DESIGN AND IMPLEMENTATION OF FTP CLIENT

COURSE TITLE: *INTERNET APPLICATION*

Name: *FENGYE (2016213363)*

ZHANGLUDAN (2016213358)

Date:

1. Overview

Our project is a simple FTP client which is implemented by C language in Linux operation system. We implement all the basic functions and extension requirements. Our project connects FTP using TCP protocol, it can receive the FTP replies coming from server and translate it into natural language, then change the user's command into FTP command before sending. When we input the correct username and password which is hidden, we can login successfully. Besides, our project supports user's interactive operation and provide these commands: list (ls), directory operation (pwd, cd, mkdir), upload a file (put), download a file (get), delete a file (delete), renaming a file(rename), transfer mode (ascii and binary), and exit (quit). For invalid commands, missing parameters, requested file already existed, the project is able to handle errors. After we finished these basic functions, we implemented the extension requirements: active mode for data connection, resuming transmission from break-point and limiting transmission data rate. Finally, we add the progress bar for get function.

2. RequirementsAnalysis

The steps:

Before we start writing the code for the project, we review the related knowledge, and learn the FTP protocol command. After we review all the lectures, we start coding.Firstly, we write the frame of the project in Xcode, then we divide all the basic functions into 2 groups. One finishes some functions, including cd, mkdir, pwd, delete, binary, ascii and quit. The other one is responsible for some other functions, including ls, rename, passive, get and put. When we finish all these basic functions, we try these functions in MacOS, using "gcc" to compile. Then, all the functions are useable. After that, we start to write code for the extension requirements. Similarly, one member finishes the reget and progress bar function. The other one write the code about active mode and speed limit. Several days later, we finish all the functions although we met some problems. Then we test all the functions in MacOS to check all the functions are useable. Finally, We set the port forwarding in Virtual Box and put the code into Linux system to test all the functions.

3. PreliminaryDesign

3.1 Decomposition of functional modules:

1. Standard input reading and transfer.

   Such as, transfering "ls" to "LIST", then store in the command data structure.

2. Send and receive request/response in commond connection.

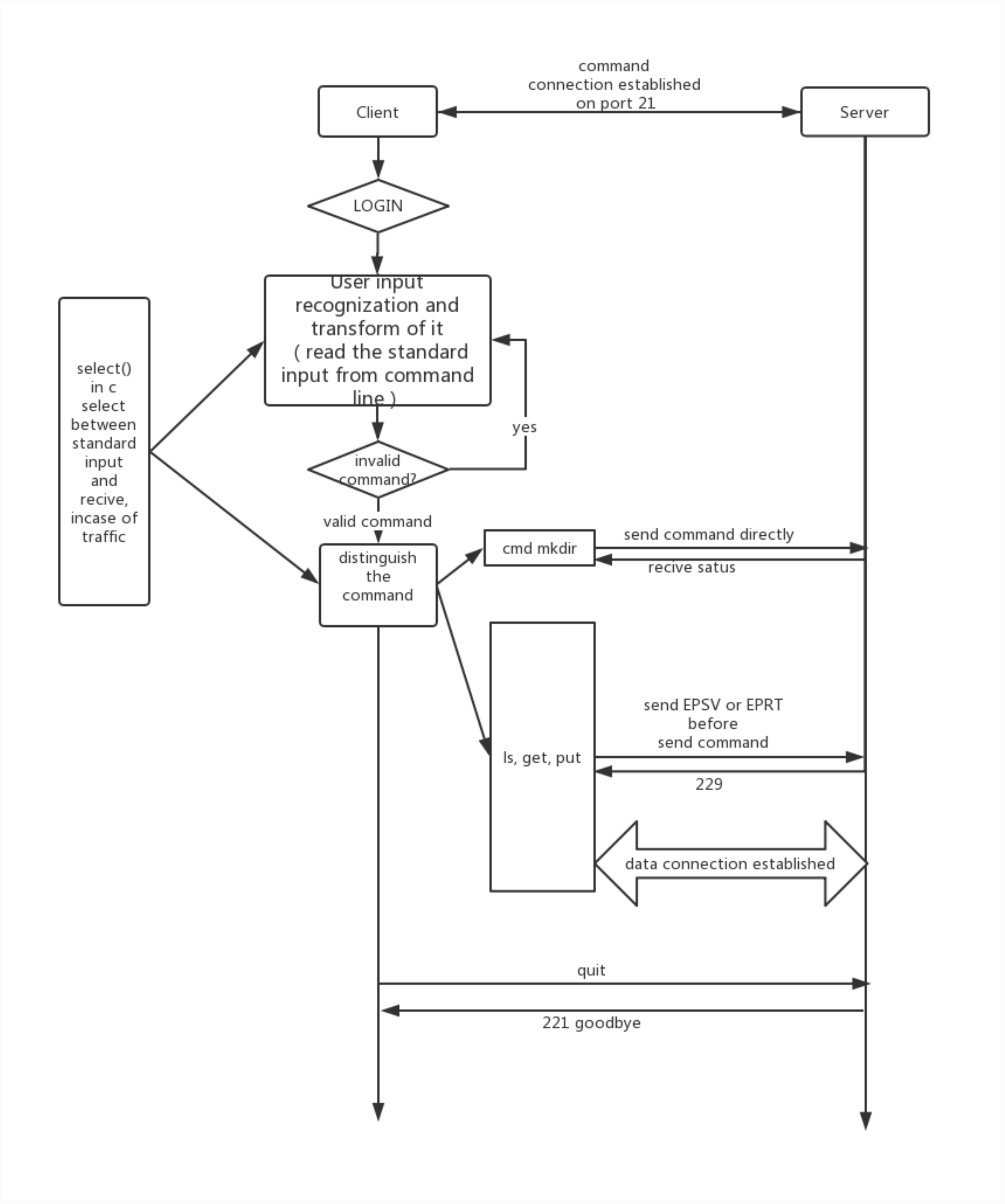
   And implementing some commands which do not need data control. Such as "cd", "pwd"

3. Establish data connection.

   Some command such as LIST, need to establish a data connection.

We tried our best to decompose the project into a series of independent modules, but unfortunately, we fail to do so, because the coupling between command-connection and data-connection still in a high level.

