TP1 - Distributed System

Le Van Truong

November 2024

1 Introduction

This program enables basic file transfer over a TCP/IP connection using the command line. The server listens for incoming connections and receives a file from the client. Meanwhile, the client sends the file in bytes to ensure reliable delivery. Once the transfer is complete, both the server and client close their connections.

2 Protocol Design

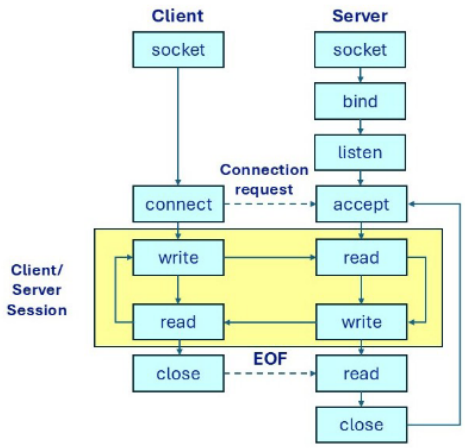

Figure 1: File Transfer Protocol Design

The protocol for the file transfer system is designed as in the figure below. The server initializes by creating a socket and binding it to a specific address and

port. The server uses `listen()`, which puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The client initiates communication by creating a socket and connecting to the server. Once the connection is established, the client-server session begins.
In this session, the client sends file data using `write()` (or `send()` in the code), and the server receives the data using `read()` (or `recv()` in the code). The server creates a file named `recv.txt` and writes the data into the file. This will continue until all the data is written into the file. After the session is completed, the connection is closed. Hence, we can receive a new file with the same data on the server's end.

3 System Organization

The provided image represents the system organization:

Figure 2: System Organization

4 Code Snippet

I have the following codes for both client and server:

4.1 server.c

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

#define SERVER_PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_socket, client_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
```

```
    char buffer[BUFFER_SIZE];

    // Create and bind the socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0); // return file
        descriptor for new socket
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(SERVER_PORT);
    bind(server_socket, (struct sockaddr *)&address, sizeof(address));

    // Listen for connections
    listen(server_socket, 5); // maximum pending connections
    printf("[*] server listening on port %d...\n", SERVER_PORT);

    // Accept a connection
    if ((client_socket = accept(server_socket, (struct sockaddr
        *)&address,
                        (socklen_t *)&addrlen)) < 0) {
        perror("Accept failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }
    printf("[*] Client connected\n");

    // Receive and save file
    FILE *file = fopen("recv.txt", "wb");
    int bytes_read;
    while ((bytes_read = recv(client_socket, buffer, BUFFER_SIZE, 0)) >
        0) {
        fwrite(buffer, 1, bytes_read, file);
    }

    printf(" [*] Received at 'recv.txt'\n");
    fclose(file);
    close(client_socket);
    close(server_socket);
    return 0;
}
```

4.2 client.c

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

```c
#define SERVER_IP "127.0.0.1"
#define SERVER_PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int client_socket;
    struct sockaddr_in address;
    char buffer[BUFFER_SIZE];

    // Create a socket
    if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Set up the server address structure
    address.sin_family = AF_INET;
    address.sin_port = htons(SERVER_PORT);

    // Convert from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &address.sin_addr) <= 0) {
        perror("inet_pton");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    // Connect to server
    if (connect(client_socket, (struct sockaddr *)&address,
         sizeof(address)) < 0) {
        perror("connect");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    printf("Connected to server %s:%d\n", SERVER_IP, SERVER_PORT);

    // Send file
    FILE *file = fopen("man.txt", "rb");
    if (file == NULL) {
        perror("File open failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    int bytes_read;
    while ((bytes_read = fread(buffer, 1, BUFFER_SIZE, file)) > 0) {
        send(client_socket, buffer, bytes_read, 0);
    }
    printf("File sent successfully.\n");
```

```
    fclose(file);

    // Close the socket
    close(client_socket);

    return 0;
}
```

5 Code Implementation

To begin with, compile and run the server.c first:

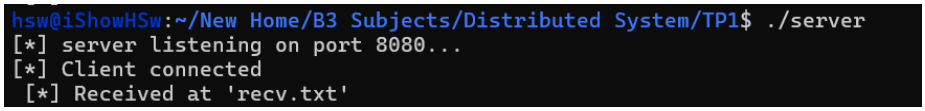

Figure 3: A running server.c

After running the server.c, the server is in the wait status.Now I compile client.c and run. In this example, I have a text file send.txt to send. In the case we want to send another file, change the file name in the client.c code. We can see that the file has been sent successfully.

Figure 4: A running client.c

On the server.c terminal, we have the result below:

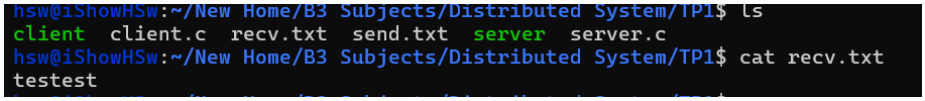

Figure 5: File received in Server terminal