Project Report

Design and Implementation of FTP Server

Course: *Internet Applications*

Name: *Lian Yibo (2018212672)*

Yang Zhengxiao (2018212680)

Date: 2021/6/16

1. Overview

The requirement of the project is implementing a simple Linux command-line terminal-based FTP client to achieve the data transfer between the FTP server and client and Wireshark can capture the communication between client and server.

The basic commands to be implemented are AUTH, PWD, CWD, PWD, CWD, LIST, MKD, DELE, RNFR/ RNTO, STOR, and RNTR. There are also two expansion functions ,which are speed limit and access levels.

2. Requirements Analysis

2.1 Development Environment

a) Installed Linux system in VM environment

b) TCP-based socket programing

c) C language and related libraries

d) gcc compiler

e) Wireshark capture analysis tool

2.2 Task decomposition and analysis

We can divide the task into two parts. The first part is to create a TCP connection between the FTP server and the client used to control the command; The second part is to convert user commands into control commands and perform some operations according to FTP response code; It is worth noting that to achieve data transmission, we need to establish a new TCP connection for data transmission before data transmission, and close the data connection after transmission.

To complete this task, we need to start the function to configure the server boot, and then complete the most critical part of socket communication.

After the normal communication between the server and the client, we have realized the functions of various commands and the processing of the command line as follows:

User command	Control command	Function
pwd	PWD	Print working directory
cd	CWD	Change working directory
ls	LIST	Display the contents and information of the files in the current directory

mkdir	MKDIR	Create a folder in the current directory
delete	DELE	Delete a file in the current directory
rename	RNFR/RNTO	Change the name of a file in the current directory
get	RNTR	Download the specified file from the server to the client
put	STOR	Upload the specified file from the client to the server

Table 1. Function list

3. Preliminary Design

3.1 Decomposition of functional modules

In our implementation of the FTP server, we have abstracted our code into **THREE** modules: Server Booting and Connection module, Handling Command module, and Close Connection module.

Module Name	Function
Server Booting and Connection module	Listen at port *21*, and accept FTP connection request then set up a TCP connection. In our program, there can only be one login client at the same time.
Handling Command module	The server will send a *220* to the client to initiate the login process. After login, the server will keep listening to the client Control command and find corresponding handling functions.
Close Connection module	When the user Control command is *QUIT*, we will start FTP connection close phase and say goodbye to the client. The current client will login out and server is still listening for more clients.

Table 2. Main functional modules

3.2 Relationship and interfaces between the modules

Server Booting and Connection module are started after we start the server program.

While the Handling Command module is the stage right after the TCP connection is established. At the start of this module is the server sending a *220* response to the

client to start up the login procedure. After login, the server will keep listening to the control commands from the client.

The Close Connection module is started when the Handling Command module receives a *QUIT* control command from the user.

As a result, all three modules are closely related, which gives a high cohesion to the system. The last two modules started because of a particular event happening at the server end.

3.3 Overall flow chart

The overall flow chart is shown in the following figure 1.

Firstly, we will boot the server to keep listening. When a client tries to establish a connection with the server, it will accept the connection request and send a *220* response to start the login process.

After login, the server will listen to user commands.

- If login not successful (except for USER and PASS command) or the receiving command is not accessible for this login user.
- When the received control command is concerned with *PORT*, the server will ask for establishing a new TCP connection for data transmission.
- When the received control command is *QUIT*, close connection with this particular client and keep listening.
- Other control commands will be handled according to corresponding functions.

