

TCP Client-Server Implementation and Study

Experiment No: AV-341-2025-Lab-6

Saurabh Kumar
SC22B146
April 1, 2025

Date and Time of experiment: March 24, 2025, 15:00 IST

Objectives

- Initiate a TCP session from client with an echo server.
- Send n packets consecutively from the client and check the number of `recv()` system calls, say k , required to receive n messages at the server, i.e., you need to observe whether $k < n$, $k = n$, or $k > n$.

Tools Used

- PC: 12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz, Windows 11 / Ubuntu 24.04 dual-booted, 64-bit, (reduced to) 4 GB RAM
- OS: Ubuntu 24.04
- Software used: VS Code, VS-Code in-built bash terminal, gcc compiler

Procedure

1. The following C program is written in VS code for creating echo server at port 5000 :

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define BUFFERSIZE 1024
#define PORT 5000

int main()
{
    int req_sock, conn_sock, bytes_recv, bytes_sent, bind_status;
    struct sockaddr_in s_server, s_client;
    int s_len = sizeof(s_server);
    char send_buf[BUFFERSIZE], recv_buf[BUFFERSIZE];

    // Creating requesting socket
    req_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

```

if (req_sock < 0)
{
    printf("Socket creation failed.\n");
    return 1;
}
else
{
    printf("Request socket creation successful with descriptor
%d\n", req_sock);
}

// Bind the socket
s_server.sin_family = AF_INET;
s_server.sin_port = htons(PORT);
s_server.sin_addr.s_addr = htonl(INADDR_ANY);

bind_status = bind(req_sock, (struct sockaddr *)&s_server,
s_len);
if (bind_status < 0)
{
    printf("Socket binding failed.\n");
    return 1;
}
else
{
    printf("Socket binding successful.\n");
}

// Enable req socket to accept connection requests
listen(req_sock, 5);

// Waiting for connection requests
printf("Waiting for connection requests.\n");
conn_sock = accept(req_sock, (struct sockaddr *)&s_client, &
s_len);
if (conn_sock == -1)
{
    printf("Connection request rejected.\n");
    close(req_sock);
    return 1;
}
else
{
    printf("Connection request accepted with socket %d\n",
conn_sock);
}

bytes_recv = recv(conn_sock, recv_buf, sizeof(recv_buf), 0);
recv_buf[bytes_recv] = '\0';

printf("%d bytes received: %s\n", bytes_recv, recv_buf);

strcpy(send_buf, recv_buf);
bytes_sent = send(conn_sock, send_buf, strlen(send_buf), 0);
printf("%d bytes sent: %s\n", bytes_sent, send_buf);

bytes_recv = recv(conn_sock, recv_buf, sizeof(send_buf), 0);
printf("%d bytes received: %s\n", bytes_recv, recv_buf);

```

```

        close(req_sock);
        close(conn_sock);

    return 0;
}

```

2. The following C program is written in VS code for creating client to request the server at localhost:5000 :

```

#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define BUFFERSIZE 1024
#define SERVERADDR "127.0.0.1"
#define SERVERPORT 5000

int main()
{
    int sock, bytes_sent, bytes_recv, conn_status;
    struct sockaddr_in s_server;
    char send_buf[BUFFERSIZE], recv_buf[BUFFERSIZE];

    // Creating socket
    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0)
    {
        printf("Socket creation failed.\n");
        return 1;
    }
    else
    {
        printf("Socket creation successful with descriptor %d.\n",
            sock);
    }

    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(SERVERPORT);
    inet_aton(SERVERADDR, &s_server.sin_addr);

    conn_status = connect(sock, (struct sockaddr *)&s_server,
        sizeof(s_server));
    if (conn_status == -1)
    {
        printf("Connection to server failed.\n");
        close(sock);
        return 1;
    }
    else
    {
        printf("Connected to server.\n");
    }

    // Reading message from keyboard and send
    printf("Enter the message: ");
}

