




2013-6-10

# Design and Implementation of FTP Client

—— Coursework report of *Internet Application*

Chen Chao (10213053)

Jin Lina (10213071)



## Content

1	Overview .....	3
1.1	Requirements.....	3
1.2	Targets.....	3
2	Requirements Analysis.....	3
2.1	Development Environment.....	3
2.2	Functional Requirements.....	3
2.2.1	Basic Functions.....	3
2.2.2	Advanced Functions.....	4
3	Preliminary Design .....	4
3.1	Decomposition of functional modules .....	4
3.2	Relationship and interface between the modules.....	5
3.3	Overall flow chart .....	6
3.4	Design of data structures .....	6
3.4.1	Struct hostent .....	6
3.4.2	Struct sockaddr.....	7
4	Detailed Design.....	7
4.1	Design analysis of functions.....	7
4.1.1	Establish connection.....	7
4.1.2	Resolve port and IP address .....	8
4.1.3	Detailed TCP connection.....	9
4.1.4	Highlight name of files .....	11
4.1.5	Limit speed .....	13
4.1.6	Show transferring progress.....	16
4.1.7	Receive and display file list.....	18
4.1.8	Get file from server.....	19
4.1.9	Put file to the server.....	20
4.2	Design analysis of FTP command.....	21
4.2.1	Login .....	21
4.2.2	pwd.....	21
4.2.3	ls .....	22
4.2.4	cd.....	22
4.2.5	get .....	22
4.2.6	put.....	23
4.2.7	mkdir .....	24
4.2.8	delete.....	24
4.2.9	rename.....	25
4.2.10	quit .....	25
5	Results .....	26
5.1	User login: .....	26
5.2	Command—help:.....	27
5.3	Command—ls: .....	27
5.4	Command—ls -a:.....	28
5.5	Command—ls -t <type>:.....	28

5.6	Command—pwd:.....	29
5.7	Command—delete <file>: .....	29
5.8	Command—mkdir <directory>:.....	29
5.9	Command—cd <directory>: .....	30
5.10	Command—rename <file> <new name>: .....	30
5.11	Command—mode: .....	31
5.12	Command—get: .....	31
5.13	Command—put:.....	33
5.14	Command—ul <speed>: .....	34
5.15	Command—dl <speed>: .....	34
5.16	Command—type:.....	35
5.17	Command—quit:.....	35
6	Summary and Conclusion .....	36
6.1	Self-evaluation .....	36
6.2	Improvement of design .....	37
7	Appendix: Source Codes .....	38

# 1 Overview

## 1.1 Requirements

- 1) Understanding FTP protocol
- 2) Analyzing network packets using Wireshark
- 3) Mastering Socket programming in Linux

## 1.2 Targets

- 1) Implementing a FTP client program
- 2) Providing a command line interface
- 3) Translating user commands into FTP commands
- 4) Accessing any FTP server

# 2 Requirements Analysis

## 2.1 Development Environment

- 1) C language and associated libraries
- 2) gcc compiler, gdb debugger, Wireshark packet capture analysis tools
- 3) Linux operating system, vsftpd server, the ftp command and its help under Linux

## 2.2 Functional Requirements

### 2.2.1 Basic Functions

- 1) Able to connect to a FTP server with domain name or IP address
- 2) Supporting Login, able to hide the input password
- 3) Using passive mode
- 4) Supporting following commands (Shown in the sheet)
- 5) Providing statistics as transfer speed and total traffic
- 6) Error handling

COMMAND NAME	DESCRIPTION
<b>pwd</b>	Print the full filename of the current working directory on the remote machine.
<b>ls</b>	Print a listing of the contents of a directory on the remote machine. The listing includes any system-dependent information that the servers choose to include.
<b>get</b>	Retrieve the remote-file and store it on the local machine. If the local file name is not specified, it is given the same name it has on the same machine.
<b>put</b>	Store a local file on the remote machine.

<b>delete</b>	Delete the file remote-file on the remote machine.
<b>rename</b>	Renames the filenames supplied according to the rule specified as the first argument.
<b>cd</b>	Change the working directory on the remote machine to remote-directory.
<b>mkdir</b>	Create directories, if they don't already exist, on the remote machine.

### 2.2.2 Advanced Functions

- 1) Help menu
- 2) Supporting active mode
- 3) Able to transfer binary files and ascii files
- 4) Able to limiting speed of data transmission
- 5) Showing transferring progress
- 6) User-friendly interface, displaying files with different colors based on file type
- 7) Supporting to resume transfer after the progress is broken

## 3 Preliminary Design

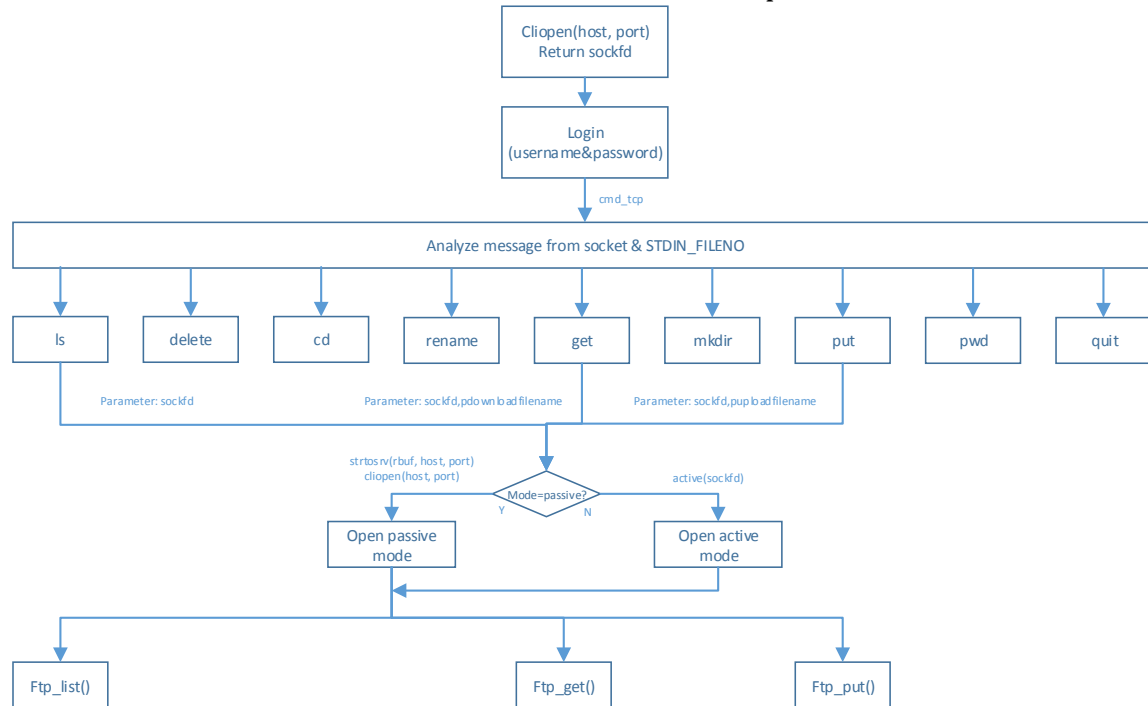
### 3.1 Decomposition of functional modules

Function	Function Role
<b>int cliopen(char *host, char *port)</b>	Establish a TCP connection from client to server.
<b>void strtosrv(char *str, char *host, char *port)</b>	Compute server's port by a pair of integers; Get server's IP address.
<b>void cmd_tcp(int sockfd)</b>	Read and write as command connection.
<b>void ftp_list(int sockfd, int index)</b>	Read and write as data transfer connection.
<b>int ftp_get(int sck, char *pDownloadFileName_s)</b>	Download file from ftp server.
<b>int ftp_put (int sck, char *pUploadFileName_s)</b>	Upload file to ftp server.
<b>int typeget(char *name)</b>	Get the type of files.
<b>void printClr(char *record, int a)</b>	Display files with different colors based on the file type.
<b>void printPanel()</b>	To display user-friendly interface while log in.
<b>void printHint()</b>	Hints of colors of different type of files.
<b>void printTransfer(int percent, double speed, int count);</b>	Print the progress bar.

<b>int timeDiff(struct timeval* tv1, struct timeval* tv2);</b>	Calculate time difference between two received packets.
<b>void goSleep(int status, double speed, int size);</b>	Hang the progress and let it sleep for some time.
<b>int active(int sockfd);</b>	Open port for active mode.

### 3.2 Relationship and interface between the modules

The boxes are modules. Arrows describe the relationship between the modules.



#### Main modules detailed description:

**Cliopen(host, port) module** is used to set up a new TCP connection between client and server. If successful, the ID of the socket will be returned as “sockfd”.

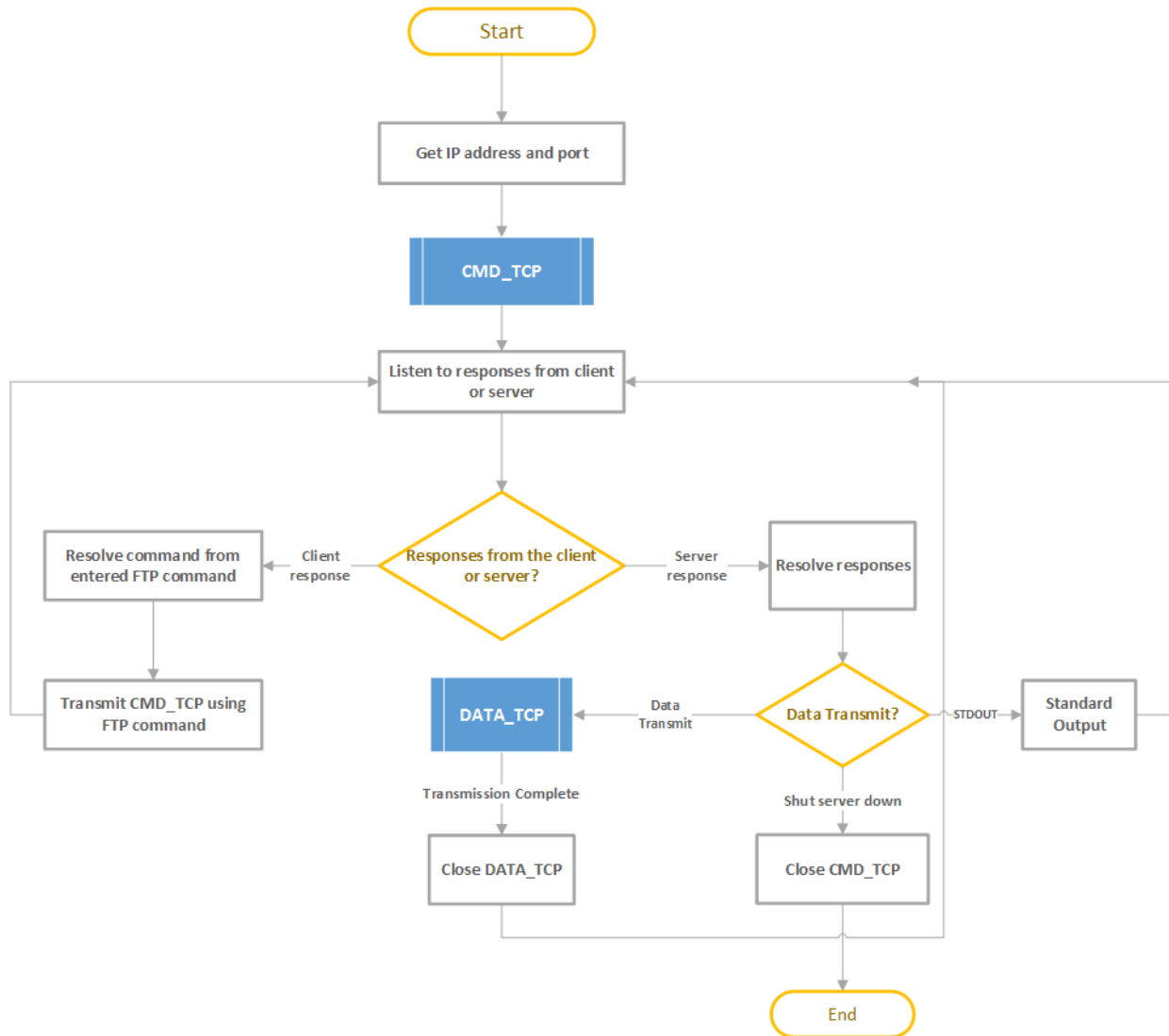
**Cmd\_tcp module** is the top module of all, which covers all commands. It reads the sockfd, and listen to the specific socket as well as STDIN. Any actions on these two IO will cause relevant operation in relevant modules.

**Strtosrv(str, host, port) module** is used to decode the string into a host address and a port number. It’s a preamble module for passive connection.

**Active(sockfd) module** is used to set up a new socket and preparing for active mode connection. Details are introduced in this report.