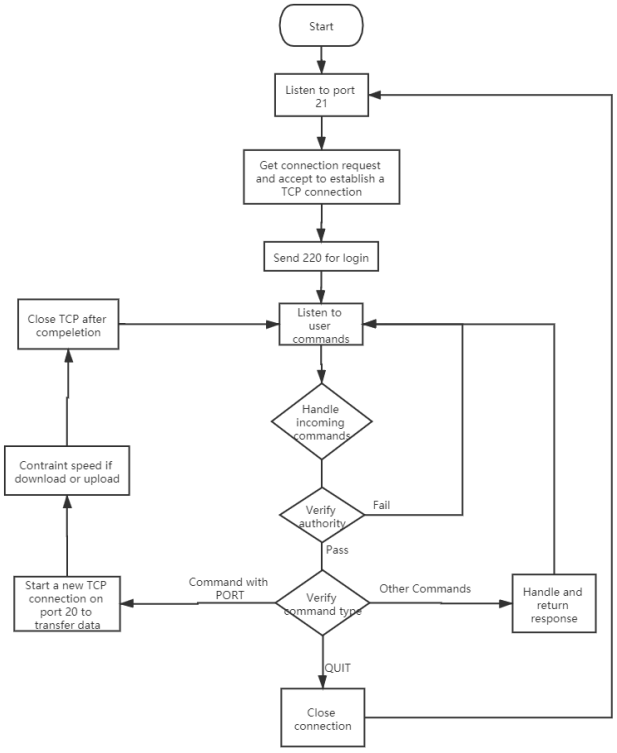

Figure 2. Overall flow chart of the FTP server

3.4 Design of data structures

① FtpClient

We defined a structure called *FtpClient* as shown in the following figure. The aim of designing this structure is to model client attributes. When the FTP server receives a *PORT* control command from the client, we will handle the argument of the *PORT* (for example, the incoming message is *PORT* 127,0,0,1,12,31) to transform it into the IP address and port number of the client.

```
//Define structure of ftp client
struct FtpClient
{
    int         _client_socket;//Client's socket
    char        _ip[MAX];//Client's ip
    //int       _data_socket;//Data socket
    int         _dataport;//Data port which is established to send data
};
```

Figure 3. FtpClient structure

The reason why we did not design a corresponding server structure is that we used the socket created by socket() function to represent the server-side, which is enough information to use in our program.

② Global Variables

There are also three global variables we declared in the ftp.c. The variable names and functions can be summarized in the following table.

Global variable name	Function
loginSuccess	Default set to 0. If login is successful, it will be set to 1. When the server receives control commands from the client, it will verify whether the user login successfully or not.
data_sock	Data socket is used for data transmission. It can only be assigned a value by connecting () function in PORT handling.
level	User authority. We have set user command “get” command and “delete” command as level-3 commands. In the meanwhile, we have also set user “test” as a level-2 authority user and user “super” as a level-3 authority user.

Table 3. Global variables

③ Error code

To make our code more concise and readable, we also defined error messages used

for perror() function in the ftp.h header file, which are shown in the following figure 3.

Two messages are also used when the client entered and left the server defined in the header file.

```
//Define error messages
#define BIND_ERROR "bind"
#define SOCKET_ERROR1 "socket"
#define LISTEN_ERROR "listen"
#define SEND_ERROR "send"
#define RECEIVE_ERROR "recv"
#define ACCEPT_ERROR "accept"
#define CLOSE_ERROR "close"
#define CONNECT_ERROR "connect"
#define REUSE_ERR "Reuse socket failed!\r\n"
#define NO_ARG "No argument provided.\r\n"
#define NO_MSG "No message to send.\r\n"
#define NO_NAME "Recieved Name : No such username\r\n"

#define ACC_CLT "Accept client 127.0.0.1 on TCP port 21.\r\n"
#define CLT_LFT "Client left the server.\r\n"
```

Figure 4. Error messages

4. Detailed Design

① Server Booting and Connection module

This module can be further broken into two individual modules, which are the server booting and listening module, and the connection establishment module.

The server booting and listening module is controlled by a functioning *startup()* in our program. The aim of this function is to take in the port number to be listening on and then after a series of function calls and error prevention. The return value of the function is the socket number of connections if successfully started up to listen to the procedure. Else, the corresponding error code will be returned and printed on the server log, which is negative numbers.

The connection establishment module is started when the server is listening and received a connection establishment request from the client. The server will use *accept()* function to accept *connect()* function from the client. Again, if this process failed, there will be an error printed on the server log. After establishing the connection, we also initialized the values of the struct *FtpClient*.

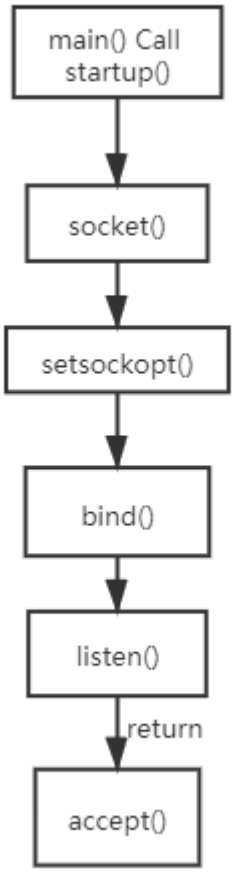

Figure 5. Flow chart of Server Booting and Connection module

② Handling Commands module

This is the core module of our design. After establishing a TCP connection, we enter another endless listening circle (using for(;;)) to listen and handle client control commands. In this part of the listening, we will keep calling a function named *handle_command()* for each iteration. The return value of this function could be 0 or 1. If the return value is 1, it will jump out of this for(;;) and close the connection between this particular client and keep listening for more clients.

