BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	**Shivam Santosh Kadam**
UID no.	**2023300099**
Experiment No.	**4(Home)**

AIM:	Network Socket Programming
OBJECTIVE:	The objective of this experiment is to make students acquainted with socket programming. And make them accustomed with applications executing on top of these sockets.
Part 3	
PROBLEM STATEMENT :	Write a client-server application for chat server. The two clients connected to the same server should be able to communicate with each other. Communication should be interactive and go on till one of them terminates.

THEORY:

- A **server** listens on port 5002 and manages communication between **two clients**. It ensures that the communication happens only if both clients are connected.
- A **client** connects to the server and sends/receives messages from the other client. It waits for the server to notify when it can start chatting (i.e., when both clients are connected).

Server:

Initialization:

1. **Socket creation:** The server creates a TCP socket using socket(AF_INET, SOCK_STREAM, 0).
2. **Binding:** The server binds the socket to port 5002 and any available local IP address (INADDR_ANY).
3. **Listening:** The server listens for incoming client connections with listen(server_fd, 3).

Connection Handling

1. **Accepting connections:**
   - The server accepts connections from clients using accept(). When a client connects, the server assigns it to an available slot in the client_sockets[] array.
   - The client's IP address is stored in client_ips[].
2. **Client assignment:**
   - If two clients are already connected, the server sends a message to the new client informing that the chat room is full and closes the connection.

- If space is available, the server assigns the new client to the first available slot (either client_sockets[0] or client_sockets[1]).

3. **Handling multiple clients:**
   - Each client is handled by a separate thread (pthread_create), ensuring that both clients can send and receive messages concurrently.
   - The handle_client() function handles the communication and disconnection of each client.

Thread Management

- Each client is handled in a separate thread, which allows for concurrent communication.
- After a thread is finished (client disconnects), it is detached to clean up resources (pthread_detach()).

Client Disconnection

- When a client disconnects, the server closes the socket, sets its entry in client_sockets[] to -1, and notifies the other client that the chat is disabled until a new client connects.

Client:

Initialization

1. **Socket creation:** The client creates a socket using socket(AF_INET, SOCK_STREAM, 0) to connect to the server.
2. **Server connection:**
   - The client connects to the server at 127.0.0.1 on port 5002 using connect().
   - If the connection is successful, the client starts listening for messages from the server.

Message Receiving

- The client creates a separate thread (pthread_create) that listens for incoming messages using recv().
- If a message indicates that both clients are connected, the client updates the chat_enabled flag to 1 (allowing message sending).
- If a message indicates that the other client is not connected or the client was disconnected, chat_enabled is set to 0, preventing message sending.

Message Sending

1. **Input reading:**
   - The client reads messages from the console using fgets().
   - Before sending a message, the client checks if chat is enabled using the chat_enabled flag.
   - If chat is not enabled, the client prints a message that it cannot send messages and prompts for new input.
2. **Message sending:**
   - If chat is enabled, the client sends the message to the server using send(). If the user types exit,

the client sends the message and then exits.

Thread Handling: The client's receive thread runs concurrently with the main program, allowing it to receive messages while the user types and sends messages.

PROGRAM:

Server3.c

```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>
#include <pthread.h>

#define PORT 5002
#define BUFFER_SIZE 1024

int client_sockets[2] = { -1, -1 };
char client_ips[2][INET_ADDRSTRLEN] = {"", ""};
pthread_t client_threads[2];
pthread_mutex_t lock;

void send_to_client(int index, const char *message) {
    if (index >= 0 && index < 2 && client_sockets[index] != -1) {
        send(client_sockets[index], message, strlen(message), 0);
    }
}

void notify_both(const char *message) {
    for (int i = 0; i < 2; i++) {
        if (client_sockets[i] != -1) {
            send(client_sockets[i], message, strlen(message), 0);
        }
    }
}

void *handle_client(void *arg) {
```

```
    int idx = *(int *)arg;
    free(arg);

    char buffer[BUFFER_SIZE];
    char msg[BUFFER_SIZE + 50];

    pthread_mutex_lock(&lock);
    int other = (idx == 0) ? 1 : 0;
    if (client_sockets[other] == -1) {
        send_to_client(idx, "Waiting for the other client to
connect...\n");
    }
    else {

        snprintf(msg, sizeof(msg),
                 "Both clients connected:\n  Client 1 (IP: %s)\n
Client 2 (IP: %s)\n",
                 client_ips[0], client_ips[1]);
        notify_both(msg);
    }
    pthread_mutex_unlock(&lock);


    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        ssize_t bytes_received = recv(client_sockets[idx], buffer,
BUFFER_SIZE - 1, 0);
        if (bytes_received <= 0) {

            pthread_mutex_lock(&lock);
            printf("Client %d (IP: %s) disconnected.\n", idx + 1,
client_ips[idx]);
            close(client_sockets[idx]);
            client_sockets[idx] = -1;

            other = (idx == 0) ? 1 : 0;
            if (client_sockets[other] != -1) {
                snprintf(msg, sizeof(msg), "Client %d (IP: %s) has
```

```
disconnected.\nChat disabled until another client connects.\n",
                        idx + 1, client_ips[idx]);
                send_to_client(other, msg);
            }
            pthread_mutex_unlock(&lock);
            break;
        }
        buffer[bytes_received] = '\0';

        pthread_mutex_lock(&lock);
        other = (idx == 0) ? 1 : 0;
        if (client_sockets[other] == -1) {
            send_to_client(idx, "Other client is not connected. You
cannot send messages until both are present.\n");
            pthread_mutex_unlock(&lock);
            continue;
        }
        snprintf(msg, sizeof(msg), "Client %d: %s", idx + 1,
buffer);
        send(client_sockets[other], msg, strlen(msg), 0);
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

int main() {
    int server_fd;
    struct sockaddr_in address;
    socklen_t addr_len = sizeof(address);

    pthread_mutex_init(&lock, NULL);

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    address.sin_family = AF_INET;
```

```
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address,
sizeof(address)) < 0) {
        perror("Bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    printf("Chat Server (modified Part-3) listening on port %d...\n",
PORT);

    while (1) {
        struct sockaddr_in client_addr;
        socklen_t client_addr_len = sizeof(client_addr);
        int new_socket = accept(server_fd, (struct sockaddr
*)&client_addr, &client_addr_len);
        if (new_socket < 0) {
            perror("Accept failed");
            continue;
        }

        char ip_str[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &client_addr.sin_addr, ip_str,
INET_ADDRSTRLEN);

        pthread_mutex_lock(&lock);
        int assigned = -1;
        for (int i = 0; i < 2; i++) {
            if (client_sockets[i] == -1) {
                client_sockets[i] = new_socket;
                strncpy(client_ips[i], ip_str, INET_ADDRSTRLEN - 1);
```

```
                assigned = i;
                printf("Client %d (IP: %s) connected.\n", i + 1,
client_ips[i]);
                break;
            }
        }
        pthread_mutex_unlock(&lock);

        if (assigned == -1) {
            char full_msg[] = "Chat room full. Only 2 clients are
allowed at a time. Try again later.\n";
            send(new_socket, full_msg, strlen(full_msg), 0);
            close(new_socket);
        }
        else {
            int *pindex = malloc(sizeof(int));
            *pindex = assigned;
            pthread_create(&client_threads[assigned], NULL,
handle_client, pindex);
            pthread_detach(client_threads[assigned]);
        }
    }

    close(server_fd);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

Client3.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>
#include <pthread.h>
```

```c
#define PORT 5002
#define BUFFER_SIZE 1024

int sock;
volatile int chat_enabled = 0; // Flag: 0 = not allowed to chat, 1 = allowed
pthread_mutex_t flag_lock = PTHREAD_MUTEX_INITIALIZER;

void *receive_messages(void *arg) {
    char buffer[BUFFER_SIZE];
    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        ssize_t bytes_received = recv(sock, buffer, BUFFER_SIZE - 1, 0);
        if (bytes_received <= 0) {
            printf("Server closed connection or error occurred.\n");
            break;
        }
        buffer[bytes_received] = '\0';
        printf("%s", buffer);

        pthread_mutex_lock(&flag_lock);
        if (strstr(buffer, "Both clients connected") != NULL)
            chat_enabled = 1;
        if (strstr(buffer, "Waiting for the other client") != NULL ||
            strstr(buffer, "has disconnected") != NULL ||
            strstr(buffer, "cannot send messages") != NULL)
            chat_enabled = 0;
        pthread_mutex_unlock(&flag_lock);
    }
    return NULL;
}

int main() {
    struct sockaddr_in serv_addr;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
```