**ftp\_list module, ftp\_get module, and ftp\_put module** perform functions of commands ls, get and put respectively.

### 3.3 Overall flow chart



### 3.4 Design of data structures

#### 3.4.1 Struct hostent

```

/* Description of data base entry for a single host. */
struct hostent {
    char *h_name;           /* official name of host. */
    char **h_aliases;       /* Alias list. */
    int h_addrtype;        /* Host address type. */
    int h_length;          /* Length of address */
    char **h_addr_list;    /* List of addresses form name server. */
    #define h_addr h_addr_list[0] /*The 1st address in the address list. */
};
  
```

### 3.4.2 Struct sockaddr

```

struct sockaddr{
unsigned short    sin_family;           /* AF_INET*/
unsigned short port  sin_port;         /* 16 bit port number */
struct  in_addr   sin_addr;           /* Internet address*/
char            sin_zero[8];          /* unused*/
};

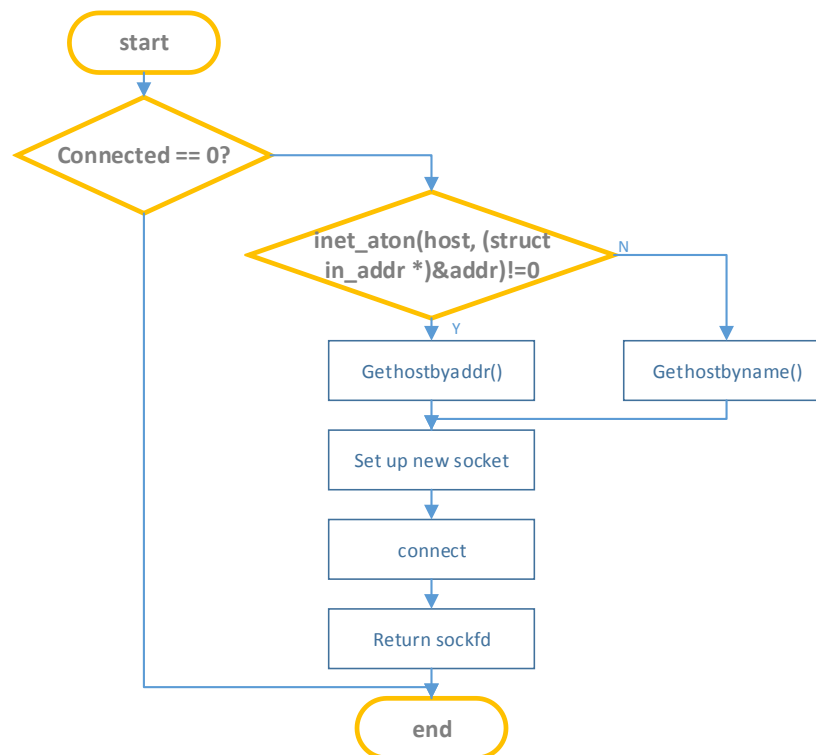
```

## 4 Detailed Design

### 4.1 Design analysis of functions

#### 4.1.1 Establish connection

Function	<i>int cliopen(char *host, char *port)</i>
<b>Performance</b>	Establish a TCP connection from client to server.
<b>Design Analysis</b>	<ol style="list-style-type: none"> <li>1) Determine the input is IP address or domain name;</li> <li>2) Get host entity from DNS server according to the input;</li> <li>3) Copy information from host entity into sockaddr_in object;</li> <li>4) Set up a new socket;</li> <li>5) Connect the socket to the server;</li> <li>6) Return the ID of the socket.</li> </ol>

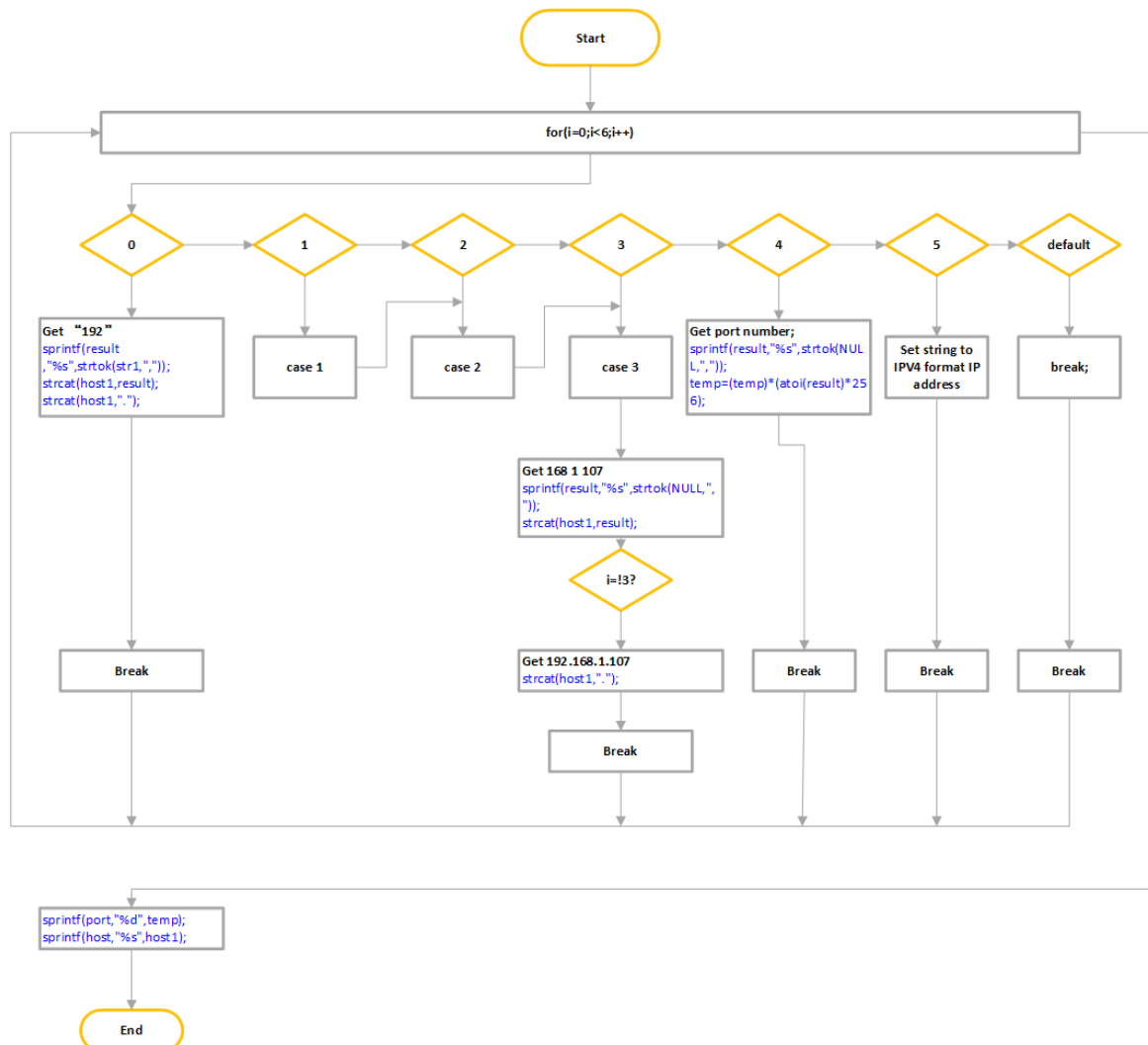




### 4.1.2 Resolve port and IP address

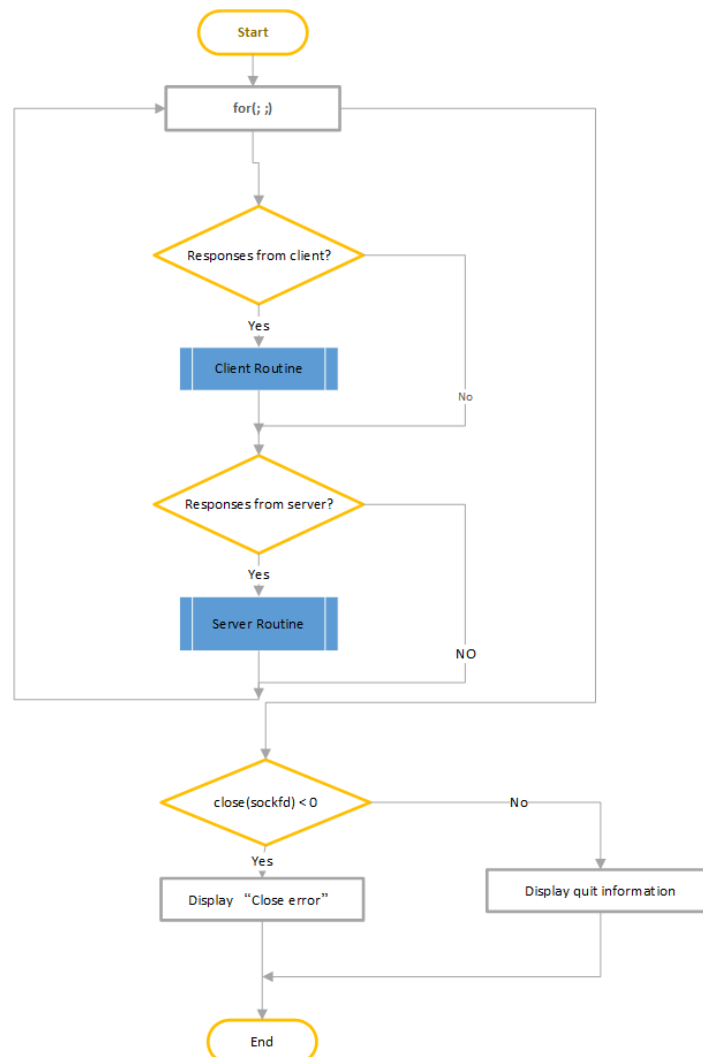
Function	<i>void strtosrv(char *str, char *host, char *port)</i>
Performance	Resolve server's port by a pair of integers; Get server's IP address.
Design Analysis	1) Compute server's port by a pair of integers and store it in char *port; 2) Split the string into several parts according to symbol '(', ',' and ')' in the string. 3) Take out 6 numbers from the string, e.g. (192.168.1.107.178.184). The former 4 numbers make up an IP address; the latter 2 numbers make up the port number by a specific operation. (Port No. = 5 <sup>th</sup> num * 256 + 6 <sup>th</sup> num) 4) Return IP address and port number in string type.

In the following flow chart, (192,168,1,107,178,184) is taken for example.