To be more concrete. Our function *handle_command()* has two submodules, which are the login and access authentication module coupled in this function, and other command-handling functions like *hand_USER()*.

The login and access authentication module are embedded before each command-handling function except for *USER* and *PASS*, for they are universal procedures of connecting to an FTP server and we will depend on the result of login correctness to check. So, the general process is shown in figure 6. Firstly, we will check if the user successfully login, and then check for the authority level of the user if there is any. If all procedures are passed, the corresponding handling function can be called. Else, corresponding error messages will be returned. The following table shows the authority of different users.

User Name	Password	Access level	Constrained functions
test	123	2	put, delete
Super	123	3	/

Table 4. Login and access control

Handling other functions module can be further separated into two categories, which are commands with *PORT* and commands without *PORT*, where containing *PORT* means there is a new TCP connection on port *20*(FTP data) is set up. We will not separate it into two modules but only to distinct them in the following table and flow chart.

Function name	Description	PORT
handle_USER()	Handle USER control command. We provide two user names “test” and “super” with access level 2 and 3, respectively.	No
handle_PASS()	Handle PASS control command. We assumed all passwords are “123” as a string. loginSucess will be assigned a value.	No
handle_PWD()	Handle PWD control command. This is to look up the current file path.	No
handle_CWD()	Handle CWD control command. This is to change the directory of the server.	No
handle_RETR()	Handle RETR control command. This is to get a file from the server with a limited of 10kB/s.	Yes
handle_LIST()	Handle LIST control command. This is to display a list of directory contents.	Yes
handle_MKD()	Handle MKD control command. This is to make a directory at the server.	No
handle_DELE()	Handle DELE control command. This to delete a file in the server.	No
handle_RNFR()	Handle RNFR control command. This is for renaming.	No
handle_RNTO()	Handle RNTO control command. This is for renaming as well.	No
handle_PUT()	Handle STOR control command. This is for upload files to the server.	Yes

Table 5. Handling different control command

The overall flow chart of this module is shown in figure 6.

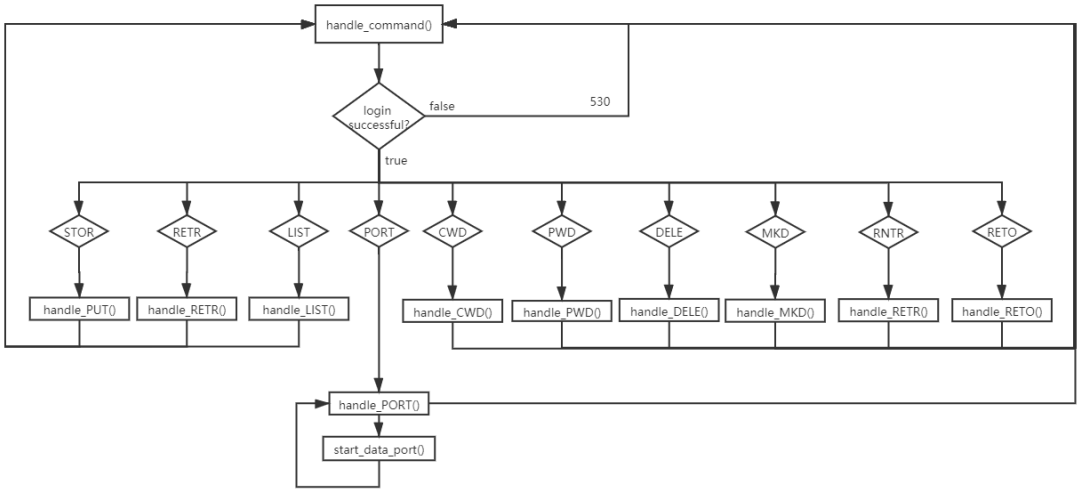

Figure 6. Flow diagram of Handling Commands module

③ Close Connection module

This is rather easy to handle with the previous work. When the server receives a *QUIT* control command from the client. The *handle_command()* will return 1 and the program will jump out of the current for(;;) , then close the connection.

