ECEN 602: Network Programming
Assignment 2

Simple Broadcast Chat Server and Client

Team 07 :
Donguk Kim (kdu1113@tamu.edu)
Rohan Dalvi (rohan.dalvi@tamu.edu)

Table of Contents

ChatGPT Generated Code

a. Server Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 8080
#define MAX_CLIENTS 10

// Structure to store client information
struct Client {
    int socket;
    char username[50];
};

// Array to store connected clients
struct Client clients[MAX_CLIENTS];
int clientCount = 0;

// Function to handle client communication
void handleClient(int clientSocket, int index) {
    char buffer[1024];
    while (1) {
        memset(buffer, 0, sizeof(buffer));
        int bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (bytesRead <= 0) {
            // Client disconnected
            printf("Client %s disconnected\n", clients[index].username);
            close(clientSocket);
            clients[index].socket = -1;
            return;
        }

        // Handle received message (e.g., broadcasting to other clients)
        // You need to implement your SBCP protocol handling here

        // Example: Broadcast the message to all other clients
        for (int i = 0; i < MAX_CLIENTS; i++) {
            if (i != index && clients[i].socket != -1) {
                send(clients[i].socket, buffer, bytesRead, 0);
            }
        }
    }
}
```

```c
int main() {
    int serverSocket, newSocket;
    struct sockaddr_in serverAddr, newAddr;
    socklen_t addrSize;
    char buffer[1024];

    // Create socket
    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket < 0) {
        perror("Socket creation error");
        exit(1);
    }

    printf("Server socket created\n");

    // Set server address parameters
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    // Bind the socket
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Binding error");
        exit(1);
    }

    printf("Binding success\n");

    // Listen for incoming connections
    if (listen(serverSocket, 10) == 0) {
        printf("Listening...\n");
    } else {
        perror("Listening error");
        exit(1);
    }

    // Accept and handle incoming connections
    addrSize = sizeof(newAddr);
    while (1) {
        newSocket = accept(serverSocket, (struct sockaddr*)&newAddr, &addrSize);

        // Store client information in the clients array
        if (clientCount < MAX_CLIENTS) {
            clients[clientCount].socket = newSocket;
            snprintf(clients[clientCount].username, sizeof(clients[clientCount].username), "Client-%d", clientCount);
            printf("Client %s connected\n", clients[clientCount].username);

            // Create a new thread or process to handle the client
            // You can also use non-blocking I/O or select/poll for handling multiple clients concurrently
            // For simplicity, we use a single-threaded approach here
            handleClient(newSocket, clientCount);

            clientCount++;
```

```c
    // Accept and handle incoming connections
    addrSize = sizeof(newAddr);
    while (1) {
        newSocket = accept(serverSocket, (struct sockaddr*)&newAddr, &addrSize);

        // Store client information in the clients array
        if (clientCount < MAX_CLIENTS) {
            clients[clientCount].socket = newSocket;
            snprintf(clients[clientCount].username, sizeof(clients[clientCount].username), "Client-%d", clientCount);
            printf("Client %s connected\n", clients[clientCount].username);

            // Create a new thread or process to handle the client
            // You can also use non-blocking I/O or select/poll for handling multiple clients concurrently
            // For simplicity, we use a single-threaded approach here
            handleClient(newSocket, clientCount);

            clientCount++;
        } else {
            // Maximum clients reached, reject the connection
            printf("Maximum clients reached. Connection rejected.\n");
            close(newSocket);
        }
    }

    return 0;
}
```

b. Client Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define PORT 8080
#define SERVER_IP "127.0.0.1"

int main() {
    int clientSocket;
    struct sockaddr_in serverAddr;
    char username[50];
    char message[1024];

    // Create socket
    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket < 0) {
        perror("Socket creation error");
        exit(1);
    }

    printf("Client socket created\n");

    // Set server address parameters
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    serverAddr.sin_addr.s_addr = inet_addr(SERVER_IP);

    // Connect to the server
    if (connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Connection error");
        exit(1);
    }

    printf("Connected to the server\n");

    // Enter a username
    printf("Enter your username: ");
    fgets(username, sizeof(username), stdin);
    username[strcspn(username, "\n")] = '\0';  // Remove the newline character

    while (1) {
        // Input a message
        printf("Enter a message: ");
        fgets(message, sizeof(message), stdin);
        message[strcspn(message, "\n")] = '\0';  // Remove the newline character

        // Send the message to the server
        send(clientSocket, message, strlen(message), 0);

        // Receive and display server's response (e.g., broadcasted messages)
        memset(message, 0, sizeof(message));
        recv(clientSocket, message, sizeof(message), 0);
        printf("Received: %s\n", message);
    }

    // Close the socket when done
    close(clientSocket);

    return 0;
}
```