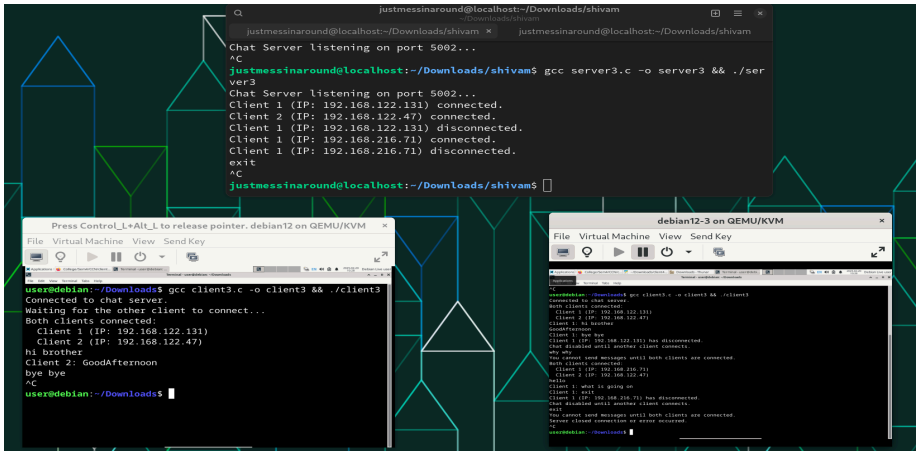
```
        perror("Socket creation error");
        exit(EXIT_FAILURE);
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        perror("Invalid address");
        close(sock);
        exit(EXIT_FAILURE);
    }

    if (connect(sock, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0) {
        perror("Connection failed");
        close(sock);
        exit(EXIT_FAILURE);
    }
    printf("Connected to chat server (modified Part-3).\n");

    pthread_t recv_thread;
    pthread_create(&recv_thread, NULL, receive_messages, NULL);
    pthread_detach(recv_thread);

    char input[BUFFER_SIZE];
    while (1) {
        memset(input, 0, BUFFER_SIZE);
        if (!fgets(input, BUFFER_SIZE, stdin)) {
            break;
        }

        pthread_mutex_lock(&flag_lock);
        int enabled = chat_enabled;
        pthread_mutex_unlock(&flag_lock);
        if (!enabled) {
            printf("You cannot send messages until both clients are
connected.\n");
```

```
            continue;
        }

        if (strncmp(input, "exit", 4) == 0) {
            send(sock, input, strlen(input), 0);
            break;
        }
        send(sock, input, strlen(input), 0);
    }

    close(sock);
    return 0;
}
```

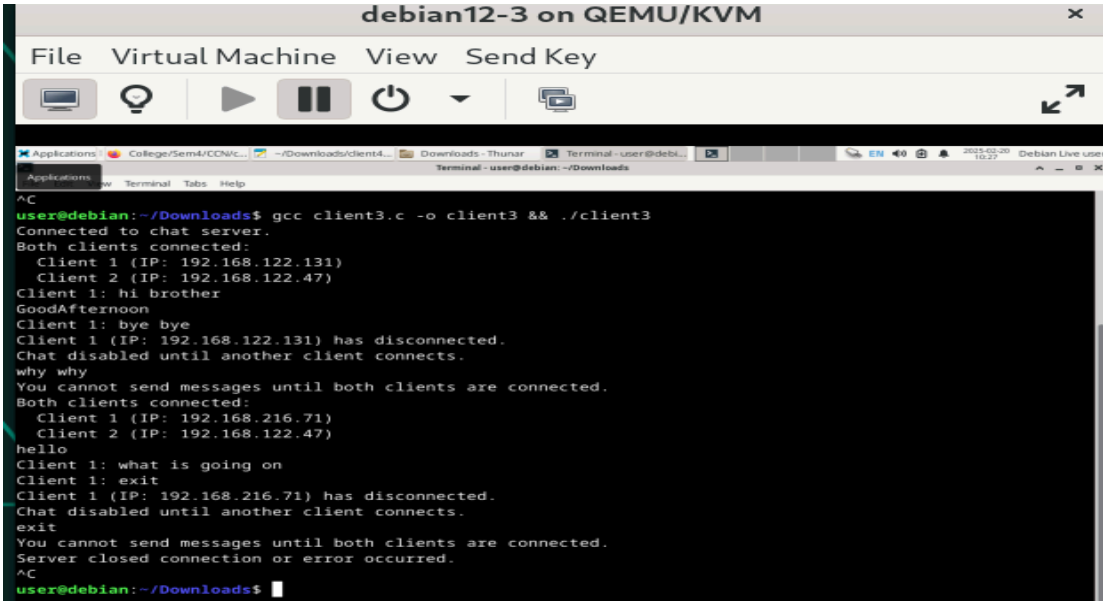
RESULT:

```
Chat Server listening on port 5002...
^C
justmessinaround@localhost:~/Downloads/shivam$ gcc server3.c -o server3 && ./ser
ver3
Chat Server listening on port 5002...
Client 1 (IP: 192.168.122.131) connected.
Client 2 (IP: 192.168.122.47) connected.
Client 1 (IP: 192.168.122.131) disconnected.
Client 1 (IP: 192.168.216.71) connected.
Client 1 (IP: 192.168.216.71) disconnected.
exit
^C
```