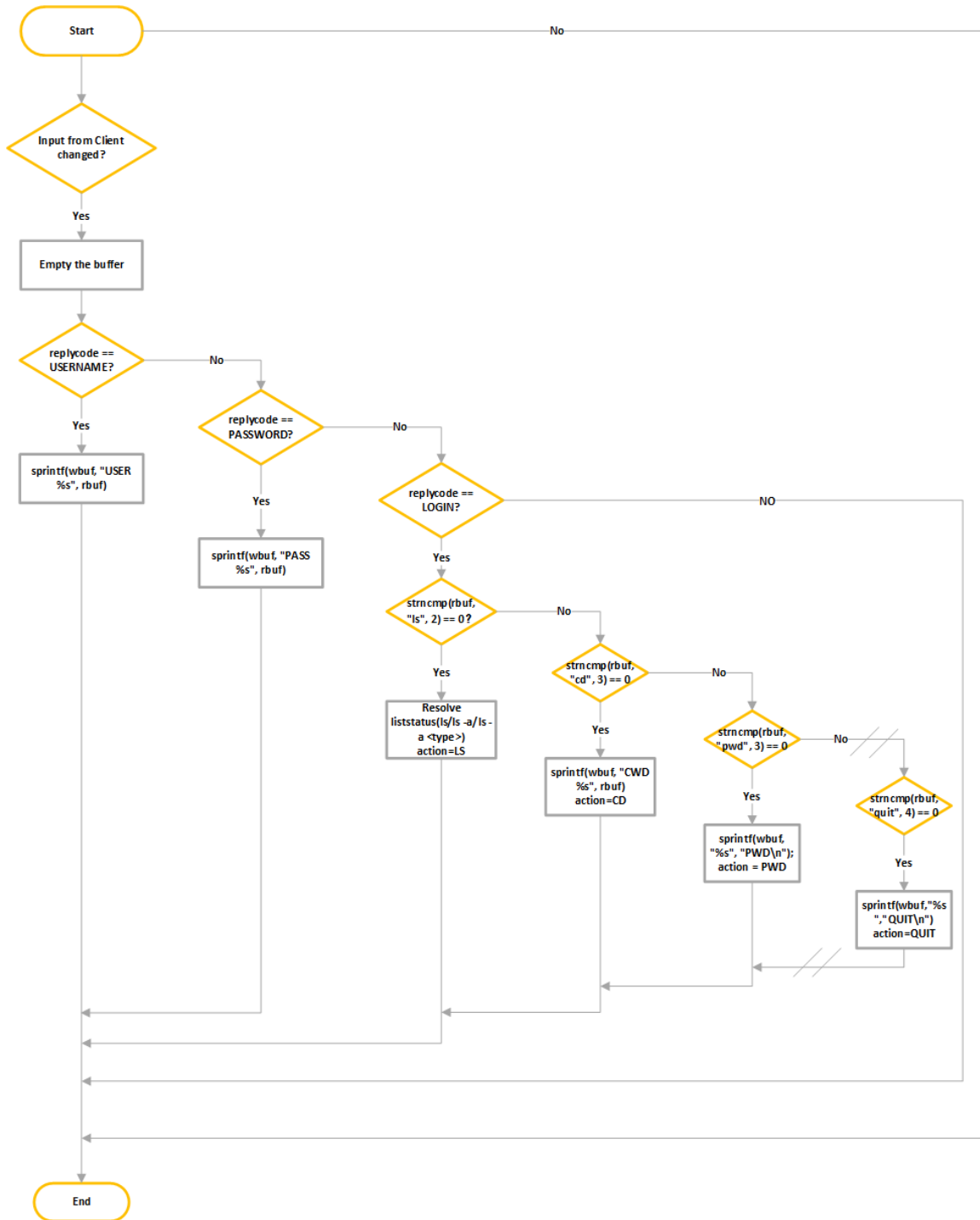


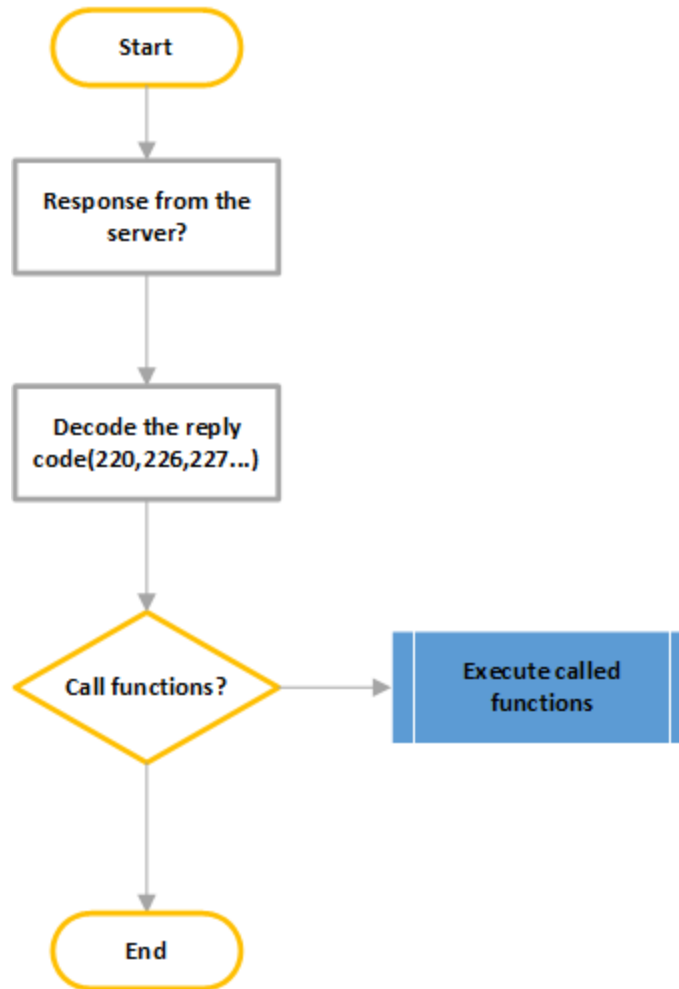
## 4.1.3 Detailed TCP connection

Function	<i>void cmd_tcp (int sockfd)</i>
<b>Performance</b>	Read and write as command connection.
<b>Design Analysis</b>	<ol style="list-style-type: none"> <li>1) Use select() to listen to actions on the socket and keyboard;</li> <li>2) A for loop is used to perform action endlessly;</li> <li>3) Use two IF judgment to decide which of socket message and keyboard input is changed.</li> <li>4) If STDIN_FILENO is invoked, read the command and perform corresponding background actions according to the input;</li> <li>5) If new message arrives in the socket, take out the first three characters of the message to see which type the message is, and do relevant operation;</li> <li>6) If QUIT, break the loop.</li> </ol>

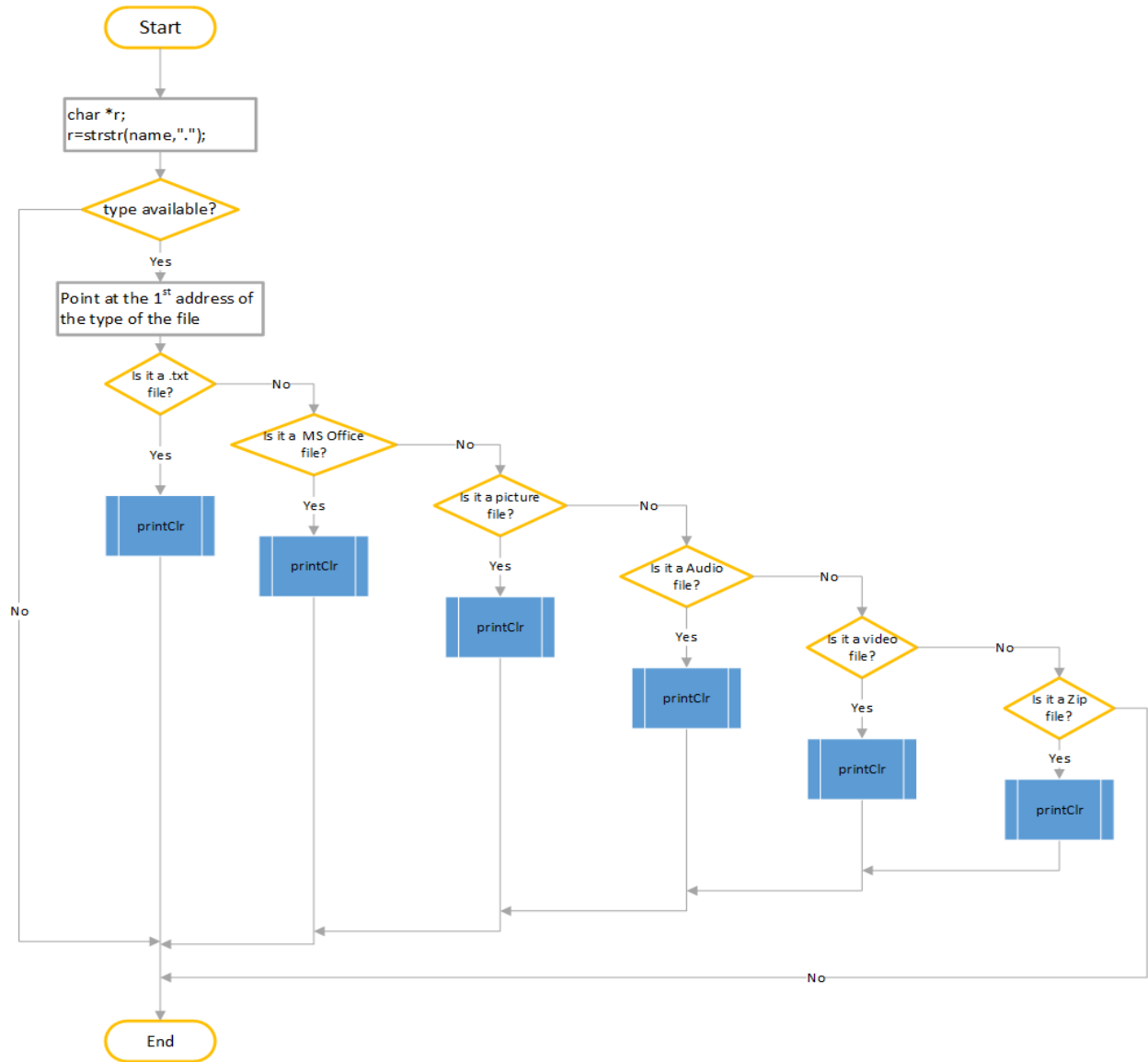


### Client Subroutine:

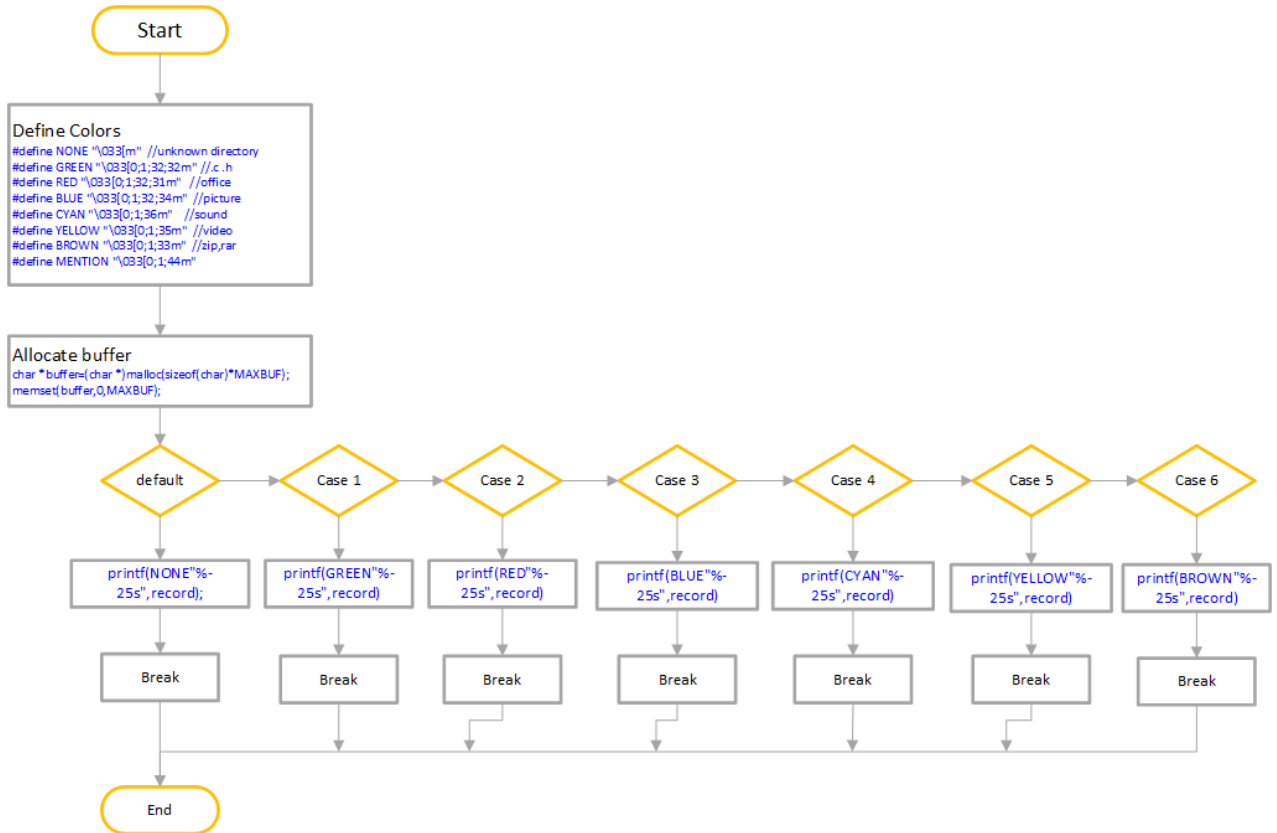


**Server Subroutine:****4.1.4 Highlight name of files**

Function	<i>void printClr(char *record,int a); int typeget(char *name);</i>
<b>Performance</b>	Display files with different colors based on the file type.
<b>Design Analysis</b>	1) Call typeget() method, take out the part of input string on the right of ',', which indicates the type of file; 2) Compare the substring to some specific type indicator such as "txt", "pdf", and return an index to indicate the type; 3) Call printClr() method, which takes in the string to print and the type indicator; 4) Use a switch structure to print the string in a specific color corresponding to the type indicator.

