Practical Work 1: TCP File transfer

Do Trong Dat - 22BI13075

November 30, 2024

Abstract

A simple file transfer protocol implemented using TCP/IP sockets in C. The server listens for incoming connections on a specified port and receives a file from the client. The client connects to the server and sends the file contents.

1 Protocol Design

The protocol follows a basic request-response pattern:

1.1 Client

The client follows the steps outlined below:

1. Resolves the hostname using `gethostbyname()`.
2. Handles errors during operations like socket creation, connection, data sending, and receiving.
3. Create a socket using `socket()` and connect to the server using `connect()`.
4. Open and read a file in chunks using `fread()`.
5. Send data in chunks using `send()`.
6. Close file and socket after communication is complete.

```
dat@LAPTOP-4BOGORQ8:/mnt/c/Users/LOQ/Desktop/ds2025/File_transfer_using_TCP_socket$ ./client
[+] Connected to server successfully
[+] File sent successfully
```

1.2 Server

The server follows these steps:

1. Listens for incoming connections on port 8080.
2. Accepts connections from clients.

3. Receives file data in chunks using `recv()`.
4. Writes the received data to a file (e.g., `test2.txt`).
5. Closes all resources after the transfer is complete.

```
dat@LAPTOP-4BOGORQ8:/mnt/c/Users/LOQ/Desktop/ds2025/File_transfer_using_TCP_socket$ ./server
[+] Server is listening on port 8080
[+] Connection accepted
[+] File received successfully
```

2 System Organization

The system consists of two main components:

2.1 Client

The client connects to the server, sends the filename, and transmits the file data in chunks.

Listing 1: Client C Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h> // For gethostbyname()

#define PORT 8080
#define SIZE 1024

int open_clientfd(char *hostname, int port) {
    int clientfd;
    struct hostent *hp;
    struct sockaddr_in serveraddr;

    if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("[-] Socket creation failed");
        return -1;
    }

    if ((hp = gethostbyname(hostname)) == NULL) {
        perror("[-] DNS resolution failed");
        return -1;
    }
```

```
    bzero((char *)&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    bcopy((char *)hp->h_addr_list[0], (char *)
    &serveraddr.sin_addr.s_addr, hp->h_length);
    serveraddr.sin_port = htons(port);

    if (connect(clientfd, (struct sockaddr *)&serveraddr,
    sizeof(serveraddr)) < 0) {
        perror("[-] Connection failed");
        return -1;
    }

    return clientfd;
}

int main() {
    char *hostname = "127.0.0.1"; // Replace with domain name if needed
    char buffer[SIZE];
    FILE *file;
    int clientfd;

    clientfd = open_clientfd(hostname, PORT);
    if (clientfd < 0) {
        return 1; // Exit if connection fails
    }
    printf("[+] Connected to server successfully\n");

    file = fopen("test.txt", "r");
    if (file == NULL) {
        perror("[-] Error opening file");
        close(clientfd);
        return 1;
    }

    ssize_t bytes_read, bytes_sent;
    while ((bytes_read = fread(buffer, 1, SIZE, file)) > 0) {
        bytes_sent = send(clientfd, buffer, bytes_read, 0); // Using send()
        if (bytes_sent < 0) {
            perror("[-] Error sending data");
            fclose(file);
            close(clientfd);
            return 1;
        }
    }

    printf("[+] File sent successfully\n");
```

```
    fclose(file);
    close(clientfd);
    return 0;
}
```

2.2 Server

The server listens for incoming connections on port 8080, receives file data in chunks, writes it to a file, and closes all resources after completion.

Listing 2: Sever C Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>
#include <strings.h> // For bzero()

#define PORT 8080
#define SIZE 1024
#define LISTENQ 10 // Maximum number of pending connections

typedef struct sockaddr SA;

int open_listenfd(int port) {
    int listenfd, optval = 1;
    struct sockaddr_in serveraddr;

    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("[-] Socket creation failed");
        return -1;
    }

    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
    (const void *)&optval, sizeof(int)) < 0) {
        perror("[-] Set socket options failed");
        return -1;
    }

    bzero((char *)&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
```

```
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port = htons((unsigned short)port);

    if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0) {
        perror("[-] Bind failed");
        return -1;
    }

    if (listen(listenfd, LISTENQ) < 0) {
        perror("[-] Listen failed");
        return -1;
    }

    return listenfd;
}

int main() {
    char buffer[SIZE];
    struct sockaddr_in clientaddr;
    socklen_t clientlen = sizeof(clientaddr);
    int listenfd, connfd, file_fd;

    listenfd = open_listenfd(PORT);
    if (listenfd < 0) {
        return 1; // Exit if server setup fails
    }
    printf("[+] Server is listening on port %d\n", PORT);

    connfd = accept(listenfd, (SA *)&clientaddr, &clientlen);
    if (connfd < 0) {
        perror("[-] Accept failed");
        close(listenfd);
        return 1;
    }
    printf("[+] Connection accepted\n");

    file_fd = open("test2.txt", O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (file_fd < 0) {
        perror("[-] Error opening/creating file");
        close(connfd);
        close(listenfd);
        return 1;
    }

    ssize_t bytes_received;
    while ((bytes_received = recv(connfd, buffer, SIZE, 0)) > 0) {
```

```
    // Using recv()
        if (write(file_fd, buffer, bytes_received) != bytes_received) {
            perror("[-] Error writing to file");
            close(file_fd);
            close(connfd);
            close(listenfd);
            return 1;
        }
    }

    if (bytes_received < 0) {
        perror("[-] Error receiving data");
    }

    printf("[+] File received successfully\n");

    close(file_fd);
    close(connfd);
    close(listenfd);
    return 0;
}
```