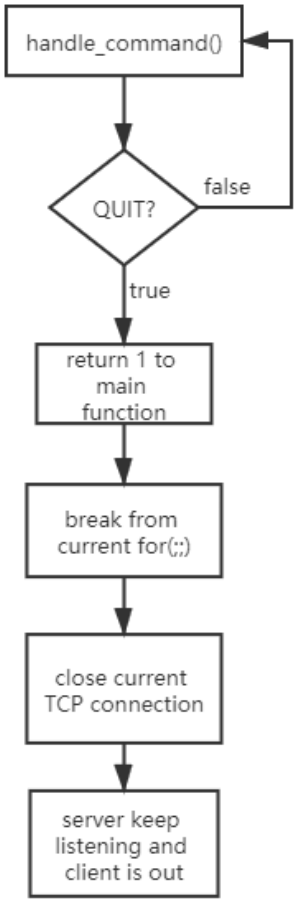

Figure 7. Flow chart of Close Connection module

④ Algorithms of small modules

Constrain speed limit to *10kB/s*

```
int cst_spd = 10;
gettimeofday(&start, NULL); //Start timer
//read files
int tt_tfc = strlen(buf) + (pkt_num - 1) * 255;
//get total traffic of the tranfer for all paktets
sleep_ms(1 + tt_tfc/cst_spd);//Sleep for a while
gettimeofday(&end, NULL);//End timer
```

Bind port *20* when creating new data socket

```
int reuse = 1;
if((setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(reuse
 ))) != 0){//socket reuse setting
    printf(REUSE_ERR);
    return -2;
}
/* Bind to the local address */
if((bind(sock,(struct sockaddr*)&servAddr, sizeof(servAddr))) < 0){
    perror(BIND_ERROR);
    return -3;
}
```

5. Results

5.1 Client

Open ftp client (*USER*、*PASS*)

```
student@BUPTIA:~$ ftp 127.0.0.1
Connected to 127.0.0.1.
220 please enter username
Name (127.0.0.1:student): test
331 please enter password
Password:
230 Login successful!.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Wrong user name

```
220 please enter username
Name (127.0.0.1:student): abc
530 No such user!.
Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Wrong password

```
220 please enter username
Name (127.0.0.1:student): test
331 please enter password
Password:
530 Wrong password!.
Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Once failed to login, you cannot use any command, for example, *pwd*

```
ftp> pwd
530 Please login with USER and PASS!
ftp>
```

When logging in with an account with low authority, some functions will be disabled (*AUTH*)

```
ftp> delete test.c
550 You have no autority!
ftp>
```

Print working directory (*PWD*)

```
ftp> pwd
257 "/home/student/ftpserver"
ftp>
```

Change working directory (*CWD*)

```
ftp> cd ..
250 Directory successfully changed.
ftp> pwd
257 "/home/student"
ftp> cd ftpserver
250 Directory successfully changed.
ftp> pwd
257 "/home/student/ftpserver"
ftp>
```

List current directory (*LIST*)

```
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 56
-rw-r--r-- 1 root    root        6 Jun 19 23:19 123.c
-rwxrwxr-x 1 student student 21923 Jun 19 23:07 ftp
-rw-r--r-- 1 student student 20776 Jun 19 23:00 ftp.c
-rw-r--r-- 1 student student  2229 Jun 19 23:00 ftp.h
WARNING! 5 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

Create a directory (*MKD*)

```
ftp> mkdir halo
257 "/home/student/ftpserver/halo" created.
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 60
-rw-r--r-- 1 root    root        6 Jun 19 23:19 123.c
-rwxrwxr-x 1 student student 21923 Jun 19 23:07 ftp
-rw-r--r-- 1 student student 20776 Jun 19 23:00 ftp.c
-rw-r--r-- 1 student student  2229 Jun 19 23:00 ftp.h
dr-xr----T 2 root    root     4096 Jun 19 23:23 halo
WARNING! 6 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

Create a folder with a duplicate name (*MKD*)

```
ftp> mkdir halo
550 Create directory operation failed.
ftp>
```

Delete files or directories (*DELE*)

```
ftp> delete halo
250 Delete opertaion successful.
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 56
-rw-r--r-- 1 root    root        6 Jun 19 23:19 123.c
-rwxrwxr-x 1 student student 21923 Jun 19 23:07 ftp
-rw-r--r-- 1 student student 20776 Jun 19 23:00 ftp.c
-rw-r--r-- 1 student student  2229 Jun 19 23:00 ftp.h
WARNING! 5 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

Delete a file that does not exist (*DELE*)

```
ftp> delete hhhhhh
550 Delete operation failed.
ftp>
```

Rename files or directories (*RNFR/RNTO*)

```
ftp> rename 123.c 321.c
350 Ready for RNTO.
250 Rename successful.
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 56
-rw-r--r-- 1 root    root        6 Jun 19 23:19 321.c
-rwxrwxr-x 1 student student 21923 Jun 19 23:07 ftp
-rw-r--r-- 1 student student 20776 Jun 19 23:00 ftp.c
-rw-r--r-- 1 student student  2229 Jun 19 23:00 ftp.h
WARNING! 5 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

Rename a file that does not exist (*RNFR*)