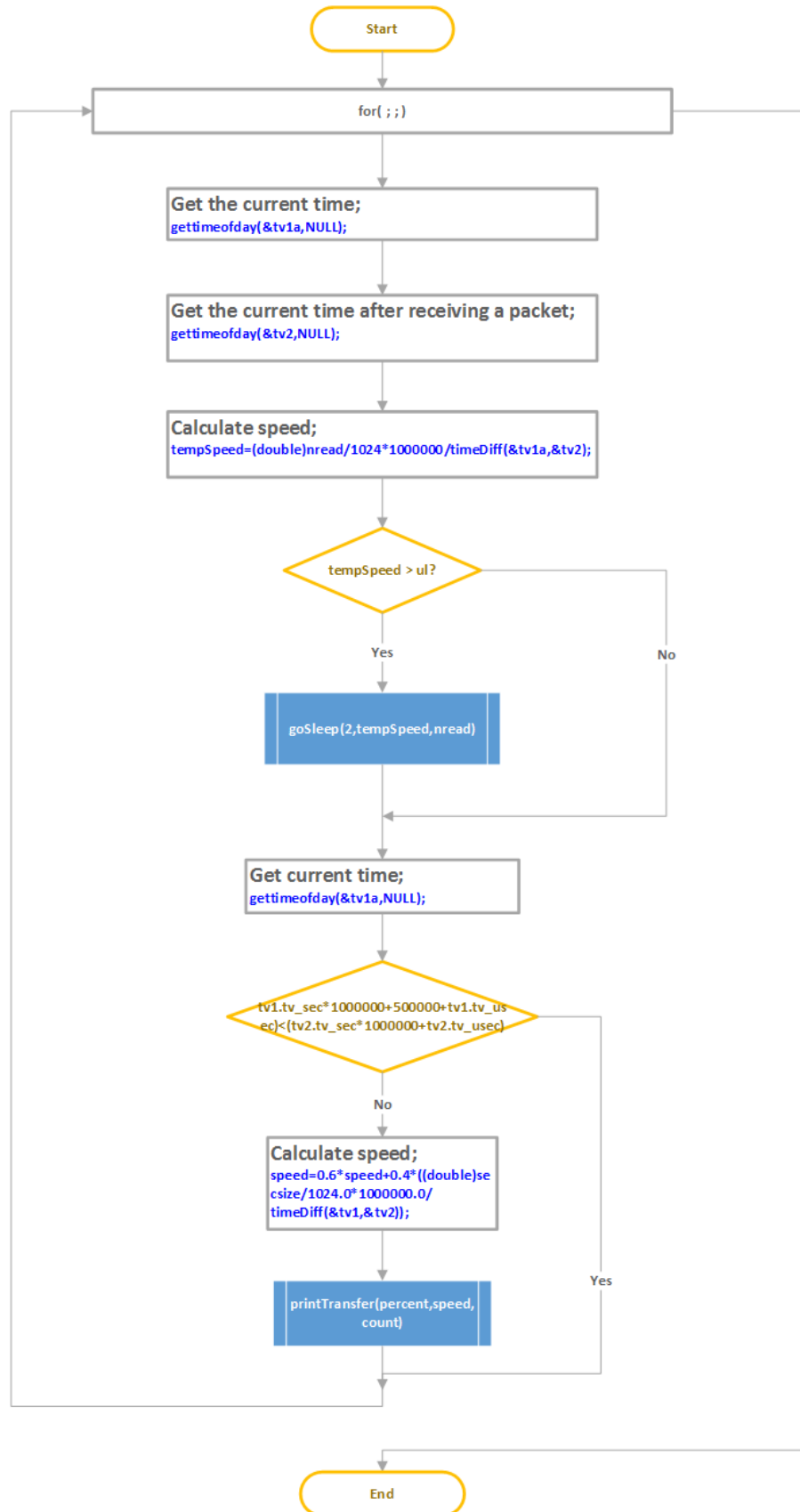


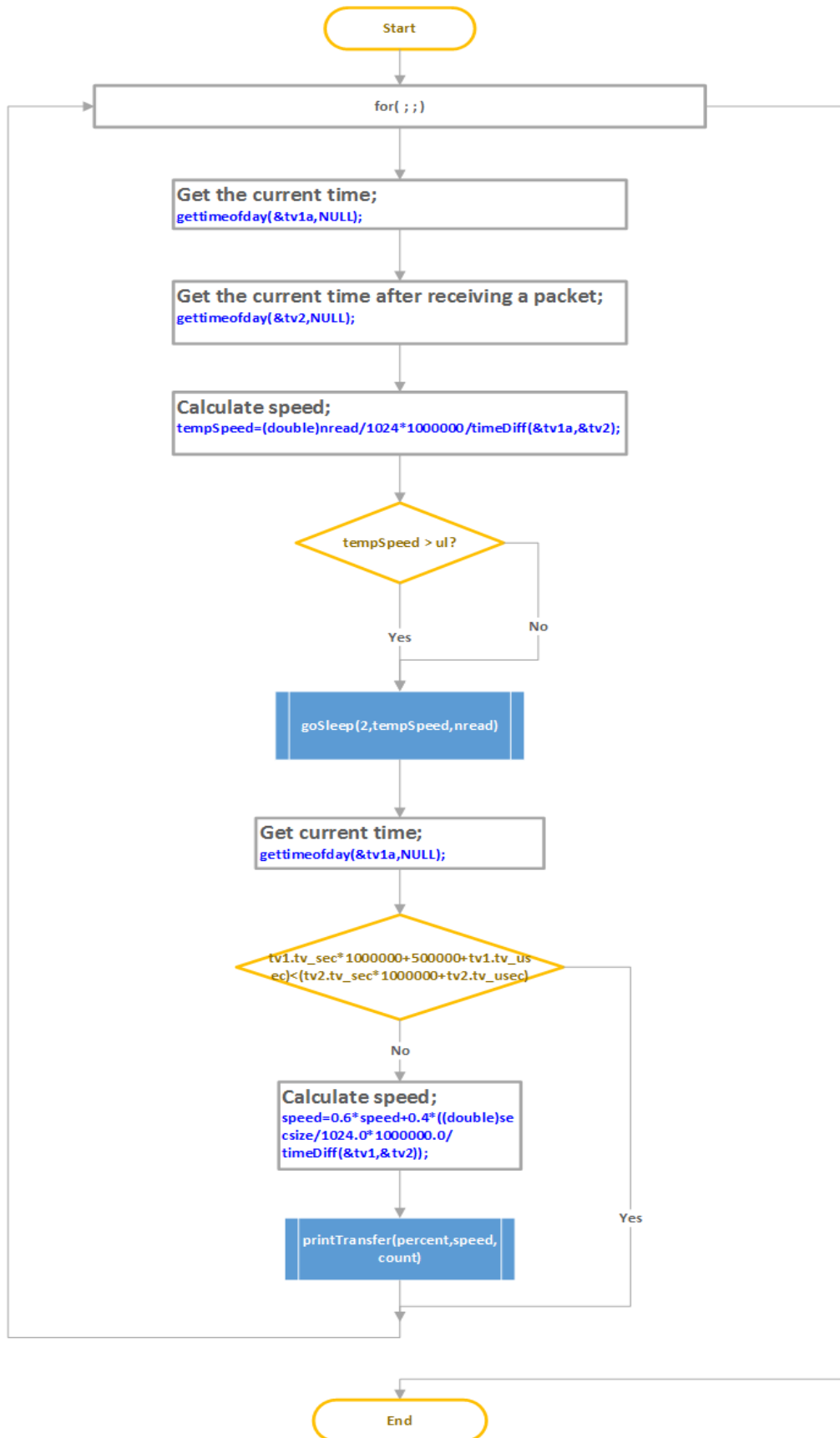
The specific flow chart of function `printClr()` is showed as following.



#### 4.1.5 Limit speed

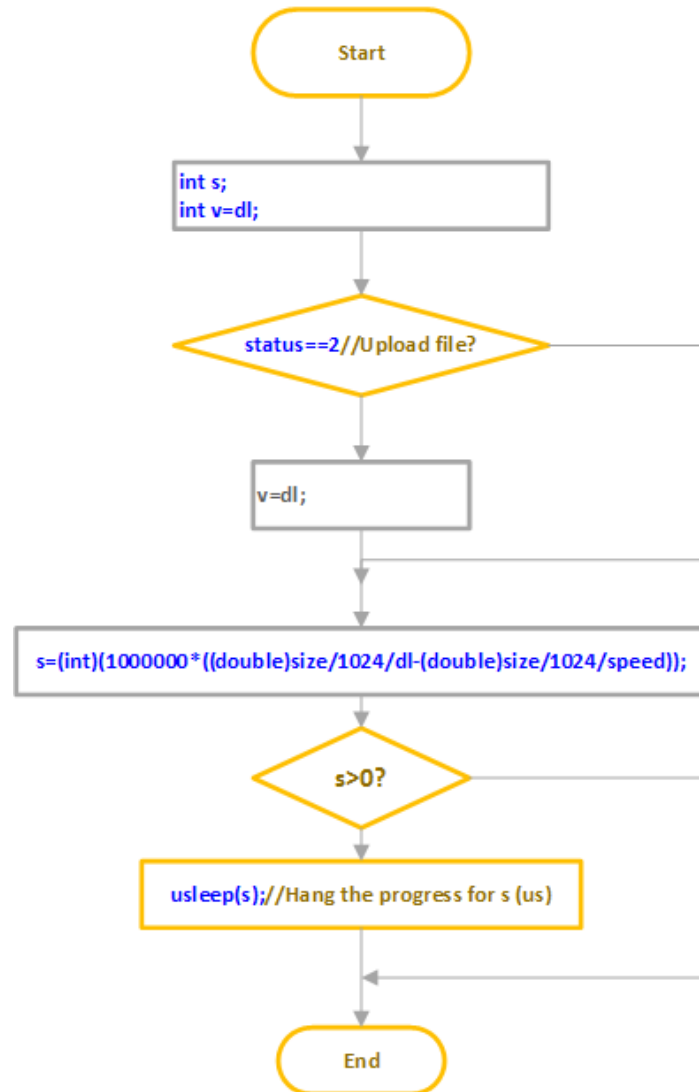
Function	<i>int ftp_get(int sck, char *pDownloadFileName_s);</i> <i>int ftp_put (int sck, char *pUploadFileName_s);</i> <i>void goSleep(int status, double speed,int size);</i>
<b>Performance</b>	Limit the speed of data transfer as it set by the user.
<b>Design Analysis</b>	1) get two time stamps before and after each read-write operation; 2) Calculate the time duration of read-write action, and calculate transport speed of each package by dividing the size of package by the duration. 3) If the speed is higher than the limitation, call goSleep() method to compensate a duration of time with no read-write action to slower the average speed. 4) Refresh and print out the speed every one second.





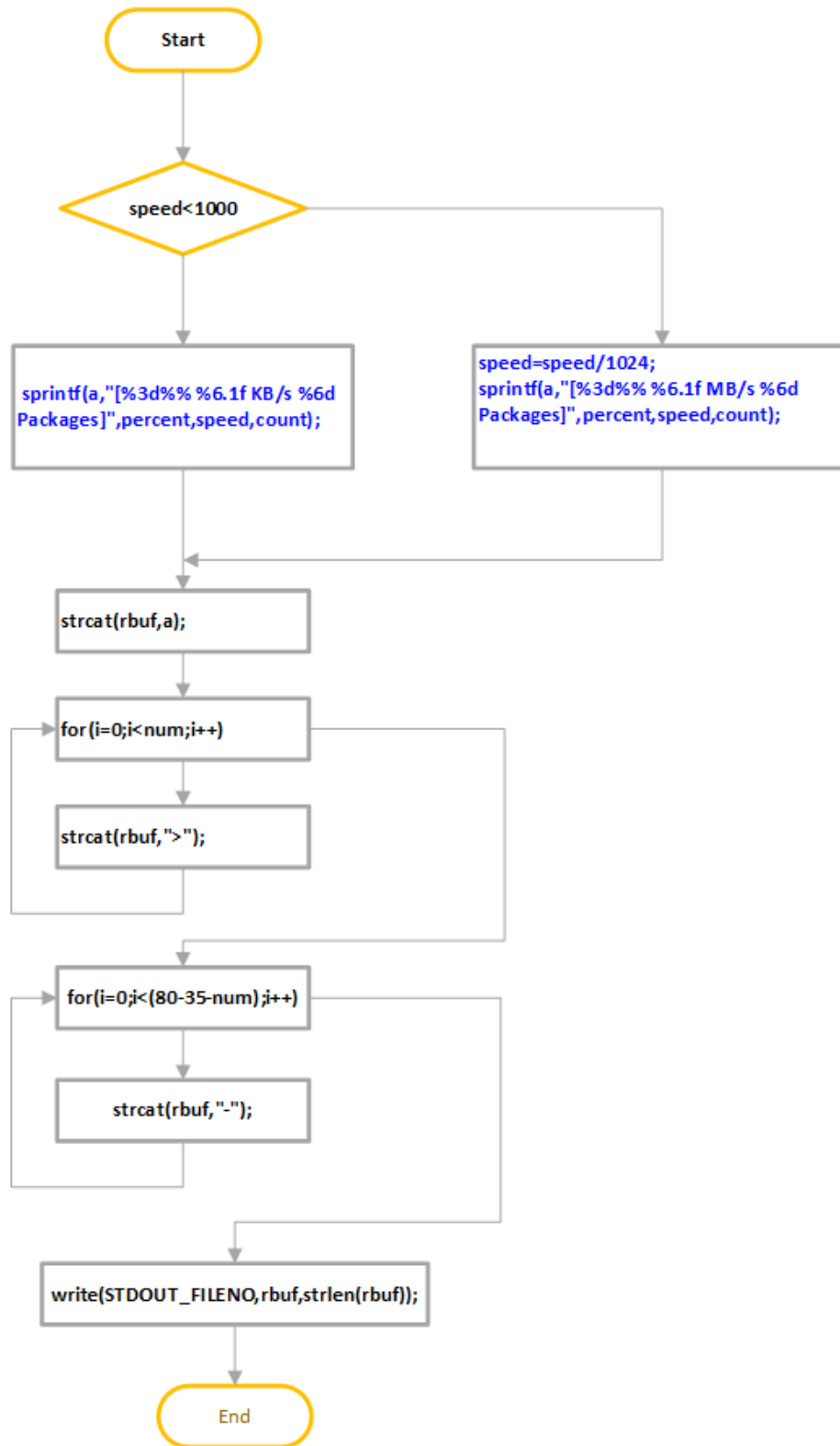


Detailed design of void `goSleep()`:



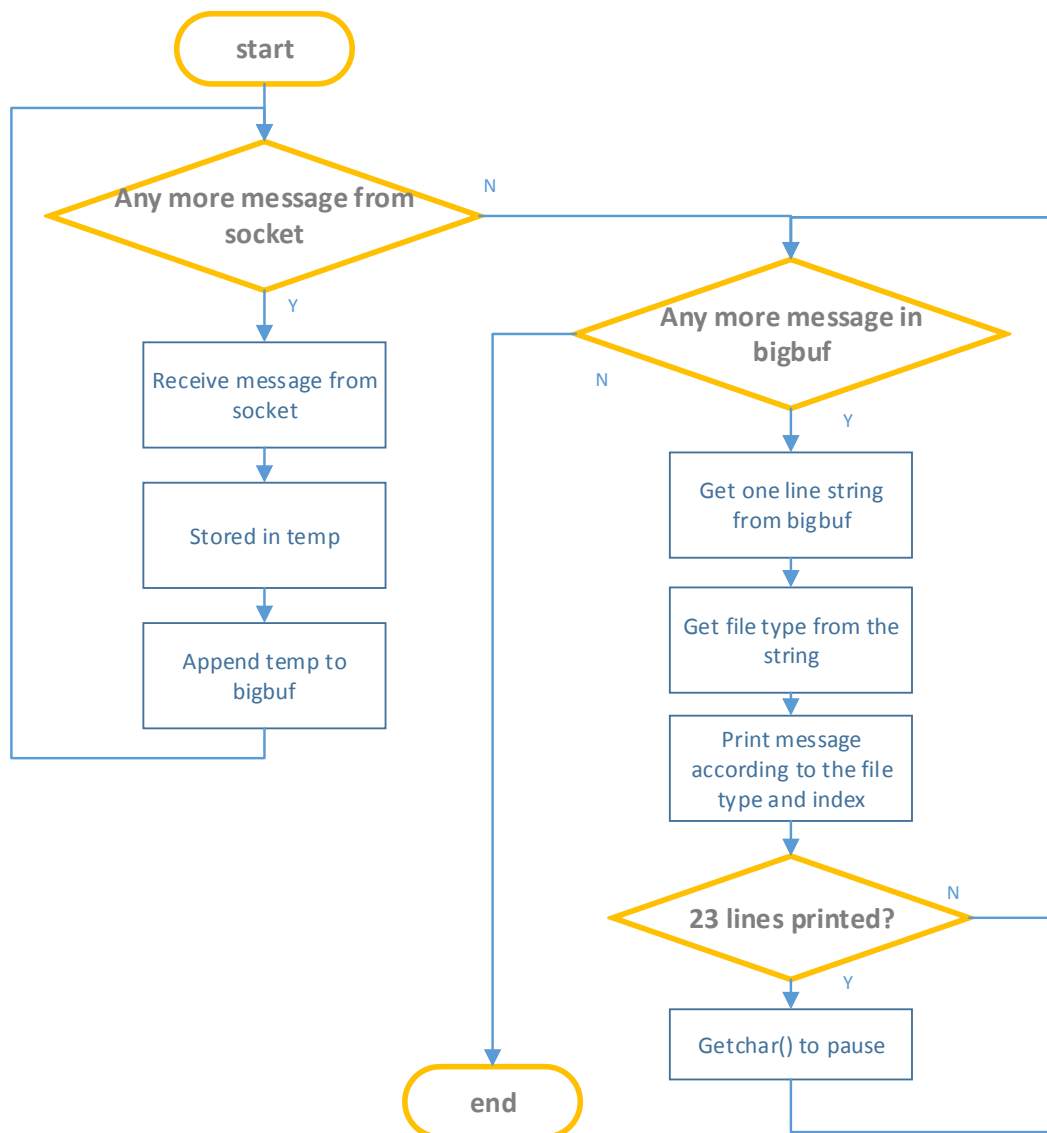
#### 4.1.6 Show transferring progress

Function	<code>void printTransfer(int percent,double speed,int count);</code>
Performance	Print the progress bar.
Design Analysis	1) Print a '\r' character to swipe the previous display and then print a new string to construct a refreshing display; 2) Use memset() method to clean up the buffer; 3) Append information to the buffer and at last print it out. 4) If the speed is higher than 1000kB/s, divide the speed by 1024 to change in unit of Mb/s.



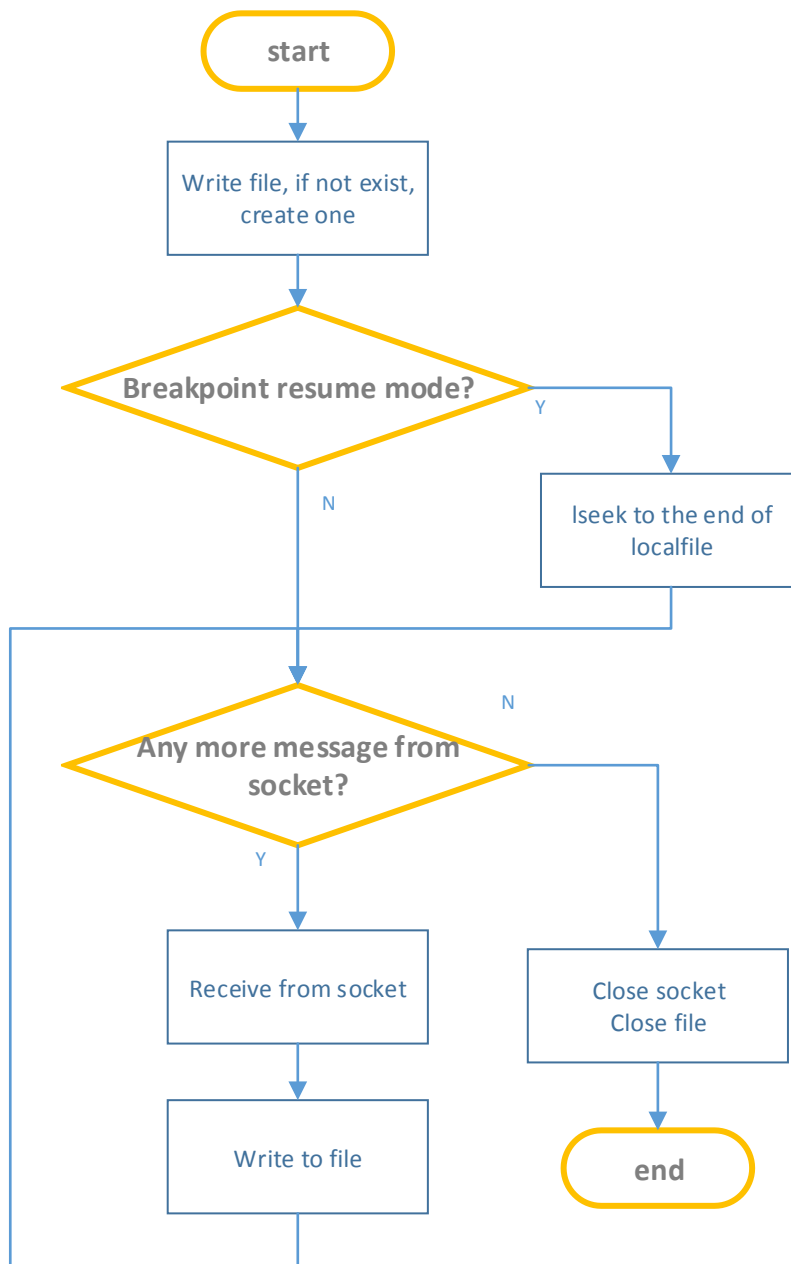
## 4.1.7 Receive and display file list

Function	<i>void ftp_list(int sockfd, int index)</i>
<b>Performance</b>	Receive package from socket and display file list.
<b>Design Analysis</b>	<ol style="list-style-type: none"> <li>1) Use a loop to receive all packages from socket and store in a big buffer.</li> <li>2) Read line by line from big buffer, and get file type of each file.</li> <li>3) Print information of file according to file type and the index,</li> <li>4) When 23 lines of messages printed, use <code>getchar()</code> to wait for continue.</li> </ol>



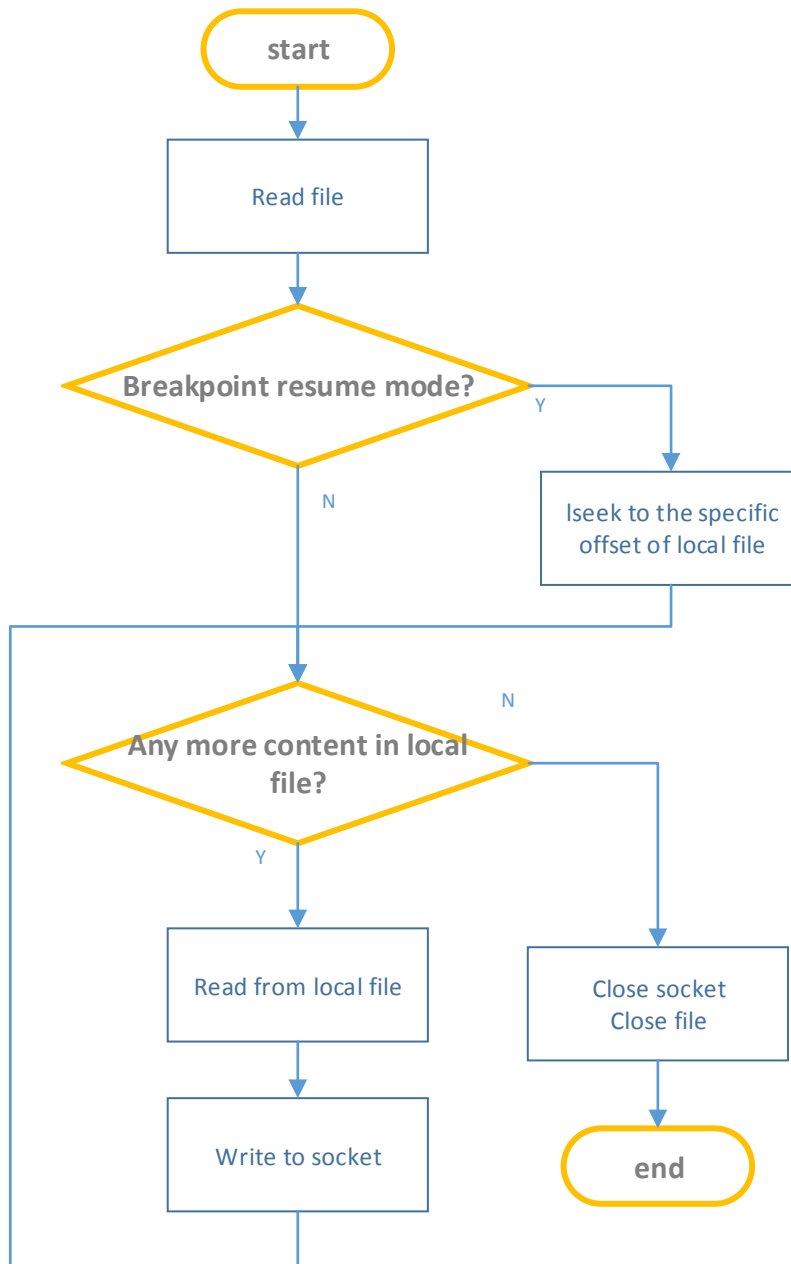
## 4.1.8 Get file from server

<b>Function</b>	<i>int ftp_get(int sck, char *pDownloadFileName_s)</i>
<b>Performance</b>	Receive file from the server and store in local client.
<b>Design Analysis</b>	1) Use a loop to receive message from socket and write in file 2) If it's in breakpoint resume mode, allocate to the end of local file before writing.



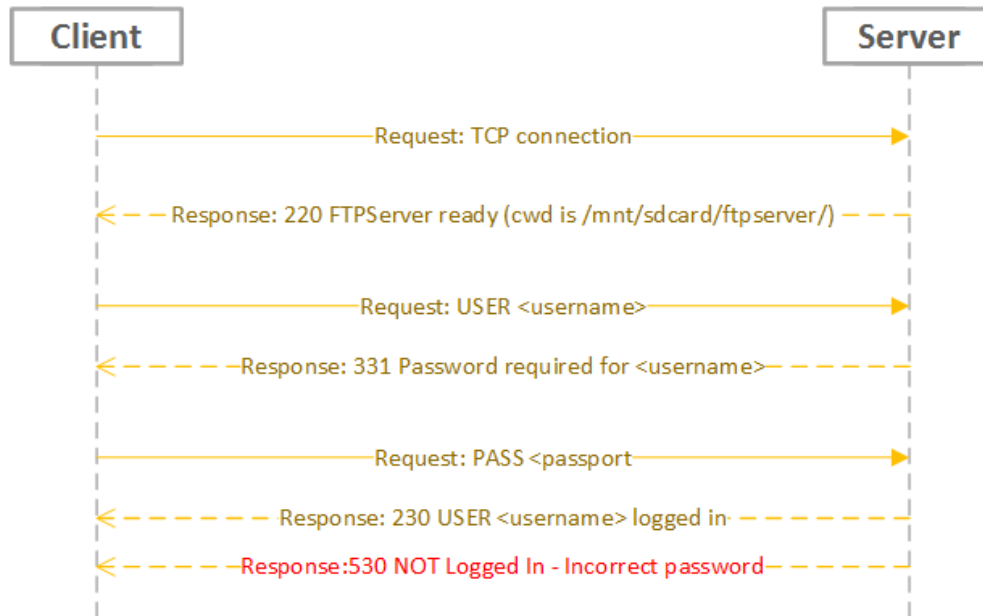
## 4.1.9 Put file to the server

<b>Function</b>	<i>int ftp_put(int sck, char *pUploadFileName_s)</i>
<b>Performance</b>	Put file to the server.
<b>Design Analysis</b>	1) Use a loop to read from local file and write to socket. 2) If it's in breakpoint resume mode, allocate to specific offset in local file.