```

```

    fgets(send_buf, sizeof(send_buf), stdin);

    bytes_sent = send(sock, send_buf, strlen(send_buf), 0);
    printf("%d bytes sent: %s\n", bytes_sent, send_buf);

    // Waiting for receiving data
    bytes_recv = recv(sock, recv_buf, sizeof(recv_buf), 0);
    recv_buf[bytes_recv] = '\0';
    printf("%d bytes received: %s\n", bytes_recv, recv_buf);

    close(sock);

    return 0;
}

```

3. For phase2, the following C program is written in VS code for creating server at port 5000 that listens to multiple requests and prints the number of receive calls required to receive all the packets:

```

#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define BUFFERSIZE 1024
#define PORT 5000

int main()
{
    int req_sock, conn_sock, bytes_recv, bind_status, recv_calls = 0;
    struct sockaddr_in s_server, s_client;
    socklen_t s_len = sizeof(s_server);
    char recv_buf[BUFFERSIZE];

    // Creating socket
    req_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (req_sock < 0)
    {
        printf("Socket creation failed.\n");
        return 1;
    }
    printf("Request socket created successfully.\n");

    // Bind the socket
    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(PORT);
    s_server.sin_addr.s_addr = htonl(INADDR_ANY);

    bind_status = bind(req_sock, (struct sockaddr *)&s_server, s_len);
    if (bind_status < 0)
    {
        printf("Socket binding failed.\n");
        return 1;
    }
    printf("Socket binding successful.\n");
}

```

```

// Listen for connections
listen(req_sock, 5);
printf("Waiting for connection requests...\n");

conn_sock = accept(req_sock, (struct sockaddr *)&s_client, &
s_len);
if (conn_sock == -1)
{
    printf("Connection request rejected.\n");
    close(req_sock);
    return 1;
}
printf("Connection request accepted.\n");

// Receive packets
printf("Receiving packets...\n");
while ((bytes_recv = recv(conn_sock, recv_buf, sizeof(recv_buf)
- 1, 0)) > 0)
{
    recv_buf[bytes_recv] = '\0';
    printf("Received (%d bytes): %s\n", bytes_recv, recv_buf);
    recv_calls++;
}

printf("\nTotal recv() calls: %d\n", recv_calls);

close(conn_sock);
close(req_sock);
return 0;
}

```

4. The following C program is written in VS code for creating client to send multiple packets to the server at localhost:5000 :

```

#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define BUFFERSIZE 1024
#define SERVERADDR "127.0.0.1"
#define SERVERPORT 5000
#define N 10 // Number of packets to send
#define ROLL_NUMBER "SC22B146"

int main()
{
    int sock, bytes_sent, conn_status;
    struct sockaddr_in s_server;
    char send_buf[BUFFERSIZE];
    char messages[N][100];

    // Create socket
    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0)
    {

```

```

        printf("Socket creation failed.\n");
        return 1;
    }
    printf("Socket creation successful with descriptor %d.\n", sock
);

    s_server.sin_family = AF_INET;
    s_server.sin_port = htons(SERVERPORT);
    inet_aton(SERVERADDR, &s_server.sin_addr);

    // Connect to server
    conn_status = connect(sock, (struct sockaddr *)&s_server,
sizeof(s_server));
    if (conn_status == -1)
    {
        printf("Connection to server failed.\n");
        close(sock);
        return 1;
    }
    printf("Connected to server.\n");

    // Prompts for messages to send
    for (int i = 0; i < N; i++) {
        printf("Enter message %d: ", i + 1);
        fgets(messages[i], sizeof(messages[i]) - 10, stdin);
        messages[i][strcspn(messages[i], "\n")] = 0;
    }

    // Send packets
    for (int i = 0; i < N; i++) {
        snprintf(send_buf, BUFFERSIZE, "%02d:%s:%s\n", i,
ROLL_NUMBER, messages[i]); // Header

        bytes_sent = send(sock, send_buf, strlen(send_buf), 0);
        if (bytes_sent < 0) {
            printf("Error sending data\n");
            return 1;
        }
        printf("Sent packet %d: %s (%d bytes)\n", i, send_buf,
bytes_sent);
    }

    close(sock);
    return 0;
}