```
ftp> rename 333.c 3333.c
550 Failed to rename.
ftp>
```

Rename a file that already exists

```
ftp> rename 333.c 3333.c
550 Failed to rename.
ftp> rename ftp.c 321.c
350 Ready for RNTO.
550 Files with the same name exist.
ftp>
```

Store files (*STOR*)

```
ftp> pwd
257 "/home/student"
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 36
drwxrwxr-x 2 student student 4096 Apr  2 11:49 exercise
drwxrwxr-x 4 student student 4096 Jun 19 23:58 ftpserver
d--------- 2 root    root    4096 Jun 11 22:46 hape
-rw-rw-r-- 1 student student  470 Jun 20 00:02 ls.txt
drwxrwxr-x 3 student student 4096 Jun 17 22:44 model
drwxr-xr-x 2 root    root    4096 May 29 21:57 packets
drwxrwxr-x 2 student student 4096 Jun 11 16:28 test
-rw-r--r-- 1 student student 1632 Jun 19 23:54 test.c
drwxrwxr-x 2 student student 4096 Jun 11 02:09 type
WARNING! 10 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

Before putting

```
ftp> pwd
257 "/home/student/ftpserver"
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 64
-rwxrwxr-x 1 student student 21923 Jun 19 23:07 ftp
-rw-r--r-- 1 student student 20776 Jun 19 23:00 ftp.c
-rw-r--r-- 1 student student  2229 Jun 19 23:00 ftp.h
drw------- 2 root    root     4096 Jun 19 23:45 halo
drw-r-x--T 2 root    root     4096 Jun 19 23:43 hello
-rw-r--r-- 1 student student   470 Jun 19 23:53 ls.txt
WARNING! 7 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

After putting

```
ftp> put test.c
local: test.c remote: test.c
200 PORT command successful.
150 Ok to send data.
226 Transfer complete.
1632 bytes sent in 0.00 secs (40865.4 kB/s)
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 68
-rwxrwxr-x 1 student student 21923 Jun 19 23:07 ftp
-rw-r--r-- 1 student student 20776 Jun 19 23:00 ftp.c
-rw-r--r-- 1 student student  2229 Jun 19 23:00 ftp.h
drw------- 2 root    root     4096 Jun 19 23:45 halo
drw-r-x--T 2 root    root     4096 Jun 19 23:43 hello
-rw-r--r-- 1 student student   470 Jun 19 23:53 ls.txt
-rw-r--r-- 1 root    root     1632 Jun 20 00:07 test.c
WARNING! 8 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

Put a file that already exists

```
ftp> put test.c
local: test.c remote: test.c
200 PORT command successful.
450 Files with the same name exist.
ftp>
```

Retrieve files (*RETR*)

```
ftp> pwd
257 "/home/student/ftpserver"
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 64
-rwxrwxr-x 1 student student 21923 Jun 19 23:07 ftp
-rw-r--r-- 1 student student 20776 Jun 19 23:00 ftp.c
-rw-r--r-- 1 student student  2229 Jun 19 23:00 ftp.h
drw------- 2 root    root     4096 Jun 19 23:45 halo
drw-r-x--T 2 root    root     4096 Jun 19 23:43 hello
-rw-r--r-- 1 student student   470 Jun 19 23:53 ls.txt
WARNING! 7 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

Before getting

```
ftp> pwd
257 "/home/student"
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 32
drwxrwxr-x 2 student student 4096 Apr  2 11:49 exercise
drwxrwxr-x 4 student student 4096 Jun 19 23:58 ftpserver
d--------- 2 root    root    4096 Jun 11 22:46 hape
drwxrwxr-x 3 student student 4096 Jun 17 22:44 model
drwxr-xr-x 2 root    root    4096 May 29 21:57 packets
drwxrwxr-x 2 student student 4096 Jun 11 16:28 test
-rw-r--r-- 1 student student 1632 Jun 19 23:54 test.c
drwxrwxr-x 2 student student 4096 Jun 11 02:09 type
WARNING! 9 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

After getting

```
ftp> get ls.txt
local: ls.txt remote: ls.txt
200 PORT command successful.
150 Opening BINARY mode data connection.
226 Transfer complete.
470 bytes received in 0.05 secs (9.5 kB/s)
ftp> cd ..
250 Directory successfully changed.
ftp> ls
200 PORT command successful.
150 Here comes the directory listening.
total 36
drwxrwxr-x 2 student student 4096 Apr  2 11:49 exercise
drwxrwxr-x 4 student student 4096 Jun 19 23:58 ftpserver
d--------- 2 root    root    4096 Jun 11 22:46 hape
-rw-rw-r-- 1 student student  470 Jun 20 00:02 ls.txt
drwxrwxr-x 3 student student 4096 Jun 17 22:44 model
drwxr-xr-x 2 root    root    4096 May 29 21:57 packets
drwxrwxr-x 2 student student 4096 Jun 11 16:28 test
-rw-r--r-- 1 student student 1632 Jun 19 23:54 test.c
drwxrwxr-x 2 student student 4096 Jun 11 02:09 type
WARNING! 10 bare linefeeds received in ASCII mode
File may not have transferred correctly.
226 Directory send OK.
ftp>
```