3 Conclusion

This labwork provided valuable hands-on experience with TCP/IP and socket programming, enabling us to implement and test a file transfer system in a client-server architecture. Challenges such as EOF management and connection errors were effectively handled through robust exception handling and edge-case testing. This experience highlighted the importance of secure and scalable communication systems, motivating future enhancements such as encryption for secure data transfers and multi-client support for increased usability. These insights have direct applications in developing file-sharing platforms and remote backup systems.

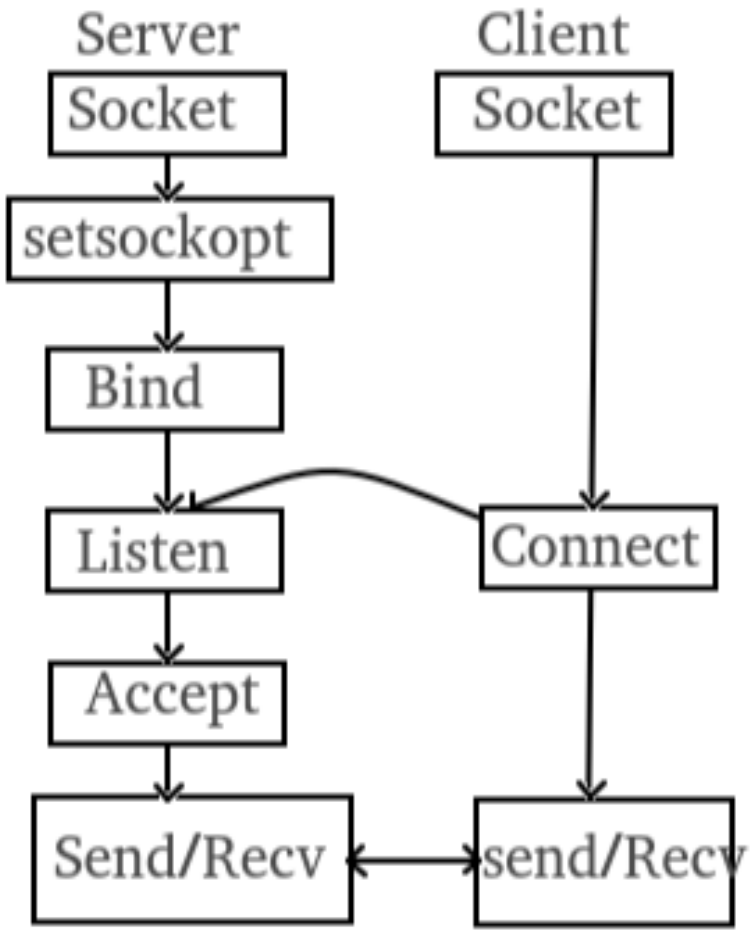

Figure 1: Client-Server Interaction for TCP File Transfer

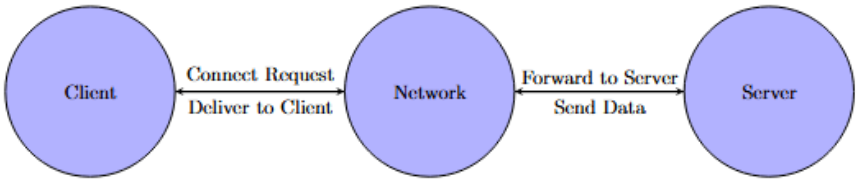

Figure 2: System Architecture for TCP File Transfer