```

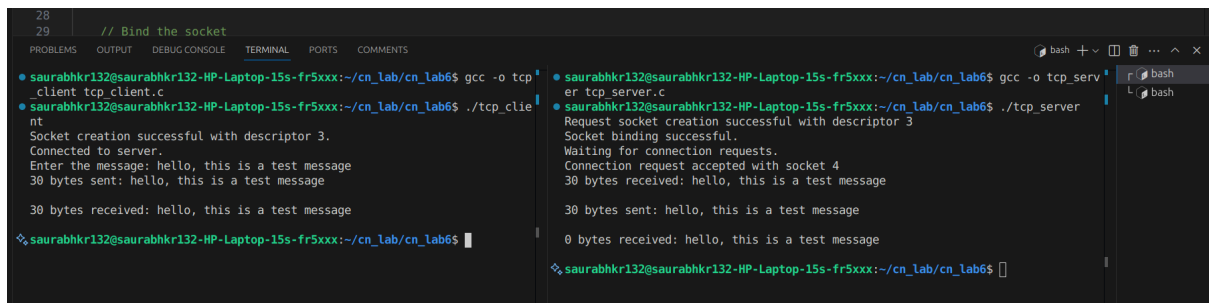
5. Compiling of code is done by the command (file must be in the current working directory):

```
gcc -o output_file_name file_to_compile.c
```

6. The compiled binary file can be run by the command:

```
./output_file_name
```

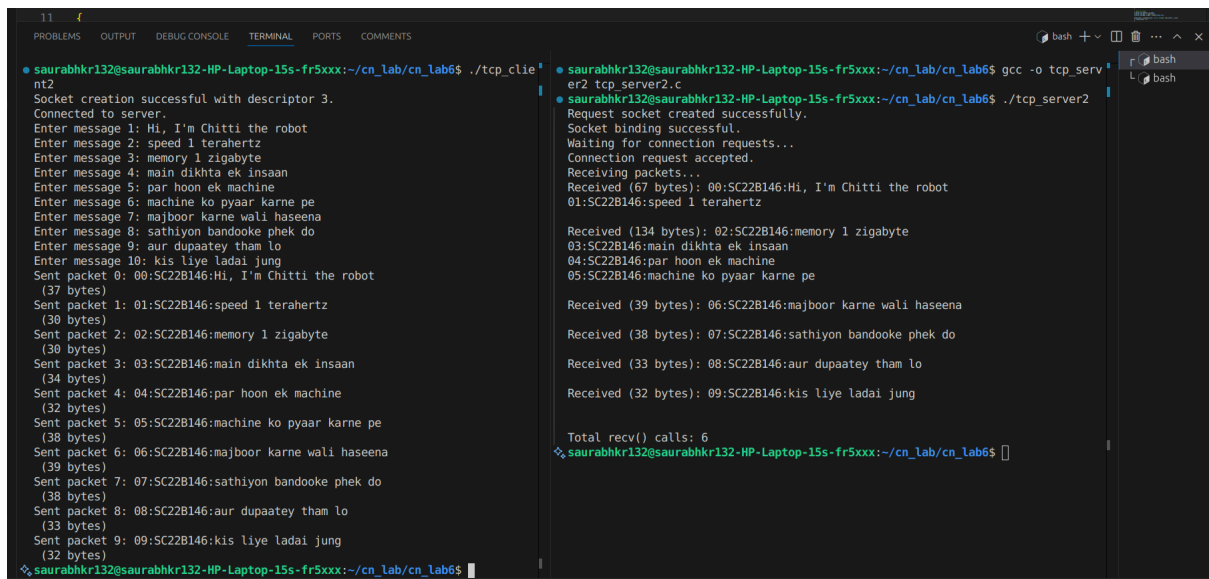
Observations



```
28 // Bind the socket
29
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$ gcc -o tcp_client tcp_client.c
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$ ./tcp_client
Socket creation successful with descriptor 3.
Connected to server.
Enter the message: hello, this is a test message
30 bytes sent: hello, this is a test message
30 bytes received: hello, this is a test message
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$

saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$ gcc -o tcp_server tcp_server.c
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$ ./tcp_server
Request socket creation successful with descriptor 3
Socket binding successful.
Waiting for connection requests.
Connection request accepted with socket 4
30 bytes received: hello, this is a test message
30 bytes sent: hello, this is a test message
0 bytes received: hello, this is a test message
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$
```