3.2 Relationship and interface between the modules Overall flow chart


command
connection established
on port 21
Client
Server
LOGIN
User input recognization and transform of it (read the standard input from command line)
select() in c select between standard input and recive, incase of traffic
yes
invalid command?
valid command
distinguish the command
cmd mkdir
send command directly
recive satus
ls, get, put
send EPSV or EPRT before send command
229
data connection established
quit
221 goodbye

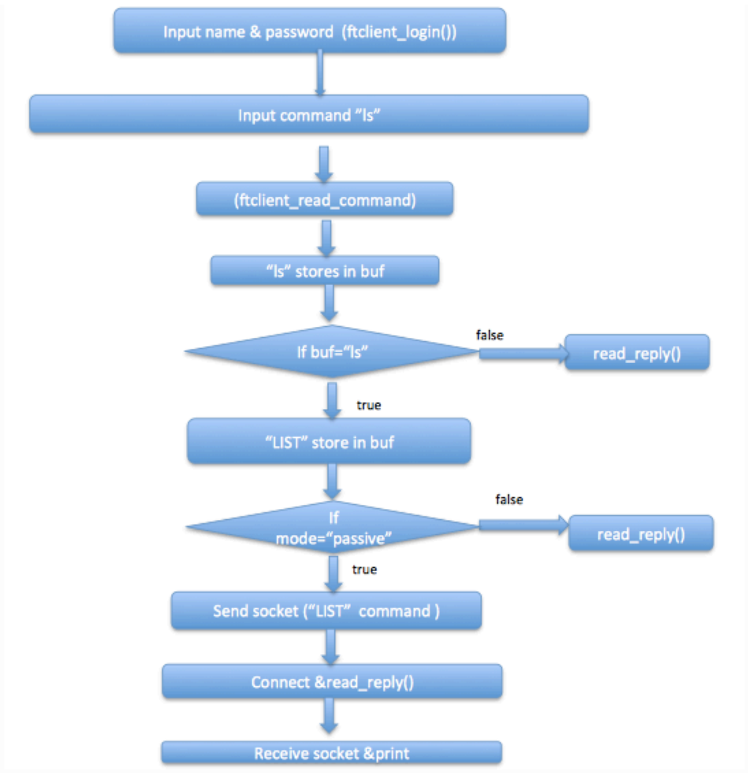
3.4 Design of data structures

Since FTP command is a structured ASCII string, we seperate it as two part, one is for the *command* such as "RETR", the other is argument such as “filename”

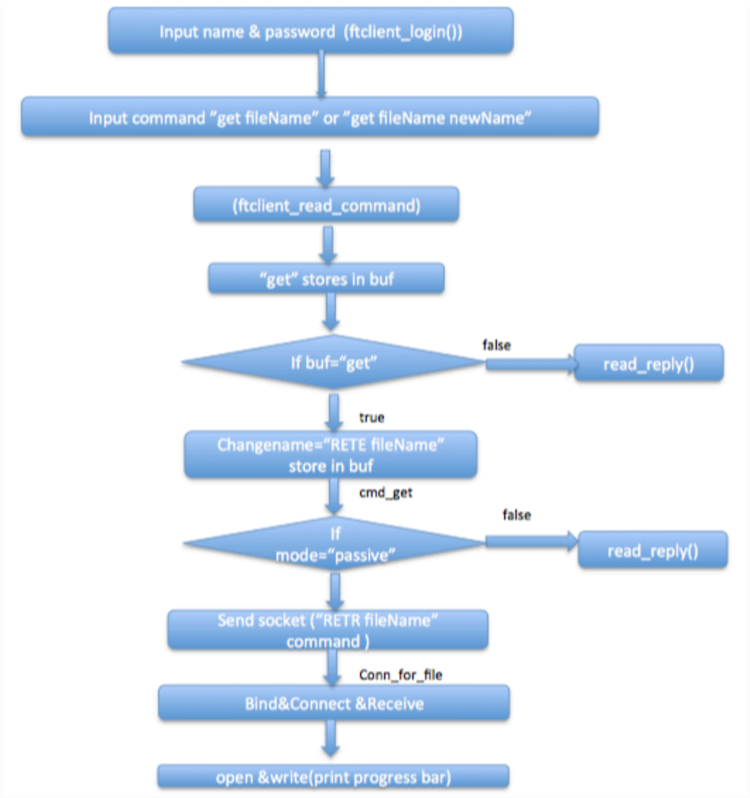
```
struct command {
char arg[255];
char code[5];
}
```

4. DetailedDesign

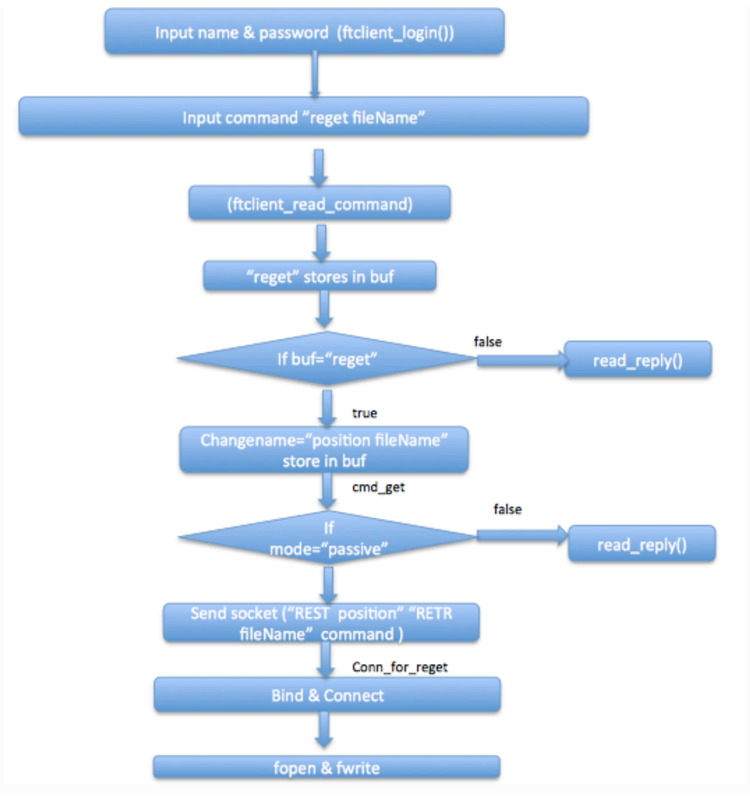
1. List function:



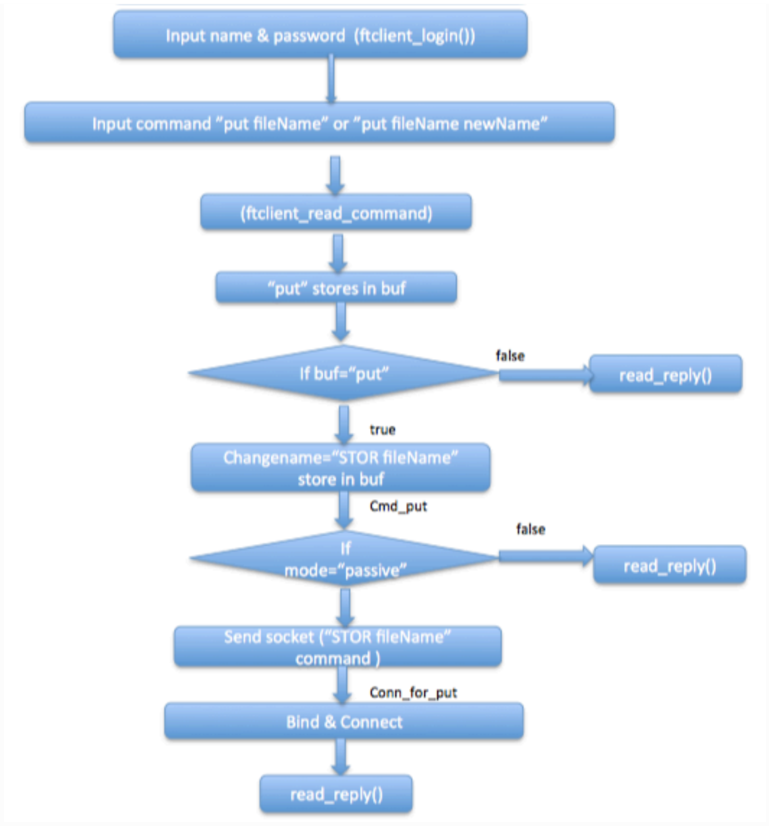
2. Get function:


Input name & password (ftclient_login())
Input command "get fileName" or "get fileName newName"
(ftclient_read_command)
"get" stores in buf
If buf="get"
false
read_reply()
true
Changename="RETE fileName" store in buf
cmd_get
false
If mode="passive"
read_reply()
Send socket ("RETR fileName" command)
Conn_for_file
Bind&Connect &Receive
open &write(print progress bar)

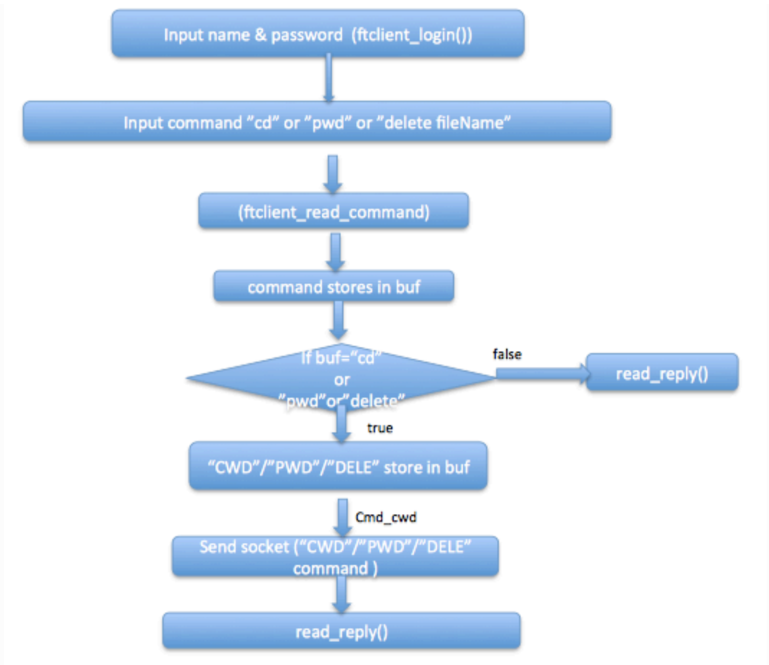
3. Reget function:


Input name & password (ftclient_login())
Input command "reget fileName"
(ftclient_read_command)
"reget" stores in buf
If buf="reget"
false
read_reply()
true
Changename="position fileName" store in buf
cmd_get
false
If mode="passive"
read_reply()
Send socket ("REST position" "RETR fileName" command)
Conn_for_reget
Bind & Connect
fopen & fwrite

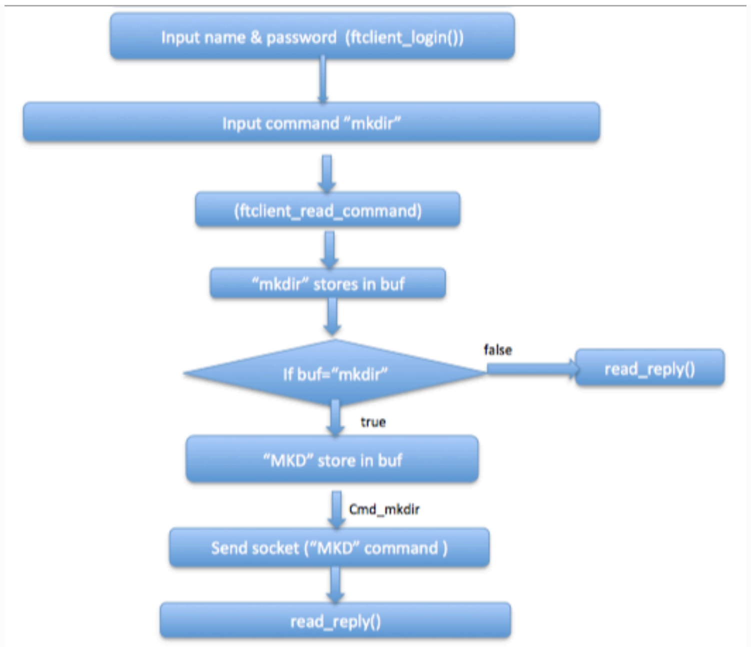
4. Put function:


Input name & password (ftclient_login())
Input command "put fileName" or "put fileName newName"
(ftclient_read_command)
"put" stores in buf
If buf="put"
false
read_reply()
true
Changename="STOR fileName" store in buf
Cmd_put
If mode="passive"
false
read_reply()
Send socket ("STOR fileName" command)
Conn_for_put
Bind & Connect
read_reply()

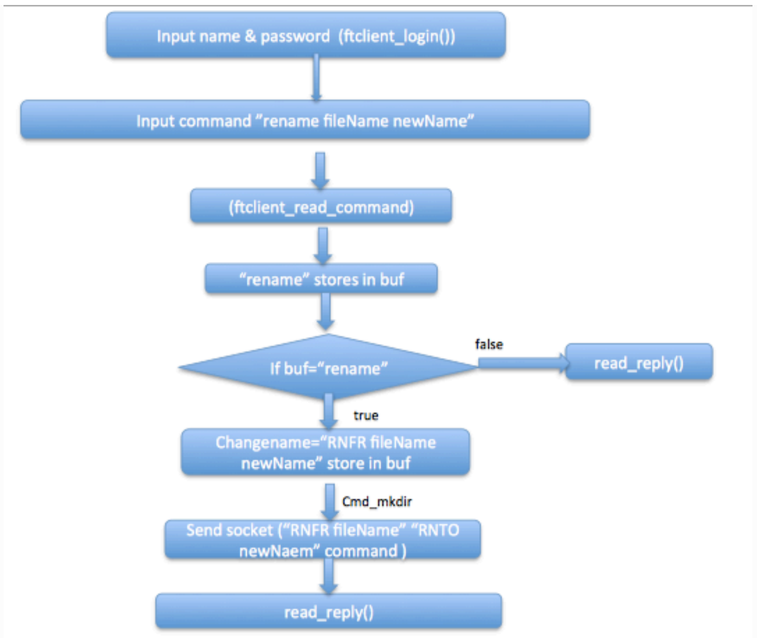
5. cd, pwd, delete function:


Input name & password (ftclient_login())
Input command "cd" or "pwd" or "delete fileName"
(ftclient_read_command)
command stores in buf
If buf="cd" or "pwd"or"delete"
false
read_reply()
true
"CWD"/"PWD"/"DELE" store in buf
Cmd_cwd
Send socket ("CWD"/"PWD"/"DELE" command)
read_reply()

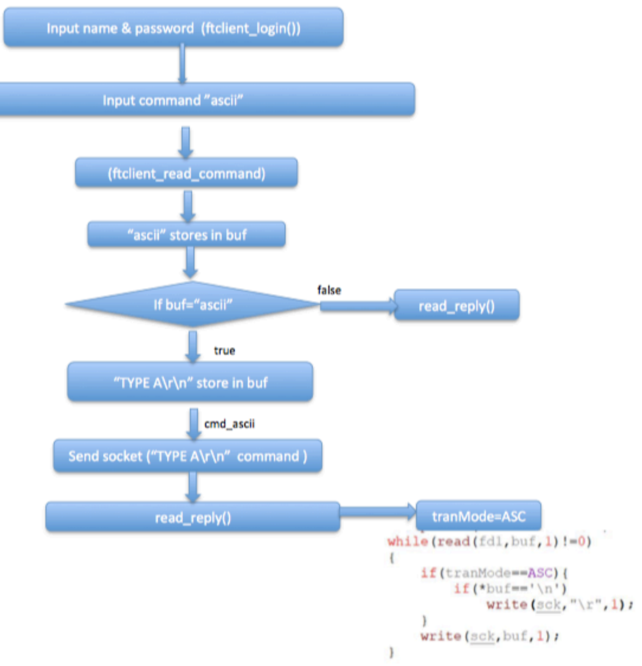
6. Mkdir (mkdir) function:



7. Rename function:


Input name & password (ftclient_login())
Input command "rename fileName newName"
(ftclient_read_command)
"rename" stores in buf
If buf="rename"
false
read_reply()
true
Changename="RNFR fileName newName" store in buf
Cmd_mkdir
Send socket ("RNFR fileName" "RNTO newNaem" command)
read_reply()

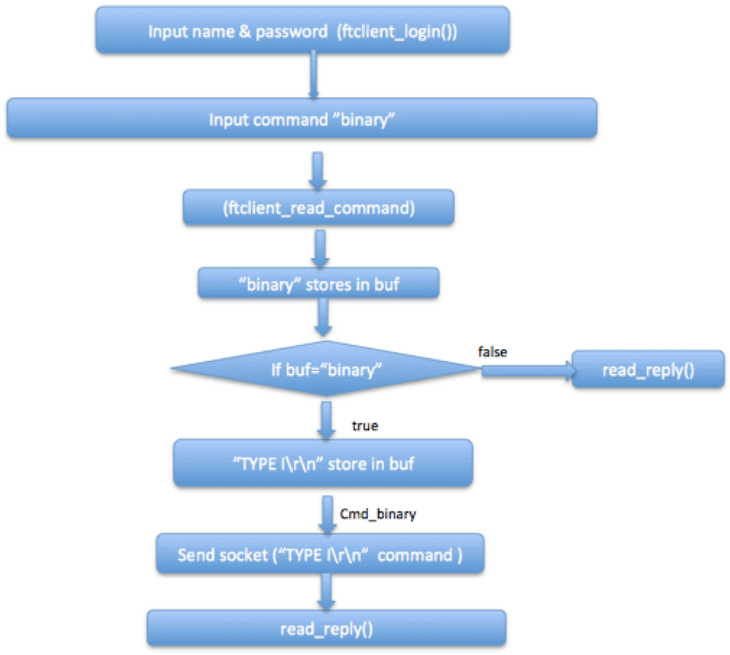
8. Ascii function:



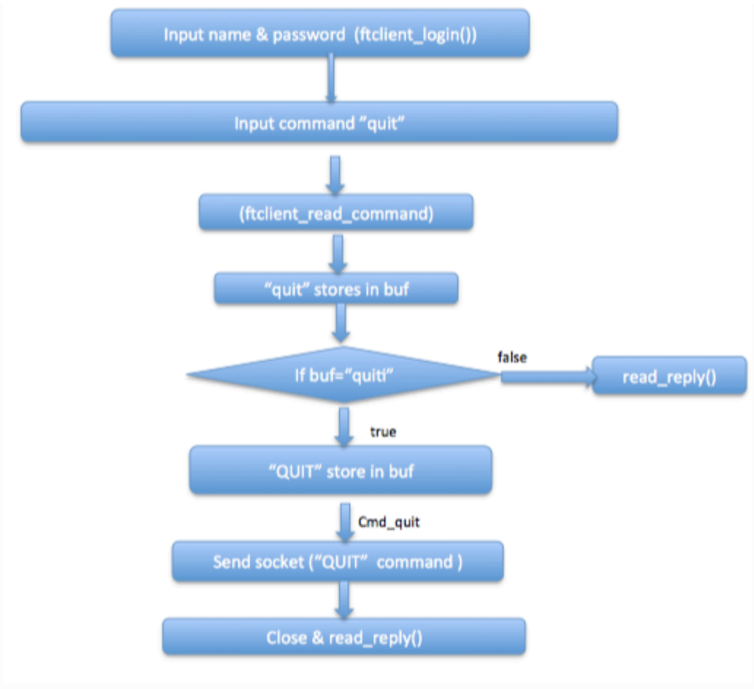
```
while(read(fdl,buf,1)!=0)
{
    if(tranMode==ASC){
        if(*buf=='\n')
            write(sck,"\r",1);
    }
    write(sck,buf,1);
}
```

When transferring files uses this part, many functions need the code about tranMode =ASC, so I display the related code.

9. Binary function:



10. Quit part:



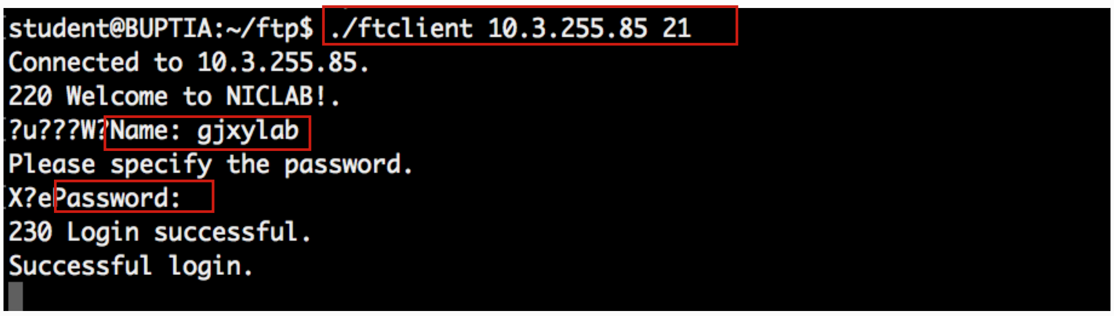
5. Results

Show the execution results with Screen Shot.

