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

Nguyen Tai Anh - 22BI13028

November 30, 2024

Contents

Abstract

This report discusses the design and implementation of a file transfer system over TCP/IP in a Command Line Interface (CLI). The system utilizes a custom-built protocol to transfer files between a client and a server over a network. This document explains the design considerations, the protocol, system organization, and implementation details, along with figures and code snippets.

1 Introduction

In modern computing, file transfer is a critical operation, especially in networked environments. The Transmission Control Protocol/Internet Protocol (TCP/IP) is the foundation of most networking systems. This report outlines the design of a simple yet effective file transfer system using TCP/IP over a Command Line Interface (CLI). This system provides a means for transferring files from one machine to another using a custom-designed protocol.

2 Protocol Design

2.1 Overview of the Protocol

The file transfer protocol (FTP) was designed to facilitate efficient and secure transfer of files between a client and a server. The protocol operates over TCP/IP, ensuring reliable communication by utilizing the underlying TCP connection's features, such as error checking, data integrity, and retransmission.

2.2 Protocol Design Steps

The following steps outline the protocol designed for file transfer:

1. **Connection Establishment:** The client initiates a connection with the server using a predefined port. The server sets up by binding to an IP address and port, then waits for connections from clients.

2. **File Request:** The client sends a request to the server for the file to be transferred.

3. **File Transfer:** The server breaks the file into smaller chunks and sends them to the client.

4. **Acknowledgment:** After receiving each chunk, the client sends an acknowledgment to the server.

5. **Completion:** Once all chunks are transferred, the server sends a completion signal to the client.

6. **Close Connection:** After the file is fully transferred, both the client and server close the connection properly.

2.3 Figure 1: Protocol Design Flow

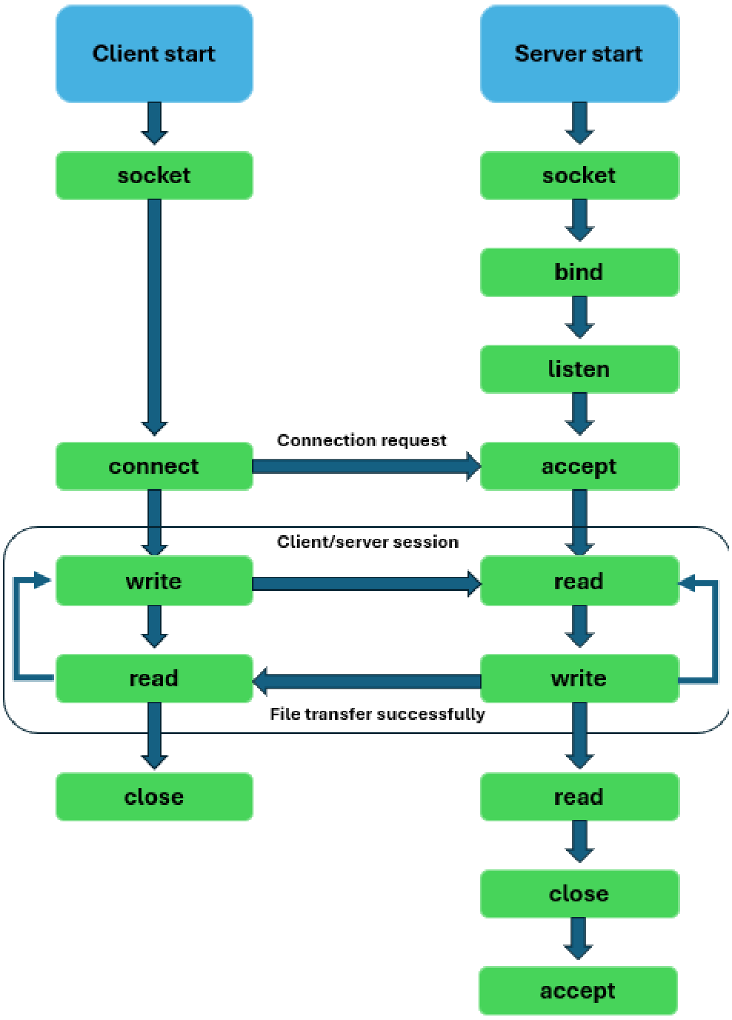

Figure 1: Protocol Design Flow

3 System Organization

3.1 System Architecture

The system is divided into two main components: the client and the server. Each component performs distinct tasks as part of the file transfer process.

- **Client:** Initiates the file transfer, sends file requests, receives file chunks, downloads file and sends acknowledgments.

- **Server:** Waits for connection requests, handles file transfer, and breaks the file into chunks for transmission.

