File Transfer over TCP/IP

Truong Quang Huy - BA12-087

November 30, 2024

Contents

1 Introduction **3**

2 Protocol Design **4**
2.1 Overview of the Protocol 4
2.2 Protocol Design Steps 4
2.3 Figure 1: Protocol Design Flow 4

3 System Organization **6**
3.1 System Architecture 6
3.2 System Organization 6

4 File Transfer Implementation **7**
4.1 Client Code 7
4.2 Server Code 9
4.3 Implementing File Transfer 11
4.3.1 Testing with myself 11

5 Conclusion **11**

1 Introduction

In contemporary computing, transferring files is a vital operation, particularly in networked environments. The Transmission Control Protocol/Internet Protocol (TCP/IP) serves as the backbone of most networking systems. This report presents the design of a straightforward yet efficient file transfer system that utilizes TCP/IP over a Command Line Interface (CLI). The system enables the transfer of files between two machines using a custom-built protocol.

2 Protocol Design

2.1 Overview of the Protocol

The file transfer protocol (FTP) was developed to enable the efficient and secure exchange of files between a client and a server. It operates over TCP/IP, leveraging the inherent capabilities of the TCP connection, such as error detection, data integrity, and retransmission, to ensure reliable communication.

2.2 Protocol Design Steps

The following steps outline the protocol designed for file transfer:

1. **Connection Establishment:** The client begins the connection process by reaching out to the server through a designated port. The server prepares by binding to an IP address and port, and then listens for incoming client connections.

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

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

4. **Acknowledgment:** Once the client receives each chunk, it sends an acknowledgment back to the server.

5. **Completion:** After all the chunks have been transferred, the server sends a completion signal to the client.

6. **Close Connection:** Once the file transfer is complete, both the client and server properly close the connection.

2.3 Figure 1: Protocol Design Flow

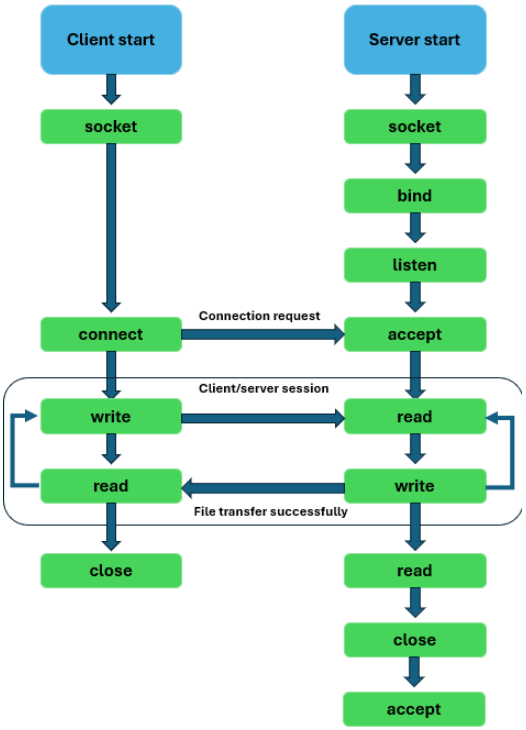

Figure 1: Enter Caption

3 System Organization

3.1 System Architecture

The system is split into two primary components: the client and the server. Each component handles specific tasks in the file transfer process.

- **Client:** Starts the file transfer, sends requests for the file, receives file chunks, downloads the file, and sends acknowledgments.
- **Server:** Waits for connection requests, handles file transfer, and breaks the file into chunks for transmission.