Our Code modified by ChatGPT

a. Server Code

```
// This is the server code for MP2
// author: rohan.dalvi@tamu.edu
#include <stdio.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "CustomStruct.h"

int ClientCount = 0;
struct ClientInfo *activeClients;

int CheckUsernameAvailability(char name[]) {
    int count = 0;
    int result = 0;
    for (count = 0; count < ClientCount; count++) {
        if (!strcmp(name, activeClients[count].username)) {
            printf("Another client with the username '%s' is trying to connect, but the username already exists. Rejecting it.\n", name);
            result = 1;
            break;
        }
    }
    return result;
}

void SendAcknowledgement(int connectionFd) {
    struct Message AckMessage;
    struct Header AckHeader;
    struct MessageAttribute AckAttribute;
    int count = 0;
    char temp[180];

    AckHeader.version = 3;
    AckHeader.type = 7;

    AckAttribute.type = 4;

    temp[0] = (char)(((int)'0') + ClientCount);
    temp[1] = ' ';
    temp[2] = '\0';
    for (count = 0; count < ClientCount - 1; count++) {
        strcat(temp, activeClients[count].username);
        if (count != ClientCount - 2)
            strcat(temp, ",");
    }
    AckAttribute.length = strlen(temp) + 1;
    strcpy(AckAttribute.payload, temp);
    AckMessage.header = AckHeader;
    AckMessage.attribute[0] = AckAttribute;

    write(connectionFd, (void *)&AckMessage, sizeof(AckMessage));
}

void SendRejection(int connectionFd, int code) {
    struct Message RejectionMessage;
    struct Header RejectionHeader;
    struct MessageAttribute RejectionAttribute;
    char temp[32];

    RejectionHeader.version = 3;
    RejectionHeader.type = 5;

    RejectionAttribute.type = 1;

    if (code == 1)
        strcpy(temp, "Username is incorrect");

    if (code == 2)
        strcpy(temp, "Client count exceeded");

    RejectionAttribute.length = strlen(temp);
    strcpy(RejectionAttribute.payload, temp);

    RejectionMessage.header = RejectionHeader;
    RejectionMessage.attribute[0] = RejectionAttribute;

    write(connectionFd, (void *)&RejectionMessage, sizeof(RejectionMessage));

    close(connectionFd);
}

void NotifyOnline(fd_set master, int serverSocket, int connectionFd, int maxFd) {
    struct Message OnlineMessage;
    int j;
    printf("Server accepted the client: %s\n", activeClients[ClientCount - 1].username);
    OnlineMessage.header.version = 3;
    OnlineMessage.header.type = 8;
    OnlineMessage.attribute[0].type = 2;
    strcpy(OnlineMessage.attribute[0].payload, activeClients[ClientCount - 1].username);
```

```
                perror("Send");
            }
        }
    }
}

void NotifyOffline(fd_set master, int clientSocket, int serverSocket, int connectionFd, int maxFd, int ClientCount) {
    struct Message OfflineMessage;
    int index, j;
    for (index = 0; index < ClientCount; index++) {
        if (activeClients[index].fd == clientSocket) {
            OfflineMessage.attribute[0].type = 2;
            strcpy(OfflineMessage.attribute[0].payload, activeClients[index].username);
        }
    }

    printf("Socket %d belonging to user '%s' is disconnected\n", clientSocket, OfflineMessage.attribute[0].payload);
    OfflineMessage.header.version = 3;
    OfflineMessage.header.type = 6;

    for (j = 0; j <= maxFd; j++) {
        if (FD_ISSET(j, &master)) {
            if (j != clientSocket && j != serverSocket) {
                if ((write(j, (void *)&OfflineMessage, sizeof(OfflineMessage))) == -1) {
                    perror("ERROR: Sending");
                }
            }
        }
    }
}

int ValidateClient(int connectionFd, int maxAllowedClients) {
    struct Message JoinMessage;
    struct MessageAttribute JoinAttribute;
    char temp[30];

    int status = 0;
    read(connectionFd, (struct Message *)&JoinMessage, sizeof(JoinMessage));

    JoinAttribute = JoinMessage.attribute[0];

    strcpy(temp, JoinAttribute.payload);

    if (ClientCount == maxAllowedClients) {
        status = 2;
        printf("A new client is trying to connect, but the client count exceeded. Rejecting it\n");
        SendRejection(connectionFd, 2); // 2- indicates client count exceeded.
        return status;
    }

    status = CheckUsernameAvailability(temp);
    if (status == 1)
        SendRejection(connectionFd, 1); // 1- indicates client already present
    else {
        strcpy(activeClients[ClientCount].username, temp);
        activeClients[ClientCount].fd = connectionFd;
        activeClients[ClientCount].ClientCount = ClientCount;
        ClientCount = ClientCount + 1;
        SendAcknowledgement(connectionFd);
    }
    return status;
}

int main(int argc, char const *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Please use the correct format: %s <hostname> <port> <max_clients>\n", argv[0]);
        exit(0);
    }

    struct Message BroadcastMessage, OfflineMessage;
    struct Message ClientMessage;
    struct MessageAttribute ClientAttribute;

    int serverSocket, connectionFd, m, k;
    unsigned int length;
    int clientStatus = 0;
    struct sockaddr_in serverAddress, *clientInfo;
    struct hostent *hostEntry;

    fd_set master;
    fd_set readSet;
    int maxFd, temp, i = 0, j = 0, x = 0, y, bytesRead, maxAllowedClients = 0;

    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        perror("ERROR: Couldn't create a socket");
        exit(0);
    } else
        printf("Server socket is created.\n");

    bzero(&serverAddress, sizeof(serverAddress));

    int enable = 1;
```

```
    fprintf(stderr, "Please use the correct format: %s <hostname> <port> <max_clients>\n", argv[0]);
    exit(0);
}

struct Message BroadcastMessage, OfflineMessage;
struct Message ClientMessage;
struct MessageAttribute ClientAttribute;

int serverSocket, connectionFd, m, k;
unsigned int length;
int clientStatus = 0;
struct sockaddr_in serverAddress, *clientInfo;
struct hostent *hostEntry;

fd_set master;
fd_set readSet;
int maxFd, temp, i = 0, j = 0, x = 0, y, bytesRead, maxAllowedClients = 0;

serverSocket = socket(AF_INET, SOCK_STREAM, 0);
if (serverSocket == -1) {
    perror("ERROR: Couldn't create a socket");
    exit(0);
} else
    printf("Server socket is created.\n");

bzero(&serverAddress, sizeof(serverAddress));

int enable = 1;
if (setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0) {
    perror("ERROR: Setsockopt\n");
    exit(1);
}

serverAddress.sin_family = AF_INET;
hostEntry = gethostbyname(argv[1]);
memcpy(&serverAddress.sin_addr.s_addr, hostEntry->h_addr, hostEntry->h_length);
serverAddress.sin_port = htons(atoi(argv[2]));

maxAllowedClients = atoi(argv[3]);

