

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

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 *myaddr , 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 *addrlen = 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.

INPUT:

OUTPUT:

CONCLUSION:

PLATFORM:

Linux

LANGUAGE:

C language.

FAQs

1. Give the differences between UDP and TCP protocols

Ans. In computer networks, two transport layer protocols are used: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). UDP is faster but less dependable since it lacks a connection and does not ensure data delivery. In contrast, TCP is connection-oriented and guarantees data delivery together with error correction and retransmission. While UDP is utilized for real-time applications like online gaming and video streaming, TCP is more dependable but slower, making it ideal for applications like web browsing and email.

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

Ans. A server in socket programming uses the accept system call to acknowledge incoming connections from clients. The accept call is made on the server socket whenever a client tries to connect to it. Until a connection request is received, it remains blocked. Bidirectional communication is made possible by the new socket that is returned in response to a connection request that has been identified. While the original server socket is still waiting for more incoming connections, this new socket is utilized to send and receive data with the connected client.

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

Ans. Using threads in socket programming has the following benefits:

- Scalability: Thread-based servers are more flexible in handling increasing loads since they can expand more readily to support a higher number of client connections.
- Concurrent Handling: By allowing the server to manage several client connections at once, threads enhance the application's overall effectiveness and responsiveness.
- Resource Utilization: By sharing memory, threads minimize the overhead associated with starting a new process for every client connection. Better resource utilization and reduced memory usage are the outcomes of this.

CODE :

server.c

```
#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>

#define MAX_CLIENTS 5

struct client_details
{
    int sockid;
    char name[80];
};

void *handle_client(void *arg)
{
    struct client_details *client = (struct client_details *)arg;
    char message[200];
    // int flag=0;

    ssize_t name_len = read(client->sockid, client->name, sizeof(client->name));
    if (name_len <= 0)
    {
        close(client->sockid);
        free(client);
    }
}
```

Aniket Inamdar
PE 25
1032201692

```
        pthread_exit(NULL);
    }
    client->name[name_len] = '\0';
    printf("%s has connected\n", client->name);
    while (1)
    {
        ssize_t message_len = read(client->sockid, message,
sizeof(message));
        if (message_len <= 0)
        {
            break;
        }
        message[message_len] = '\0';

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

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

    close(client->sockid);
    free(client);
    pthread_exit(NULL);
}

int main()
{
    int server_sockid;
    struct sockaddr_in server_addr;
    pthread_t threads[MAX_CLIENTS];

    server_sockid = socket(AF_INET, SOCK_STREAM, 0);
    if (server_sockid == -1)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_port = htons(9129);
```

Aniket Inamdar
PE 25
1032201692

```
    if (bind(server_sockid, (struct sockaddr *)&server_addr,
sizeof(server_addr)) == -1)
    {
        perror("Bind failed");
        close(server_sockid);
        exit(EXIT_FAILURE);
    }

    printf("Server is waiting for client connections - \n");
    listen(server_sockid, MAX_CLIENTS);

    while (1)
    {
        struct sockaddr_in client_address;
        socklen_t client_len = sizeof(client_address);

        int client_sockid = accept(server_sockid, (struct sockaddr
*)&client_address, &client_len);
        if (client_sockid == -1)
        {
            perror("Acceptance failed");
            continue;
        }

        struct client_details *client = (struct client_details
*)&malloc(sizeof(struct client_details));
        if (client == NULL)
        {
            perror("Memory allocation has failed");
            close(client_sockid);
            continue;
        }

        client->sockid = client_sockid;
        pthread_create(&threads[MAX_CLIENTS], NULL, handle_client, (void
*)&client);
    }

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

client.c

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
```