3.2 System Organization

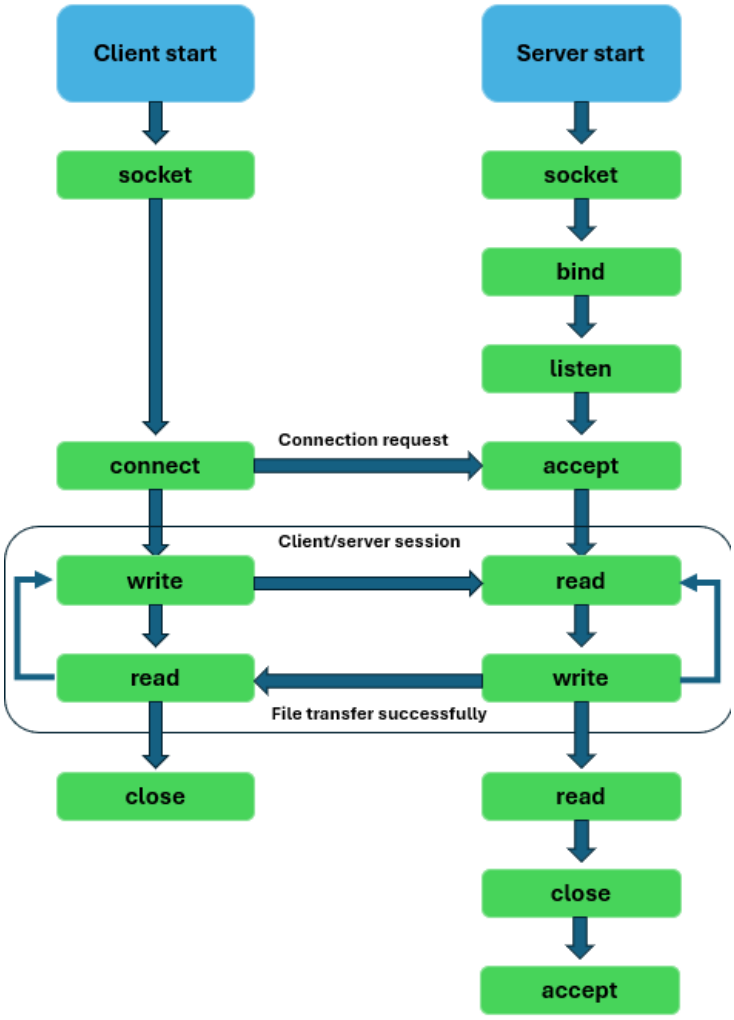

Figure 2: Protocol Design Flow

4 File Transfer Implementation

4.1 Client Code

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

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
    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_add
        if(e<0)
        {
            perror("Error in Binding");
            exit(1);
        }
        printf("Binding Successfull.\n");

        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

5 Conclusion

This report details the design and implementation of a file transfer system utilizing TCP/IP over a Command Line Interface. The system enables reliable file transfers between a client and a server. The custom protocol ensures both efficient and secure transfers, while the implementation is robust enough

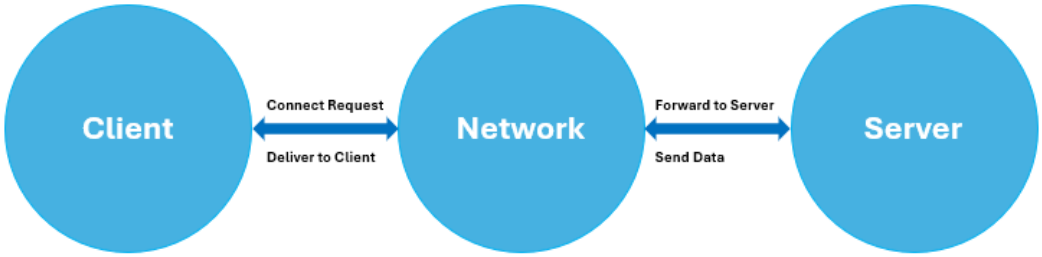

Figure 3: System Organization Diagram

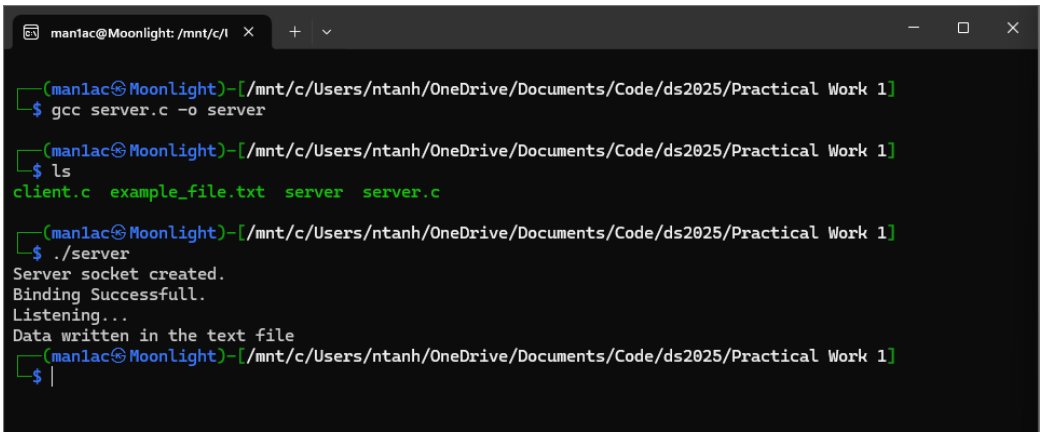

Figure 4: Server Command Line Interface

to handle files of varying sizes.