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 :

```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;
    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;
}
```

```
└─$ ./server
zsh: no such file or directory: ./server

┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ ./7ii_server
UDP Echo server is running on port 8080 ...
Received message: hello kali linus!
Received message: how was your day?
Received message: closing now!
^C
```

```
┌──(snsupratim㉿kali)-[~/Desktop/cn_lab]
└─$ ./7ii_client
Connected to the server. Type messages to send:
Client: hello kali linus!
Server: hello kali linus!
Client: how was your day?
Server: how was your day?
Client: closing now!
Server: closing now!
Client: ^C
```

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 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;
}
```