Tushar Deshmukh
Final Year CSE PE27
1032201698

TITLE:

Write a program to implement Echo server using socket programming

AIM:

To implement Client-Server architecture as echo server using

1. Socket programming
2. Multi-threading

OBJECTIVE:

To understand the concept of socket programming, multi-threading, and echo servers.

THEORY:

Most inter process communication uses the client server model. One of the two processes, the client, typically to make a request for information. The system calls for establishing a connection for the client and server are as follows:
Client side:

1. Socket ()
2. Connect ()
3. Read ()
4. Write ()

Server Side:

1. Socket ()
2. Bind ()
3. Listen ()
4. Accept ()
5. Read ()
6. Write ()

SOCKET TYPES:

When a socket is created, the program has to specify the address domain and the socket type. Two processes can communicate with each other only if their sockets are of same type and in the same domain. There are two widely used domains:

1. UNIX domain
2. internet domain

There are two widely used socket types.

1. Stream sockets
2. Datagram sockets

Tushar Deshmukh
Final Year CSE PE27
1032201698

Stream sockets treat communication as continuous stream of characters, while datagram sockets have to read the entire message at once. Each uses its own communication protocol. Stream sockets use TCP, which is reliable, stream oriented protocol and datagram sockets are UDP, which is unreliable and message oriented.

SYSTEM CALL FORMATS:

1. int Socket (int family, int type, int protocol)

Type: f Socket type
    SOCK_DGRAM------- UDP
    SOCK_STREAM ----- TCP

Protocol: Type = 0-------------- TCP
    Type = 1-------------- UDP

The above call returns socket descriptor.

2. int Bind (int SOCK_FD,struct sockaddr *myadddr, int addrlen)
use:
    Binding socket with the address.

3. Listen (int SOCK_FD, int backlog)
Where,
Backlog = number of requests that can be queued up to the server .

4. Accept (int SOCK_FD, struct sockaddr *peer, int *addrlen)
Where,
int *adddrlen = length of the structure

5. Connect (int SOCK_FD, struct sock addr *servaddr, socklen_t addrlen)

RANGE OF PORTS:
1. Wellknown ports: 0-1023
2. Registered: 1024 – 59151
    (Controlled by IANA)
3. Dynamic (Ephemeral): 49152-65535

Reserved port in UNIX is any port < 1024

MULTITHREADED ECHO SERVER:

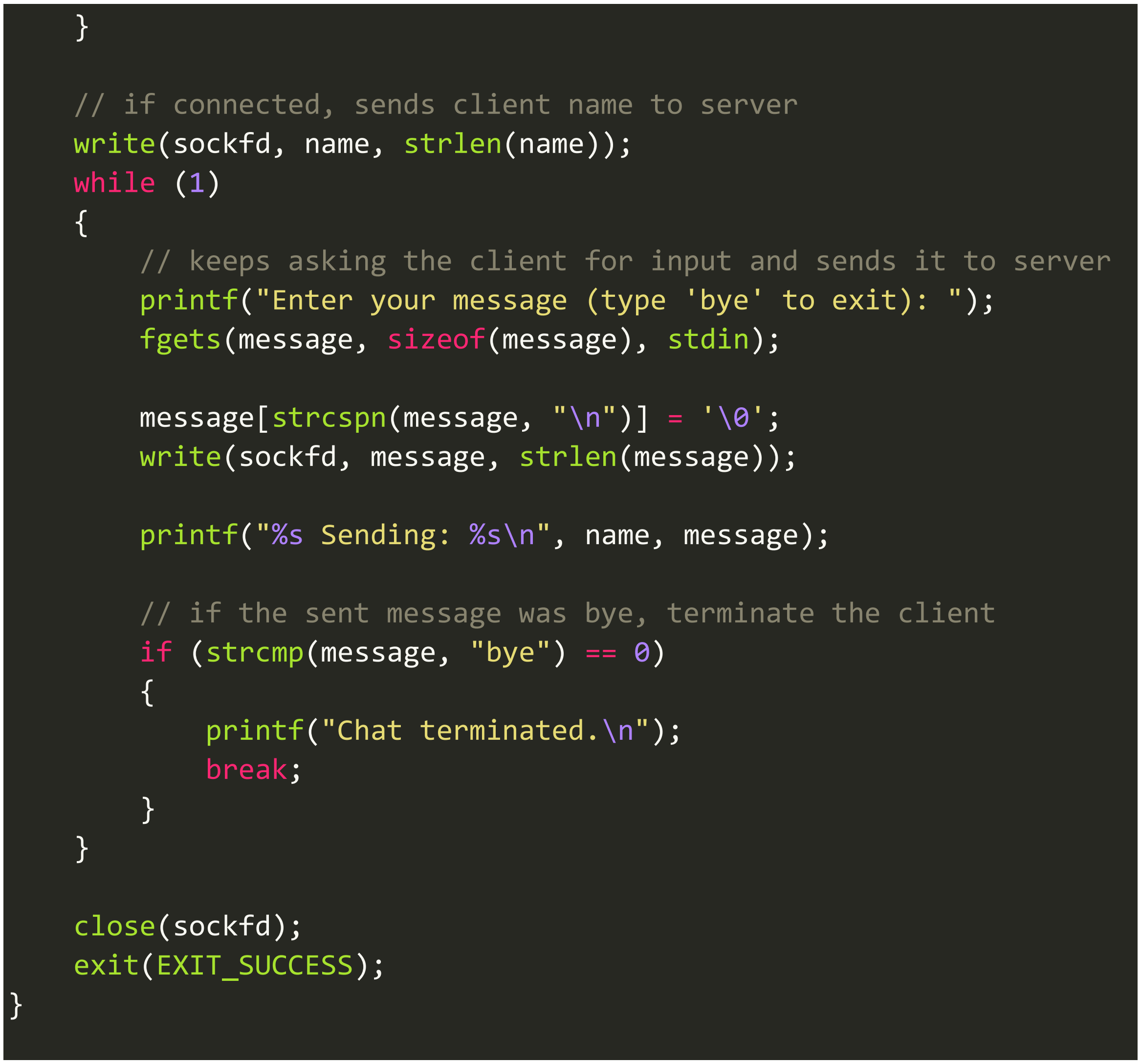
Multiple clients request for service to the same server. Server creates threads to serve the clients. Threads are light weight processes. It provides concurrency. On the server side, process keeps listening to the requests made by clients. As soon as the connection has to be made with clients, server creates the threads.

Tushar Deshmukh
Final Year CSE PE27
1032201698

CODE:

1. **Client**

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

int main()
{
    int sockfd;
    struct sockaddr_in server_address;
    char name[80], message[200];

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

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    // Error handling for socket creation
    if (sockfd == -1)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_address.sin_family = AF_INET;
    // change the inet_addr to server's ip and change the port to server's port
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_address.sin_port = htons(9129);