User manual:

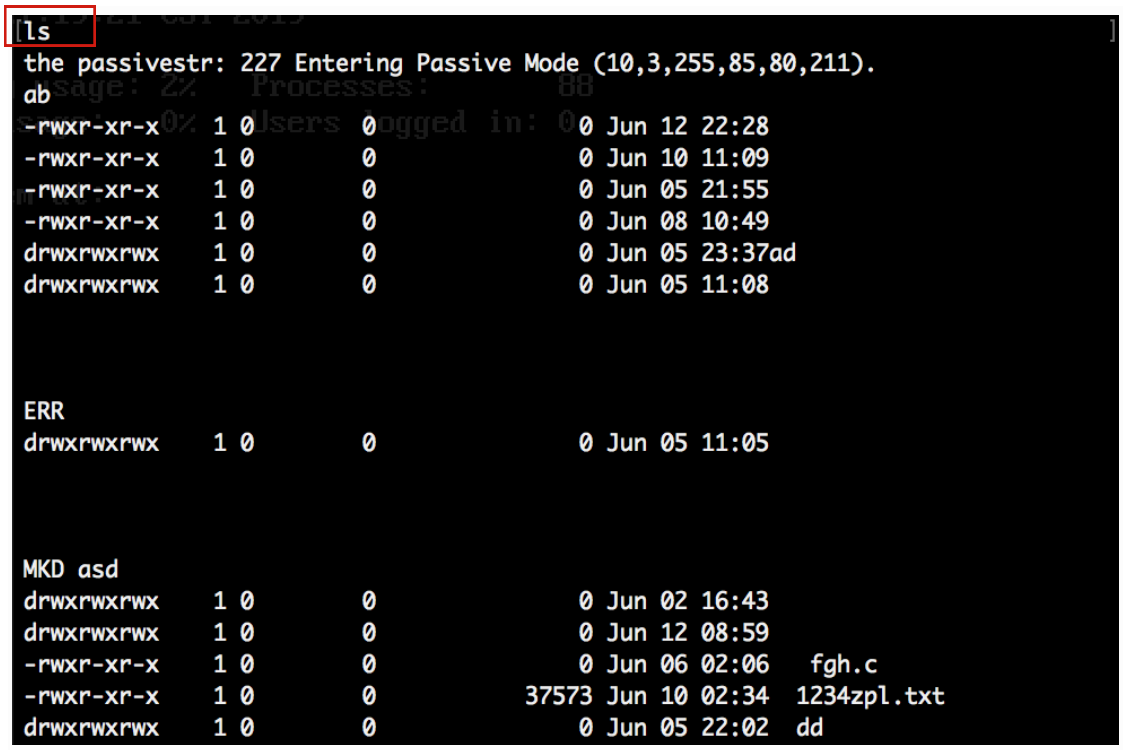
(1). Input the correct username & password to login

compile ftclientfinal.c



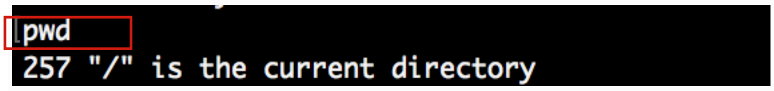
```
student@BUPTIA:~/ftp$ ./ftclient 10.3.255.85 21
Connected to 10.3.255.85.
220 Welcome to NICLAB!.
?u???W?Name: gjxylab
Please specify the password.
X?ePassword:
230 Login successful.
Successful login.
```

(2). Input "ls" can see all the files



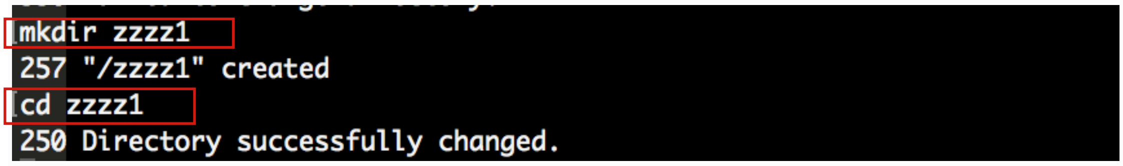
```
[ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,80,211).
ab
-rwxr-xr-x    1 0        0              0 Jun 12 22:28
-rwxr-xr-x    1 0        0              0 Jun 10 11:09
-rwxr-xr-x    1 0        0              0 Jun 05 21:55
-rwxr-xr-x    1 0        0              0 Jun 08 10:49
drwxrwxrwx    1 0        0              0 Jun 05 23:37ad
drwxrwxrwx    1 0        0              0 Jun 05 11:08



ERR
drwxrwxrwx    1 0        0              0 Jun 05 11:05



MKD asd
drwxrwxrwx    1 0        0              0 Jun 02 16:43
drwxrwxrwx    1 0        0              0 Jun 12 08:59
-rwxr-xr-x    1 0        0              0 Jun 06 02:06   fgh.c
-rwxr-xr-x    1 0        0          37573 Jun 10 02:34  1234zpl.txt
drwxrwxrwx    1 0        0              0 Jun 05 22:02  dd
```

(3). Input “pwd” can see the current directory



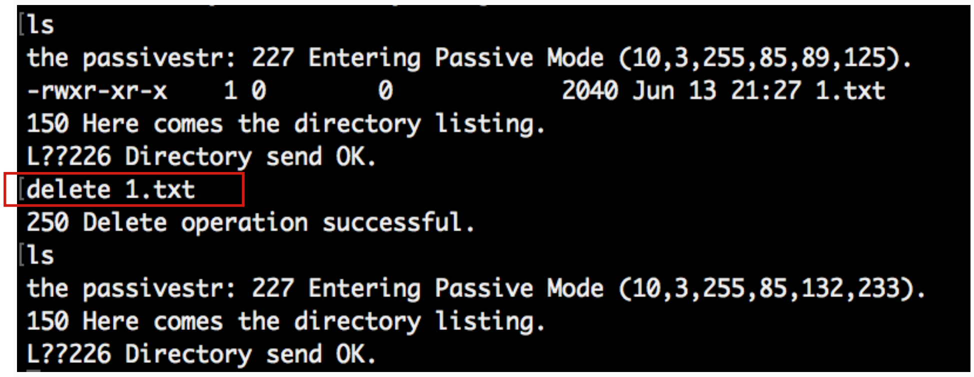
```
[pwd
257 "/" is the current directory
```

(4). Input “mkdir” to make a new directory, then ”cd directoryName” can change the directory.