activeClients = (struct ClientInfo *)malloc(maxAllowedClients * sizeof(struct ClientInfo));
clientInfo = (struct sockaddr_in *)malloc(maxAllowedClients * sizeof(struct sockaddr_in));

if ((bind(serverSocket, (struct sockaddr *)&serverAddress, sizeof(serverAddress))) != 0) {
    perror("ERROR: Couldn't bind the socket\n");
    exit(0);
} else
    printf("Binding successful.\n");

if ((listen(serverSocket, maxAllowedClients)) != 0) {
    perror("ERROR: Couldn't listen.\n");
    exit(0);
} else
    printf("Listening successful.\n");

FD_SET(serverSocket, &master);
maxFd = serverSocket;

for (;;) {
    readSet = master;
    if (select(maxFd + 1, &readSet, NULL, NULL, NULL) == -1) {
        perror("ERROR: select");
        exit(4);
    }

    for (i = 0; i <= maxFd; i++) {
        if (FD_ISSET(i, &readSet)) {
            if (i == serverSocket) {
                // Incoming connection
                length = sizeof(clientInfo[ClientCount]);
                connectionFd = accept(serverSocket, (struct sockaddr *)&clientInfo[ClientCount], &length);

                // Checking the validity of accept
                if (connectionFd < 0) {
                    perror("ERROR: Couldn't accept \n");
                    exit(0);
                } else {
                    temp = maxFd;
                    FD_SET(connectionFd, &master);
                    if (connectionFd > maxFd) {
                        maxFd = connectionFd;
                    }
                    clientStatus = ValidateClient(connectionFd, maxAllowedClients);
                    if (clientStatus == 0)
                        NotifyOnline(master, serverSocket, connectionFd, maxFd);

                    else if (clientStatus == 1) {
                        // Username already present. Restore maxFD and remove it from the set
                        maxFd = temp;
                        FD_CLR(connectionFd, &master);
                    } else {
                        // Username already present. Restore maxFD and remove it from the set
                        maxFd = temp;
                        FD_CLR(connectionFd, &master); // clear connectionFd to remove this client
```

```
                        clientStatus = ValidateClient(connectionFd, maxAllowedClients);
                        if (clientStatus == 0)
                            NotifyOnline(master, serverSocket, connectionFd, maxFd);

                        else if (clientStatus == 1) {
                            // Username already present. Restore maxFD and remove it from the set
                            maxFd = temp;
                            FD_CLR(connectionFd, &master);
                        } else {
                            // Username already present. Restore maxFD and remove it from the set
                            maxFd = temp;
                            FD_CLR(connectionFd, &master); // clear connectionFd to remove this client
                        }
                    }
                } else {
                    // OLD connections
                    if ((bytesRead = read(i, (struct Message *)&ClientMessage, sizeof(ClientMessage))) <= 0) {
                        // got error or connection closed by the client
                        if (bytesRead == 0)
                            NotifyOffline(master, i, serverSocket, connectionFd, maxFd, ClientCount);
                        else
                            perror("ERROR In receiving");

                        // Cleaning up after closing the erroneous socket
                        close(i);
                        FD_CLR(i, &master); // remove from the master set
                        for (k = 0; k < ClientCount; k++) {
                            if (activeClients[k].fd == i) {
                                m = k;
                                break;
                            }
                        }

                        for (x = m; x < (ClientCount - 1); x++) {
                            activeClients[x] = activeClients[x + 1];
                        }
                        ClientCount--;
                    } else {
                        int payloadLength = 0;
                        char temp[16];
                        // Checking if the existing user becomes idle
                        if (ClientMessage.header.type == 9) {
                            BroadcastMessage = ClientMessage;
                            BroadcastMessage.attribute[0].type = 2;
                            for (y = 0; y < ClientCount; y++) {
                                if (activeClients[y].fd == i) {
                                    strcpy(BroadcastMessage.attribute[0].payload, activeClients[y].username);
                                    strcpy(temp, activeClients[y].username);
                                    payloadLength = strlen(activeClients[y].username);
                                    temp[payloadLength] = '\0';
                                    printf("User '%s' is idle\n", temp);
                                }
                            }
                        } else {
                            // Non-zero-count message received from the client
                            ClientAttribute = ClientMessage.attribute[0]; // message
                            BroadcastMessage = ClientMessage;

                            BroadcastMessage.header.type = 3;
                            BroadcastMessage.attribute[1].type = 2; // username
                            payloadLength = strlen(ClientAttribute.payload);
                            strcpy(temp, ClientAttribute.payload);
                            temp[payloadLength] = '\0';

                            // Message forwarded to clients
                            for (y = 0; y < ClientCount; y++) {
                                if (activeClients[y].fd == i)
                                    strcpy(BroadcastMessage.attribute[1].payload, activeClients[y].username);
                            }
                            printf("User '%s': %s", BroadcastMessage.attribute[1].payload, temp);
                        }

                        for (j = 0; j <= maxFd; j++) {
                            if (FD_ISSET(j, &master)) {
                                // Message broadcasted to everyone except to myself and the listener
                                if (j != i && j != serverSocket) {
                                    if ((write(j, (void *)&BroadcastMessage, bytesRead)) == -1) {
                                        perror("send");
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    close(serverSocket);

    return 0;
}
```