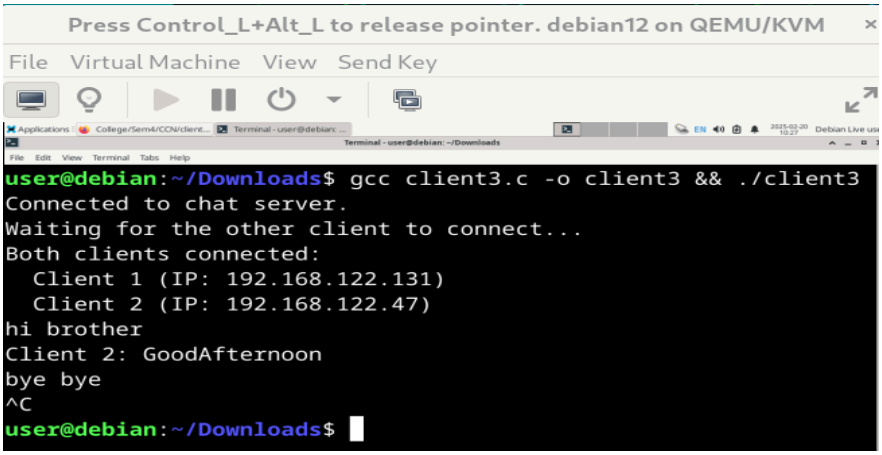
Server



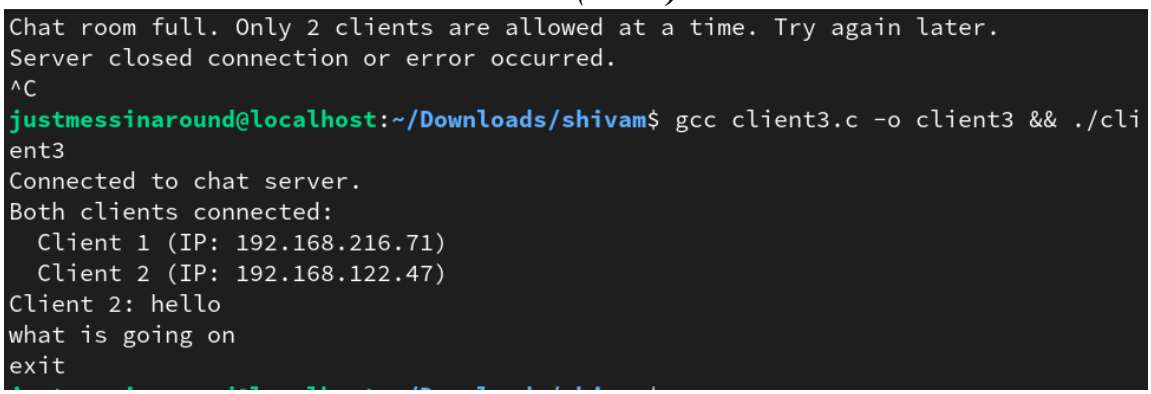
Overview of client-server communication in a two-person chat



Client 1



Client 2(Initial)



Client 3(new client2)

BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Part 4	
PROBLEM STATEMENT:	Upgrade the code in part-3 chat server to act as both group and direct chats.

THEORY:

- A **server** listens on port 5003 and allows communication between **multiple clients**. It supports both **group chat** and **direct messages** between clients.
- A **client** connects to the server and can send either group messages or private messages to other users.

Server:

Client Management

1. **Accepting Connections:** The server waits for clients to connect using `accept()`. It stores client information in the `clients[]` array.
2. **Client Assignment:** If a client connects, the server assigns them to the first available slot in `clients[]`. If the server is at full capacity (10 clients), the new client is rejected.
3. **Thread Handling:** The server listens for messages from a client using `recv()` and forwards the message to the appropriate recipients.

Message Handling

1. **Broadcast Messages:** The server supports group chat by broadcasting messages to all clients except the sender using `broadcast_message()`.
2. **Direct Messages:** To send a private message to another user, the client prefixes their message with `@username`. The server then sends the message to the target client using `direct_message()`.

Server Shutdown: The server listens for a termination signal (`Ctrl+C`) and closes all active connections using `close()` when shutting down. It notifies clients about the shutdown.

Client:

Message Receiving

The client runs a separate thread (`pthread_create`) to continuously listen for incoming messages from the server using `recv()`.

Received messages are printed to the console. If the message is a direct message or group chat message, the client displays it accordingly.

Message Sending

1. **Input Handling:** The client reads messages from the user using `fgets()`. The user can send either a group message or a direct message.
2. **Message Formatting:** If the user types a message starting with `@`, the client formats it as a direct message to the specified user. Otherwise, it sends the message to the group.
3. **Message Sending:** The client sends messages to the server using `send()`. If the user types `exit`, the client disconnects from the server.

