Distributed System - Practical 1

Do Bao Nhi - 22BI13349

November 30, 2024

Contents

List of Figures

1 Design Protocol

We create a socket (communication endpoint) using the `socket()` function for both the client and the server. This enables them to exchange data using IP addresses.

1.1 Client Side

- The `SERVER_IP` and `SERVER_PORT` are defined as the target server's IP address and port number, respectively.
- The `connect()` function is then used to establish a connection between the client and the server.
  - The `connect()` function sends a **SYN** packet to the server and waits for the server to send back a **SYN-ACK** packet.
  - Once the **SYN-ACK** is received, the client sends an **ACK** packet to complete the connection establishment process.
- Once the connection is established, the client transfers the file to the server using the `send()` function.
- The client reads data from the file in chunks and sends it over the established connection.
- Finally, the `close()` function is used to terminate the communication.

1.2 Server Side

- The `PORT` number is defined on which the server will listen for incoming requests.
- The `listen()` function is used to transition the server socket to passive mode, enabling it to wait for connection requests sent by the client (via `connect()`).
- When the server receives a **SYN** packet from the client:
  - The server's TCP stack automatically sends a **SYN-ACK** packet back to the client.
  - The server transitions to the **SYN_RECEIVED** state while waiting for the client's **ACK** response.
- Once the client sends the **ACK**, the connection is established.
- The `accept()` function is called to return a new socket dedicated to the established connection.
- At this point, the client and server can exchange data.
- The server uses the `recv()` function to receive data sent by the client.
- The received data is then written to a file using the `fprintf()` function.
- After all the data has been written to the file, the server socket is closed using the `close()` function.

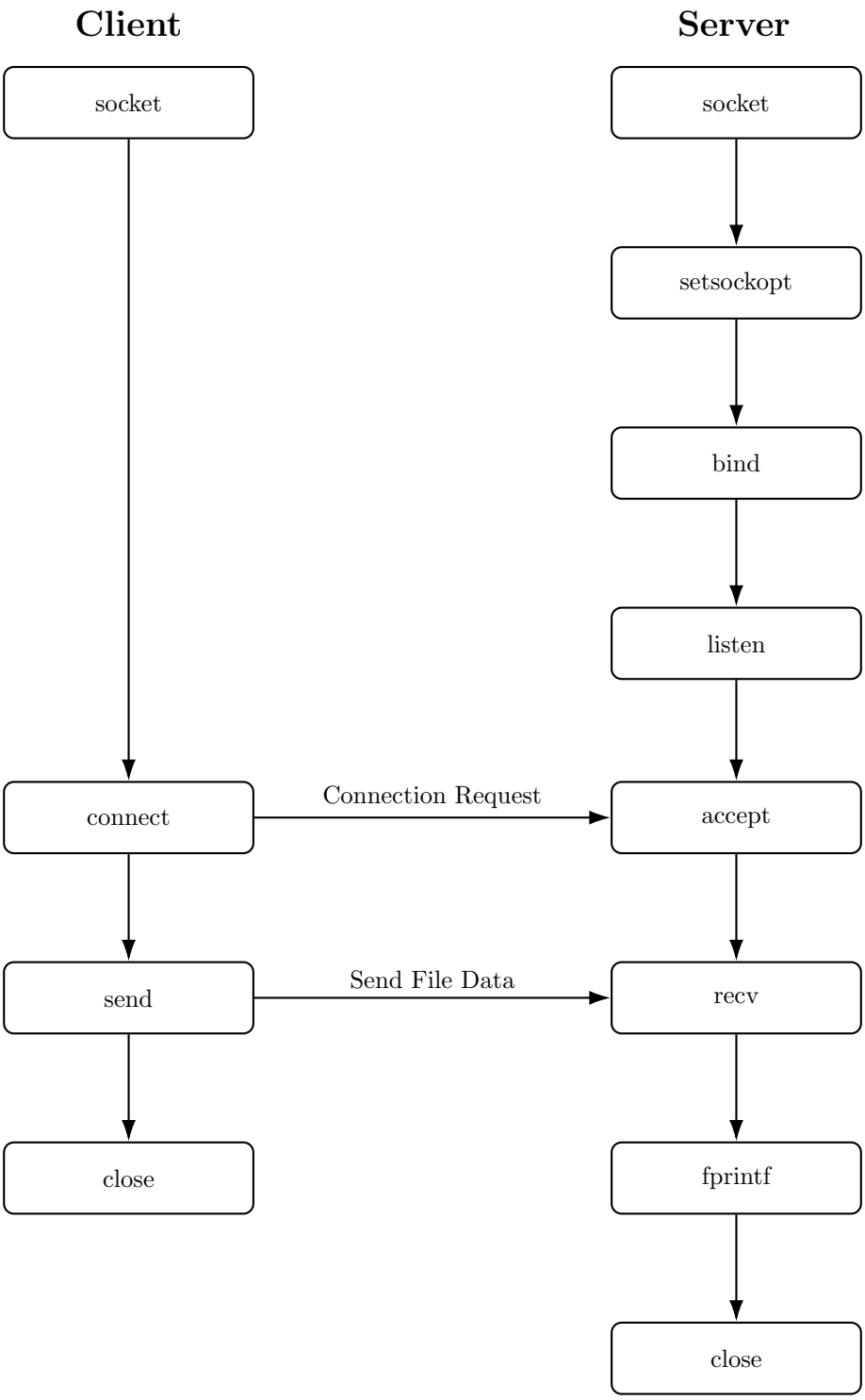

Figure 1: Client-Server Interaction Protocol

2 System Architecture

The system consists of a server and a client, which communicate with each other using a stream socket. A stream socket allows processes to utilize the **Transmission Control Protocol (TCP)** for communication. Since TCP is a connection-oriented protocol, it requires a connection to be established before data can be transferred. The responsibilities of each component are as follows:

Client

- Sends a **SYN** request to initiate the connection.
- Receives a **SYN-ACK** response from the server and then sends an **ACK** to complete the handshake.
- Once the connection is established, the client sends a file to the server.