Exit the system (*QUIT*)

```
ftp> quit
221 Goodbye.
student@BUPTIA:~$
```

5.2 Server

Open ftp server

```
student@BUPTIA:~/ftpserver$ sudo ./tcp
[sudo] password for student:
listening now
```

When the port is occupied

```
student@BUPTIA:~/ftpserver$ sudo ./tcp
[sudo] password for student:
bind: Address already in use
accept: Bad file descriptor
send: Bad file descriptor
recv: Socket operation on non-socket
User control command :
student@BUPTIA:~/ftpserver$
```

Login (*USER*、*PASS*)

```
Accept client 127.0.0.1 on TCP port 21.
User control command :USER super
Recieved Name : super
Sent 331 to client.
User control command :PASS 123
Recieved Password : 123
Sent 230 to client.
User control command :SYST
Sent 215 to client.
```

Print working directory (*PWD*)

```
User control command :PWD
PWD prints 257 "/home/student"
Sent 257 to client.
```

Change working directory successfully (*CWD*)

```
User control command :CWD ftpserver
Failed to change, sent 550 to client.
```

Failed to change working directory (*CWD*)

```
User control command :CWD hhhhh
Failed to change, sent 550 to client.
```

List current directory (*LIST*)

```
User control command :PORT 127,0,0,1,151,126
Started up TCP connection with client at port 20.
Created new data port :38782, client ip :127.0.0.1.
Sent 200 to client.
User control command :LIST
Prepare to list, sent 150 to client.
Successful to list, sent 226 to client.
```

Create a directory successfully (*MKD*)

```
User control command :MKD halo
Successful to make directory, sent 257 to client.
```

Failed to create a directory (*MKD*)

```
User control command :MKD halo
Failed to make directory, sent 550 to client.
```

Delete files successfully (*DELE*)

```
User control command :DELE halo
Successful to delete, sent 250 to client.
```

Failed to delete files (*DELE*)

```
User control command :DELE halo
Failed to delete, sent 550 to client.
```

Rename a file successfully (*RNFR/RNTO*)

```
User control command :RNFR halo
Success for RNFR, sent 350 to client.
User control command :RNTO hello
Successful to rename, sent 250 to client.
```

The file does not exist (*RNFR)*

```
User control command :RNFR halo
Failed for RNFR, sent 550 to client.
```

The file with the same name exists (*RNTO*)

```
User control command :RNFR halo
Success for RNFR, sent 350 to client.
User control command :RNTO hello
Failed for RNTO, sent 550 to client.
```

Store files (*STOR*)

```
User control command :TYPE I
Set type to Binary
User control command :PORT 127,0,0,1,136,253
Started up TCP connection with client at port 20.
Created new data port :35069, client ip :127.0.0.1.
Sent 200 to client.
User control command :STOR test.c
File doesn't existd, try to create a new one.
Sent 150 to client.
total traffic: 18988 Bytes recieved in 76 pakets
recieving speed :4.875559 kB/s
Successful to put, send 226 to client.
```

The file with the same name exists (*STOR*)

```
User control command :PORT 127,0,0,1,136,20
Started up TCP connection with client at port 20.
Created new data port :34836, client ip :127.0.0.1.
Sent 200 to client.
User control command :STOR test.c
Failed for STOR, sent 450 to client.
```

Retrieve files (*RETR*)

```
User control command :TYPE I
Set type to Binary
User control command :PORT 127,0,0,1,213,173
Started up TCP connection with client at port 20.
Created new data port :54701, client ip :127.0.0.1.
Sent 200 to client.
User control command :RETR ls.txt
Sent 150 to client.
total traffic: 470 Bytes sent in 2 pakets
sending speed :9.527044 kB/s
Sent 226 to client
```