PROGRAM:

Server4.c

```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>
#include <pthread.h>
#include <signal.h>

#define PORT 5003
#define BUFFER_SIZE 1024
#define MAX_CLIENTS 10
#define USERNAME_LEN 32

typedef struct {
    int socket;
    char username[USERNAME_LEN];
    char ip[INET_ADDRSTRLEN];
    int active;
} ClientInfo;

ClientInfo clients[MAX_CLIENTS];
pthread_mutex_t lock;

void broadcast_message(const char *message, int exclude_sock) {
    pthread_mutex_lock(&lock);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].active && clients[i].socket != exclude_sock)
```

```
{
            send(clients[i].socket, message, strlen(message), 0);
        }
    }
    pthread_mutex_unlock(&lock);
}

void direct_message(const char *sender, const char *recipient, const
char *message) {
    pthread_mutex_lock(&lock);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].active && strcmp(clients[i].username,
recipient) == 0) {
            char dm_buffer[BUFFER_SIZE + 50];
            snprintf(dm_buffer, sizeof(dm_buffer), "[DM from %s]:
%s\n", sender, message);
            send(clients[i].socket, dm_buffer, strlen(dm_buffer),
0);
            break;
        }
    }
    pthread_mutex_unlock(&lock);
}

void remove_client(int index) {
    pthread_mutex_lock(&lock);
    clients[index].active = 0;
    close(clients[index].socket);
    pthread_mutex_unlock(&lock);
}

void send_user_list(int client_socket, int self_index) {
    char list_buffer[BUFFER_SIZE];
    memset(list_buffer, 0, BUFFER_SIZE);
    strcat(list_buffer, "Currently connected users: [");
    int first = 1;
    pthread_mutex_lock(&lock);
    for (int i = 0; i < MAX_CLIENTS; i++) {
```

```
        if (clients[i].active && i != self_index) {
            if (!first) {
                strcat(list_buffer, ", ");
            }
            strcat(list_buffer, clients[i].username);
            first = 0;
        }
    }
    pthread_mutex_unlock(&lock);
    if (first) {
        strcat(list_buffer, "None");
    }
    strcat(list_buffer, "]\n");
    send(client_socket, list_buffer, strlen(list_buffer), 0);
}

void *client_handler(void *arg) {
    int index = *(int *)arg;
    free(arg);

    int sock = clients[index].socket;
    char username[USERNAME_LEN];
    strcpy(username, clients[index].username);
    char buffer[BUFFER_SIZE];
    char msg[BUFFER_SIZE + 50];

    // Send the list of currently connected users to this new
client.
    send_user_list(sock, index);

    // Send welcome message.
    char welcome_msg[BUFFER_SIZE];
    snprintf(welcome_msg, sizeof(welcome_msg),
             "Welcome %s! Type '@username message' for a direct
message, or just type to broadcast.\n",
             username);
    send(sock, welcome_msg, strlen(welcome_msg), 0);
```

```
    snprintf(msg, sizeof(msg), "%s has joined the chat.\n",
username);
    broadcast_message(msg, sock);

    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        ssize_t valread = recv(sock, buffer, BUFFER_SIZE - 1, 0);
        if (valread <= 0) {
            pthread_mutex_lock(&lock);
            printf("Client %s (IP: %s) disconnected.\n", username,
clients[index].ip);
            pthread_mutex_unlock(&lock);
            remove_client(index);
            snprintf(msg, sizeof(msg), "%s has left the chat.\n",
username);
            broadcast_message(msg, -1);
            break;
        }
        buffer[valread] = '\0';

        if (strncmp(buffer, "exit", 4) == 0) {
            pthread_mutex_lock(&lock);
            printf("Client %s (IP: %s) exited.\n", username,
clients[index].ip);
            pthread_mutex_unlock(&lock);
            remove_client(index);
            snprintf(msg, sizeof(msg), "%s has left the chat.\n",
username);
            broadcast_message(msg, -1);
            break;
        }

        if (buffer[0] == '@') {
            char *space_ptr = strchr(buffer, ' ');
            if (space_ptr != NULL) {
                *space_ptr = '\0';
                char *recipient = buffer + 1;  // Skip '@'
                char *dm_message = space_ptr + 1;
```

```
                direct_message(username, recipient, dm_message);
            }
        }
        else {
            snprintf(msg, sizeof(msg), "%s: %s", username, buffer);
            broadcast_message(msg, sock);
        }
    }
    return NULL;
}

void handle_signal(int sig) {
    printf("\nServer shutting down. Closing all connections.\n");
    pthread_mutex_lock(&lock);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i].active) {
            send(clients[i].socket, "Server is shutting down. Chat
will end.\n", 43, 0);
            close(clients[i].socket);
            clients[i].active = 0;
        }
    }
    pthread_mutex_unlock(&lock);
    exit(0);
}

int main() {
    int server_fd;
    struct sockaddr_in address;
    socklen_t addr_len = sizeof(address);

    pthread_mutex_init(&lock, NULL);

    for (int i = 0; i < MAX_CLIENTS; i++) {
        clients[i].active = 0;
        clients[i].socket = -1;
        memset(clients[i].username, 0, USERNAME_LEN);
        memset(clients[i].ip, 0, INET_ADDRSTRLEN);
```

```
    }

    // Setup signal handler for Ctrl-C.
    signal(SIGINT, handle_signal);

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY; // Listen on all
interfaces
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address,
sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, MAX_CLIENTS) < 0) {
        perror("listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    printf("Group/Direct Chat Server (Part-4) listening on port
%d...\n", PORT);

    while (1) {
        struct sockaddr_in client_addr;
        socklen_t client_addr_len = sizeof(client_addr);
        int new_socket = accept(server_fd, (struct sockaddr
*)&client_addr, &client_addr_len);
        if (new_socket < 0) {
            perror("accept failed");
            continue;
```

```
        }

        char ip_str[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &client_addr.sin_addr, ip_str,
INET_ADDRSTRLEN);

        char prompt[] = "Enter your name: ";
        send(new_socket, prompt, strlen(prompt), 0);

        char name_buf[USERNAME_LEN];
        memset(name_buf, 0, USERNAME_LEN);
        ssize_t name_read = recv(new_socket, name_buf, USERNAME_LEN
- 1, 0);
        if (name_read <= 0) {
            close(new_socket);
            continue;
        }
        name_buf[name_read] = '\0';
        char *nl = strchr(name_buf, '\n');
        if (nl) *nl = '\0';

        pthread_mutex_lock(&lock);
        int taken = 0;
        for (int i = 0; i < MAX_CLIENTS; i++) {
            if (clients[i].active && strcmp(clients[i].username,
name_buf) == 0) {
                taken = 1;
                break;
            }
        }
        pthread_mutex_unlock(&lock);

        if (taken) {
            char err_msg[] = "Name already taken. Disconnecting.\n";
            send(new_socket, err_msg, strlen(err_msg), 0);
            close(new_socket);
            continue;
        }
```

```
        pthread_mutex_lock(&lock);
        int index = -1;
        for (int i = 0; i < MAX_CLIENTS; i++) {
            if (!clients[i].active) {
                index = i;
                clients[i].active = 1;
                clients[i].socket = new_socket;
                strncpy(clients[i].username, name_buf, USERNAME_LEN
- 1);
                strncpy(clients[i].ip, ip_str, INET_ADDRSTRLEN - 1);
                break;
            }
        }
        pthread_mutex_unlock(&lock);

        if (index == -1) {
            char err_msg[] = "Server full. Disconnecting.\n";
            send(new_socket, err_msg, strlen(err_msg), 0);
            close(new_socket);
            continue;
        }

        printf("Client %s connected from IP %s (Socket %d).\n",
name_buf, ip_str, new_socket);

        pthread_t tid;
        int *pindex = malloc(sizeof(int));
        *pindex = index;
        pthread_create(&tid, NULL, client_handler, pindex);
        pthread_detach(tid);
    }

    close(server_fd);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

Client4.c