b. Client Code

```
// This is the client code for MP2
// author: kdu1113@tamu.edu
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <strings.h>
#include <errno.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <stdbool.h>
#include "chat.h"
#include <netdb.h>
#include <signal.h>
#include "CustomStruct.h"

#define MAX_BUFF_SIZE (1200)
#define MAX_MESSAGE_LEN (512)

// Read from socket and do whatever the client needs to do
int readMessagefromServer(int sockfd) {
    struct Message stCustomMessage;
    int rslt = 0;
    int nbytes = 0;
    int value, i;

    nbytes = read(sockfd, (struct Message *)&stCustomMessage, sizeof(stCustomMessage));
    if (nbytes <= 0) {
        perror("Server Not Connected.\n");
        kill(0, SIGINT);
        exit(1);
    }

    // (3) FWD
    if (stCustomMessage.header.type == MSG_TYPE_FWD) {
        if ((stCustomMessage.attribute[0].payload != NULL || stCustomMessage.attribute[0].payload != '\0') &&
            (stCustomMessage.attribute[1].payload != NULL || stCustomMessage.attribute[1].payload != '\0') &&
            stCustomMessage.attribute[0].type == ATTR_TYPE_MESSAGE && stCustomMessage.attribute[1].type == ATTR_TYPE_USER_NAME) {
            printf("%s : %s ", stCustomMessage.attribute[1].payload, stCustomMessage.attribute[0].payload);
        }
        return 0;
    }

    // (5) NAK
    if (stCustomMessage.header.type == MSG_TYPE_NAK) {
        if ((stCustomMessage.attribute[0].payload != NULL || stCustomMessage.attribute[0].payload != '\0') &&
            stCustomMessage.attribute[0].type == 1) {
            printf("Disconnected.NAK Message from Server is %s \n", stCustomMessage.attribute[0].payload);
        }
        return 1;
    }

    // (6) OFFLINE
    if (stCustomMessage.header.type == MSG_TYPE_OFFLINE) {
        if ((stCustomMessage.attribute[0].payload != NULL || stCustomMessage.attribute[0].payload != '\0') &&
            stCustomMessage.attribute[0].type == 2) {
            printf("User '%s' is now OFFLINE \n", stCustomMessage.attribute[0].payload);
        }
        return 0;
    }

    // (7) ACK
    if (stCustomMessage.header.type == MSG_TYPE_ACK) {
        if ((stCustomMessage.attribute[0].payload != NULL || stCustomMessage.attribute[0].payload != '\0') &&
            stCustomMessage.attribute[0].type == 4) {
            printf("ACK Message from Server is %s \n", stCustomMessage.attribute[0].payload);
        }
        return 0;
    }

    // (8) ONLINE
    if (stCustomMessage.header.type == MSG_TYPE_ONLINE) {
        if ((stCustomMessage.attribute[0].payload != NULL || stCustomMessage.attribute[0].payload != '\0') &&
            stCustomMessage.attribute[0].type == 2) {
            printf("User '%s' is ONLINE \n", stCustomMessage.attribute[0].payload);
        }
        return 0;
    }

    // (9) IDLE Message
    if (stCustomMessage.header.type == MSG_TYPE_IDLE) {
        if ((stCustomMessage.attribute[0].payload != NULL || stCustomMessage.attribute[0].payload != '\0') &&
            stCustomMessage.attribute[0].type == 2) {
            printf("User '%s' is in IDLE state \n", stCustomMessage.attribute[0].payload);
        }
        return 0;
    }

    return 0;
}

int isValidIpAddr(char *addr_str) {
    struct sockaddr_in tmp_sock;
    int rslt = inet_pton(AF_INET, addr_str, &(tmp_sock.sin_addr));
    if (rslt == 0) {
        return false;
```

```
}

void send_join(int sock_fd, char *username) {
    struct Message stMessage;
    struct MessageAttribute stAttribute;

    memset(&stMessage, 0, sizeof(stMessage));
    memset(&stAttribute, 0, sizeof(stAttribute));

    stAttribute.type = ATTR_TYPE_USER_NAME;
    stAttribute.length = strlen(username) + 1;
    strcpy(stAttribute.payload, username);

    stMessage.header.version = 3;
    stMessage.header.type = MSG_TYPE_JOIN;
    stMessage.attribute[0] = stAttribute;

    printf("username : %s\n", stAttribute.payload);

    write(sock_fd, (void *)&stMessage, sizeof(stMessage));

    // Wait for server's reply
    sleep(1);
    if (readMessagefromServer(sock_fd) == 1) {
        close(sock_fd);
        exit(0);
    }
}

void send_message(int sock_fd) {
    struct Message stMessage;
    struct MessageAttribute stAttribute;

    memset(&stMessage, 0, sizeof(stMessage));
    memset(&stAttribute, 0, sizeof(stAttribute));

    int n = 0;
    char buff[MAX_BUFF_SIZE];
    fd_set readfds;
    FD_ZERO(&readfds);
    FD_SET(STDIN_FILENO, &readfds);

    select(STDIN_FILENO + 1, &readfds, NULL, NULL, NULL);

    if (FD_ISSET(STDIN_FILENO, &readfds)) {
        n = read(STDIN_FILENO, buff, sizeof(buff));
        if (n > 0) {
            buff[n] = '\0';
        }
    }

    stMessage.header.version = 3;
    stMessage.header.type = MSG_TYPE_SEND;

    strcpy(stAttribute.payload, buff);
    stAttribute.type = ATTR_TYPE_MESSAGE;

    stMessage.attribute[0] = stAttribute;

    write(sock_fd, (void *)&stMessage, sizeof(stMessage));
}

void send_timeout(int sock_fd) {
    struct Message stMessage;
    struct MessageAttribute stAttribute;

    memset(&stMessage, 0, sizeof(stMessage));
    memset(&stAttribute, 0, sizeof(stAttribute));

    stMessage.header.version = 3;
    stMessage.header.type = MSG_TYPE_IDLE;

    write(sock_fd, (void *)&stMessage, sizeof(stMessage));
}

int main(int argc, char *argv[]) {
    int tcp_socket;
    struct sockaddr_in servaddr;
    printf("starting client.\n");

    // Step0. Check errors
    if (argc < 4) {
        printf("[Error] Usage: ./client <username> <IP address> <port>\n");
        exit(-1);
    }
    if (isValidIpAddr(argv[2]) == false) {
        printf("[Error] Invalid IP address.\n");
        exit(-1);
    }

    // Step1. Create a Socket
    tcp_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (tcp_socket == -1) {
        printf("[Error] socket creation failed.\n");
        exit(-1);
    } else {
        // socket successfully created. Do nothing.
#if (DEBUG_PRINT == 1)
        printf("Socket created.\n");
```

```
    stMessage.attribute[0] = stAttribute;

    write(sock_fd, (void *)&stMessage, sizeof(stMessage));
}

void send_timeout(int sock_fd) {
    struct Message stMessage;
    struct MessageAttribute stAttribute;

    memset(&stMessage, 0, sizeof(stMessage));
    memset(&stAttribute, 0, sizeof(stAttribute));

    stMessage.header.version = 3;
    stMessage.header.type = MSG_TYPE_IDLE;

    write(sock_fd, (void *)&stMessage, sizeof(stMessage));
}

int main(int argc, char *argv[]) {
    int tcp_socket;
    struct sockaddr_in servaddr;
    printf("starting client.\n");

    // Step0. Check errors
    if (argc < 4) {
        printf("[Error] Usage: ./client <username> <IP address> <port>\n");
        exit(-1);
    }
    if (isValidIpAddr(argv[2]) == false) {
        printf("[Error] Invalid IP address.\n");
        exit(-1);
    }

    // Step1. Create a Socket
    tcp_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (tcp_socket == -1) {
        printf("[Error] socket creation failed.\n");
        exit(-1);
    } else {
        // socket successfully created. Do nothing.
#if (DEBUG_PRINT == 1)
        printf("Socket created.\n");
#endif
    }

    // Step2. Connect
    servaddr.sin_family = AF_INET;
    struct hostent *hostret = gethostbyname(argv[2]);
    memcpy(&servaddr.sin_addr.s_addr, hostret->h_addr, hostret->h_length);
    servaddr.sin_addr.s_addr = inet_addr(argv[2]);
    servaddr.sin_port = htons(atoi(argv[3]));

    if (connect(tcp_socket, (const struct sockaddr *)&servaddr, sizeof(servaddr)) == -1) {
        printf("[Error] Connect Failed.\n");
        exit(-1);
    }

#if (DEBUG_PRINT == 1)
    printf("Connected to the server.");
#endif

    char *username = argv[1];

    // Send Join Command
    send_join(tcp_socket, username);
    printf("Server connection successful. \n");

    char buff[MAX_BUFF_SIZE];
    fd_set master;
    fd_set new;
    fd_set read_fds;

    FD_ZERO(&master);
    FD_ZERO(&new);
    FD_ZERO(&read_fds);

    FD_SET(tcp_socket, &master);
    FD_SET(STDIN_FILENO, &new);

    struct timeval tv;
    int secs, usecs;
    tv.tv_sec = 10;
    tv.tv_usec = 0;
    secs = (int)tv.tv_sec;
    usecs = (int)tv.tv_usec;

    pid_t pid;
    pid = fork();

    if (pid == 0) {
        // select stdin
        while (1) {
            read_fds = new;
            tv.tv_sec = 10;
            tv.tv_usec = 0;

            if (select(STDIN_FILENO + 1, &read_fds, NULL, NULL, &tv) == -1) {
                perror("select failed.");
                exit(0);
            }
            // if stdin ready
            if (FD_ISSET(STDIN_FILENO, &read_fds)) {
                // create a packet and send it to the server.
                send_message(tcp_socket);
                continue;
            } else if (tv.tv_sec == 0 && tv.tv_usec == 0) {
                printf("Time out!! No user input for %d secs %d usecs\n", secs, usecs);
                tv.tv_sec = 10;
                tv.tv_usec = 0;
                read_fds = new;
                send_timeout(tcp_socket);
                continue;
            }
        }
    } else {
        // master
        while (1) {
            read_fds = master;
            // select server
            if (select(tcp_socket + 1, &read_fds, NULL, NULL, NULL) == -1) {
                perror("select failed.");
                exit(1);
            }
            // if data ready from socket
            if (FD_ISSET(tcp_socket, &read_fds)) {
                // read the message and do sth.
                readMessagefromServer(tcp_socket);
            }
        }
        kill(pid, SIGINT);
    }

    printf("\n Connection Ends \n");

    return 0;
}
```