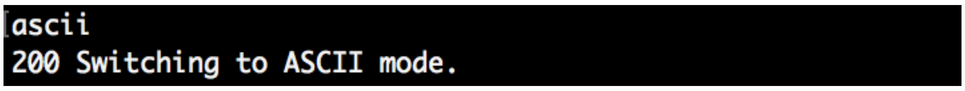
```
[mkdir zzzz1
257 "/zzzz1" created
[cd zzzz1
250 Directory successfully changed.
```

(5). Input ”delete fileName” can delete the related file



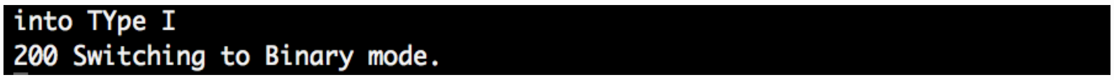
```
[ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,89,125).
-rwxr-xr-x    1 0        0            2040 Jun 13 21:27 1.txt
150 Here comes the directory listing.
L??226 Directory send OK.
[delete 1.txt
250 Delete operation successful.
[ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,132,233).
150 Here comes the directory listing.
L??226 Directory send OK.
```

(6). Input ”ascii ” can into ascii mode



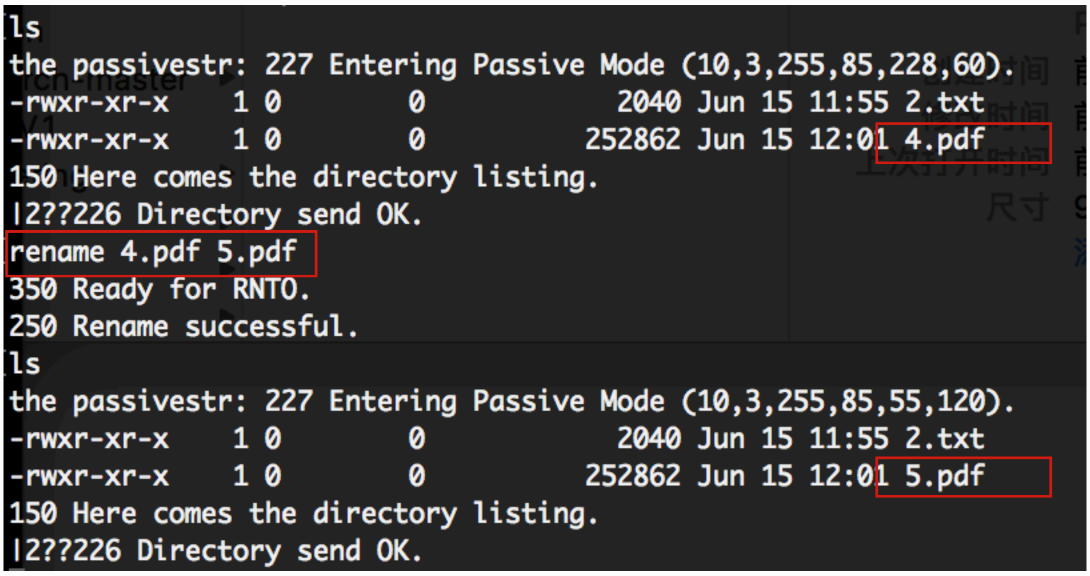
```
ascii
200 Switching to ASCII mode.
```

(7). Input “binary” can into binary mode



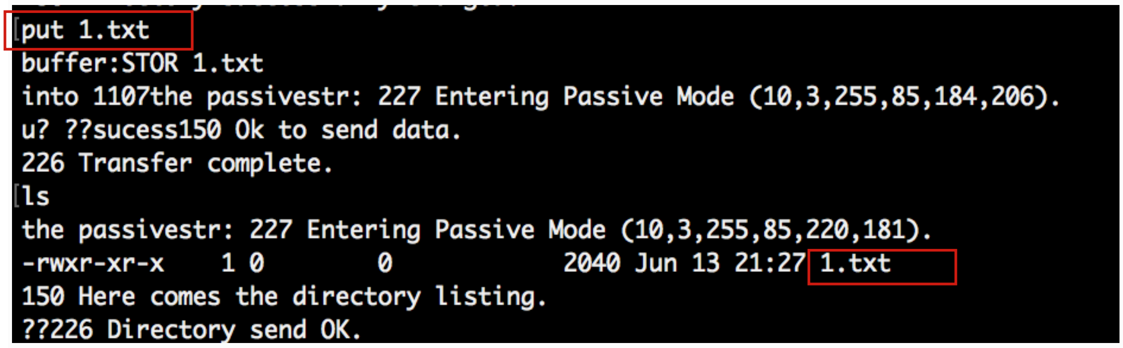
```
into TYpe I
200 Switching to Binary mode.
```

(7.5).Input “rename” to change the filename



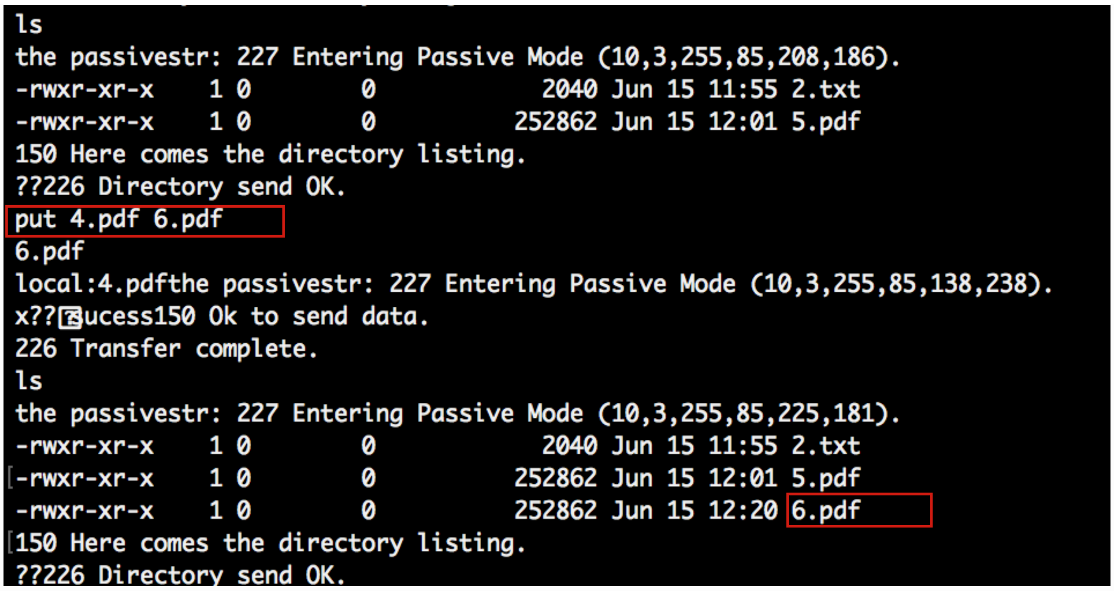
```
ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,228,60).
-rwxr-xr-x    1 0        0            2040 Jun 15 11:55 2.txt
-rwxr-xr-x    1 0        0          252862 Jun 15 12:01 4.pdf
150 Here comes the directory listing.
|2??226 Directory send OK.
rename 4.pdf 5.pdf
350 Ready for RNTO.
250 Rename successful.
ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,55,120).
-rwxr-xr-x    1 0        0            2040 Jun 15 11:55 2.txt
-rwxr-xr-x    1 0        0          252862 Jun 15 12:01 5.pdf
150 Here comes the directory listing.
|2??226 Directory send OK.
```

(8). input “put localFileName” can put a local file to server.