```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>
#include <pthread.h>

#define PORT 5003
#define BUFFER_SIZE 1024

int sock;

void *receive_messages(void *arg) {
    char buffer[BUFFER_SIZE];
    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        ssize_t valread = recv(sock, buffer, BUFFER_SIZE - 1, 0);
        if (valread <= 0) {
            printf("Server closed connection. Exiting chat.\n");
            exit(0);
        }
        buffer[valread] = '\0';
        printf("%s", buffer);
    }
    return NULL;
}

int main() {
    struct sockaddr_in serv_addr;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation error");
        exit(EXIT_FAILURE);
    }
```

```
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        perror("Invalid address");
        close(sock);
        exit(EXIT_FAILURE);
    }

    if (connect(sock, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0) {
        perror("Connection failed");
        close(sock);
        exit(EXIT_FAILURE);
    }
    printf("Connected to Group/Direct Chat Server (Part-4).\n");

    char prompt[BUFFER_SIZE];
    ssize_t valread = recv(sock, prompt, BUFFER_SIZE - 1, 0);
    if (valread > 0) {
        prompt[valread] = '\0';
        printf("%s", prompt);
    }

    char name_buf[BUFFER_SIZE];
    fgets(name_buf, BUFFER_SIZE, stdin);
    send(sock, name_buf, strlen(name_buf), 0);

    pthread_t recv_thread;
    pthread_create(&recv_thread, NULL, receive_messages, NULL);
    pthread_detach(recv_thread);

    char buffer[BUFFER_SIZE];
    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        if (!fgets(buffer, BUFFER_SIZE, stdin))
            break;
```

