

Distributed System - Practical Work 1 Report

File transfer over TCP/IP in Command Line Interface (CLI)

Nguyen Hai Nam - 22BI13322

November 2024

1 Introduction

This implements a basic file transfer over TCP/IP in CLI (Command Line Interface) between a server and a client via sockets. The server waits for connections and receives a file from the client. The client sends the file in chunks, ensuring reliable transfer before both sides close their sockets.

2 Protocol Design

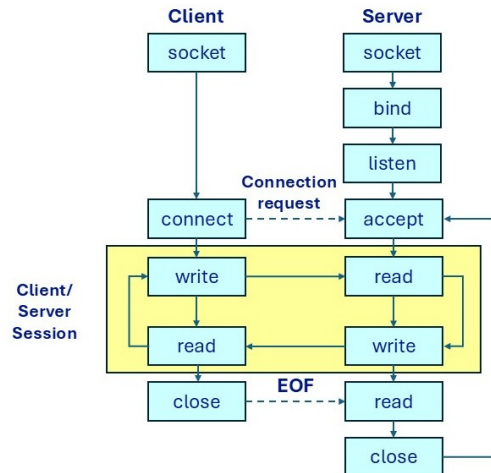


Figure 1: File Transfer Protocol Design

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 `received_file.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:

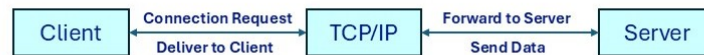


Figure 2: System Organization

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] = {0};
  
```

```

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

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

// Bind the socket to the address and port number
if (bind(server_socket, (struct sockaddr *)&address,
    sizeof(address)) < 0) {
    perror("Bind failed");
    close(server_socket);
    exit(EXIT_FAILURE);
}

// Listen for connections
if (listen(server_socket, 3) < 0) {
    perror("Listen failed");
    close(server_socket);
    exit(EXIT_FAILURE);
}
printf("Server listening on port %d...\n", SERVER_PORT);

// Accept a client 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 file
FILE *file = fopen("received_file.txt", "wb");
if (file == NULL) {
    perror("File open failed");
    close(client_socket);
    close(server_socket);
    exit(EXIT_FAILURE);
}

int bytes_read;
while ((bytes_read = recv(client_socket, buffer, BUFFER_SIZE, 0)) >
    0) {

```

```

        fwrite(buffer, 1, bytes_read, file);
    }
    printf("File received and saved as 'received_file.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>

#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);

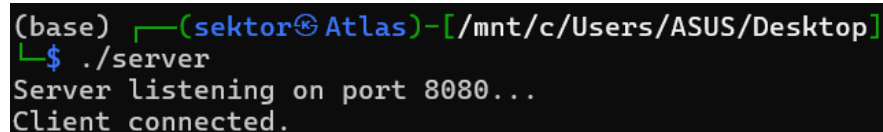
// Close the socket
close(client_socket);

return 0;
}

```

5 Code Implementation

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



```

(base) └─(sektor@Atlas)-[/mnt/c/Users/ASUS/Desktop]
└─$ ./server
Server listening on port 8080...
Client connected.

```

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 man.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.

```
(base) └─(sektor@Atlas)-[/mnt/c/Personal/Work/USTH/B3/ds/TP1]
└─$ ls
client  client.c  man.txt  server.c

(base) └─(sektor@Atlas)-[/mnt/c/Personal/Work/USTH/B3/ds/TP1]
└─$ ./client
Connected to server 127.0.0.1:8080
File sent successfully.
```

Figure 4: A running client.c

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

```
File received and saved as 'received_file.txt'

(base) └─(sektor@Atlas)-[/mnt/c/Users/ASUS/Desktop]
└─$ cat received_file.txt
lorem ipsum
```

Figure 5: File received in Server terminal