Exit the system (*QUIT*)

```
User control command :QUIT
```

5.3 Wireshark capture results

USER、PASS

301	8.202986000	127.0.0.1	127.0.0.1	FTP	93 Response: 220 please enter username
330	10.483462000	127.0.0.1	127.0.0.1	FTP	78 Request: USER super
332	10.483598000	127.0.0.1	127.0.0.1	FTP	93 Response: 331 please enter password
342	11.245912000	127.0.0.1	127.0.0.1	FTP	76 Request: PASS 123
343	11.245929000	127.0.0.1	127.0.0.1	FTP	90 Response: 230 Login successful!.
345	11.246065000	127.0.0.1	127.0.0.1	FTP	72 Request: SYST
346	11.246073000	127.0.0.1	127.0.0.1	FTP	85 Response: 215 UNIX Type: L8

PWD

361	12.766583000	127.0.0.1	127.0.0.1	FTP	71 Request: PWD
362	12.766631000	127.0.0.1	127.0.0.1	FTP	97 Response: 257 "/home/student/ftpserver"

CWD

29514	324.700846000	127.0.0.1	127.0.0.1	FTP	74 Request: CWD ..
29515	324.700866000	127.0.0.1	127.0.0.1	FTP	103 Response: 250 Directory successfully changed.
29540	327.589948000	127.0.0.1	127.0.0.1	FTP	76 Request: CWD halo
29541	327.589987000	127.0.0.1	127.0.0.1	FTP	99 Response: 550 Failed to change directory.

LIST

420	20.753533000	127.0.0.1	127.0.0.1	FTP	90 Request: PORT 127,0,0,1,200,140
424	20.753579000	127.0.0.1	127.0.0.1	FTP	96 Response: 200 PORT command successful.
426	20.753917000	127.0.0.1	127.0.0.1	FTP	72 Request: LIST
427	20.753926000	127.0.0.1	127.0.0.1	FTP	107 Response: 150 Here comes the directory listening.
434	20.790688000	127.0.0.1	127.0.0.1	FTP	90 Response: 226 Directory send OK.

LIST (ftp-data)

1754	8.388377000	127.0.0.1	127.0.0.1	FTP-DATA	343 FTP Data: 277 bytes

MKD

44354	413.107332000	127.0.0.1	127.0.0.1	FTP	76 Request: MKD halo
44355	413.107477000	127.0.0.1	127.0.0.1	FTP	111 Response: 257 "/home/student/ftpserver/halo" created.
44379	415.408867000	127.0.0.1	127.0.0.1	FTP	76 Request: MKD halo
44380	415.408994000	127.0.0.1	127.0.0.1	FTP	106 Response: 550 Create directory operation failed.

DELE

49149	447.177012000	127.0.0.1	127.0.0.1	FTP	77 Request: DELE halo
49150	447.177154000	127.0.0.1	127.0.0.1	FTP	100 Response: 250 Delete opertaion successful.
49193	453.136659000	127.0.0.1	127.0.0.1	FTP	78 Request: DELE hallo
49194	453.136682000	127.0.0.1	127.0.0.1	FTP	96 Response: 550 Delete operation failed.

RNFR/RNTO

78911	652.636830000	127.0.0.1	127.0.0.1	FTP	79 Request: RNFR test.c
78912	652.636854000	127.0.0.1	127.0.0.1	FTP	87 Response: 350 Ready for RNTO.
78914	652.637231000	127.0.0.1	127.0.0.1	FTP	77 Request: RNTO ts.c
78915	652.637336000	127.0.0.1	127.0.0.1	FTP	90 Response: 250 Rename successful.
78964	659.940918000	127.0.0.1	127.0.0.1	FTP	79 Request: RNFR test.c
78965	659.941068000	127.0.0.1	127.0.0.1	FTP	89 Response: 550 Failed to rename.

STOR

2114	63.010639000	127.0.0.1	127.0.0.1	FTP	74 Request: TYPE I
2115	63.010758000	127.0.0.1	127.0.0.1	FTP	97 Response: 200 Switching to Binary mode.
2117	63.010932000	127.0.0.1	127.0.0.1	FTP	90 Request: PORT 127,0,0,1,154,206
2121	63.011016000	127.0.0.1	127.0.0.1	FTP	96 Response: 200 PORT command successful.
2122	63.011442000	127.0.0.1	127.0.0.1	FTP	79 Request: STOR test.c
2123	63.011482000	127.0.0.1	127.0.0.1	FTP	88 Response: 150 Ok to send data.
2134	63.050993000	127.0.0.1	127.0.0.1	FTP	90 Response: 226 Transfer complete.

