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 :

```c
#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
        }
```

```
8i_server.c: In function 'handle_sigchld':
8i_server.c:16:12: warning: implicit declaration of function 'waitpi
d' [-Wimplicit-function-declaration]
   16 |     while (waitpid(-1, NULL, WNOHANG) > 0);
      |            ^~~~~~~

┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ ./8i_server
Concurrent TCP echo server is running on port 8080...
Connected to client
Connected to client
Received from client: hello from client1
Received from client: hello from client2
Received from client: closing client1
Received from client: closing client2
```

```
┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ vim 8i_client.c

┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ gcc 8i_client.c -o 8i_client

┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ ./8i_client
Connected to the server as Client 1. Type messages to send:
Client 1: hello from client1
Server echoed back to Client 1: hello from client1
Client 1: closing client1
Server echoed back to Client 1: closing client1
Client 1: ^C
```

```
└─$ gcc 8i_clients.c -o 8i_clients

┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ ./8i_clients
Connected to the server as Client 2. Type messages to send:
Client 2: hello from client2
Server echoed back to Client 2: hello from client2
Client 2: closing client2
Server echoed back to Client 2: closing client2
Client 2: ^C

┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$
```

CLIENT1 SIDE CODE :

```c
#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;
}
```

CLIENT2 SIDE CODE :

```c
#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;
}
```