Our Code

a. Server Code

```
// This is the server code for MP2
// author: rohan.dalvi@tamu.edu
#include <stdio.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "CustomStruct.h"

int ClientCount = 0;
struct ClientInfo *activeClients;

int CheckUsernameAvailability(char name[]) {
    int count = 0;
    int result = 0;
    for (count = 0; count < ClientCount; count++) {
        if (!strcmp(name, activeClients[count].username)) {
            printf("Another client with the username '%s' is trying to connect, but the username
already exists. Rejecting it.\n", name);
            result = 1;
            break;
        }
    }
    return result;
}

void SendAcknowledgement(int connectionFd) {
    struct Message AckMessage;
    struct Header AckHeader;
    struct MessageAttribute AckAttribute;
    int count = 0;
    char temp[180];

    AckHeader.version = 3;
    AckHeader.type = 7;

    AckAttribute.type = 4;

    temp[0] = (char)(((int)'0') + ClientCount);
    temp[1] = ' ';
```

```
    temp[2] = '\0';
    for (count = 0; count < ClientCount - 1; count++) {
        strcat(temp, activeClients[count].username);
        if (count != ClientCount - 2)
            strcat(temp, ",");
    }
    AckAttribute.length = strlen(temp) + 1;
    strcpy(AckAttribute.payload, temp);
    AckMessage.header = AckHeader;
    AckMessage.attribute[0] = AckAttribute;

    write(connectionFd, (void *)&AckMessage, sizeof(AckMessage));
}

void SendRejection(int connectionFd, int code) {
    struct Message RejectionMessage;
    struct Header RejectionHeader;
    struct MessageAttribute RejectionAttribute;
    char temp[32];

    RejectionHeader.version = 3;
    RejectionHeader.type = 5;

    RejectionAttribute.type = 1;

    if (code == 1)
        strcpy(temp, "Username is incorrect");

    if (code == 2)
        strcpy(temp, "Client count exceeded");

    RejectionAttribute.length = strlen(temp);
    strcpy(RejectionAttribute.payload, temp);

    RejectionMessage.header = RejectionHeader;
    RejectionMessage.attribute[0] = RejectionAttribute;

    write(connectionFd, (void *)&RejectionMessage, sizeof(RejectionMessage));

    close(connectionFd);
}

void NotifyOnline(fd_set master, int serverSocket, int connectionFd, int maxFd) {
    struct Message OnlineMessage;
    int j;
    printf("Server accepted the client: %s\n", activeClients[ClientCount - 1].username);
    OnlineMessage.header.version = 3;
    OnlineMessage.header.type = 8;
    OnlineMessage.attribute[0].type = 2;
    strcpy(OnlineMessage.attribute[0].payload, activeClients[ClientCount - 1].username);
```

```
    for (j = 0; j <= maxFd; j++) {
        if (FD_ISSET(j, &master)) {
            if (j != serverSocket && j != connectionFd) {
                if ((write(j, (void *)&OnlineMessage, sizeof(OnlineMessage))) == -1) {
                    perror("send");
                }
            }
        }
    }
}

void NotifyOffline(fd_set master, int clientSocket, int serverSocket, int connectionFd, int maxFd,
int ClientCount) {
    struct Message OfflineMessage;
    int index, j;
    for (index = 0; index < ClientCount; index++) {
        if (activeClients[index].fd == clientSocket) {
            OfflineMessage.attribute[0].type = 2;
            strcpy(OfflineMessage.attribute[0].payload, activeClients[index].username);
        }
    }

    printf("Socket %d belonging to user '%s' is disconnected\n", clientSocket,
OfflineMessage.attribute[0].payload);
    OfflineMessage.header.version = 3;
    OfflineMessage.header.type = 6;

    for (j = 0; j <= maxFd; j++) {
        if (FD_ISSET(j, &master)) {
            if (j != clientSocket && j != serverSocket) {
                if ((write(j, (void *)&OfflineMessage, sizeof(OfflineMessage))) == -1) {
                    perror("ERROR: Sending");
                }
            }
        }
    }
}

int ValidateClient(int connectionFd, int maxAllowedClients) {
    struct Message JoinMessage;
    struct MessageAttribute JoinAttribute;
    char temp[30];

    int status = 0;
    read(connectionFd, (struct Message *)&JoinMessage, sizeof(JoinMessage));

    JoinAttribute = JoinMessage.attribute[0];

    strcpy(temp, JoinAttribute.payload);

    if (ClientCount == maxAllowedClients) {
```

```
        status = 2;
        printf("A new client is trying to connect, but the client count exceeded. Rejecting it\n");
        SendRejection(connectionFd, 2); // 2- indicates client count exceeded.
        return status;
    }

    status = CheckUsernameAvailability(temp);
    if (status == 1)
        SendRejection(connectionFd, 1); // 1- indicates client already present
    else {
        strcpy(activeClients[ClientCount].username, temp);
        activeClients[ClientCount].fd = connectionFd;
        activeClients[ClientCount].ClientCount = ClientCount;
        ClientCount = ClientCount + 1;
        SendAcknowledgement(connectionFd);
    }
    return status;
}

int main(int argc, char const *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Please use the correct format: %s <hostname> <port> <max_clients>\n",
argv[0]);
        exit(0);
    }

    struct Message BroadcastMessage, OfflineMessage;
    struct Message ClientMessage;
    struct MessageAttribute ClientAttribute;

    int serverSocket, connectionFd, m, k;
    unsigned int length;
    int clientStatus = 0;
    struct sockaddr_in serverAddress, *clientInfo;
    struct hostent *hostEntry;

    fd_set master;
    fd_set readSet;
    int maxFd, temp, i = 0, j = 0, x = 0, y, bytesRead, maxAllowedClients = 0;

    serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        perror("ERROR: Couldn't create a socket");
        exit(0);
    } else
        printf("Server socket is created.\n");

    bzero(&serverAddress, sizeof(serverAddress));

    int enable = 1;
    if (setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0) {
```

```
        perror("ERROR: Setsockopt\n");
        exit(1);
    }

    serverAddress.sin_family = AF_INET;
    hostEntry = gethostbyname(argv[1]);
    memcpy(&serverAddress.sin_addr.s_addr, hostEntry->h_addr, hostEntry->h_length);
    serverAddress.sin_port = htons(atoi(argv[2]));

    maxAllowedClients = atoi(argv[3]);

    activeClients = (struct ClientInfo *)malloc(maxAllowedClients * sizeof(struct ClientInfo));
    clientInfo = (struct sockaddr_in *)malloc(maxAllowedClients * sizeof(struct sockaddr_in));

    if ((bind(serverSocket, (struct sockaddr *)&serverAddress, sizeof(serverAddress))) != 0) {
        perror("ERROR: Couldn't bind the socket\n");
        exit(0);
    } else
        printf("Binding successful.\n");

    if ((listen(serverSocket, maxAllowedClients)) != 0) {
        perror("ERROR: Couldn't listen.\n");
        exit(0);
    } else
        printf("Listening successful.\n");

    FD_SET(serverSocket, &master);
    maxFd = serverSocket;

    for (;;) {
        readSet = master;
        if (select(maxFd + 1, &readSet, NULL, NULL, NULL) == -1) {
            perror("ERROR: select");
            exit(4);
        }

        for (i = 0; i <= maxFd; i++) {
            if (FD_ISSET(i, &readSet)) {
                if (i == serverSocket) {
                    // Incoming connection
                    length = sizeof(clientInfo[ClientCount]);
                    connectionFd = accept(serverSocket, (struct sockaddr *)&clientInfo[ClientCount],
&length);

                    // Checking the validity of accept
                    if (connectionFd < 0) {
                        perror("ERROR: Couldn't accept \n");
                        exit(0);
                    } else {
                        temp = maxFd;
                        FD_SET(connectionFd, &master);
```

```
                if (connectionFd > maxFd) {
                    maxFd = connectionFd;
                }
                clientStatus = ValidateClient(connectionFd, maxAllowedClients);
                if (clientStatus == 0)
                    NotifyOnline(master, serverSocket, connectionFd, maxFd);

                else if (clientStatus == 1) {
                    // Username already present. Restore maxFD and remove it from the set
                    maxFd = temp;
                    FD_CLR(connectionFd, &master);
                } else {
                    // Username already present. Restore maxFD and remove it from the set
                    maxFd = temp;
                    FD_CLR(connectionFd, &master); // clear connectionFd to remove this client
                }
            }
        } else {
            // OLD connections
            if ((bytesRead = read(i, (struct Message *)&ClientMessage, sizeof(ClientMessage)))
<= 0) {
                // got error or connection closed by the client
                if (bytesRead == 0)
                    NotifyOffline(master, i, serverSocket, connectionFd, maxFd, ClientCount);
                else
                    perror("ERROR In receiving");

                // Cleaning up after closing the erroneous socket
                close(i);
                FD_CLR(i, &master); // remove from the master set
                for (k = 0; k < ClientCount; k++) {
                    if (activeClients[k].fd == i) {
                        m = k;
                        break;
                    }
                }

                for (x = m; x < (ClientCount - 1); x++) {
                    activeClients[x] = activeClients[x + 1];
                }
                ClientCount--;
            } else {
                int payloadLength = 0;
                char temp[16];
                // Checking if the existing user becomes idle
                if (ClientMessage.header.type == 9) {
                    BroadcastMessage = ClientMessage;
                    BroadcastMessage.attribute[0].type = 2;
                    for (y = 0; y < ClientCount; y++) {
                        if (activeClients[y].fd == i) {
```

```
                        strcpy(BroadcastMessage.attribute[0].payload,
activeClients[y].username);
                        strcpy(temp, activeClients[y].username);
                        payloadLength = strlen(activeClients[y].username);
                        temp[payloadLength] = '\0';
                        printf("User '%s' is idle\n", temp);
                    }
                }
            } else {
                // Non-zero-count message received from the client
                ClientAttribute = ClientMessage.attribute[0]; // message
                BroadcastMessage = ClientMessage;

                BroadcastMessage.header.type = 3;
                BroadcastMessage.attribute[1].type = 2; // username
                payloadLength = strlen(ClientAttribute.payload);
                strcpy(temp, ClientAttribute.payload);
                temp[payloadLength] = '\0';

                // Message forwarded to clients
                for (y = 0; y < ClientCount; y++) {
                    if (activeClients[y].fd == i)
                        strcpy(BroadcastMessage.attribute[1].payload,
activeClients[y].username);
                }
                printf("User '%s': %s", BroadcastMessage.attribute[1].payload, temp);
            }

            for (j = 0; j <= maxFd; j++) {
                if (FD_ISSET(j, &master)) {
                    // Message broadcasted to everyone except to myself and the listener
                    if (j != i && j != serverSocket) {
                        if ((write(j, (void *)&BroadcastMessage, bytesRead)) == -1) {
                            perror("send");
                        }
                    }
                }
            }
        }
    }
}

close(serverSocket);

return 0;
}
```

