/\*

Name: Rohit Saini

Erp: 1032200897

Panel: C

RollNo: PC-41 \*/

**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 \*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.

**INPUT:**

3+4

**OUTPUT:**

3+4 Answer=7

**CONCLUSION:**

Echo server is implemented using sockets and multithreading.

**PLATFORM:**

Linux

**LANGUAGE:**

C language.

**CODE:**

**//Client**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

int main()

{

    int client\_socket;

    struct sockaddr\_in server\_addr;

    char buffer[1024];

    client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

    if (client\_socket == -1)

    {

        perror("Socket creation error");

        return 1;

    }

    memset(&server\_addr, 0, sizeof(server\_addr));

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = htons(8080);

    server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

    {

        perror("Connection error");

        return 1;

    }

    printf("Connected to server.\n");

    while (1)

    {

        printf("Enter a message ('exit' to quit): ");

        fgets(buffer, sizeof(buffer), stdin);

        // Remove trailing newline

        buffer[strlen(buffer) - 1] = '\0';

        if (send(client\_socket, buffer, strlen(buffer), 0) == -1)

        {

            perror("Send error");

            break;

        }

        if (strcmp(buffer, "exit") == 0)

        {

            break;

        }

        ssize\_t bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);

        if (bytes\_received == -1)

        {

            perror("Receive error");

            break;

        }

        printf("Server response: %.\*s\n", (int)bytes\_received, buffer);

    }

    close(client\_socket);

    return 0;

}

/\*

gcc client.c -o client -Wall

\*/

//Server Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <pthread.h>

int evaluateExpression(const char \*expression)

{

    int num1, num2;

    char operator;

    sscanf(expression, "%d%c%d", &num1, &operator, & num2);

    switch (operator)

    {

    case '+':

        return num1 + num2;

    default:

        printf("Unsupported operator: %c\n", operator);

        return 0;

    }

}

int add(const char \*data)

{

    int result = evaluateExpression(data);

    // printf("Result: %d\n", result);

    return result;

}

void \*clientHandler(void \*clientSocketPtr)

{

    int client\_socket = \*((int \*)clientSocketPtr);

    char buffer[1024];

    while (1)

    {

        ssize\_t bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);

        if (bytes\_received == -1)

        {

            perror("Receive error");

            break;

        }

        if (bytes\_received == 0 || strcmp(buffer, "exit") == 0)

        {

            break;

        }

        printf("Received data: %.\*s\n", (int)bytes\_received, buffer);

        int ans = add(buffer);

        printf("Answer: %d\n", ans);

        char ansStr[20];

        snprintf(ansStr, sizeof(ansStr), " Answer: %d", ans);

        strcat(buffer, ansStr);

        send(client\_socket, buffer, strlen(buffer), 0);

    }

    close(client\_socket);

    return NULL;

}

int main()

{

    int server\_socket;

    struct sockaddr\_in server\_addr, client\_addr;

    socklen\_t client\_len = sizeof(client\_addr);

    server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

    if (server\_socket == -1)

    {

        perror("Socket creation error");

        return 1;

    }

    memset(&server\_addr, 0, sizeof(server\_addr));

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    server\_addr.sin\_port = htons(8080);

    if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

    {

        perror("Binding error");

        return 1;

    }

    if (listen(server\_socket, 5) == -1)

    {

        perror("Listening error");

        return 1;

    }

    printf("Server listening on port 8080...\n");

    while (1)

    {

        int client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_len);

        if (client\_socket == -1)

        {

            perror("Accept error");

            return 1;

        }

        printf("Connected by %s:%d\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

        pthread\_t thread;

        if (pthread\_create(&thread, NULL, clientHandler, &client\_socket) != 0)

        {

            perror("Thread creation error");

            return 1;

        }

        pthread\_detach(thread);

    }

    close(server\_socket);

    return 0;

}

/\*

gcc server.c -o server -Wall

 \*/

FAQs

1. Give the differences between UDP and TCP protocols
2. How does the accept system call work in socket programming
3. What is the advantage of using threads in socket programming