    // error handling for server connection
    if (connect(sockfd, (struct sockaddr *)&server_address, sizeof(server_address)) == -1)
    {
        perror("Connection failed");
        close(sockfd);
        exit(EXIT_FAILURE);
```

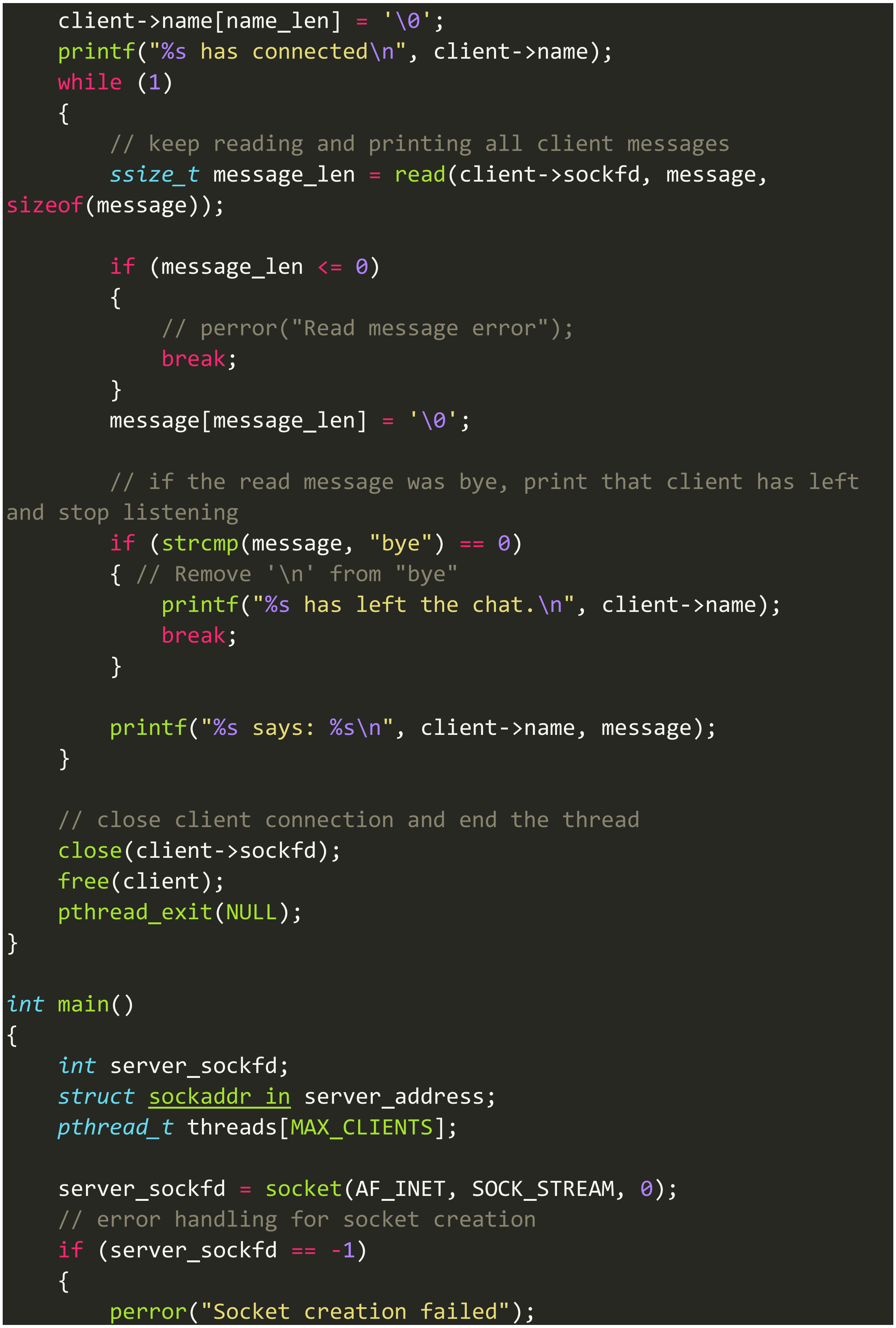


```
    }

    // if connected, sends client name to server
    write(sockfd, name, strlen(name));
    while (1)
    {
        // keeps asking the client for input and sends it to server
        printf("Enter your message (type 'bye' to exit): ");
        fgets(message, sizeof(message), stdin);

        message[strcspn(message, "\n")] = '\0';
        write(sockfd, message, strlen(message));

        printf("%s Sending: %s\n", name, message);

        // if the sent message was bye, terminate the client
        if (strcmp(message, "bye") == 0)
        {
            printf("Chat terminated.\n");
            break;
        }
    }

