**COMPUTER NETWORK LAB ASSIGNMENTS :**

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

**Implementation of IPC in iterative modalities:**

**II.** Write a program to develop an iterative echo server using UDP 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;

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

    char buffer[BUFFER\_SIZE];

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

    // Create socket file descriptor

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

        perror("Socket creation failed");

        exit(EXIT\_FAILURE);

    }

    // Define the server address

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

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

    server\_address.sin\_addr.s\_addr = INADDR\_ANY;

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

    // Bind the socket to the specified IP and port

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

        perror("Bind failed");

        close(server\_fd);

        exit(EXIT\_FAILURE);

    }

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

    while (1) {

        memset(buffer, 0, BUFFER\_SIZE);

        // Receive message from client

        int bytes\_received = recvfrom(server\_fd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*)&client\_address, &client\_len);

        if (bytes\_received < 0) {

            perror("Receive failed");

            continue;

        }

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

        // Echo the message back to the client

        sendto(server\_fd, buffer, bytes\_received, 0, (struct sockaddr \*)&client\_address, client\_len);

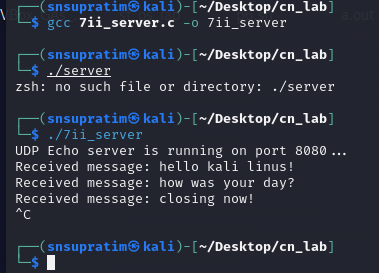
    }

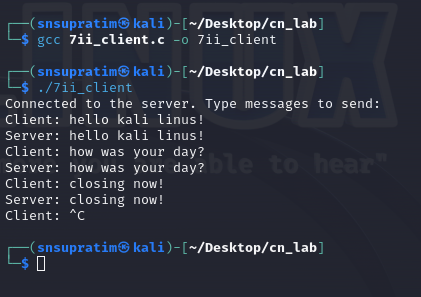
    // Close the socket (though we usually never reach here in an iterative server)

    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 sockfd;

    struct sockaddr\_in server\_address;

    char buffer[BUFFER\_SIZE];

    socklen\_t server\_len = sizeof(server\_address);

    // Create socket file descriptor

    if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

        perror("Socket creation failed");

        exit(EXIT\_FAILURE);

    }

    // Define the server address

    memset(&server\_address, 0, sizeof(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(sockfd);

        exit(EXIT\_FAILURE);

    }

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

    while (1) {

        printf("Client: ");

        fgets(buffer, BUFFER\_SIZE, stdin);

        // Send message to server

        sendto(sockfd, buffer, strlen(buffer), 0, (const struct sockaddr \*)&server\_address, server\_len);

        // Receive echoed message from server

        int bytes\_received = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*)&server\_address, &server\_len);

        if (bytes\_received < 0) {

            perror("Receive failed");

            break;

        }

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

        printf("Server: %s", buffer);

    }

    // Close the socket

    close(sockfd);

    return 0;

}

CLIENT SIDE CODE :