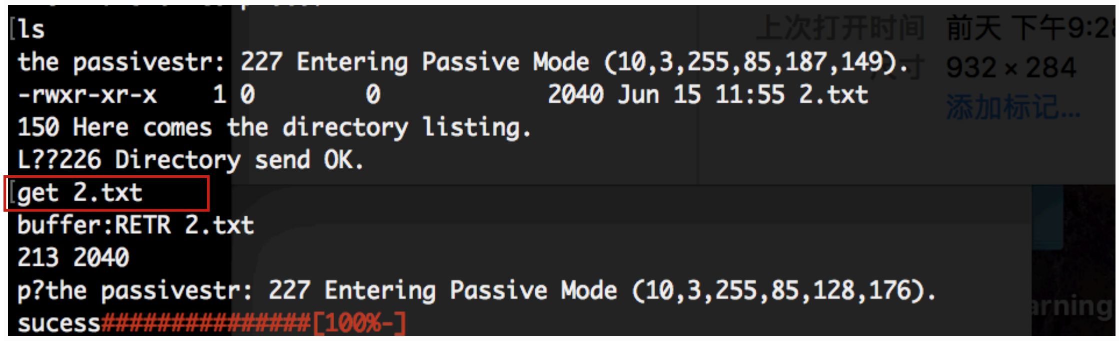


```
put 1.txt
buffer:STOR 1.txt
into 1107the passivestr: 227 Entering Passive Mode (10,3,255,85,184,206).
u? ??sucess150 Ok to send data.
226 Transfer complete.
ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,220,181).
-rwxr-xr-x    1 0        0           2040 Jun 13 21:27 1.txt
150 Here comes the directory listing.
??226 Directory send OK.
```

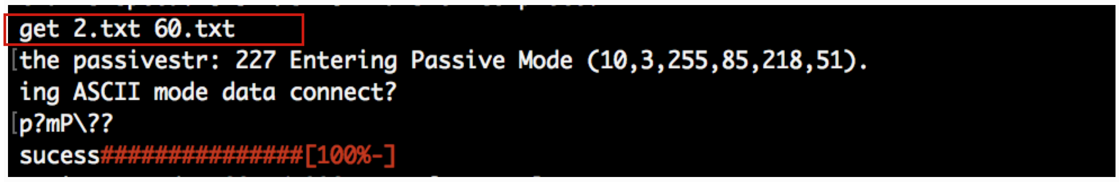
(9). Input ”put localFileName remoteFileName” can put a local file to server and then the related file is stored with the newname


```
ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,208,186).
-rwxr-xr-x    1 0        0            2040 Jun 15 11:55 2.txt
-rwxr-xr-x    1 0        0          252862 Jun 15 12:01 5.pdf
150 Here comes the directory listing.
??226 Directory send OK.
put 4.pdf 6.pdf
6.pdf
local:4.pdfthe passivestr: 227 Entering Passive Mode (10,3,255,85,138,238).
x??ucess150 Ok to send data.
226 Transfer complete.
ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,225,181).
-rwxr-xr-x    1 0        0            2040 Jun 15 11:55 2.txt
-rwxr-xr-x    1 0        0          252862 Jun 15 12:01 5.pdf
-rwxr-xr-x    1 0        0          252862 Jun 15 12:20 6.pdf
150 Here comes the directory listing.
??226 Directory send OK.
```

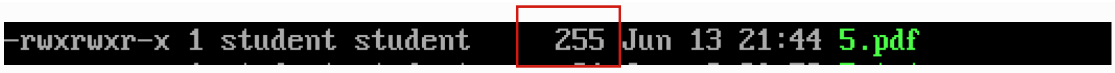
(10). Input "get fileName" can get a existing file from the server


```
ls
the passivestr: 227 Entering Passive Mode (10,3,255,85,187,149).
-rwxr-xr-x    1 0        0            2040 Jun 15 11:55 2.txt
150 Here comes the directory listing.
L??226 Directory send OK.
get 2.txt
buffer:RETR 2.txt
213 2040
p?the passivestr: 227 Entering Passive Mode (10,3,255,85,128,176).
sucess###############[100%-]
```

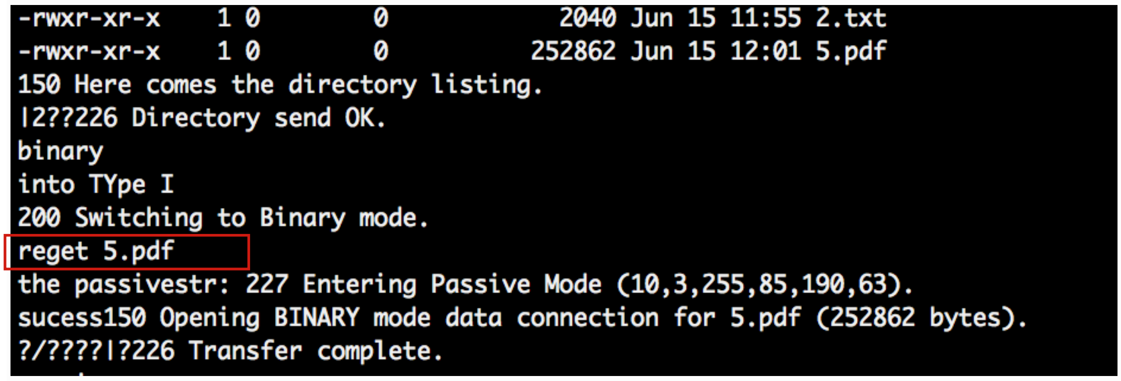
(11). Input "get fileName remoteName " can get a file from the server and store the file with newName in client.


```
get 2.txt 60.txt
the passivestr: 227 Entering Passive Mode (10,3,255,85,218,51).
ing ASCII mode data connect?
p?mP\??
sucess###############[100%-]
```

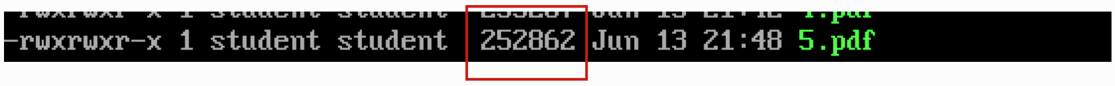
(12). If the process was terminate accidently, you download the file form break point by entering "binary" then "reget filename".



```
-rwxrwxr-x 1 student student     255 Jun 13 21:44 5.pdf
```