b. Client Code

```
// This is the client code for MP2
// author: kdu1113@tamu.edu
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<strings.h>
#include<errno.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<stdbool.h>
#include "chat.h"
#include<netdb.h>
#include <signal.h>
#include "CustomStruct.h"

#define MAX_BUFF_SIZE (1200)
#define MAX_MESSAGE_LEN (512)

// Read from socket and do whatever the client needs to do
int readMessagefromServer(int sockfd){
  struct Message stCustomMessage;
  int rslt = 0;
  int nbytes=0;
  int value, i;

  nbytes=read(sockfd,(struct Message *) &stCustomMessage,sizeof(stCustomMessage));
  if(nbytes <=0){
    perror("Server Not Connected.\n");
    kill(0, SIGINT);
    exit(1);
  }

  // (3) FWD
  if(stCustomMessage.header.type==MSG_TYPE_FWD)
  {
    if((stCustomMessage.attribute[0].payload!=NULL ||
stCustomMessage.attribute[0].payload!='\0') &&
       (stCustomMessage.attribute[1].payload!=NULL ||
stCustomMessage.attribute[1].payload!='\0') &&
       stCustomMessage.attribute[0].type==ATTR_TYPE_MESSAGE &&
stCustomMessage.attribute[1].type==ATTR_TYPE_USER_NAME)
    {
```

```
      printf("%s : %s ", stCustomMessage.attribute[1].payload,
stCustomMessage.attribute[0].payload);
    }
    return 0;
  }

  // (5) NAK
  if(stCustomMessage.header.type==MSG_TYPE_NAK)
  {
    if((stCustomMessage.attribute[0].payload!=NULL ||
stCustomMessage.attribute[0].payload!='\0') &&
        stCustomMessage.attribute[0].type==1)
    {
          printf("Disconnected.NAK Message from Server is %s
\n",stCustomMessage.attribute[0].payload);
    }
    return 1;
  }

  // (6) OFFLINE
  if(stCustomMessage.header.type==MSG_TYPE_OFFLINE)
  {
    if((stCustomMessage.attribute[0].payload!=NULL ||
stCustomMessage.attribute[0].payload!='\0') &&
      stCustomMessage.attribute[0].type==2)
    {
          printf("User '%s' is now OFFLINE \n",stCustomMessage.attribute[0].payload);
    }
    return 0;
  }

  // (7) ACK
  if(stCustomMessage.header.type==MSG_TYPE_ACK)
  {
           if((stCustomMessage.attribute[0].payload!=NULL ||
stCustomMessage.attribute[0].payload!='\0') &&
        stCustomMessage.attribute[0].type==4)
    {
             printf("ACK Message from Server is %s \n",stCustomMessage.attribute[0].payload);
    }
    return 0;
  }

  // (8) ONLINE
  if(stCustomMessage.header.type==MSG_TYPE_ONLINE)
  {
    if((stCustomMessage.attribute[0].payload!=NULL ||
stCustomMessage.attribute[0].payload!='\0') &&
       stCustomMessage.attribute[0].type==2)
    {
          printf("User '%s' is ONLINE \n",stCustomMessage.attribute[0].payload);
```

```
    }
    return 0;
  }

  // (9) IDLE Message
  if(stCustomMessage.header.type==MSG_TYPE_IDLE)
  {
    if((stCustomMessage.attribute[0].payload!=NULL ||
stCustomMessage.attribute[0].payload!='\0') &&
       stCustomMessage.attribute[0].type==2)
    {
      printf("User '%s' is in IDLE state \n",stCustomMessage.attribute[0].payload);
    }
    return 0;
  }

  return 0;
}

int isValidIpAddr(char* addr_str)
{
        struct sockaddr_in tmp_sock;
        int rslt = inet_pton(AF_INET, addr_str, &(tmp_sock.sin_addr));
        if (rslt == 0)
        {
                return false;
        }
        return true;
}

void send_join(int sock_fd, char* username){
  SBCP_Attribute stAttr;
  SBCP_Message stMessage;

  memset(&stMessage, 0, sizeof(stMessage));
  memset(&stAttr, 0, sizeof(stAttr));

  stAttr.type = ATTR_TYPE_USER_NAME;
  stAttr.length = strlen(username) + 1;
  strcpy(stAttr.payload, username);

  stMessage.vrsn = SBCP_PROTOCOL_VERSION;
  stMessage.type = MSG_TYPE_JOIN;
  memcpy(&(stMessage.stAttribute1), &stAttr, sizeof(stAttr));

  printf("username : %s\n", stAttr.payload);

  write(sock_fd,(void *) &stMessage,sizeof(stMessage));
  // if (writen(sock_fd, (void *) &stMessage, sizeof(stMessage)) == -1)
  // {
  //   printf("[ERROR]Writen failed.\n");
```

```
  //   exit(-1);
  // }

  // Wait for server's reply
  sleep(1);
  if(readMessagefromServer(sock_fd) == 1)
  {
    close(sock_fd);
    exit(0);
  }
}

void send_message(int sock_fd){
  SBCP_Attribute stAttr;
  SBCP_Message stMessage;

  memset(&stMessage, 0, sizeof(stMessage));
  memset(&stAttr, 0, sizeof(stAttr));

  int n = 0;
  char buff[MAX_BUFF_SIZE];
  fd_set readfds;
  FD_ZERO(&readfds);
  FD_SET(STDIN_FILENO, &readfds);

  select(STDIN_FILENO+1, &readfds, NULL, NULL, NULL);

  if (FD_ISSET(STDIN_FILENO, &readfds))
  {
    n = read(STDIN_FILENO, buff, sizeof(buff));
    if(n > 0){
      buff[n] = '\0';
    }
  }

  stMessage.type = MSG_TYPE_SEND;
  strcpy(stAttr.payload,buff);
  stAttr.type = ATTR_TYPE_MESSAGE;
  memcpy(&(stMessage.stAttribute1), &stAttr, sizeof(stAttr));
  write(sock_fd ,(void *) &stMessage, sizeof(stMessage));
}

void send_timeout(int sock_fd){
  SBCP_Attribute stAttr;
  SBCP_Message stMessage;

  memset(&stMessage, 0, sizeof(stMessage));
  memset(&stAttr, 0, sizeof(stAttr));

  stMessage.vrsn=SBCP_PROTOCOL_VERSION;
  stMessage.type=MSG_TYPE_IDLE;
```

```
  write(sock_fd,(void *) &stMessage, sizeof(stMessage));
}

int main(int argc, char* argv[])
{
	int tcp_socket;
	struct sockaddr_in servaddr;
  printf("starting client.\n");

	// Step0. Check errors
	if (argc < 4)
	{
		printf("[Error] Usage: ./client <username> <IP address> <port>\n");
		exit(-1);
	}
	if (isValidIpAddr(argv[2]) == false)
	{
		printf("[Error] Invalid IP address.\n");
		exit(-1);
	}

	// Step1. Create a Socket
	tcp_socket = socket(AF_INET, SOCK_STREAM, 0);
	if (tcp_socket == -1)
	{
		printf("[Error] socket creation failed.\n");
		exit(-1);
	}
	else
	{
		// socket successfully created. Do nothing.
#if (DEBUG_PRINT == 1)
		printf("Socket created.\n");
#endif
	}

	// Step2. Connect
	servaddr.sin_family = AF_INET;
  struct hostent* hostret = gethostbyname(argv[2]);
  memcpy(&servaddr.sin_addr.s_addr, hostret->h_addr,hostret->h_length);
	servaddr.sin_addr.s_addr = inet_addr(argv[2]);
	servaddr.sin_port = htons(atoi(argv[3]));

	if (connect(tcp_socket, (const struct sockaddr *)&servaddr, sizeof(servaddr)) == -1)
	{
		printf("[Error] Connect Failed.\n");
		exit(-1);
	}

#if (DEBUG_PRINT == 1)
	printf("Connected to the server.");
```

```
#endif

  char* username = argv[1];

  // Send Join Command
  send_join(tcp_socket, username);
  printf("Server connection successful. \n");

  char buff[MAX_BUFF_SIZE];
  char send_buff[MAX_BUFF_SIZE];
  fd_set master;
  fd_set new;
  fd_set read_fds;

  FD_ZERO(&master);
  FD_ZERO(&new);
  FD_ZERO(&read_fds);

  FD_SET(tcp_socket, &master);
  FD_SET(STDIN_FILENO, &new);

  SBCP_Message stMessage;
  SBCP_Attribute stAttribute;

  struct timeval tv;
  int secs, usecs;
  tv.tv_sec = 10;
  tv.tv_usec = 0;
  secs=(int) tv.tv_sec;
  usecs=(int) tv.tv_usec;

  pid_t pid;
  pid=fork();

  if(pid==0){
    // select stdin
    while(1){
      read_fds=new;
      tv.tv_sec = 10;
      tv.tv_usec = 0;

      if(select(STDIN_FILENO+1, &read_fds, NULL, NULL, &tv) == -1)
      {
        perror("select faiiled.");
        exit(0);
      }
      // if stdin ready
      if (FD_ISSET(STDIN_FILENO, &read_fds)){
        // create a packet and send it to the server.
        send_message(tcp_socket);
        continue;
```

```
      }
      else if(tv.tv_sec==0 && tv.tv_usec==0){
        printf("Time out!! No user input for %d secs %d usecs\n", secs, usecs);
        tv.tv_sec=10;
        tv.tv_usec=0;
        read_fds=new;
        send_timeout(tcp_socket);
        continue;
                                }
    }
  }
  else {
    // master
    while(1){
      read_fds=master;
      // select server
      if (select(tcp_socket+1,&read_fds,NULL,NULL,NULL) == -1)
      {
        perror("select failed.");
        exit(1);
      }
      // if data ready from socket
      if (FD_ISSET(tcp_socket, &read_fds)){
        // read the message and do sth.
        readMessagefromServer(tcp_socket);
      }
    }
    kill(pid, SIGINT);
  }

  printf("\n Connection Ends \n");

  return 0;
}
```