Figure 1: Server-client TCP Communication



```
11
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$ ./tcp_client
nt2
Socket creation successful with descriptor 3.
Connected to server.
Enter message 1: Hi, I'm Chitti the robot
Enter message 2: speed 1 terahertz
Enter message 3: memory 1 zigabyte
Enter message 4: main dikhta ek insaan
Enter message 5: par hoon ek machine
Enter message 6: machine ko pyaar karne pe
Enter message 7: majboor karne wali haseena
Enter message 8: sathiyon bandooke phek do
Enter message 9: aur dupaatey tham lo
Enter message 10: kis liye ladai jung
Sent packet 0: 00:SC22B146:Hi, I'm Chitti the robot
(37 bytes)
Sent packet 1: 01:SC22B146:speed 1 terahertz
(30 bytes)
Sent packet 2: 02:SC22B146:memory 1 zigabyte
(30 bytes)
Sent packet 3: 03:SC22B146:main dikhta ek insaan
(34 bytes)
Sent packet 4: 04:SC22B146:par hoon ek machine
(32 bytes)
Sent packet 5: 05:SC22B146:machine ko pyaar karne pe
(30 bytes)
Sent packet 6: 06:SC22B146:majboor karne wali haseena
(39 bytes)
Sent packet 7: 07:SC22B146:sathiyon bandooke phek do
(38 bytes)
Sent packet 8: 08:SC22B146:aur dupaatey tham lo
(33 bytes)
Sent packet 9: 09:SC22B146:kis liye ladai jung
(32 bytes)
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$

saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$ gcc -o tcp_server2 tcp_server2.c
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$ ./tcp_server2
Request socket created successfully.
Socket binding successful.
Waiting for connection requests...
Connection request accepted.
Receiving packets...
Received (67 bytes): 00:SC22B146:Hi, I'm Chitti the robot
01:SC22B146:speed 1 terahertz
Received (134 bytes): 02:SC22B146:memory 1 zigabyte
03:SC22B146:main dikhta ek insaan
04:SC22B146:par hoon ek machine
05:SC22B146:machine ko pyaar karne pe
Received (39 bytes): 06:SC22B146:majboor karne wali haseena
Received (38 bytes): 07:SC22B146:sathiyon bandooke phek do
Received (33 bytes): 08:SC22B146:aur dupaatey tham lo
Received (32 bytes): 09:SC22B146:kis liye ladai jung
Total recv() calls: 6
saurabhkr132@saurabhkr132-HP-Laptop-15s-fr5xxx:~/cn_lab/cn_lab6$
```

Figure 2: Server-client TCP Communication with multiple packets

Conclusions

- In the first phase, a basic tcp server and a client are created using socket programming in which the client sends a request to the server and the server replies back with the same message.
- In the second phase, we successfully established a reliable communication channel between the client and the server for multiple packets. The client sent multiple consecutive messages, each containing a sequence number and sender roll number, while the server received and processed these messages accordingly.

One key observation was the number of `recv()` calls (k) at the server compared to the number of messages (n) sent by the client. Depending on network behavior and buffer management, the server could receive all messages at once ($k < n$), exactly as sent ($k = n$), or in fragmented packets ($k > n$). This highlights the importance of handling variable receive behaviour in TCP-based communication. The experiment demonstrated TCP's ordered, reliable, and congestion-controlled nature, making it suitable for scenarios requiring error-free data transfer.