**COMPUTER NETWORK LAB ASSIGNMENTS :**

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

**Implementation of IPC in concurrent modalities:**

I. Write a program to develop a concurrent echo server using TCP 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>

#include <sys/types.h>

#include <sys/socket.h>

#include <signal.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

// Signal handler to reap zombie processes

void handle\_sigchld(int sig) {

    while (waitpid(-1, NULL, WNOHANG) > 0);

}

int main() {

    int server\_fd, client\_fd;

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

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

    char buffer[BUFFER\_SIZE];

    // Register signal handler for SIGCHLD to avoid zombie processes

    signal(SIGCHLD, handle\_sigchld);

    // Create socket

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

        perror("Socket failed");

        exit(EXIT\_FAILURE);

    }

    // Define the 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);

    }

    // Listen for incoming connections

    if (listen(server\_fd, 3) < 0) {

        perror("Listen failed");

        close(server\_fd);

        exit(EXIT\_FAILURE);

    }

    printf("Concurrent TCP echo server is running on port %d...\n", PORT);

    while (1) {

        // Accept an incoming connection

        if ((client\_fd = accept(server\_fd, (struct sockaddr \*)&client\_address, &client\_len)) < 0) {

            perror("Accept failed");

            continue;

        }

        // Fork a new process to handle the client

        if (fork() == 0) {

            // Child process

            close(server\_fd); // Close the server socket in the child process

            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;

                }

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

                // Echo the message back to the client

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

            }

            close(client\_fd); // Close client socket in the child process

            exit(0); // Exit child process

        }

        // Parent process closes client\_fd as it's being handled by child

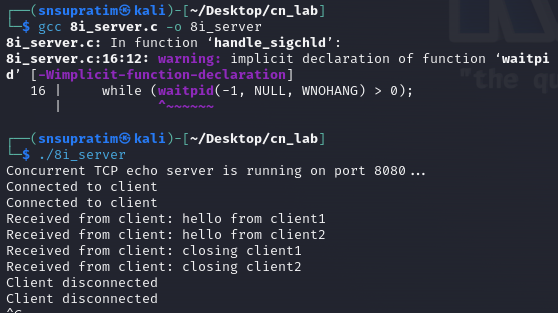
        close(client\_fd);

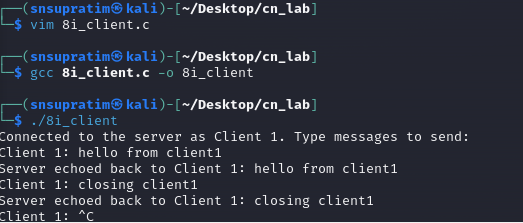
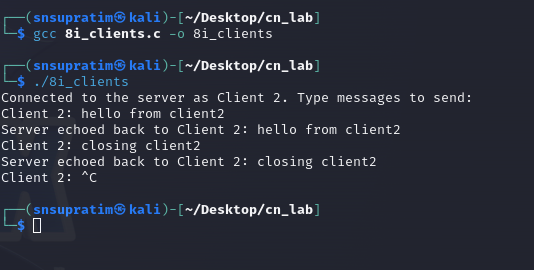
    }

    close(server\_fd); // Close server socket

    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

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

        perror("Socket creation error");

        exit(EXIT\_FAILURE);

    }

    // Define the server address

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

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

    // Convert IP address 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 as Client 2. Type messages to send:\n");

    // Send and receive messages in a loop

    while (1) {

        printf("Client 2: ");

        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 disconnected\n");

            break;

        }

        buffer[bytes\_received] = '\0';

        printf("Server echoed back to Client 2: %s", buffer);

    }

    // Close the socket

    close(sock);

    return 0;

}

CLIENT1 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;

}