**COMPUTER NETWORK LAB ASSIGNMENTS :**

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

**Implementation of IPC in concurrent modalities:**

II. Write a program to develop a concurrent echo server using UDP socket by implementing

at least two clients.

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 sockfd;

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

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

    char buffer[BUFFER\_SIZE];

    // Create UDP socket

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

        perror("Socket creation failed");

        exit(EXIT\_FAILURE);

    }

    // Clear and configure server address structure

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

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

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

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

    // Bind the socket to the specified port

    if (bind(sockfd, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

        perror("Bind failed");

        close(sockfd);

        exit(EXIT\_FAILURE);

    }

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

    // Main loop to handle incoming messages

    while (1) {

        // Receive message from a client

        int len = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*)&client\_addr, &client\_len);

        if (len < 0) {

            perror("Receive failed");

            continue;

        }

        buffer[len] = '\0';  // Null-terminate the received string

        printf("Received from client: %s\n", buffer);

        // Send the received message back to the client

        sendto(sockfd, buffer, len, 0, (struct sockaddr \*)&client\_addr, client\_len);

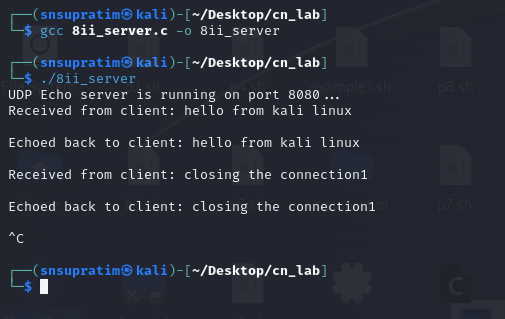
        printf("Echoed back to client: %s\n", buffer);

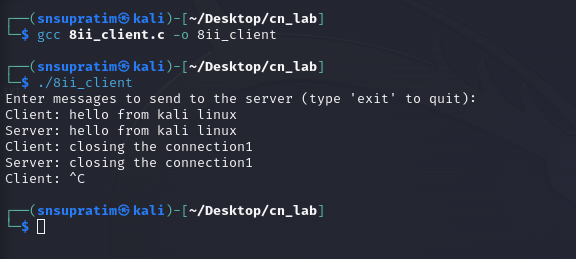
    }

    close(sockfd);

    return 0;

}





#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1"  // IP address of the server

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

    int sockfd;

    struct sockaddr\_in server\_addr;

    char buffer[BUFFER\_SIZE];

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

    // Create UDP socket

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

        perror("Socket creation failed");

        exit(EXIT\_FAILURE);

    }

    // Configure server address structure

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

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

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

    server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    printf("Enter messages to send to the server (type 'exit' to quit):\n");

    while (1) {

        printf("Client: ");

        fgets(buffer, BUFFER\_SIZE, stdin);

        // Send message to the server

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

        // If client sends "exit", break the loop and close the client

        if (strncmp(buffer, "exit", 4) == 0) {

            printf("Exiting...\n");

            break;

        }

        // Receive echo from the server

        int len = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, NULL, NULL);

        if (len < 0) {

            perror("Receive failed");

            break;

        }

        buffer[len] = '\0';  // Null-terminate the received string

        printf("Server: %s", buffer);

    }

    close(sockfd);

    return 0;

}

CLIENT SIDE CODE :

CLIENT2 SIDE CODE :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1"  // IP address of the server

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

    int sockfd;

    struct sockaddr\_in server\_addr;

    char buffer[BUFFER\_SIZE];

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

    // Create UDP socket

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

        perror("Socket creation failed");

        exit(EXIT\_FAILURE);

    }

    // Configure server address structure

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

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

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

    server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    printf("Enter messages to send to the server (type 'exit' to quit):\n");

    while (1) {

        printf("Client: ");

        fgets(buffer, BUFFER\_SIZE, stdin);

        // Send message to the server

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

        // If client sends "exit", break the loop and close the client

        if (strncmp(buffer, "exit", 4) == 0) {

            printf("Exiting...\n");

            break;

        }

        // Receive echo from the server

        int len = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, NULL, NULL);

        if (len < 0) {

            perror("Receive failed");

            break;

        }

        buffer[len] = '\0';  // Null-terminate the received string

        printf("Server: %s", buffer);

    }

    close(sockfd);

    return 0;

}