```
        if (strncmp(buffer, "exit", 4) == 0) {
            send(sock, buffer, strlen(buffer), 0);
            break;
        }
        send(sock, buffer, strlen(buffer), 0);
    }

    close(sock);
    return 0;
}
```

RESULT:

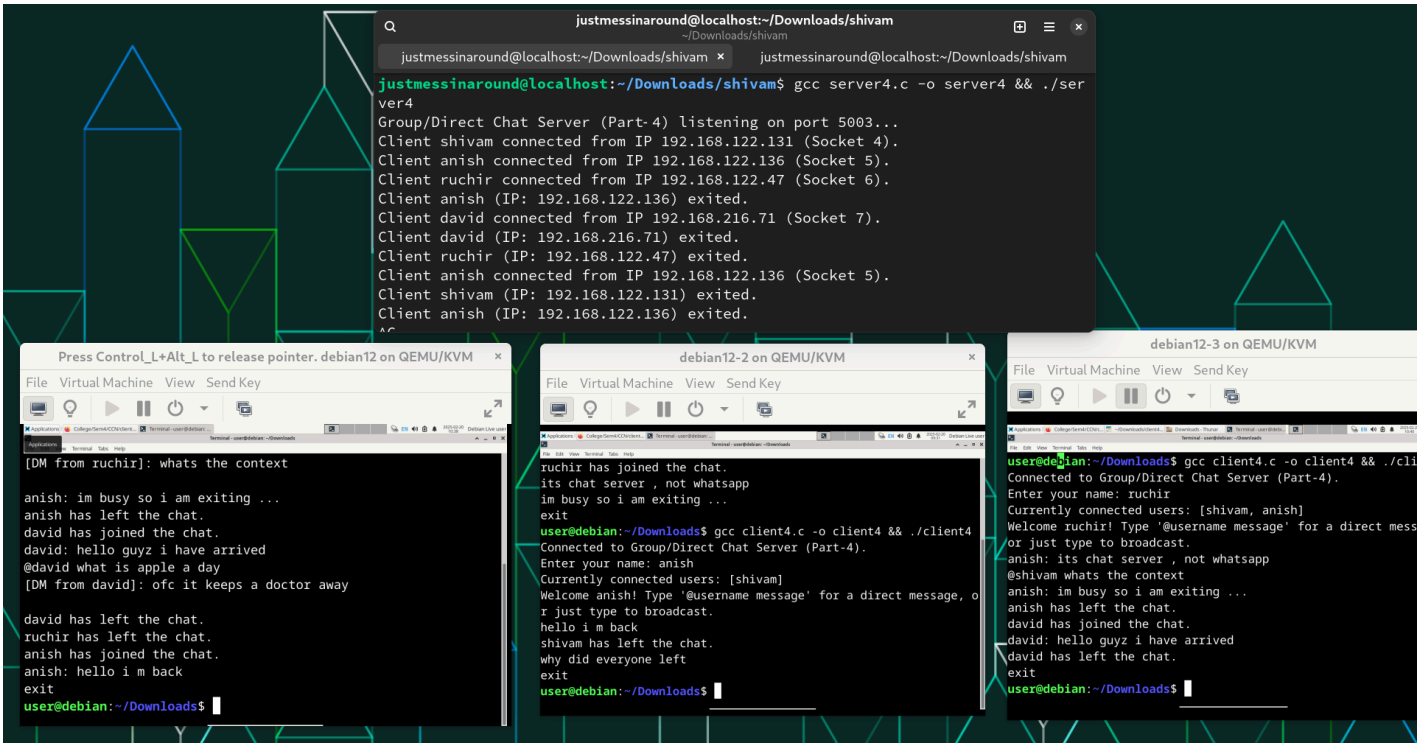
```
justmessinaround@localhost:~/Downloads/shivam$ gcc server4.c -o server4 && ./server4
Group/Direct Chat Server (Part-4) listening on port 5003...
Client shivam connected from IP 192.168.122.131 (Socket 4).
Client anish connected from IP 192.168.122.136 (Socket 5).
Client ruchir connected from IP 192.168.122.47 (Socket 6).
Client anish (IP: 192.168.122.136) exited.
Client david connected from IP 192.168.216.71 (Socket 7).
Client david (IP: 192.168.216.71) exited.
Client ruchir (IP: 192.168.122.47) exited.
Client anish connected from IP 192.168.122.136 (Socket 5).
Client shivam (IP: 192.168.122.131) exited.
Client anish (IP: 192.168.122.136) exited.
^C
Server shutting down. Closing all connections.
```

Server

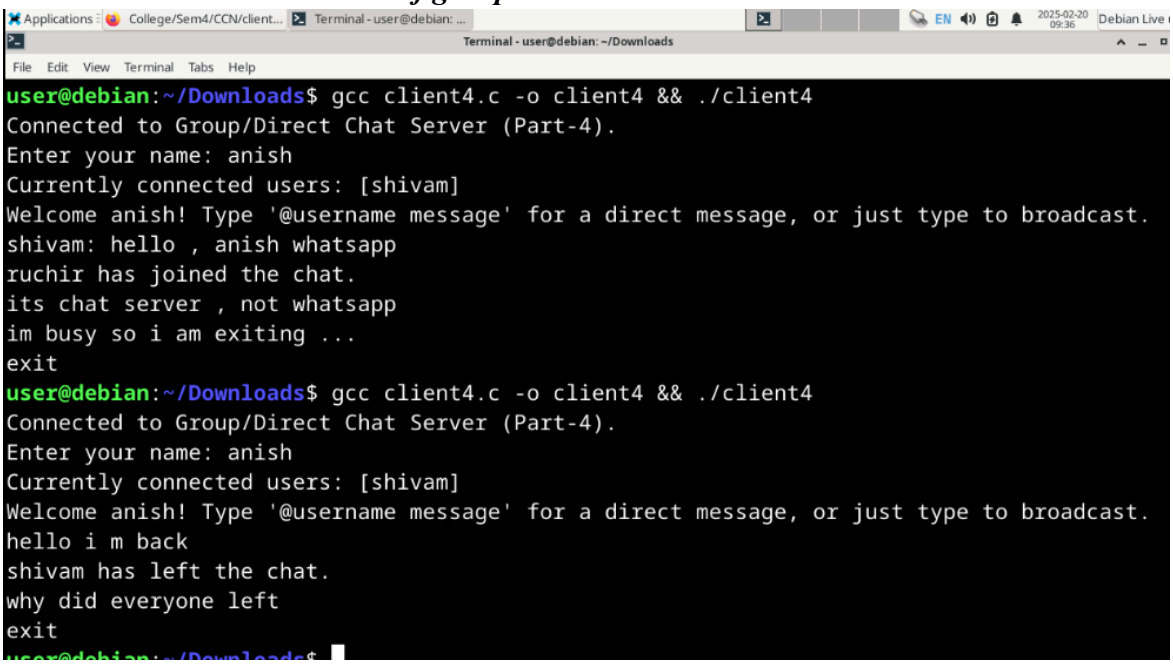
```
Applications  Firefox  Terminal - user@debian: ...  xfce4-screenshooter
Terminal - user@debian: ~/Downloads
File  Edit  View  Terminal  Tabs  Help
user@debian:~/Downloads$ gcc client4.c -o client4 && ./client4
Connected to Group/Direct Chat Server (Part-4).
Enter your name: shivam
Currently connected users: [None]
Welcome shivam! Type '@username message' for a direct message, or just type to broadcast.
anish has joined the chat.
hello , anish whatsapp
ruchir has joined the chat.
anish: its chat server , not whatsapp
[DM from ruchir]: whats the context