## 4.2 Design analysis of FTP command

### 4.2.1 Login



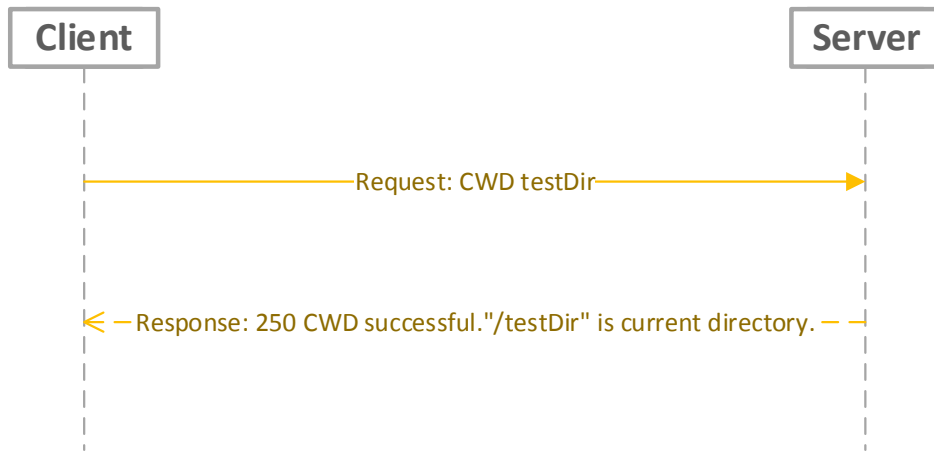
### 4.2.2 pwd



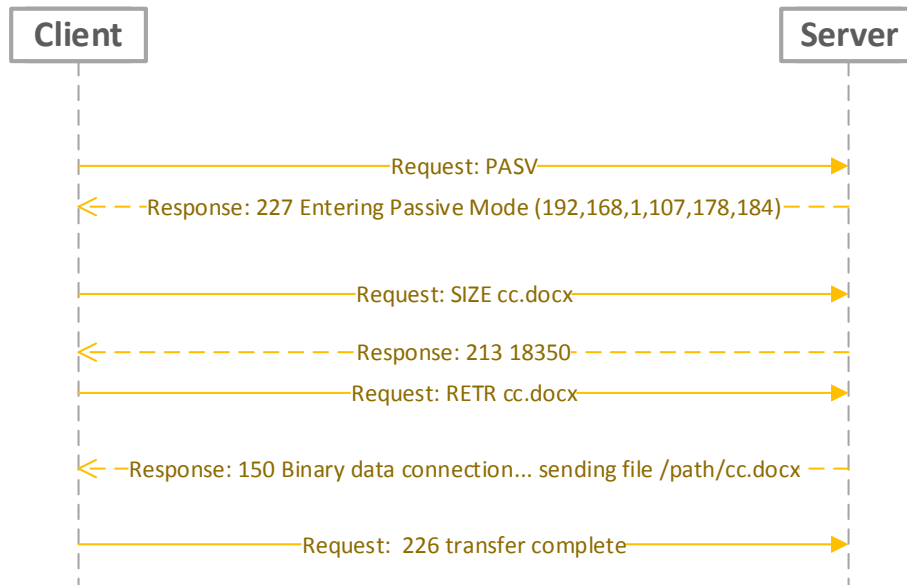
#### 4.2.3 ls



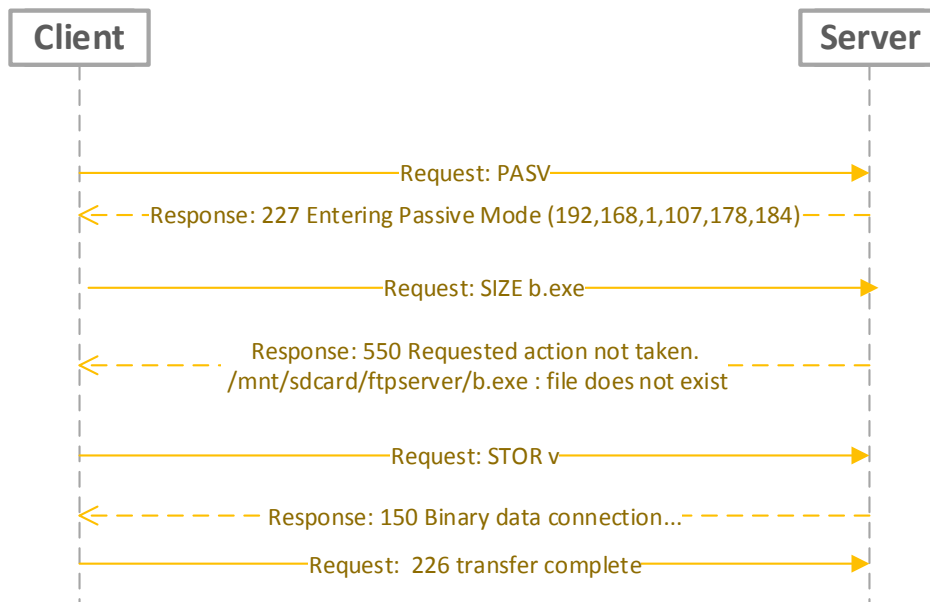
#### 4.2.4 cd



#### 4.2.5 get



#### 4.2.6 put

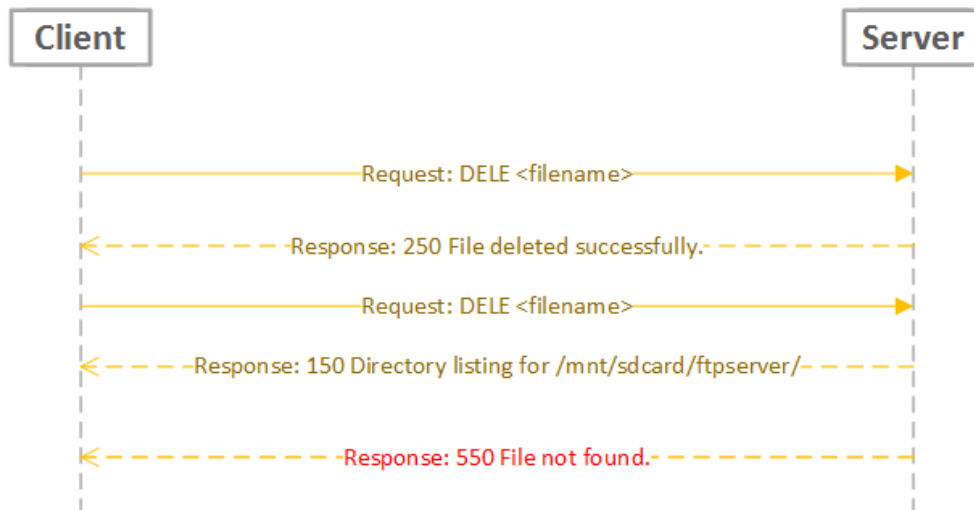




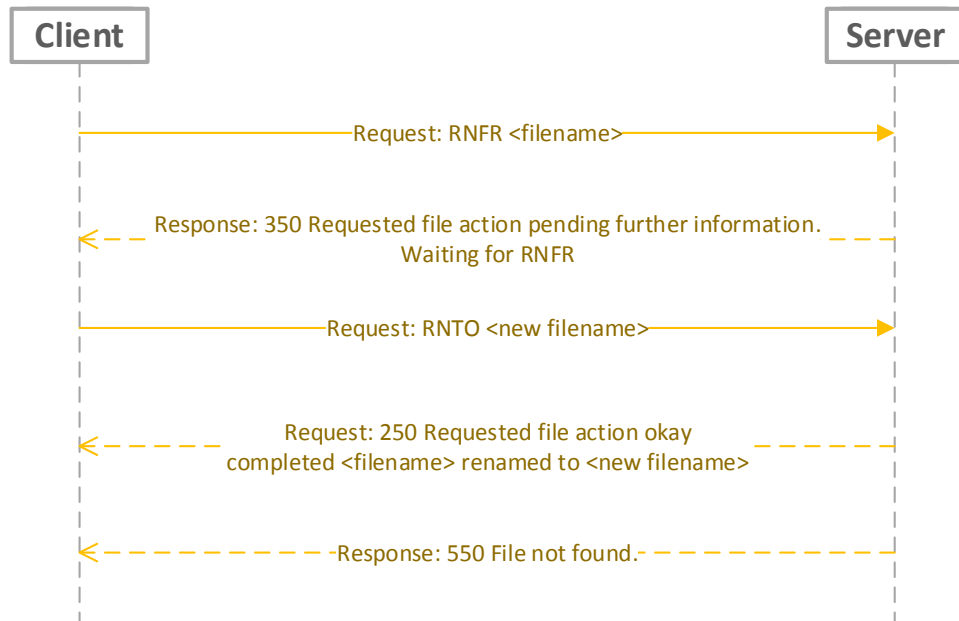
#### 4.2.7 mkdir



#### 4.2.8 delete



#### 4.2.9 rename



#### 4.2.10 quit



## 5 Results

### 5.1 User login:

```

nina [Running]

#####
#####
##          ##          ##
##          ##          ##
#####
##          ##          ##
##          ##          ##
#####
#####
Authors: Chao Chen 10213053 2010215110
         Lina Jin  10213071 2010215110

(Client)

*****
Welcome! Please log in.                                     ftp.mayan.cn
*****
Username: test
your password:

```

```

*****Login Successful!*****
ftp-HiaHia>

```

#### Error detection:

A warning comes out when accept wrong username or passport.

```

*****
Welcome! Please log in.                                     ftp.mayan.cn
*****
Username: test
your password:

*****Login failed. Incorrect username or password.*****
Username:

```

## 5.2 Command—help:

Enter “help” to get specific description of each command. The “Help Menu” showed below briefly illustrate all the functions designed for the FTP client.

```
ftp-HiaHia>help
*****Help Menu*****
cd <directory>      --- Move to a specific directory.
pwd                --- Return the name of current directory.
delete <file>       --- Delete a file.
rename <file> <new name> --- Rename a file.
mkdir <directory>   --- Set up a new directory under current directory.
ls                 --- View file list in current directory.
ls -a              --- In detail view file list in current directory.
ls -t <type>        --- Highlight file of type <type> in a list
get <file>          --- Download a file from the server.
put <file>          --- Upload a file onto the server.
mode               --- View current connection mode.
mode <mode>         --- Change connection mode(1-passive;0-active)
type               --- View current data type.
type <mode>         --- Change data type(a-ASCII;i-BINARY)
ul                 --- View the upload speed limitation.
ul <speed>          --- Limit upload speed under a specific value.
dl                 --- View the download speed limitation.
dl <speed>          --- Limit download speed under a specific value.
help               --- View help information.
quit               --- Log out and exit.
*****
ftp-HiaHia>_
```

## 5.3 Command—ls:

```
Connecting in Passive mode...
227 Entering Passive Mode (114,255,40,147,234,70)
150 Connection accepted
*****
(Image file)-(Text file)-(Audio file)-(Office file)-(Video file)-(Zip file)
*****
1.txtip          111.rar          12345
111.zip          11111111         2
123456          2                2.ppt
2012.txt         22.txt           29.png
7.mp3            888.zip           a
m.pdf           a.pdfa.pdf       a.txt
aa              aa.gz            aaa
aaaa           b.pdf            c.rmub
c.txt           chen1            chen2
dn.jpg          dsdd             fuck.zip
gao111.png      gaohuan          gest
gre.pdf         grp11            hehe.mp3
hehe.mp3        ja.txt           ll
myfile          new111           p
piecebear       sample           sample.pdf
sun.pdf         test             Test.java
test1           test_dir         wbl
ww.ww           yuan.png         zef
zh1x.rar        zhu              zhx
zz.pdf          zzz              zzzz
■ ■ .mp3

226 Transfer OK
ftp-HiaHia>
```

## 5.4 Command—ls -a:

```

nina [Running]
227 Entering Passive Mode (114,255,40,147,234,72)
150 Connection accepted
*****
(Image file)-(Text file)-(Audio file)-(Office file)-(Video file)-(Zip file)
*****
-rw-r--r-- 1 ftp ftp          31 Jun 07 14:23 1.txtip
-rw-r--r-- 1 ftp ftp    1366884 Jun 07 15:38 111.rar
-rw-r--r-- 1 ftp ftp    2861360 Jun 07 14:18 111.zip
-rw-r--r-- 1 ftp ftp           0 Jun 07 13:28 11111111
drwxr-xr-x 1 ftp ftp           0 Jun 07 13:53 12345
drwxr-xr-x 1 ftp ftp           0 Jun 07 13:49 123456
drwxr-xr-x 1 ftp ftp           0 Jun 07 13:21 2
-rw-r--r-- 1 ftp ftp    1126446 Jun 07 13:18 2.ppt
-rw-r--r-- 1 ftp ftp     10533 Jun 07 14:58 2012.txt
-rw-r--r-- 1 ftp ftp      1024 Jun 07 13:17 22.txt
-rw-r--r-- 1 ftp ftp      1410 Jun 07 14:35 29.png
-rw-r--r-- 1 ftp ftp           0 Jun 07 13:34 7.mp3
-rw-r--r-- 1 ftp ftp     32556 Jun 07 13:47 888.zip
drwxr-xr-x 1 ftp ftp           0 Jun 07 14:42 a
-rw-r--r-- 1 ftp ftp    6532170 Jun 07 14:02 a m.pdf
-rw-r--r-- 1 ftp ftp           0 Jun 07 14:10 a.pdfa.pdf
-rw-r--r-- 1 ftp ftp           0 Jun 07 14:23 a.txt
drwxr-xr-x 1 ftp ftp           0 Jun 07 14:59 aa
-rw-r--r-- 1 ftp ftp    13807945 Jun 07 15:08 aa.gz
-rw-r--r-- 1 ftp ftp       1024 Jun 07 15:07 aaa
drwxr-xr-x 1 ftp ftp           0 Jun 07 14:00 aaaa
-rw-r--r-- 1 ftp ftp    493024 Jun 07 14:14 b.pdf
-rw-r--r-- 1 ftp ftp    1126596 Jun 07 14:24 c.rmub
Press Enter to continue.
-

```

## 5.5 Command—ls -t <type>:

Highlight file of type <type> in a list.

```

drwxr-xr-x 1 ftp ftp           0 Jun 07 13:53 12345
drwxr-xr-x 1 ftp ftp           0 Jun 07 13:49 123456
drwxr-xr-x 1 ftp ftp           0 Jun 07 13:21 2
-rw-r--r-- 1 ftp ftp    1126446 Jun 07 13:18 2.ppt
-rw-r--r-- 1 ftp ftp     10533 Jun 07 14:58 2012.txt
-rw-r--r-- 1 ftp ftp      1024 Jun 07 13:17 22.txt
-rw-r--r-- 1 ftp ftp      1410 Jun 07 14:35 29.png
-rw-r--r-- 1 ftp ftp           0 Jun 07 13:34 7.mp3
-rw-r--r-- 1 ftp ftp     32556 Jun 07 13:47 888.zip
drwxr-xr-x 1 ftp ftp           0 Jun 07 14:42 a
-rw-r--r-- 1 ftp ftp    6532170 Jun 07 14:02 a m.pdf
-rw-r--r-- 1 ftp ftp           0 Jun 07 14:10 a.pdfa.pdf
-rw-r--r-- 1 ftp ftp           0 Jun 07 14:23 a.txt
drwxr-xr-x 1 ftp ftp           0 Jun 07 14:59 aa
-rw-r--r-- 1 ftp ftp    13807945 Jun 07 15:08 aa.gz
-rw-r--r-- 1 ftp ftp       1024 Jun 07 15:07 aaa
drwxr-xr-x 1 ftp ftp           0 Jun 07 14:00 aaaa
-rw-r--r-- 1 ftp ftp           0 Jun 07 15:41 aaad
-rw-r--r-- 1 ftp ftp    493024 Jun 07 14:14 b.pdf
Press Enter to continue.

