University of Science and Technology of Hanoi

Distributed System

PRACTICAL WORK 1: TCP FILE TRANSFER

Group members:

Nguyen Phuong Thao (BI9-212)

Doan Tuyet Mai (BI9-162)

Trinh Thao Phuong (BI9-191)

Phung Kim Son (BI9-202)

Pham Minh Long (BI9-146)

Hanoi, Feb 2021

1. Protocol design

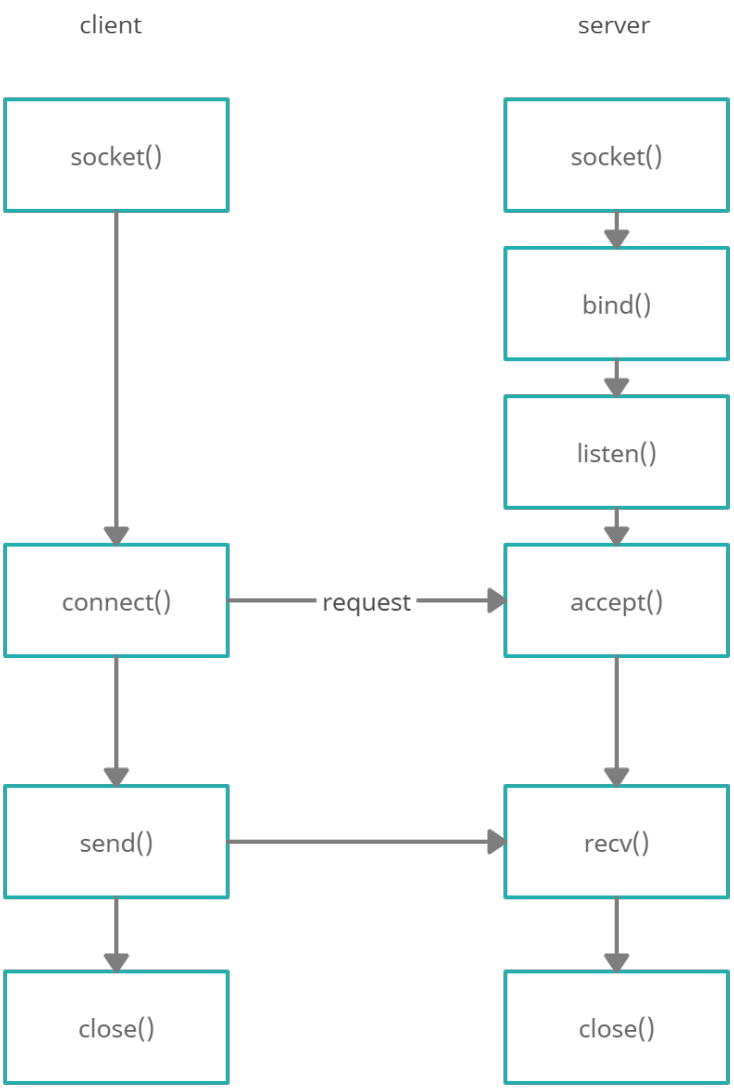

Figure 1: Protocol design

In the server side, at first a socket is created, then it is assigned an address using `bind()` function and wait for the client. Then in the client side, after a socket is created, the function `connect()` is used to connect the client to the server. The server uses `accept()` function to accept the connection from the client. Once connection has succeeded, the client starts to send the file to the connected server and the server receives the file. After transmission is completed, sockets must be closed at both endpoints.

2. System organization

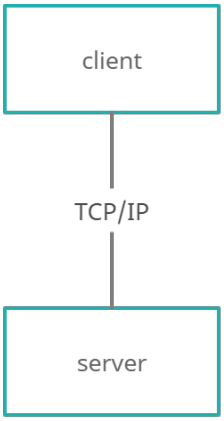

Figure 2: System organization

The server socket will be bind to port with host byte order 12345 after created. It will then listening to data received. The socket will start listening for the connections and accept the connection from the client. Both client and server sending each other data until ones decided to close the socket in open session.

3. File transfer implementation

Full code implementation could be found here, in the `code` directory.

3.1 Client-side

The code for client-side implements these following steps:

1. Create a socket, initialize the address of the socket and connect to the server.
2. Generate a file containing message from the client.
3. Open the generated file.
4. Send the generated file to the server.
5. Close the socket.
6. The program is stopped.