    close(sockfd);
    exit(EXIT_SUCCESS);
}
```

Tushar Deshmukh
Final Year CSE PE27
1032201698

2. Server

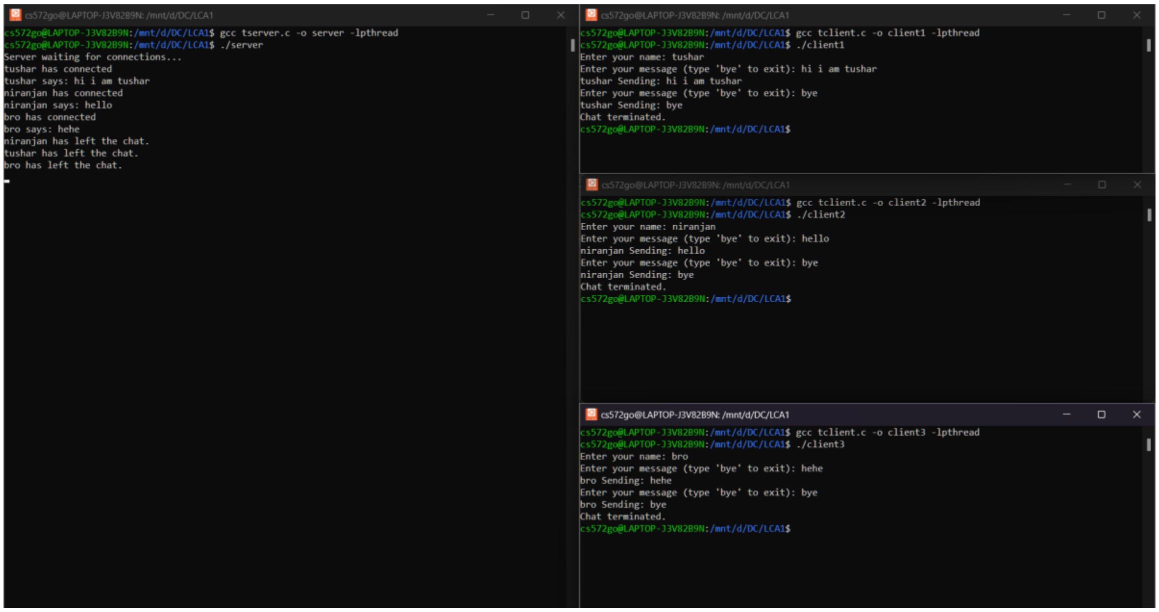
```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>

// max client threads allowd
#define MAX_CLIENTS 5

// global struct for client id and client name
struct client_info
{
    int sockfd;
    char name[80];
};

void *handle_client(void *arg)
{
    // client handling thread
    // stays alive till a client is connected
    // ends when client is terminated

    struct client_info *client = (struct client_info *)arg;
    char message[200];
    // int flag=0;

    // read client name once a client connects
    ssize_t name_len = read(client->sockfd, client->name,
sizeof(client->name));

    // error handling for name
    if (name_len <= 0)
    {
        // perror("Read name error");
        close(client->sockfd);
        free(client);
        pthread_exit(NULL);
    }
```



```
    client->name[name_len] = '\0';
    printf("%s has connected\n", client->name);
    while (1)
    {
        // keep reading and printing all client messages
        ssize_t message_len = read(client->sockfd, message,
sizeof(message));

        if (message_len <= 0)
        {
            // perror("Read message error");
            break;
        }
        message[message_len] = '\0';

        // if the read message was bye, print that client has left
and stop listening
        if (strcmp(message, "bye") == 0)
        { // Remove '\n' from "bye"
            printf("%s has left the chat.\n", client->name);
            break;
        }

        printf("%s says: %s\n", client->name, message);
    }

    // close client connection and end the thread
    close(client->sockfd);
    free(client);
    pthread_exit(NULL);
}

int main()
{
    int server_sockfd;
    struct sockaddr_in server_address;
    pthread_t threads[MAX_CLIENTS];

    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    // error handling for socket creation
    if (server_sockfd == -1)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_address.sin_family = AF_INET;
    // change the inet_addr to desired ip and change the port to
desired port
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_address.sin_port = htons(9129);

    // error handling for binding ip address
    if (bind(server_sockfd, (struct sockaddr *)&server_address,
sizeof(server_address)) == -1)
    {
        perror("Bind failed");
        close(server_sockfd);
        exit(EXIT_FAILURE);
    }

    // server has started and is waiting for clients
    printf("Server waiting for connections...\n");
    listen(server_sockfd, MAX_CLIENTS);

    while (1)
    {
        // keeps listening for clients
        // spawns a thread when new client connects
        // closes thread when client leaves
        struct sockaddr_in client_address;
        socklen_t client_len = sizeof(client_address);

        int client_sockfd = accept(server_sockfd, (struct sockaddr
*)&client_address, &client_len);