anish: im busy so i am exiting ...
anish has left the chat.
david has joined the chat.
david: hello guyz i have arrived
@david what is apple a day
[DM from david]: ofc it keeps a doctor away

david has left the chat.
ruchir has left the chat.
anish has joined the chat.
anish: hello i m back
exit
```

Client1(shivam)

BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering



Overview of group-chat server-client interaction



Client2(anish)

BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

```
user@debian:~/Downloads$ gcc client4.c -o client4 && ./client4
Connected to Group/Direct Chat Server (Part-4).
Enter your name: ruchir
Currently connected users: [shivam, anish]
Welcome ruchir! Type '@username message' for a direct message, or just type to broadcast.
anish: its chat server , not whatsapp
@shivam whats the context
anish: im busy so i am exiting ...
anish has left the chat.
david has joined the chat.
david: hello guyz i have arrived
david has left the chat.
exit
```

Client3(ruchir)

```
justmessinaround@localhost:~/Downloads/shivam$ gcc client4.c -o client4 && ./client4

Connected to Group/Direct Chat Server (Part-4).
Enter your name: david
Currently connected users: [shivam, ruchir]
Welcome david! Type '@username message' for a direct message, or just type to broadcast.
hello guyz i have arrived
[DM from shivam]: what is apple a day

@shivam ofc it keeps a doctor away
exit
```

Client4(david)

CONCLUSION:	In this experiment, I learned the basics of socket programming, implementing both group and direct chat in a client-server model. I used multi-threading, message broadcasting, and direct messaging to manage client communication. The server handled multiple clients simultaneously, ensuring smooth interaction through mechanisms like mutex locks to avoid race conditions. The client, on the other hand, used threads to continuously receive messages while allowing the user to send either group or private messages.