We have used 2 Header Files (CustomStruct.h & Chat.h) for the client-server codes.

a. CustomStruct.h

```
#ifndef CUSTOM_STRUCT_H
#define CUSTOM_STRUCT_H

// Header structure
struct Header {
    unsigned int version : 9;
    unsigned int type : 7;
    int length;
};

// Message Attribute structure
struct MessageAttribute {
    int type;
    int length;
    char payload[512];
};

// Message structure
struct Message {
    struct Header header;
    struct MessageAttribute attribute[2];
};

// Client information structure
struct ClientInfo {
    int fd;
    char username[16];
    int ClientCount;  // Add this member
};

#endif  // CUSTOM_STRUCT_H
```

b. Chat.h

```
#include<string.h>

typedef unsigned char UINT8;
typedef unsigned short UINT16;
typedef unsigned int UINT32;

#define SBCP_PROTOCOL_VERSION   3   // protocol version is 3

typedef enum {
    ATTR_TYPE_REASON = 1,
    ATTR_TYPE_USER_NAME = 2,
    ATTR_TYPE_CLIENT_COUNT = 3,
    ATTR_TYPE_MESSAGE = 4,
    ATTR_TYPE_INVALID = 0x7F,     // 7 bits max.
} AttrType;

typedef enum {
    MSG_TYPE_JOIN = 2,
    MSG_TYPE_FWD = 3,
    MSG_TYPE_SEND = 4,
    // Bonus Types
    MSG_TYPE_NAK = 5,
    MSG_TYPE_OFFLINE = 6,
    MSG_TYPE_ACK = 7,
    MSG_TYPE_ONLINE = 8,
    MSG_TYPE_IDLE = 9,
    MSG_TYPE_INVALID = 0xFFFF,   // 9 bits max. -> assign 2 bytes.
} MsgType;

typedef struct _SBCP_Attribute{
    int type;
    int length;
    char payload[512];
}SBCP_Attribute;

typedef struct _SBCP_Message{
    unsigned int vrsn : 9;
    unsigned int type : 7;
    int length;
    SBCP_Attribute stAttribute1;
    SBCP_Attribute stAttribute2;
}SBCP_Message;
```

```
// Macros
#if 0
#define GET_MESSAGE_VRSN(stMessage) ((stMessage.vrsn_and_type) >> 7 &
(UINT16)(0x1FF))
#define GET_MESSAGE_TYPE(stMessage) ((stMessage.vrsn_and_type) & (UINT16)(0x7F))
#define SET_MESSAGE_VRSN(pMessage, set_value)
(((SBCP_Message*)(pMessage))->vrsn_and_type |= (UINT16)((set_value & (0x1FF)) << 7))
#define SET_MESSAGE_TYPE(pMessage, set_value)
(((SBCP_Message*)(pMessage))->vrsn_and_type |= (UINT16)((set_value & (0x7F))))
#define CLEAR_MESSAGE_VRSN(pMessage)
(((SBCP_Message*)(pMessage))->vrsn_and_type &= (UINT16)(0x7F))
#define CLEAR_MESSAGE_TYPE(pMessage)
(((SBCP_Message*)(pMessage))->vrsn_and_type &= (UINT16)(0x1FF << 7))

#define GET_MESSAGE_VRSN(stMessage) ((stMessage.vrsn_and_type) >> 8 &
(UINT16)(0xFF))
#define GET_MESSAGE_TYPE(stMessage) ((stMessage.vrsn_and_type) & (UINT16)(0xFF))
#define SET_MESSAGE_VRSN(pMessage, set_value)
(((SBCP_Message*)(pMessage))->vrsn_and_type |= (UINT16)((set_value & (0xFF)) << 8))
#define SET_MESSAGE_TYPE(pMessage, set_value)
(((SBCP_Message*)(pMessage))->vrsn_and_type |= (UINT16)((set_value & (0xFF))))
#define CLEAR_MESSAGE_VRSN(pMessage)
(((SBCP_Message*)(pMessage))->vrsn_and_type &= (UINT16)(0xFF))
#define CLEAR_MESSAGE_TYPE(pMessage)
(((SBCP_Message*)(pMessage))->vrsn_and_type &= (UINT16)(0xFF << 8))
#endif

#if 0
void set_sbcp_attribute(SBCP_Attribute* _pAttribute, AttrType _eAttrType, UINT16 _length,
char* _payload){
    _pAttribute->length = _length;
    _pAttribute->type = _eAttrType;
    strcpy(_pAttribute->payload, _payload);
}

void set_sbcp_message(SBCP_Message* _pSbcpMessage, UINT16 _nVrsn, MsgType
_eMessageType, UINT16 _nLength, SBCP_Attribute _stAttribute1, SBCP_Attribute
_stAttribute2){
    CLEAR_MESSAGE_VRSN(_pSbcpMessage);
    CLEAR_MESSAGE_TYPE(_pSbcpMessage);
    SET_MESSAGE_VRSN(_pSbcpMessage, _nVrsn);
    SET_MESSAGE_TYPE(_pSbcpMessage, _eMessageType);
    _pSbcpMessage->length = _nLength;
```

```
    _pSbcpMessage->stAttribute1 = _stAttribute1;
    _pSbcpMessage->stAttribute2 = _stAttribute2;
    // printf("Message Type: %d\n", _eMessageType);
    // printf("version and type: 0x%x\n", _pSbcpMessage->vrsn_and_type);
    // printf("Message Len: %d\n", _pSbcpMessage->length);
    // printf("Attr Type: %d, Attr Len: %d", _pSbcpMessage->stAttribute1.type,
_pSbcpMessage->stAttribute1.length);
}

#endif
```

Makefile

```
output: client.o server.o
	gcc client.o -o client
	gcc server.o -o server

client.o: client.c
	gcc -c client.c

server.o: server.c
	gcc -c server.c

clean:
	rm -f client server *.o core
```

README.md

This is the readme file for ECEN602 - MP2.