```

## 5.6 Command—**pwd**:

```
*****Login Successful!*****
ftp-HiaHia>pwd
257 "/" is current directory.
ftp-HiaHia>_
```

## 5.7 Command—**delete <file>**:

```
ftp-HiaHia>delete 111.rar
250 File deleted successfully
ftp-HiaHia>
```

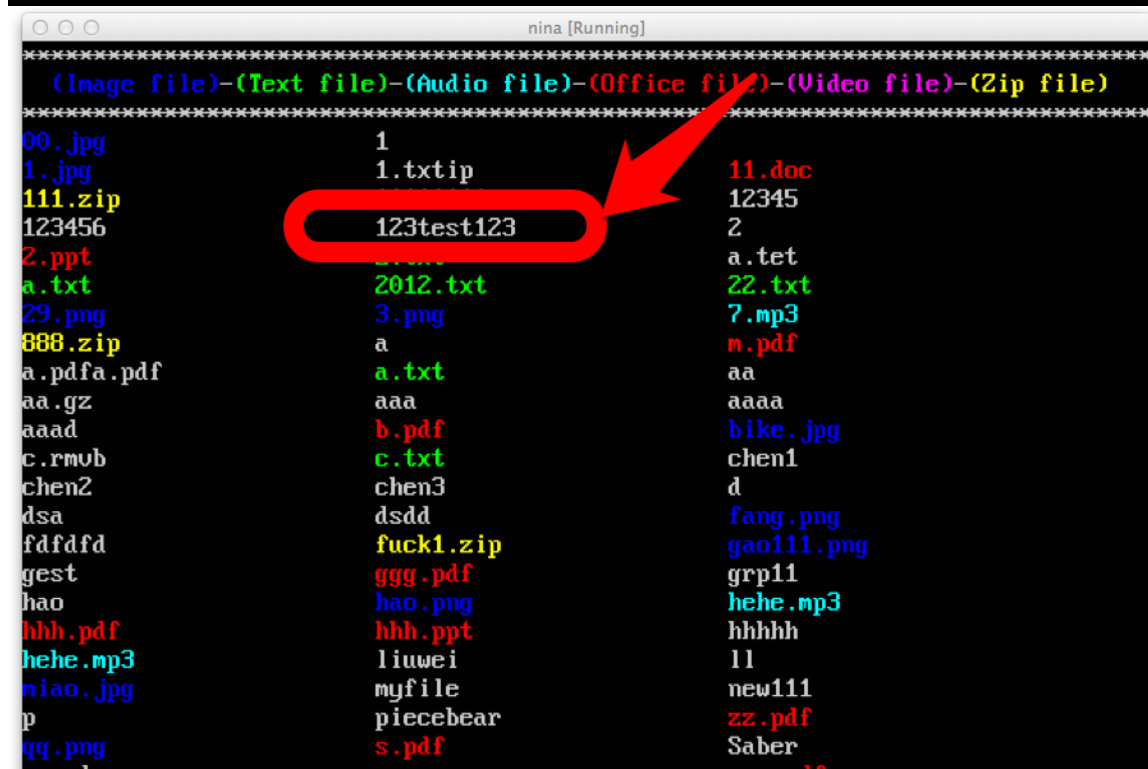
### Error detection:

A fail reply is printed while delete a file that doesn't exist.

```
ftp-HiaHia>delete nosuchfile.c
Failed --- File not found. Please check the file name.
ftp-HiaHia>
```

## 5.8 Command—**mkdir <directory>**:

```
ftp-HiaHia>mkdir 123test123
257 "/123test123" created successfully
ftp-HiaHia>
```



**Error detection:**

While the client tries to make a directory that has already existed, the result comes out as the following shot showed.

```
ftp-HiaHia>mkdir 123test123
Failed --- Directory name already exists. Please choose another.
ftp-HiaHia>
```

**5.9 Command—`cd <directory>`:**

```
*****Login Successful!*****
ftp-HiaHia>cd
Usage: cd <dir_name>
ftp-HiaHia>cd 123456
250 CWD successful. "/123456" is current directory.
ftp-HiaHia>pwd
257 "/123456" is current directory.
ftp-HiaHia>
```

**5.10 Command—`rename <file> <new name>`:**

```
ftp-HiaHia>rename 11.doc 11rename.doc
350 File exists, ready for destination name.
250 file renamed successfully
ftp-HiaHia>_
```

```
*****
(Image file)-(Text file)-(Audio file)-(Office file)-(Video file)-(Zip file)
*****
00.jpg          1
1.jpg           1.txtip
111.zip         11111111
12345           123456
2              2.ppt
a.tet           a.txt
22.txt          2012.txt
7.mp3           3.png
m.pdf           a
aa              a.txt
aaaa            aaa
b.pdf           abcd
                c.rmvb
                1-2
```

**Renamed file.** (A yellow arrow points from this text to the file **11rename.doc** in the list.)

**Error detection:**

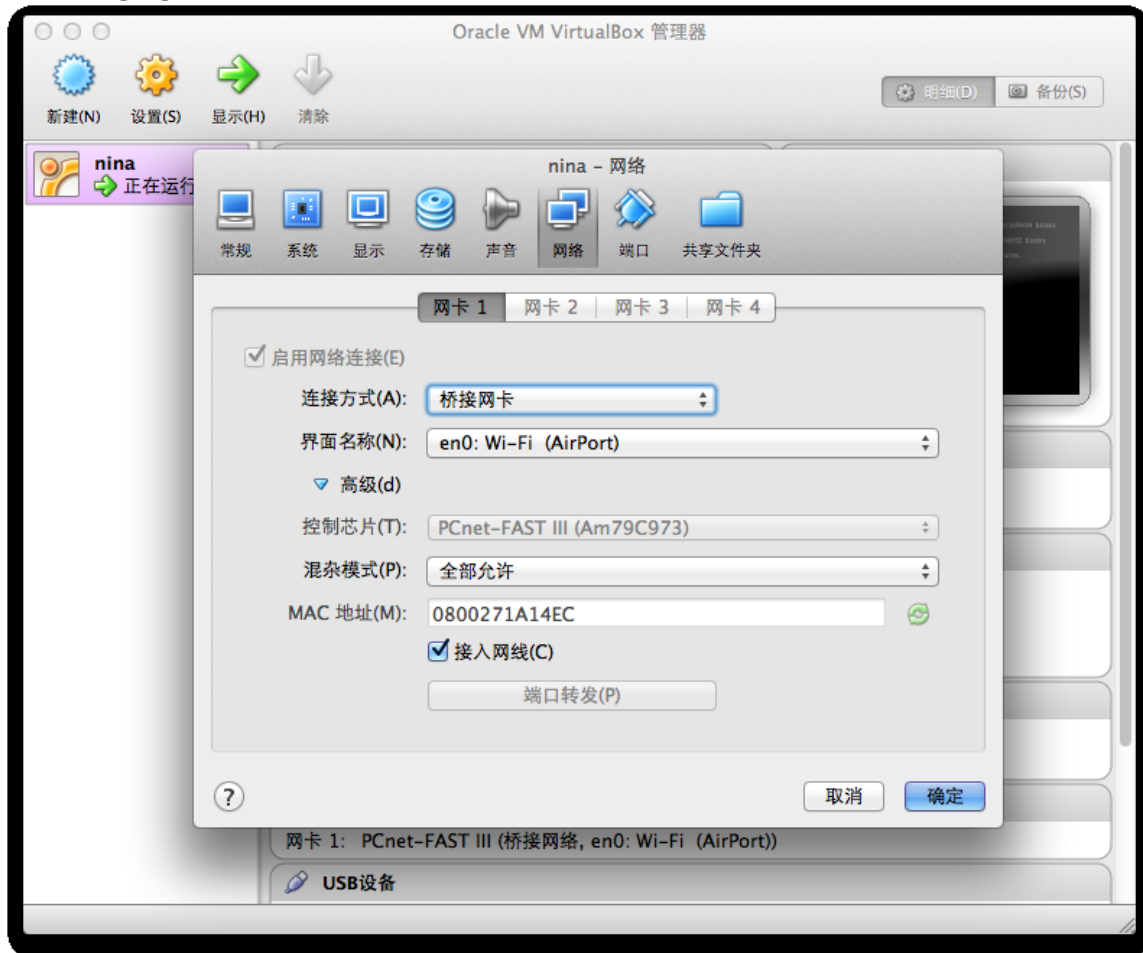
A warning is given while a client tries to rename a file that doesn't exist from the beginning.

```
ftp-HiaHia>rename 10.jpg 1.jpg
Failed --- File or directory not found. Please check the filename.
ftp-HiaHia>503 Bad sequence of commands!
```

### 5.11 Command—mode:

```
ftp-HiaHia>mode
Inform --- Data transport is in PASSIVE mode.
ftp-HiaHia>mode 0
Inform --- Data transport changed to ACTIVE mode.
ftp-HiaHia>mode
Inform --- Data transport is in ACTIVE mode.
ftp-HiaHia>mode 1
Inform --- Data transport changed to PASSIVE mode.
ftp-HiaHia>mode
Inform --- Data transport is in PASSIVE mode.
ftp-HiaHia>
```

To activate active mode, network settings of VirtualBox should to be specified as the following figure shows.



### 5.12 Command—get:

get in passive mode:









### 5.16 Command—**type**:

```
*****Login Successful!*****
ftp-HiaHia>type
Inform --- Data type is unknown,please set a type.
ftp-HiaHia>type a
Inform --- Setting type in ASCII.
200 Type set to A
ftp-HiaHia>type i
Inform --- Setting type in BINARY.
200 Type set to I
```

### 5.17 Command—**quit**:

```
*****Login Successful!*****
ftp-HiaHia>quit

*****
  Thank you for using ftp client.      Authored by Chao Chen and Lina Jin.
*****
nina@ubuntu:/mnt$
```

## 6 Summary and Conclusion

### 6.1 Self-evaluation

**Group member:** Chen Chao

**Task Assignment:**

✚ Code:

- Password hidden;
- Passive mode and active mode;
- FTP commands—ls, put, get;
- Transition between ASCII and Binary;
- Transfer binary files;
- Limit speed of data transmission;
- Show transferring progress;
- Resume transfer after the progress is broken

✚ Report:

- Relevant screen shots in **Result** and flow-charts in **Detailed Design** according to task assignment in the code part.
- **Relationship and interface between the modules**
- **Requirement Analysis**
- **Improvement of design**

**Self-evaluation:**

It's one of the hardest project I've ever participated in. Problems occurred over the whole procedures: command sequence analysis, data structure designing, debugging and so on. I developed ability to handle large scare project and that to solve challenges I've never seen. At the very beginning we have little resource about how to set about the task. Knowledge about FTP on lecture notes are superficial. To conquer this problem, I innovatively installed an ftp server on mobile phone to catch command sequence. With this I can cover all error conditions by sequence. Another problem is to debug the code. Bugs can be divided into functional bugs and non-functional bugs. One example of functional bugs is the function of "ls". Command sent to server should be "LIST -al" rather than "LIST", despite that the latter works on ftp.mayan.cn, it's invalid on other ftp servers. Nonfunctional bugs are often shown as Segmentation Error (core dumped). Most of them are caused by illegal uses of pointers. When error occurs, I insert printf() within every two lines to locate the error point, and adapt the specific line of code. Overall I learned a lot in this course work.

**Group member:** Jin Lina

**Task Assignment:**

✚ Code:

- Login (Hostname&IP);
- FTP commands—pwd, cd, delete, ls, mkdir, rename, quit;
- Highlight name of files;
- Statistics;
- Error handling;

✚ Report:

- Relevant screen shots in **Result** and flow-charts in **Detailed Design** according to task assignment in the code part.
- **Decomposition of functional modules**
- **Overall flow chart**
- **Design of data structures**
- Sequence diagram of each FTP commands in **Design Analysis**

**Self-evaluation:**

At the time of the whole project is to draw to a close, it is a huge relief to write a summary. At the beginning, we reviewed relevant lecture slides to get more specific information about how FTP works. Soon we found the overall structure in the slides is far more from the knowledge we need to acquire to design a FTP Client. With the experience of utilizing Wireshark to capture and analysis the process of DHCP and DNS, we also use this capture tool to obtain more details about the process of FTP commands this time. The sequence diagrams drew in the Design Analysis part are based on the Wireshark capture. When it comes to the design of flow chart, Microsoft Visio is used and the procedures of drawing flow charts and coding are fulfilled simultaneously. I learned so much about network programing and File Transfer Protocol, it is such a good experience filled with pain and pleasure. At last, give all my sincere thanks to my terrific partner; the whole project won't operate well without all the efforts he made.