Server

- Listens for incoming connection requests.
- Receives the **SYN** request from the client and responds with a **SYN-ACK** packet.
- Receives the client's **ACK** to complete the connection establishment.
- After the connection is established, the server receives the file from the client and writes the data to a file.

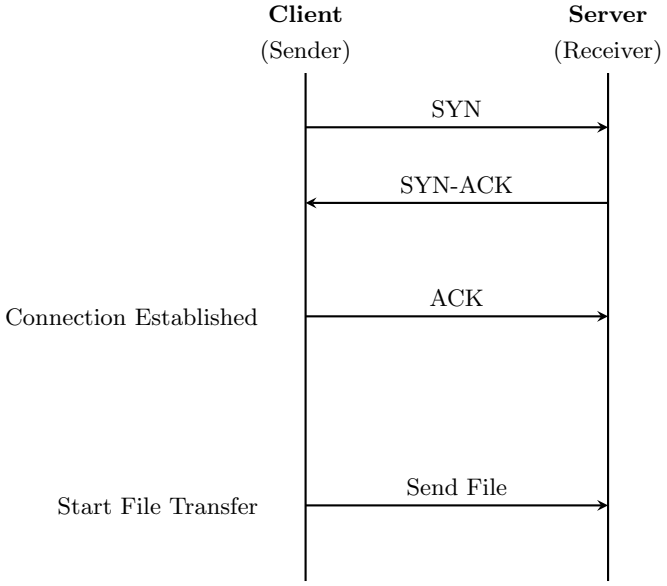

Figure 2: TCP Handshake and File Transfer Process

3 Implementation

3.1 send_file() Function in the client side

The `send_file()` function is responsible for sending a file from the client to the server. It reads the file in chunks and transmits each chunk to the server over an established socket connection.

```
void send_file(FILE *fp, int client_socket) {
    int n;
    char buffer[BUFFER_SIZE];
    while(fgets(buffer, BUFFER_SIZE, fp) != NULL) {
        if(send(client_socket, buffer, sizeof(buffer), 0) == -1) {
            perror("[-] Sending file failed");
            exit(EXIT_FAILURE);
        }
        bzero(buffer, BUFFER_SIZE);
    }
}
```

Listing 1: send_file() Function in the Client Side

3.2 recv_file() Function in the server side

The `recv_file()` function is responsible for receiving a file sent by the client. It reads data from the client socket in chunks, writes the received data into a file, and continues until the connection is closed or an error occurs.

```
void recv_file(int client_socket) {
    int n;
    FILE *fp;
    char *fn = "message.txt";
    char buffer[BUFFER_SIZE];

    fp = fopen(fn, "w");
    // Loop until the connection is closed or an error occurs
    while (1) {
        // recv() returns the number of bytes read
        n = recv(client_socket, buffer, BUFFER_SIZE, 0);
        if (n <= 0) {
            break;
        }
        // Write the received data to message.txt
        fprintf(fp, "%s", buffer);
        bzero(buffer, BUFFER_SIZE); // Clear all data in the buffer
    }
    fclose(fp);
}
```

Listing 2: recv_file() Function in the Server Side

3.3 Result

3.3.1 Client output

The following image shows the client-side output captured during the execution of the program

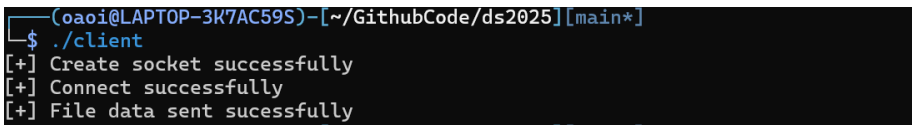

Figure 3: Client-side output.

3.3.2 Server output

The following image shows the server-side output captured during the execution of the program.

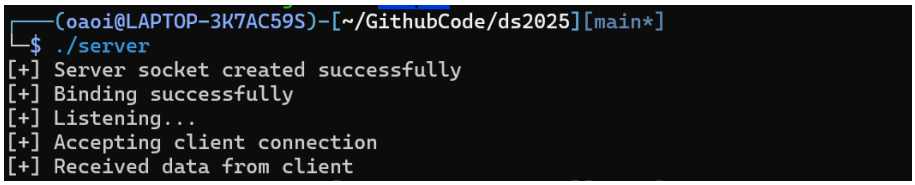

Figure 4: Server-side output.

This image indicates that the client send the **send.txt** file to the server over connection. Then server received the file data from client and write these data to **message.txt** file.

Figure 5: File data transferred.