STOR (ftp-data)

52053	669.085850000	127.0.0.1	127.0.0.1	FTP-DATA	1698 FTP Data: 1632 bytes

RETR

100861	789.108457000	127.0.0.1	127.0.0.1	FTP	74 Request: TYPE I
100862	789.108473000	127.0.0.1	127.0.0.1	FTP	97 Response: 200 Switching to Binary mode.
100864	789.108626000	127.0.0.1	127.0.0.1	FTP	90 Request: PORT 127,0,0,1,148,124
100868	789.108796000	127.0.0.1	127.0.0.1	FTP	96 Response: 200 PORT command successful.
100869	789.109276000	127.0.0.1	127.0.0.1	FTP	77 Request: RETR ts.c
100870	789.109361000	127.0.0.1	127.0.0.1	FTP	108 Response: 150 Opening BINARY mode data connection.
101027	789.146681000	127.0.0.1	127.0.0.1	FTP	90 Response: 226 Transfer complete.

RETR (ftp-data)

9384	137.151472000	127.0.0.1	127.0.0.1	FTP-DATA	321 FTP Data: 255 bytes
9387	137.199653000	127.0.0.1	127.0.0.1	FTP-DATA	281 FTP Data: 215 bytes

QUIT

107110	839.421280000	127.0.0.1	127.0.0.1	FTP	72 Request: QUIT
107111	839.421357000	127.0.0.1	127.0.0.1	FTP	80 Response: 221 Goodbye.

5.4 Files before and after putting

```
student@BUPTIA:~$ vi test.c

int handle_RETR(int sock,char* arg){
    char file_name[MAX];
    char temp[MAX];
    memset(file_name,'\0',sizeof(file_name));
    sscanf(arg,"%s %s",file_name,temp);

    send_msg(sock,"150 Opening BINARY mode data connection.\r\n");
    printf("Sent 150 to client.\r\n");

    FILE *ck;
    if((ck = fopen(temp,"r")) != NULL){
        send_msg(sock,"450 Files with the same name exist.\r\n");
        printf("Failed for STOR, sent 450 to client.\r\n");
        return;
    }
    FILE *fp;
    fp = fopen(file_name, "r");
    struct timeval start;
    struct timeval end;
    int pkt_num = 0;
    int cst_spd = 10;

    if(fp == NULL){
        printf("The file %s do not exist.\r\n", file_name);
                close(data_sock);
        return 0;
    }else{

        fseek(fp, 0, SEEK_END);
        int size = ftell(fp);

        fseek(fp, 0, SEEK_SET);
                char buf[MAX];
                memset(buf,'\0',MAX);

        gettimeofday(&start, NULL);
```

```
student@BUPTIA:~/ftpserver$ vi test.c

int handle_RETR(int sock,char* arg){
    char file_name[MAX];
    char temp[MAX];
    memset(file_name,'\0',sizeof(file_name));
    sscanf(arg,"%s %s",file_name,temp);

    send_msg(sock,"150 Opening BINARY mode data connection.\r\n");
    printf("Sent 150 to client.\r\n");

    FILE *ck;
    if((ck = fopen(temp,"r")) != NULL){
        send_msg(sock,"450 Files with the same name exist.\r\n");
        printf("Failed for STOR, sent 450 to client.\r\n");
        return;
    }
    FILE *fp;
    fp = fopen(file_name, "r");
    struct timeval start;
    struct timeval end;
    int pkt_num = 0;
    int cst_spd = 10;

    if(fp == NULL){
        printf("The file %s do not exist.\r\n", file_name);
                close(data_sock);
        return 0;
    }else{

        fseek(fp, 0, SEEK_END);
        int size = ftell(fp);

        fseek(fp, 0, SEEK_SET);
                char buf[MAX];
                memset(buf,'\0',MAX);

        gettimeofday(&start, NULL);
```

6. Summary and Conclusion

Because it's a two-person team, the assignment of tasks will be more flexible. In the beginning, Yibo Lian was responsible for building the server. He was responsible for building a system that can communicate with TCP protocol and realizing the login function. The specific code part was the completion of startup, USER, PASS and recv_ msg、Send_ MSG, PORT, and main functions. Then Zhengxiao Yang is responsible for the completion of some functions in FTP, and the code parts are PWD, CWD, list, MKDIR, DELE, RNFR / RNTO. So far, the FTP system construction and basic commands have been realized, and then the two completed the writing of STOR command and RNTR command respectively, and finally completed the writing of the whole simple FTP system.

In this process, we can complete the task on time and according to the quality, and every night two people will meet to discuss the progress of the work, and according to the progress of the task acceptance and distribution. The whole construction period is completed very quickly under the efficient organization. However, there are still some improvements in this process. For example, we should use vsftpd to test and record the whole system before writing. Because when implementing specific FTP commands, some functions such as LIST cannot work normally due to the lack of PORT command. This part of the problem should be understood and solved before writing. For example, advance the priority of the PORT function and complete the writing of the port function in advance.