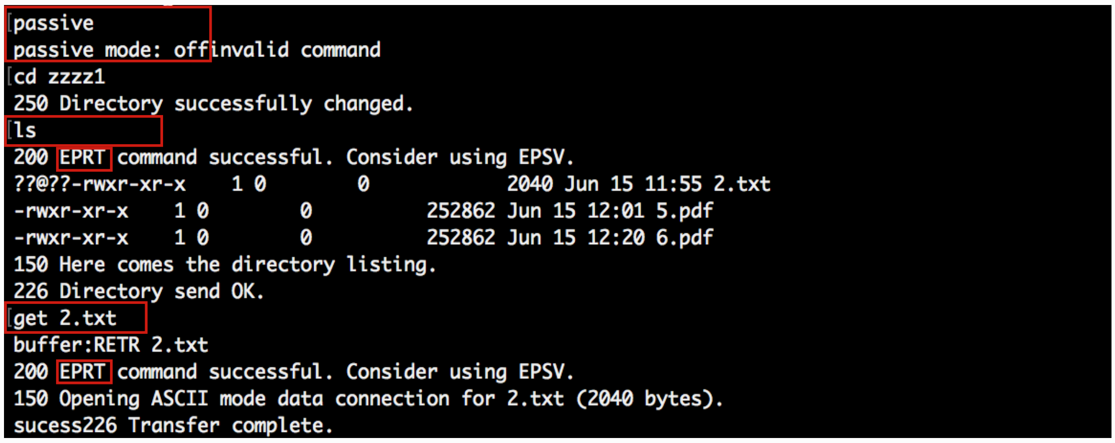


```
-rwxr-xr-x    1 0        0            2040 Jun 15 11:55 2.txt
-rwxr-xr-x    1 0        0          252862 Jun 15 12:01 5.pdf
150 Here comes the directory listing.
|2??226 Directory send OK.
binary
into TYpe I
200 Switching to Binary mode.
reget 5.pdf
the passivestr: 227 Entering Passive Mode (10,3,255,85,190,63).
sucess150 Opening BINARY mode data connection for 5.pdf (252862 bytes).
?/????|?226 Transfer complete.
```



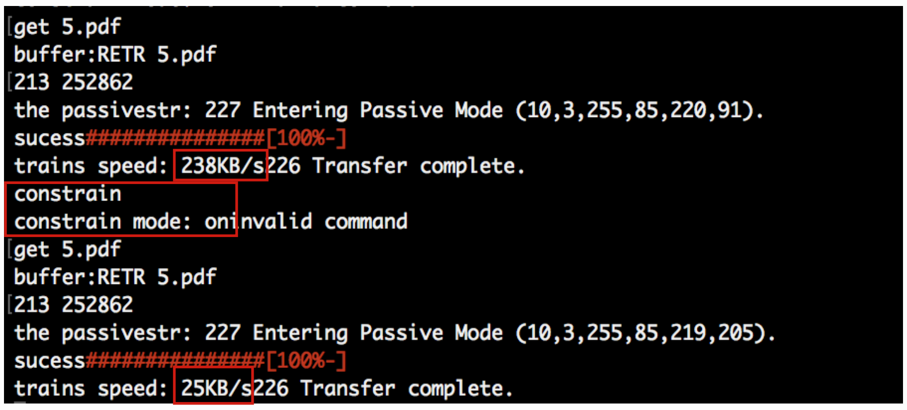
```
-rwxrwxr-x 1 student student  252862 Jun 13 21:48 5.pdf
```

(13).Default passive mode, input "passive" , then the passive mode is off, into active mode, input "passive" again, passive mode is on.



```
passive
passive mode: offinvalid command
cd zzzz1
250 Directory successfully changed.
ls
200 EPRT command successful. Consider using EPSV.
??@??-rwxr-xr-x    1 0        0            2040 Jun 15 11:55 2.txt
-rwxr-xr-x    1 0        0          252862 Jun 15 12:01 5.pdf
-rwxr-xr-x    1 0        0          252862 Jun 15 12:20 6.pdf
150 Here comes the directory listing.
226 Directory send OK.
get 2.txt
buffer:RETR 2.txt
200 EPRT command successful. Consider using EPSV.
150 Opening ASCII mode data connection for 2.txt (2040 bytes).
sucess226 Transfer complete.
```

(14).Default constrain off ,input "constrain" then transfer to "constrain on", input constrain again, return to constrain off mode.



```
get 5.pdf
buffer:RETR 5.pdf
213 252862
the passivestr: 227 Entering Passive Mode (10,3,255,85,220,91).
sucess##############[100%-]
trains speed: 238KB/s226 Transfer complete.
constrain
constrain mode: oninvalid command
get 5.pdf
buffer:RETR 5.pdf
213 252862
the passivestr: 227 Entering Passive Mode (10,3,255,85,219,205).
sucess##############[100%-]
trains speed: 25KB/s226 Transfer complete.
```

6. SummaryandConclusion

FENGYE:

Firstly, I was responsible for implementing the reading of standard input and establish commend connection and data connection for both passive mode and active mode. After that, my partner implement some functions which do not need to establish the data connection and break point download based on it, and as for me, I completed the upload and download function(both passive mode and active mode) and transmit rate control. Meanwhile, I also built get [local] [remote]/put [local][remote] function which can change the file name while upload and download, and rename function(which is quit similar to the previous function).

By doing this project, I have a more precise understanding of network programming and how th FTP protocal and proxy works. More specifically, I've acknowledged that the ftp control command is a practice of ASCII string but add stucture to the message body, which is very similar to HTTP header. And how a protcal works, we regulate the message format and representation, take FTP as an example, RFC958 regulate which part to represent the command and which part to represent the arguments and the mapping of command to function to be implementing(nc is great tool of learning it). Then, both the server and client implement their function based on this standard. Frankly, ftp message format is much easier than DNS and DHCP ,but now with this comprehensive understanding of principle, I think we can handle it as well. And we all agree C language is a basic programming language, unlike java or python, we have to understand how different operating systems deal with it, and how to allocate the memory. Even the same function in standard library performs inconsistent in different operation systems, such as select(), and clock(). While it is much complicated than programming in Java, but C show me some way to handling underlying content, such as ntohs and htons is a way to dealing with inconsistant of byte order of host and network(big-endian and little-endian), which we do not have to concern in Java.

Zhang Ludan:

Finish the functions: cd, pwd, mkdir, delete, ascii, binary, reget and progress bar. Actually, at first I was unfamiliar with FTP protocol, so I learned some related knowledge and then made some code practice about it. At the same time, I use Wireshark to catch the package about FTP protocol to learn the principles of all the commands.After that, I finish the basic functions (including cd, pwd, mkdir, delete, ascii). My partner and I divide the extension requirements into 2 group. I finish the reget function which resumes transmission from break-point. This function is implemented in the binary mode, so at first I should change into binary mode and then reget the file. Finally, I add the progress bar for the get function.

Self-evaluation:

Through this project, I learn more about network programing and FTP protoal. Through catch packages using WireShark, I learned all the principles of different commands and functions. Understand the port forwarding and socket connection. At the same time, I review the knowledge of

C lanfuage. Especially learn some new mothods. In fact, I can improve the reget function since it is written under the binary mode. Then I can improve the progress bar to be more pretty.

Some flaws to be refined: Addmittedly, we were not doing a great job on error tolerants, such as when the client's directory have existed a file with exact same file name, I should notice the client there is a same named file in this directory, measure to cover it? then should I send the RETR command. Another flaw is that the code is a little redundant, many code can be actually encapsulated as a function and reuse. But if we had more time to prepare it, we can make the code more efficient and readable.

Appendix: Source Codes