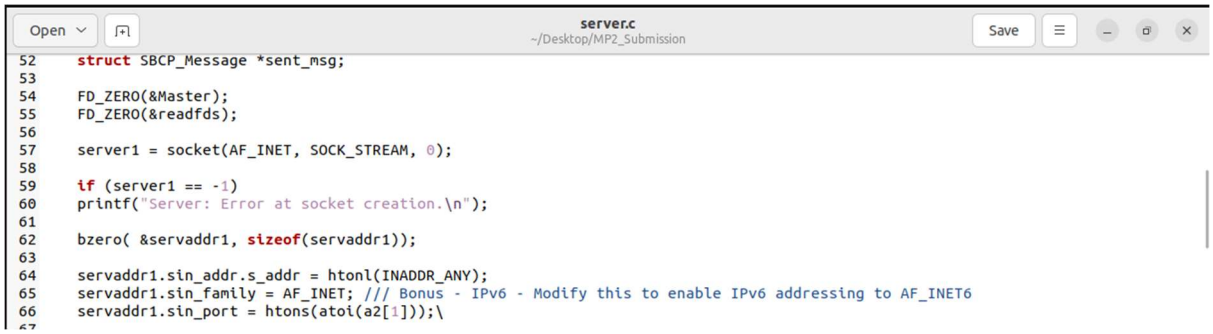
4. Offline



5. IDLE Timer



6. IPv6
   a. Server




```c
    struct SBCP_Message *sent_msg;

    FD_ZERO(&Master);
    FD_ZERO(&readfds);

    server1 = socket(AF_INET, SOCK_STREAM, 0);

    if (server1 == -1)
    printf("Server: Error at socket creation.\n");

    bzero( &servaddr1, sizeof(servaddr1));

    servaddr1.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr1.sin_family = AF_INET; /// Bonus - IPv6 - Modify this to enable IPv6 addressing to AF_INET6
    servaddr1.sin_port = htons(atoi(a2[1]));\
```

   b. Client

```
//For IPv6, use the below client_create_socket method.

// int client_create_socket(int *listen_fd) {
//     struct sockaddr_in6 server_addr;

//     if ((*listen_fd = socket(AF_INET6, SOCK_STREAM, 0)) == -1) {
//         printf("ERROR : Socket creation failed");
//         return -1;
//     }

//     memset(&server_addr, 0, sizeof(server_addr));
//     server_addr.sin6_family = AF_INET6;
//     if (inet_pton(AF_INET6, serverIP, &server_addr.sin6_addr) != 1) {
//         printf("ERROR : Invalid server IP address");
//         close(*listen_fd);
//         return -1;
//     }
//     server_addr.sin6_port = htons(serverPort);

//     if (connect(*listen_fd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr_in6)) != 0) {
//         printf("ERROR : Connection to server failed");
//         close(*listen_fd);
//         return -1;
//     }

//     return 0;
// }
```

References:

[1] Unix Network Programming, Volume 1, The Sockets Networking API, 3rd Edition

[2] Beej's Guide to Network programming