Aniket Inamdar
PE 25
1032201692

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

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

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

    sockid = socket(AF_INET, SOCK_STREAM, 0);
    if (sockid == -1) {
        perror("Socket creation has failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_port = htons(9129);

    if (connect(sockid, (struct sockaddr *)&server_addr,
sizeof(server_addr)) == -1) {
        perror("Connection has failed!");
        close(sockid);
        exit(EXIT_FAILURE);
    }
    write(sockid, name, strlen(name));
    while (1) {
        printf("Input message ('bye' to exit): ");
        fgets(message, sizeof(message), stdin);

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

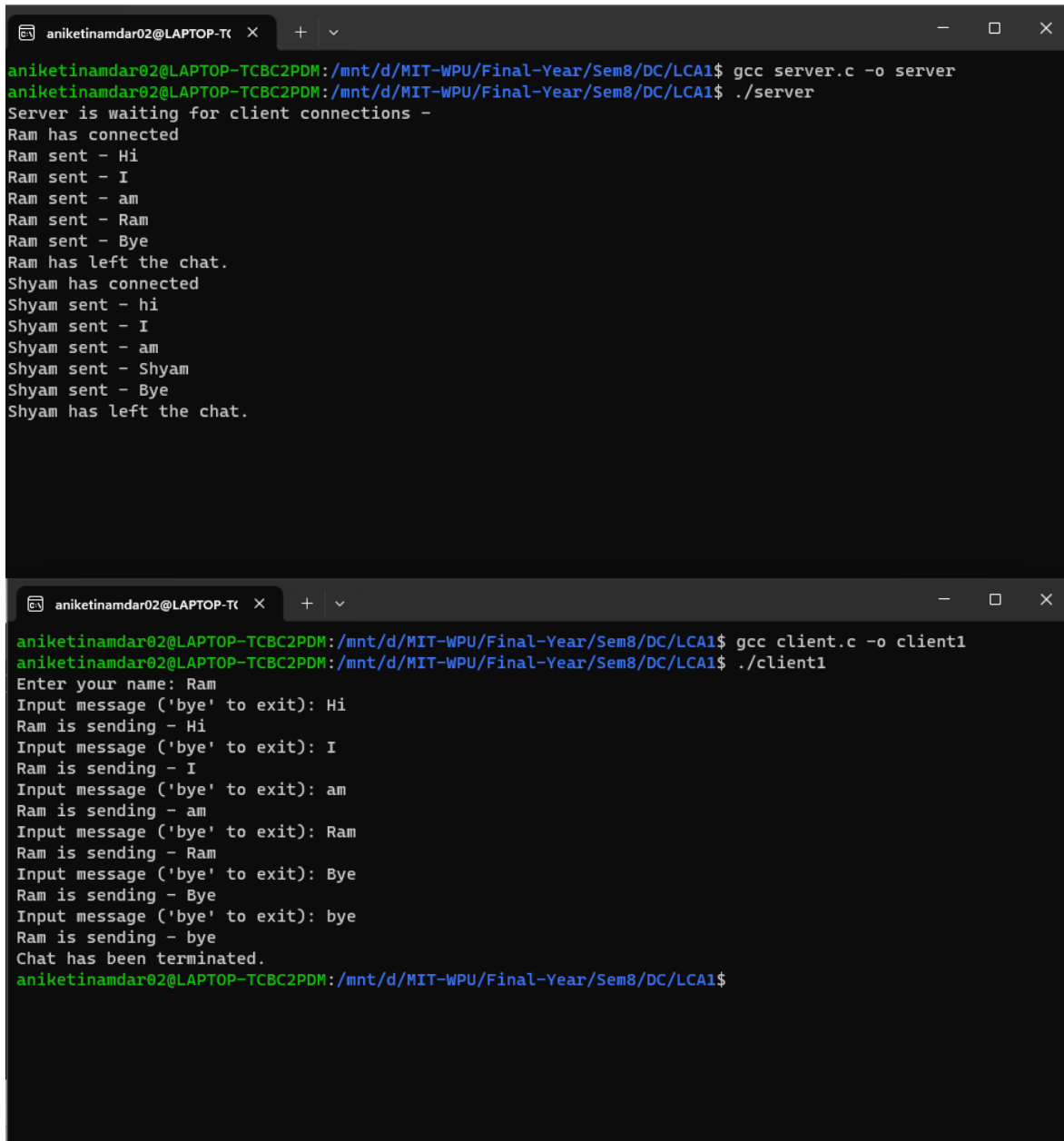
        printf("%s is sending - %s\n", name, message);

