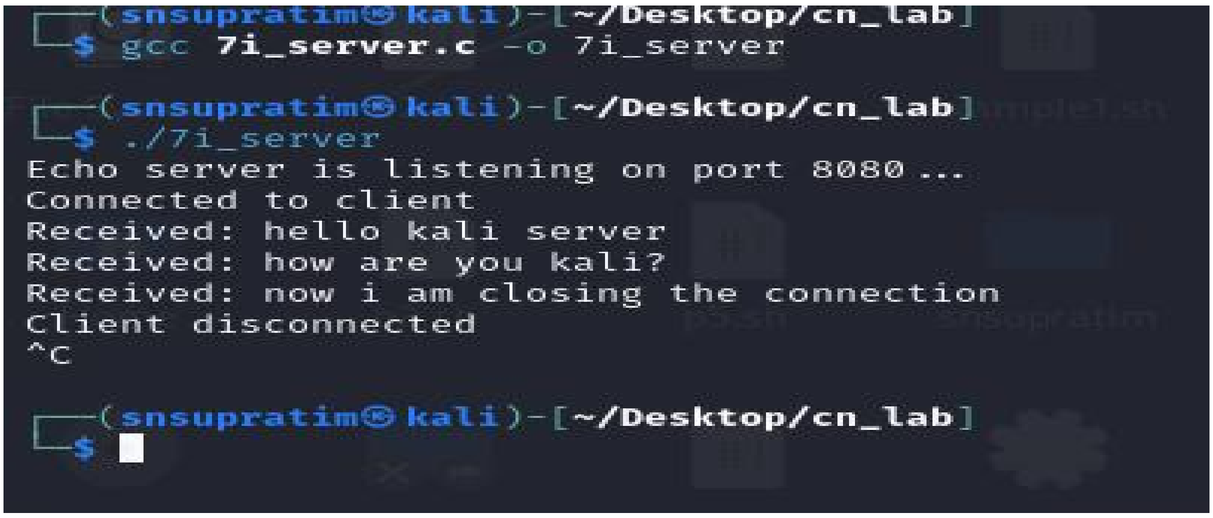
COMPUTER NETWORK LAB ASSIGNMENTS :
SUPRATIM NAG/CSE-AIM/22/57 :

Implementation of IPC in iterative modalities:

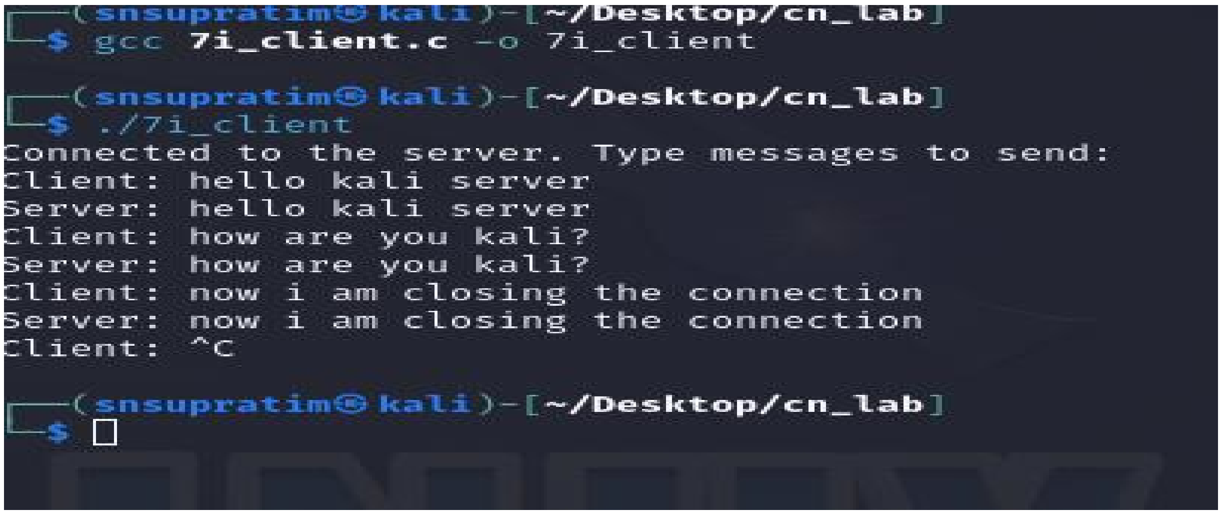
I . Write a program to develop an iterative echo server using TCP socket.

SERVER SIDE CODE :

```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 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;
}
```



```
┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ gcc 7i_server.c -o 7i_server

┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ ./7i_server
Echo server is listening on port 8080 ...
Connected to client
Received: hello kali server
Received: how are you kali?
Received: now i am closing the connection
Client disconnected
^C

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



```
┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ gcc 7i_client.c -o 7i_client

┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ ./7i_client
Connected to the server. Type messages to send:
Client: hello kali server
Server: hello kali server
Client: how are you kali?
Server: how are you kali?
Client: now i am closing the connection
Server: now i am closing the connection
Client: ^C

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

CLIENT 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 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;
}
```