Firstly, to create a socket:

```
// create socket
int serv = socket(AF_INET, SOCK_STREAM, 0);
```

where `AF_INET` is an address family that is used to designate the type of addresses that your socket can communicate with, and `SOCK_STREAM` indicates that this is a TCP socket.

Next, take the hostname from the second argument (argument at the index one) from the command line. From that we could get the address of the server with `gethostbyname()`. Set the port of the socket the same value as of the server so that they can be connected. Then establish a connection to the server.

```c
#include <sys/socket.h>
#include <netdb.h>

struct sockaddr_in ad;
socklen_t ad_length = sizeof(ad);
struct hostent *hep;

// init address
hep = gethostbyname(argv[1]);
memset(&ad, 0, sizeof(ad));
ad.sin_family = AF_INET;
ad.sin_addr = *(struct in_addr *)hep->h_addr_list[0];
ad.sin_port = htons(12345); // translates host byte order to network byte order

// connect to server
connect(serv, (struct sockaddr *)&ad, ad_length);
```

After the client and the server being connected, the file transfer process could be implemented.

First, generate a file containing the message:

```c
void write_file(char* content){
   FILE *fp;
   fp = fopen("sample.txt", "w");
   fprintf(fp, content);
   fclose(fp);
}
```

Then to send the file, first, we read the file that will be sent to have the input of `FILE` datatype, extract the data with `fgets()`, then send over the connection with the `send()` function:

```c
#define SIZE 1024
void send_file(FILE *fp, int sockfd){
    int n;
    char data[SIZE] = {0};

    while (fgets(data, SIZE, fp) != NULL){
        // send data over connection
      if (send(sockfd, data, sizeof(data), 0) == -1){
         perror("Can't send the file!");
         exit(1);
      }
      bzero(data, SIZE);
    }
}

// read generated file
fp = fopen("sample.txt", "r");
if (fp == NULL){
   perror("Error in reading file.");
   exit(1);
}
send_file(fp, serv);
```

where `sockfd` is the the file descriptor associated with the socket.

Finally, close the socket and stop the program:

```c
// close connection
close(serv);
```

3.2 Server-side

The code for the server-side implements these following steps:

1. Create a socket, initialize the address of the socket and connect to the server.
2. Generate a new file.
3. Received the data from the client.
4. Open the generated file and write into file.
5. Close the socket.
6. The program is stopped.

Create a socket, initialize the address of the socket and connect to the server.

```
 // create the socket
ss = socket(AF_INET, SOCK_STREAM, 0);
// bind the socket to port 12345
memset(&ad, 0, sizeof(ad));
ad.sin_family = AF_INET;
ad.sin_addr.s_addr = INADDR_ANY;
ad.sin_port = htons(12345);
bind(ss, (struct sockaddr *)&ad, ad_length);
```

The server start listening for the connections.

```
listen(ss, 0);
```

We have a function to write into file.

```
void write_file(int sockfd){
   int n;
   char buffer[SIZE];

   FILE *fp;
   fp = fopen("received_sample.txt","w");
   while (1){
      n = recv(sockfd, buffer, SIZE, 0);
      if (n <= 0){
         break;
         return;
      }
      fprintf(fp, "%s", buffer);
      bzero(buffer, SIZE);
   }
   return;
}
```

The server accept the connection from the client and write data received into a new file with above write file function.

```
while (1) {
       // an incoming connection
       cli = accept(ss, (struct sockaddr *)&ad, &ad_length);

       pid = fork();
       if (pid == 0) {
           printf("client connected\n");
           write_file(cli);
           printf("Received file.");
```

```
            return 0;
        }
        else {
            // continue the loop to accept more clients
            continue;
        }
    }
```

Finally, close the socket and stop the program:

```
// close connection
close(cli);
```

4. Contribution

Student	**Student ID**	**Contribution**
Pham Minh Long	BI9-146	Protocol design
Phung Kim Son	BI9-202	System organization
Trinh Thao Phuong	BI9-191	Draw figures
Doan Tuyet Mai	BI9-162	Code for file transfer Explanation for the implementation of Client-side
Nguyen Phuong Thao	BI9-212	Explanation for the implementation of Server-side