        if (strcmp(message, "bye") == 0) {
            printf("Chat has been terminated.\n");
            break;
        }
    }
}
```

Aniket Inamdar
PE 25
1032201692

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

Output :



```
aniketinamdar02@LAPTOP-TCBC2PDM: /mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ gcc server.c -o server  
aniketinamdar02@LAPTOP-TCBC2PDM: /mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ ./server  
Server is waiting for client connections -  
Ram has connected  
Ram sent - Hi  
Ram sent - I  
Ram sent - am  
Ram sent - Ram  
Ram sent - Bye  
Ram has left the chat.  
Shyam has connected  
Shyam sent - hi  
Shyam sent - I  
Shyam sent - am  
Shyam sent - Shyam  
Shyam sent - Bye  
Shyam has left the chat.  
  
aniketinamdar02@LAPTOP-TCBC2PDM: /mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ gcc client.c -o client1  
aniketinamdar02@LAPTOP-TCBC2PDM: /mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ ./client1  
Enter your name: Ram  
Input message ('bye' to exit): Hi  
Ram is sending - Hi  
Input message ('bye' to exit): I  
Ram is sending - I  
Input message ('bye' to exit): am  
Ram is sending - am  
Input message ('bye' to exit): Ram  
Ram is sending - Ram  
Input message ('bye' to exit): Bye  
Ram is sending - Bye  
Input message ('bye' to exit): bye  
Ram is sending - bye  
Chat has been terminated.  
aniketinamdar02@LAPTOP-TCBC2PDM: /mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$
```


Aniket Inamdar

PE 25

1032201692

```
aniketinamdar02@LAPTOP-TCBC2PDM:~$ cd /mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1
aniketinamdar02@LAPTOP-TCBC2PDM:/mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ gcc client.c -o client2
aniketinamdar02@LAPTOP-TCBC2PDM:/mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ ./client2
Enter your name: Shyam
Input message ('bye' to exit): hi
Shyam is sending - hi
Input message ('bye' to exit): I
Shyam is sending - I
Input message ('bye' to exit): am
Shyam is sending - am
Input message ('bye' to exit): Shyam
Shyam is sending - Shyam
Input message ('bye' to exit): Bye
Shyam is sending - Bye
Input message ('bye' to exit): bye
Shyam is sending - bye
Chat has been terminated.
aniketinamdar02@LAPTOP-TCBC2PDM:/mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ |
```

```
Radha has connected
Derek has connected
Radha sent - Hi
Derek sent - Hello
Radha sent - Bye
Derek sent - Bye
Radha has left the chat.
Derek has left the chat.
|
aniketinamdar02@LAPTOP-TCBC2PDM:/mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ ./client1
Enter your name: Radha
Input message ('bye' to exit): Hi
Radha is sending - Hi
Input message ('bye' to exit): Bye
Radha is sending - Bye
Input message ('bye' to exit): bye
Radha is sending - bye
Chat has been terminated.
aniketinamdar02@LAPTOP-TCBC2PDM:/mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ ./client2
Enter your name: Derek
Input message ('bye' to exit): Hello
Derek is sending - Hello
Input message ('bye' to exit): Bye
Derek is sending - Bye
Input message ('bye' to exit): bye
Derek is sending - bye
Chat has been terminated.
aniketinamdar02@LAPTOP-TCBC2PDM:/mnt/d/MIT-WPU/Final-Year/Sem8/DC/LCA1$ |
```