**COMPUTER NETWORK LAB ASSIGNMENTS :**

**SUPRATIM NAG/CSE-AIM/22/57 :**

**Implementation of IPC in iterative modalities:**

**I .** Write a program to develop an iterative echo server using TCP socket.

SERVER SIDE CODE :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

    int server\_fd, client\_fd;

    struct sockaddr\_in address;

    int addrlen = sizeof(address);

    char buffer[BUFFER\_SIZE];

    // Create socket file descriptor

    if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

        perror("Socket failed");

        exit(EXIT\_FAILURE);

    }

    // Set up server address and port

    address.sin\_family = AF\_INET;

    address.sin\_addr.s\_addr = INADDR\_ANY;

    address.sin\_port = htons(PORT);

    // Bind the socket to the network address and port

    if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

        perror("Bind failed");

        close(server\_fd);

        exit(EXIT\_FAILURE);

    }

    // Listen for incoming connections

    if (listen(server\_fd, 3) < 0) {

        perror("Listen failed");

        close(server\_fd);

        exit(EXIT\_FAILURE);

    }

    printf("Echo server is listening on port %d...\n", PORT);

    while (1) {

        // Accept a connection from a client

        if ((client\_fd = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

            perror("Accept failed");

            continue;

        }

        printf("Connected to client\n");

        // Echo loop: receive message and send it back to client

        while (1) {

            memset(buffer, 0, BUFFER\_SIZE);

            int bytes\_received = recv(client\_fd, buffer, BUFFER\_SIZE, 0);

            if (bytes\_received == 0) {

                printf("Client disconnected\n");

                break;

            }

            if (bytes\_received < 0) {

                perror("Receive failed");

                break;

            }

            printf("Received: %s", buffer);

            // Echo the message back to the client

            send(client\_fd, buffer, bytes\_received, 0);

        }

        // Close the client socket

        close(client\_fd);

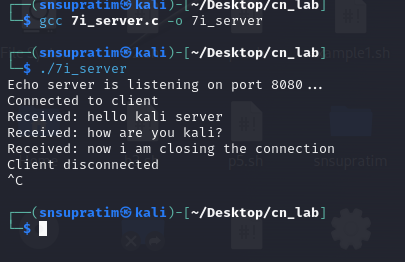
}

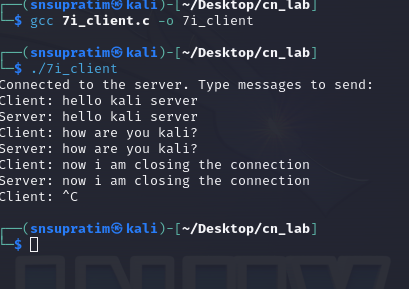
    // Close the server socket

    close(server\_fd);

    return 0;

}





#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

    int sock = 0;

    struct sockaddr\_in server\_address;

    char buffer[BUFFER\_SIZE] = {0};

    // Create socket file descriptor

    if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

        perror("Socket creation error");

        exit(EXIT\_FAILURE);

    }

    // Define server address

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

    server\_address.sin\_port = htons(PORT);

    // Convert IPv4 address from text to binary form

    if (inet\_pton(AF\_INET, "127.0.0.1", &server\_address.sin\_addr) <= 0) {

        perror("Invalid address/Address not supported");

        close(sock);

        exit(EXIT\_FAILURE);

    }

    // Connect to the server

    if (connect(sock, (struct sockaddr \*)&server\_address, sizeof(server\_address)) < 0) {

        perror("Connection failed");

        close(sock);

        exit(EXIT\_FAILURE);

    }

    printf("Connected to the server. Type messages to send:\n");

    // Send and receive messages in a loop

    while (1) {

        printf("Client: ");

        fgets(buffer, BUFFER\_SIZE, stdin);

        // Send message to server

        send(sock, buffer, strlen(buffer), 0);

        // Receive echoed message from server

        int bytes\_received = recv(sock, buffer, BUFFER\_SIZE, 0);

        if (bytes\_received <= 0) {

            printf("Server closed the connection\n");

            break;

        }

        buffer[bytes\_received] = '\0';  // Null-terminate received message

        printf("Server: %s", buffer);

    }

    // Close the socket

    close(sock);

    return 0;

}

CLIENT SIDE CODE :