Practical 3: Client–Server Communication using Socket Programming in C

Aim

To implement **client–server communication** using socket programming in C and demonstrate data transfer between two systems in a network.

Concepts

- **Socket:** An endpoint for communication between two machines.
- **Client-Server Model:**
  - **Server:** Always running, waits for requests from clients.
  - **Client:** Connects to server, sends requests, receives responses.
- **Important C Socket Functions:**
  1. `socket()` – Create socket
  2. `bind()` – Assign IP/port
  3. `listen()` – Wait for clients
  4. `accept()` – Accept connection
  5. `connect()` – Connect client to server
  6. `send() / recv()` – Exchange data
  7. `close()` – Close socket

Network Setup in Cisco Packet Tracer

- Use two PCs connected via a switch.
- Assign IPs:
  - Server PC: 192.168.1.1
  - Client PC: 192.168.1.2
- Verify with `ping`.

Implementation

Server Program (server.c)

```c
// server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 12345

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Hello Client, message received!";

    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt
        , sizeof(opt));

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) <
         0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }
    printf("Server listening on port %d...\n", PORT);

    new_socket = accept(server_fd, (struct sockaddr *)&address, (
        socklen_t*)&addrlen);
    if (new_socket < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    read(new_socket, buffer, 1024);
    printf("Client says: %s\n", buffer);

    send(new_socket, hello, strlen(hello), 0);
    printf("Response sent to client.\n");

    close(new_socket);
    close(server_fd);
```

```
    return 0;
}
```

Client Program (client.c)

```
// client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 12345

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    char *hello = "Hello Server, this is Client.";

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Socket creation error");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "192.168.1.1", &serv_addr.sin_addr) <= 0) {
        printf("Invalid address/ Address not supported\n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))
        < 0) {
        perror("Connection Failed");
        return -1;
    }

    send(sock, hello, strlen(hello), 0);
    printf("Message sent to server.\n");

    read(sock, buffer, 1024);
    printf("Server says: %s\n", buffer);

    close(sock);
    return 0;
}
```

Compilation & Execution

1. On **Server PC**:

```
gcc server.c -o server
./server
```

2. On **Client PC**:

```
gcc client.c -o client
./client
```

Expected Output

Server side:

```
Server listening on port 12345...
Client says: Hello Server, this is Client.
Response sent to client.
```

Client side:

```
Message sent to server.
Server says: Hello Client, message received!
```

Analogy

The process is like a **telephone call**:

- Server = person waiting for calls.
- Client = person who dials the number.
- Conversation = messages exchanged.

Learning Outcomes

- Understand how processes communicate across a network.
- Learn the flow of socket programming in C.
- Observe real-time data transfer between two systems.