3.2 Figure 2: System Organization

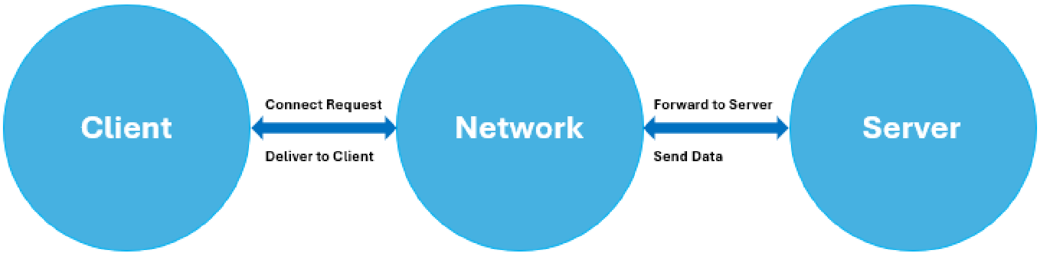

Figure 2: System Organization Diagram

4 File Transfer Implementation

4.1 Client Code

The following code snippet shows the implementation of the client in Python using TCP/IP for file transfer.

Listing 1: Client Code for File Transfer

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

#define SIZE 1024

void send_file(FILE *fp, int sockfd)
{
    char data[SIZE] = {0};

    while(fgets(data, SIZE, fp)!=NULL)
    {
        if(send(sockfd, data, sizeof(data), 0)== -1)
        {
```

```
            perror("Error in sendung data");
            exit(1);
        }
        bzero(data, SIZE);
    }
}

int main()
{
    char *ip = "127.0.0.1";
    int port = 8080;
    int e;

    int sockfd;
    struct sockaddr_in server_addr;
    FILE *fp;
    char *filename = "example.txt";
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd<0)
    {
        perror("Error in socket");
        exit(1);
    }
     printf("Server socket created. \n");

     server_addr.sin_family = AF_INET;
     server_addr.sin_port = port;
     server_addr.sin_addr.s_addr = inet_addr(ip);

     e = connect(sockfd, (struct sockaddr*)&server_addr
     , sizeof(server_addr));
     if(e == -1)
     {
         perror("Error in Connecting");
         exit(1);
     }
     printf("[+]Connected to server.\n");
     fp = fopen(filename, "r");
     if(fp == NULL)
     {
         perror("[-]Error in reading file.");
```

```
        exit(1);
    }
    send_file(fp,sockfd);
    printf("File data send successfully.\n");
    close(sockfd);
    printf("Disconnected from the server.\n");
    return 0;

}
```

4.2 Server Code

The following code snippet shows the server-side implementation to handle the incoming file and save it.

Listing 2: Server Code for File Transfer

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

#define SIZE 1024

void write_file(int sockfd)
{
    int n;
    FILE *fp;
    char *filename = "received_file.txt";
    char buffer[SIZE];

    fp = fopen(filename, "w");
    if(fp==NULL)
    {
        perror("Error in creating file.");
        exit(1);
    }
    while(1)
    {
        n = recv(sockfd, buffer, SIZE, 0);
        if(n<=0)
        {
```

```
        break;
        return;
      }
      fprintf(fp, "%s", buffer);
      bzero(buffer, SIZE);
    }
    return;

}

int main ()
{
    char *ip = "127.0.0.1";
    int port = 8080;
    int e;

    int sockfd, new_sock;
    struct sockaddr_in server_addr, new_addr;
    socklen_t addr_size;
    char buffer[SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd<0)
    {
        perror("Error in socket");
        exit(1);
    }
     printf("Server socket created. \n");

     server_addr.sin_family = AF_INET;
     server_addr.sin_port = port;
     server_addr.sin_addr.s_addr = inet_addr(ip);

     e = bind(sockfd,(struct sockaddr*)&server_addr
     , sizeof(server_addr));
     if(e<0)
     {
          perror("Error in Binding");
          exit(1);
     }
     printf("Binding Successfull.\n");
```

```c
        e = listen(sockfd, 10);
        if(e==0)
        {
            printf("Listening...\n");
        }
        else
        {
            perror("Error in Binding");
            exit(1);
        }
        addr_size = sizeof(new_addr);
        new_sock = accept(sockfd,(struct sockaddr*)&new_addr
        , &addr_size);

        write_file(new_sock);
        printf("Data written in the text file ");
}
```