Roles:

1. Rohan Dalvi - write the code for server.c and run test cases.
2. Donguk Kim - write the code for client.c, debugging.

Architecture of the code: The program consists of two main parts, the client and the server. The server and the client each have their own main function. There are several assumptions between the client and the server. The Server hosts a chat service, and multiple clients can join the chat with JOIN requests, SEND messages, and receive the messages that other clients send which are broadcasted by the server with an FWD message.

Usage:

1. Go to the MP2 folder, Build using the make command.
   make

2. In one terminal, start the server by
   ./server 127.0.0.1 2222 4 (arguments: IP address, port, max number of clients)

3. Open another terminal, start the client by
   ./client user1 127.0.0.1 2222 (arguments: username, IP address, port)

4. Open another terminal, and start the client using a different username.
   ./client user2 127.0.0.1 2222 (arguments: username, IP address, port)

5. Input the string and press enter on the client terminal.
   Howdy, World!!!

6. To leave the chat, go to the client terminal and press ctrl+c.
   ./client 127.0.0.1 2222

Test Cases

Test Case 1: Normal operation of the chat client with three clients connected.

SERVER

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (03:02:45 10/02/23)
:: ./server 127.0.0.1 2222 3
Server socket is created.
Binding successful.
Listening successful.
Server accepted the client: user1
Server accepted the client: user2
User 'user1' is idle
User 'user2' is idle
User 'user1' is idle
Server accepted the client: user3
User 'user2' is idle
User 'user1' is idle
User 'user3' is idle
```

USER 1

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (02:50:45 10/02/23)
:: ./client user1 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 1
Server connection successful
User 'user2' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user2' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now ONLINE
User 'user2' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now IDLE
```

USER 2

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (02:50:42 10/02/23)
:: ./client user2 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 2 user1
Server connection successful
User 'user1' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
User 'user3' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
User 'user3' is now IDLE
```

USER 3

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (02:50:30 10/02/23)
:: ./client user3 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 3 user1,user2
Server connection successful
User 'user2' is now IDLE
User 'user1' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user2' is now IDLE
User 'user1' is now IDLE
```

<u>Test Case 2:</u> Server rejects a client with a duplicate username

SERVER

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:26:30 10/02/23)
:: ./server 127.0.0.1 2222 3
Server socket is created.
Binding successful.
Listening successful.
Server accepted the client: user1
User 'user1' is idle
User 'user1' is idle
User 'user1' is idle
Server accepted the client: user2
User 'user1' is idle
User 'user2' is idle
User 'user1' is idle
User 'user2' is idle
Another client with the username 'user2' is trying to connect, but the username already exists. Rejecting it.
```

USER 1

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:27:45 10/02/23)
:: ./client user1 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 1
Server connection successful
Time out!! No user input for 10 secs 0 usecs
Time out!! No user input for 10 secs 0 usecs
Time out!! No user input for 10 secs 0 usecs
User 'user2' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user2' is now IDLE
```

USER 2

```
:: ./client user2 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 2 user1
Server connection successful
User 'user1' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
Time out!! No user input for 10 secs 0 usecs
```

USER 3

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:28:43 10/02/23)
:: ./client user2 127.0.0.1 2222
Socket creation is successful
Disconnected.NAK Message from Server is Username is incorrect

[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:28:55 10/02/23)
::
```

Test Case 3: Server allows a previously used username to be reused

SERVER

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:37:48 10/02/23)
:: ./server 127.0.0.1 2222 3
Server socket is created.
Binding successful.
Listening successful.
Server accepted the client: user1
Server accepted the client: user2
User 'user1' is idle
Server accepted the client: user3
User 'user2' is idle
Socket 4 belonging to user 'user1' is disconnected
Server accepted the client: user1
User 'user3' is idle
User 'user2' is idle
User 'user1' is idle
```

USER 1

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:35:14 10/02/23)
:: ./client user1 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 1
Server connection successful
User 'user2' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now ONLINE
User 'user2' is now IDLE
^C

[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:38:43 10/02/23)
:: ./client user1 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 3 user2,user3
Server connection successful
User 'user3' is now IDLE
User 'user2' is now IDLE
```

USER 2

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:37:48 10/02/23)
:: ./client user2 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 2 user1
Server connection successful
User 'user1' is now IDLE
User 'user3' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now OFFLINE
User 'user1' is now ONLINE
User 'user3' is now IDLE
Time out!! No user input for 10 secs 0 usecs
```

<u>Test Case 4:</u> Server rejects the client because it exceeds the maximum number of clients allowed (In this case 3)

SERVER

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:37:48 10/02/23)
:: ./client user2 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 2 user1
Server connection successful
User 'user1' is now IDLE
User 'user3' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now OFFLINE
User 'user1' is now ONLINE
User 'user3' is now IDLE
Time out!! No user input for 10 secs 0 usecs
```

USER 1

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:41:45 10/02/23)
:: ./client user1 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 1
Server connection successful
User 'user2' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now ONLINE
User 'user2' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now IDLE
User 'user2' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now IDLE
User 'user2' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now IDLE
User 'user2' is now IDLE
```

USER 2

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:41:45 10/02/23)
:: ./client user2 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 2 user1
Server connection successful
User 'user1' is now IDLE
User 'user3' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
User 'user3' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
User 'user3' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
User 'user3' is now IDLE
```

USER 3

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:41:45 10/02/23)
:: ./client user3 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 3 user1,user2
Server connection successful
User 'user2' is now IDLE
User 'user1' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user2' is now IDLE
User 'user1' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user2' is now IDLE
User 'user1' is now IDLE
```

USER 4

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:37:48 10/02/23)
:: ./client user4 127.0.0.1 2222
Socket creation is successful
Disconnected.NAK Message from Server is Client count exceeded

[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:44:19 10/02/23)
::
```

<u>Test Case 5:</u> BONUS FEATURES IMPLEMENTED

(1) ACK, NAK, ONLINE, and OFFLINE. The server uses the ACK message to provide an explicit confirmation of the client's JOIN.

(2) The client sends an IDLE message to the server when the user doesn't use the chat session for more than 10 seconds.

SERVER

```
:: ./server 127.0.0.1 2222 3
Server socket is created.
Binding successful.
Listening successful.
Server accepted the client: user1
Server accepted the client: user2
User 'user1' is idle
Server accepted the client: user3
User 'user2' is idle
User 'user1' is idle
A new client is trying to connect, but the client count exceeded. Rejecting it
User 'user3' is idle
User 'user2' is idle
User 'user1' is idle
Socket 5 belonging to user 'user2' is disconnected
```

USER 1

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (14:41:39 10/02/23)
:: ./client user1 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 1
Server connection successful
User 'user2' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now ONLINE
User 'user2' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user3' is now IDLE
User 'user2' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user2' is now OFFLINE
```

USER 2

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (14:41:39 10/02/23)
:: ./client user2 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 2 user1
Server connection successful
User 'user1' is now IDLE
User 'user3' is now ONLINE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
User 'user3' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
^C
```

USER 3

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (14:41:39 10/02/23)
:: ./client user3 127.0.0.1 2222
Socket creation is successful
ACK Message from Server is 3 user1,user2
Server connection successful
User 'user2' is now IDLE
User 'user1' is now IDLE
Time out!! No user input for 10 secs 0 usecs
User 'user2' is now IDLE
User 'user1' is now IDLE
User 'user2' is now OFFLINE
Time out!! No user input for 10 secs 0 usecs
User 'user1' is now IDLE
Time out!! No user input for 10 secs 0 usecs
```

USER 4

```
[rohan.dalvi]@hera3 ~/ECEN602/Test> (13:44:19 10/02/23)
:: ./client user3 127.0.0.1 2222
Socket creation is successful
Disconnected.NAK Message from Server is Client count exceeded

[rohan.dalvi]@hera3 ~/ECEN602/Test> (14:45:21 10/02/23)
::
```