        // error handling for client connection to server
        if (client_sockfd == -1)
        {
            perror("Accept failed");
            continue;
        }

        // new client has connected
        struct client_info *client = (struct client_info
*)malloc(sizeof(struct client_info));
```

```
        // error handling for new client
        if (client == NULL)
        {
            perror("Memory allocation failed");
            close(client_sockfd);
            continue;
        }

        // set new client id and spawn thread
        client->sockfd = client_sockfd;
        pthread_create(&threads[MAX_CLIENTS], NULL, handle_client,
(void *)client);
    }

    close(server_sockfd);
    exit(EXIT_SUCCESS);
}
```

OUTPUT:


cs572go@LAPTOP-J3V82B9N: /mnt/d/DC/LCA1
cs572go@LAPTOP-J3V82B9N: /mnt/d/DC/LCA1
cs572go@LAPTOP-J3V82B9N: /mnt/d/DC/LCA1
cs572go@LAPTOP-J3V82B9N: /mnt/d/DC/LCA1

```
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$ gcc tserver.c -o server -lpthread
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$ ./server
Server waiting for connections...
tushar has connected
tushar says: hi i am tushar
niranjan has connected
niranjan says: hello
bro has connected
bro says: hehe
niranjan has left the chat.
tushar has left the chat.
bro has left the chat.
```

```
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$ gcc tclient.c -o client1 -lpthread
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$ ./client1
Enter your name: tushar
Enter your message (type 'bye' to exit): hi i am tushar
tushar Sending: hi i am tushar
Enter your message (type 'bye' to exit): bye
tushar Sending: bye
Chat terminated.
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$
```

```
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$ gcc tclient.c -o client2 -lpthread
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$ ./client2
Enter your name: niranjan
Enter your message (type 'bye' to exit): hello
niranjan Sending: hello
Enter your message (type 'bye' to exit): bye
niranjan Sending: bye
Chat terminated.
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$
```

```
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$ gcc tclient.c -o client3 -lpthread
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$ ./client3
Enter your name: bro
Enter your message (type 'bye' to exit): hehe
bro Sending: hehe
Enter your message (type 'bye' to exit): bye
bro Sending: bye
Chat terminated.
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA1$
```

Tushar Deshmukh
Final Year CSE PE27
1032201698

CONCLUSION:

Implemented a Chat server using sockets in C.

PLATFORM:

Windows:

VSCode
Windows Subsystem for Linux (Ubuntu 22.04 LTS)

LANGUAGE:

C language.

Tushar Deshmukh
Final Year CSE PE27
1032201698

FAQs

1. Give the differences between UDP and TCP protocols

Answer:

Aspect	UDP	TCP
Connection	Connectionless	Connection-oriented
Reliability	Unreliable	Reliable
Order of Delivery	No guarantee	In-order delivery assured
Header Overhead	Low	High
Flow Control	No flow control	Flow control mechanisms
Error Checking	Limited error checking	Extensive error checking

2. How does the accept system call work in socket programming

Answer:

In socket programming, the accept system call is used by a server to handle incoming client connections. When the server is in a listening state, accept waits for a client to establish a connection. Once a connection request arrives, accept creates a new socket specifically for communication with that client.

The new socket is unique to the connected client, allowing the server to interact with multiple clients simultaneously. The original listening socket remains open and continues to accept new connections, so the server can serve multiple clients simultaneously.

3. What is the advantage of using threads in socket programming

Answer:

Advantages of using threads in socket programming are as follows-

a. Concurrency: Threads allow multiple clients to be served simultaneously without blocking the main application.
b. Responsiveness: It can quickly create a new thread to handle each incoming connection, ensuring that clients receive timely responses.
c. Resource Efficiency: Threads share the same process memory space, reducing overhead compared to creating separate processes for each client.
d. Scalability: Additional threads can be created as needed to accommodate increasing client connections, providing scalability and flexibility.