4.3 Implementing File Transfer

4.3.1 Testing with myself

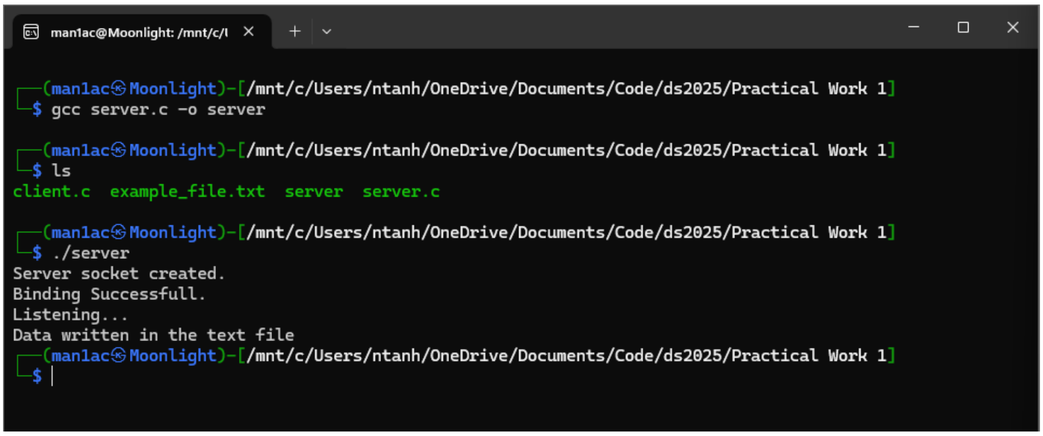

Figure 3: Server Command Line Interface

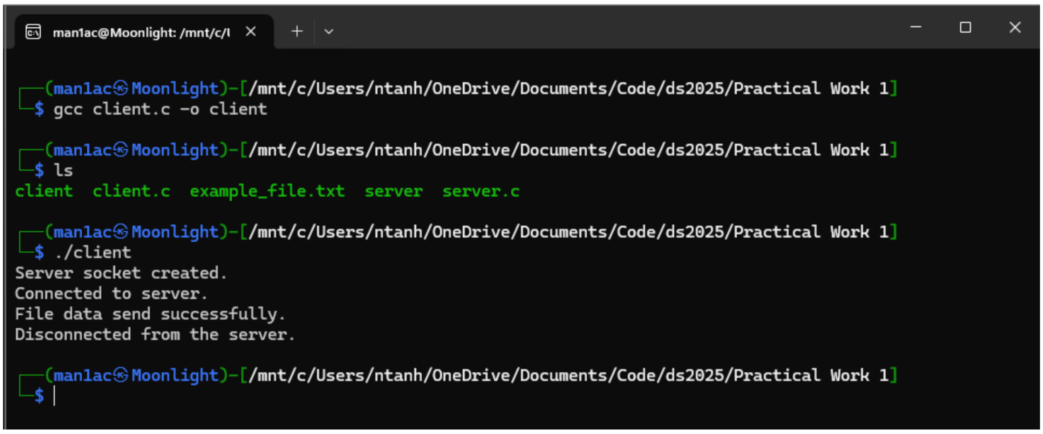

Figure 4: Client Command Line Interface

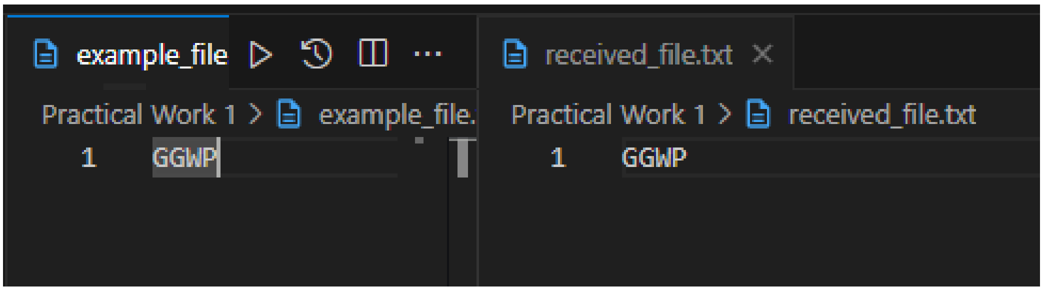

Figure 5: example file and received file

4.3.2 Testing with groupmate

Change the server and client IP address to server IP address to connect two different computers together

```
int main ()
{
    char *ip = "192.168.113.131"; //Both client and server
    int port = 8080;
    int e;
```

Figure 6: Server Command Line Interface - NhiDB computer

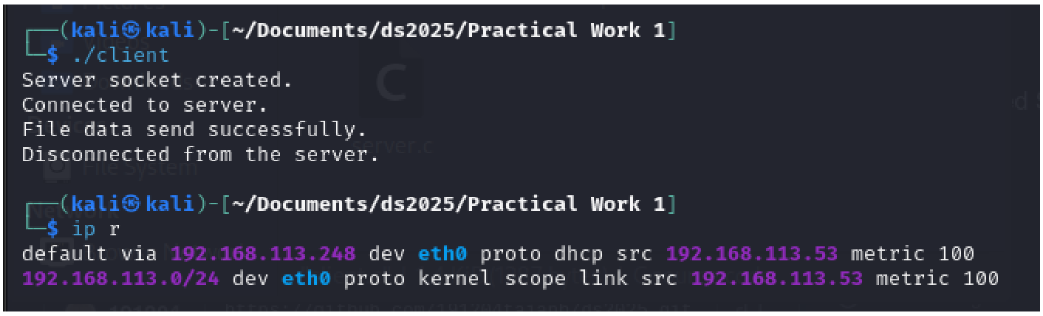

Figure 7: Client Command Line Interface - my computer

5 Conclusion

In this report, we have outlined the design and implementation of a file transfer system using TCP/IP in a command-line interface. The system allows for reliable file transfer between a client and a server. The custom protocol ensures efficient and secure transfer, and the implementation is robust enough to handle different file sizes.