## 6.2Improvement of design

This ftp client is developed based on the pattern provided by teacher and has some more improvement.

This program optimizes the message display and has specialized method used for printing message.

This program covers most kinds of reply messages, especially those indicate error (5XX and 4XX). And error message are interpreted by discarding the original received message and reprint the wbuf.

This program concentrate much in details. For example, when displaying transport speed of getting files, the speed will be shown as Kb/s or Mb/s according to the specific speed.

However, there remain several points to improve. For instance, when limiting speed,

the principle is to compensate a duration of time after each package received, so that  $t$  increase when  $s$  is not changed, and  $v=s/t$  decrease. There are some better solutions to do this.

## 7 Appendix: Source Codes

```
/* Coursework of Internet Application
   FTP Client
   Author: Chen Chao 10213053 2010215110; Jin Lina 10213071 2010215110
*/

/* header files */
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h> /* getservbyname(), gethostbyname() */
#include <errno.h> /* for definition of errno */
#include <string.h> /* for memset() */
#include <sys/socket.h> /* for socket() */
#include <unistd.h> /* for close() */
#include <arpa/inet.h> /* for sockaddr_in and inet_addr() */
#include <sys/select.h> /* for select() */
#include <sys/time.h> /* timeval */
// #include <time.h>
#include <termios.h> /* Hide password */
#include <netinet/in.h> /* sockaddr_in hostEnt */
#include <signal.h>
#include <sys/ioctl.h>
#include <linux/if.h>
#include <sys/stat.h> /* open() */
#include <sys/types.h> /* open() */
#include <fcntl.h> /* open() */

/* define macros */
#define MAXBUF 1024
#define STDIN_FILENO 0
#define STDOUT_FILENO 1

/* define colors for high light. */
#define NONE "\033[m" //unknown directory
#define GREEN "\033[0;1;32;32m" //c.h
#define RED "\033[0;1;32;31m" //office
#define BLUE "\033[0;1;32;34m" //picture
#define CYAN "\033[0;1;36m" //sound
#define YELLOW "\033[0;1;35m" //video
#define BROWN "\033[0;1;33m" //zip,rar
#define MENTION "\033[0;1;44m"

/* define FTP reply code */
#define USERNAME 220
#define PASSWORD 331
#define LOGIN 230
#define PATHNAME 257
#define CLOSEDATA 226
#define ACTIONOK 250

/* Define action code for distinguishing messages when receiving. */
#define NO 0 /* no action */
#define LS 1
#define CD 2
```

```
#define PWD 3
#define MKDIR 4
#define DELETE 5
#define RENAME 6
#define RENAME1 7
#define GETFILE 8
#define GETFILE1 9 // get file in breakpoint resume mode
#define PUTFILE 10
#define PUTFILE1 11 //put file in breakpoint resume mode
#define TYPE 12

#define FILENAME 1
#define DIRECTORYNAME 2

/* define data channel transport mode*/
#define ACTIVE 0
#define PASSIVE 1

/* define data type*/
#define BINARY 0
#define ASCII 1
#define UNKNOWN 2

/* Define global variables */
char *host; /* hostname or dotted-decimal string */
char *port; /* port number string */
char *rbuf, *rbuf1; /* pointer that is malloc'ed */
char *wbuf, *wbuf1; /* pointer that is malloc'ed */
char *dir; /* directory, sometimes used to store file name */
char *logo = "ftp-HiaHia>";
int size, dl, ul, setting, slp, mode, connected, init, action, listStatus, sock_active, type;
struct termios new_settings, stored_settings;
struct hostent *h;
struct sockaddr_in local_host;

/* Announce functions */
int cliopen(char *host, char *port);
void strtosrv(char *str, char *host, char *port);
int typeget(char *name);
int min(int a, int b);
void printClr(char *record, int a);
void printPanel();
void printHint();
void printTransfer(int percent, double speed, int count);
void cmd_tcp(int sockfd);
void ftp_list(int sockfd, int index);
int ftp_get(int sck, char *pDownloadFileName_s);
int ftp_put(int sck, char *pUploadFileName_s);
int timeDiff(struct timeval* tv1, struct timeval* tv2);
void goSleep(int status, double speed, int size);
int checkName(char *name, int index);
int active(int sockfd);

int main(int argc, char *argv[]) {
    /* initialize parameters */
    int fd;
    type = UNKNOWN;
    connected = 0;
    init = 0;
    ul = 2048;
```



```
    dl = 9999;
    listStatus = 0;
    slp = 1;
    action = NO;
    mode = PASSIVE;

    if (0 != argc - 2) {
        printf("%s\n", "missing <hostname>");
        exit(0);
    }

    host = argv[1];
    port = "21";

    //code here: Allocate the read and write buffers before open().
    if ((rbuf = (char *) malloc(sizeof (char)*(MAXBUF))) == NULL || (rbuf1 = (char *) malloc(sizeof
(char)*MAXBUF)) == NULL || (wbuf = (char *) malloc(sizeof (char)*MAXBUF)) == NULL || (wbuf1 = (char *)
malloc(sizeof (char)*MAXBUF)) == NULL) {
        printf("Failed to allocate memory for buffers.");
        exit(0);
    }
    fd = cliopen(host, port);
    printPanel();
    printf("\n*****\n");
    printf("Welcome! Please log in.");
    printf("%57s", host);
    printf("\n*****\n");
    cmd_tcp(fd);

    exit(0);
}

/* Establish a TCP connection from client to server */
int cliopen(char *host, char *port) {
    //code here
    int sockfd;

    struct sockaddr_in servaddr;
    struct in_addr addr;

    if (connected == 0) {
        //If the input is IP address, do the following
        if (inet_aton(host, (struct in_addr *) &addr) != 0) {
            if ((h = gethostbyaddr((char *) &addr, sizeof (addr), AF_INET)) == NULL) {
                printf("\nError occurs when gethostbyaddr.\n");
                exit(1);
            }
            //If the input is domain name, do the following.
        } else if ((h = gethostbyname(host)) == NULL) {
            printf("Error occurs when gethostbyname.\n");
            exit(0);
        }
    }
    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) //Set up a socket.
    {
        printf("Error occurs when establishing socket");
        exit(0);
    }

    // Copy informaton into sockaddr_in object */
```

```
memset(&servaddr, 0, sizeof (servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = ((struct in_addr *) (h->h_addr))->s_addr;
servaddr.sin_port = htons(atoi(port));

/* Connect */
if (connect(sockfd, (struct sockaddr*) &servaddr, sizeof (servaddr)) == -1) {
    printf("connect() failed.\n");
    exit(0);
}
return sockfd;
}

/*
Compute server's port by a pair of integers and store it in char *port
Get server's IP address and store it in char *host
*/
void strtosrv(char *str, char *host, char *port) {
    //code here
    char *result;
    char *host1;
    char *str1;

    int temp = 1;
    int i;
    result = (char *) malloc(sizeof (char)*5);
    host1 = (char *) malloc(40);
    memset(result, 0, 5);
    memset(host1, 0, 40);
    str1 = strstr(str, "(");
    str1 = str1 + 1;
    //take out 6 numbers from the string.
    for (i = 0; i < 6; i++) {
        switch (i) {
            case 0:
                sprintf(result, "%s", strtok(str1, ","));
                strcat(host1, result);
                strcat(host1, ".");
                break;
            case 1: case 2: case 3:
                sprintf(result, "%s", strtok(NULL, ","));
                strcat(host1, result);
                if (i != 3)
                    strcat(host1, ".");
                break;
            case 4:
                sprintf(result, "%s", strtok(NULL, ","));
                temp = (temp)*(atoi(result)*256);
                break;
            case 5:
                sprintf(result, "%s", strtok(NULL, ","));
                temp = temp + atoi(result);
                break;
            default:
                break;
        }
    }
    sprintf(port, "%d", temp);
    sprintf(host, "%s", host1);
}
```

```
/* Read and write as command connection */
void
cmd_tcp(int sockfd) {
    int maxfdp1, nread, nwrite, fd, replycode, file1, cursize, i, set, new_sock;
    char host[16];
    char port[6];
    fd_set rset;
    char *temp = (char *) malloc(sizeof (char)*50);

    FD_ZERO(&rset);
    maxfdp1 = sockfd + 1; /* check descriptors [0..sockfd] */

    /* Main loop of the program. */
    for (;;) {
        FD_SET(STDIN_FILENO, &rset);
        FD_SET(sockfd, &rset);

        if (select(maxfdp1, &rset, NULL, NULL, NULL) < 0)
            printf("select error\n");

        /* data to read on stdin */
        if (FD_ISSET(STDIN_FILENO, &rset)) {
            memset(rbuf, 0, MAXBUF);
            //memset(wbuf,0,MAXBUF);
            if ((nread = read(STDIN_FILENO, rbuf, MAXBUF)) < 0)
                printf("read error from stdin\n");
            nwrite = nread + 5;

            /* send username */
            if (replycode == USERNAME) {
                sprintf(wbuf, "USER %s", rbuf);
                if (write(sockfd, wbuf, nwrite) != nwrite)
                    printf("write error\n");
            } //code here: send password
            else if (replycode == PASSWORD) {
                sprintf(wbuf, "PASS %s", rbuf);
                nwrite = strlen(wbuf);
                if (write(sockfd, wbuf, nwrite) != nwrite)
                    printf("write error\n");
            }

            /* send command */
            else if (replycode == LOGIN || replycode == CLOSEDATA || replycode == PATHNAME || replycode
== ACTIONOK) {
                /* ls - list files and directories
                * this client provide three display mode
                * according to the input.
                */
                if (strncmp(rbuf, "ls", 2) == 0) {
                    memset(wbuf, 0, MAXBUF);
                    action = LS;
                    if (strcmp(rbuf, "ls -a\n") == 0)
                        listStatus = 1;
                    else if (strcmp(rbuf, "ls\n") == 0)
                        listStatus = 2;
                    else if (strncmp(rbuf, "ls -t ", 6) == 0 && strlen(rbuf) > 6) {
                        listStatus = 3;
                        strtok(rbuf, " ");
                        strtok(NULL, " ");
                        dir = (char *) malloc(sizeof (char)*10);
                    }
                }
            }
        }
    }
}
```

```
        strcpy(dir, strtok(NULL, "\\r\\n"));
    } else
        listStatus = 1;
    /* Command varies according to transport mode. */
    if (mode == PASSIVE) {
        memset(wbuf, 0, MAXBUF);
        sprintf(wbuf, "Connecting in Passive mode...\\n");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        memset(wbuf, 0, MAXBUF);
        sprintf(wbuf, "%s", "PASV\\n");

    } else if (mode == ACTIVE) {
        memset(wbuf, 0, MAXBUF);
        sprintf(wbuf, "Connecting in Active mode...\\n");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        fd = active(sockfd);
    }
    if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
        printf("write error\\n");
    continue;
}
// code here: cd - change working directory/
else if (strncmp(rbuf, "cd", 2) == 0) {
    memset(wbuf, 0, MAXBUF);
    /* Input error detection */
    if (strlen(rbuf) < 4) {
        sprintf(wbuf, "Usage: cd <dir_name>\\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } else {
        action = CD;
        strtok(rbuf, " ");
        dir = strtok(NULL, " ");
        sprintf(wbuf, "CWD %s", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\\n");
        continue;
    }
}
/* pwd - print working directory */
else if (strncmp(rbuf, "pwd", 3) == 0) {
    memset(wbuf, 0, MAXBUF);
    action = PWD;
    sprintf(wbuf, "%s", "PWD\\n");
    write(sockfd, wbuf, 4);
    // printf("write error\\n");
    continue;
}
// code here: quit - quit from ftp server
else if (strncmp(rbuf, "quit", 4) == 0) {
    memset(wbuf, 0, MAXBUF);
    sprintf(wbuf, "%s", "QUIT\\n");
    if (write(sockfd, wbuf, 5) != 5)
        printf("write error\\n");
    continue;
}
/* mkdir - set up a new directory */
else if (strncmp(rbuf, "mkdir", 5) == 0) {
    memset(wbuf, 0, MAXBUF);
    /* Input error detection */
    if (strlen(rbuf) < 7) {
        sprintf(wbuf, "Usage: mkdir <dir_name>\\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    }
}
```

```
    } else {
        action = MKDIR;
        dir = rbuf + 6;
        /* Check name validation */
        if (checkName(dir, DIRECTORYNAME) == 0) {
            action = NO;
            sprintf(wbuf, "Error --- Illegal name.\nftp-HiaHia>");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            continue;
        }
        sprintf(wbuf, "MKD %s", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    }
} /* delete - delete file on the server */
else if (strncmp(rbuf, "delete", 6) == 0) {
    memset(wbuf, 0, MAXBUF);
    /* Input error detection */
    if (strlen(rbuf) < 8) {
        sprintf(wbuf, "Usage: delete <file_name>\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } else {
        action = DELETE;
        dir = rbuf + 6;
        sprintf(wbuf, "DELE %s", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    }
}

} /* rename - rename a file on the server */
else if (strncmp(rbuf, "rename", 6) == 0) {
    char *filename;
    /* Input error detection */
    if (strlen(rbuf) < 10) {
        sprintf(wbuf, "Usage: rename <file_name> <new_name>\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } else {
        action = RENAME;
        strtok(rbuf, " ");
        // obtain filename from string
        if ((filename = strtok(NULL, " ")) == NULL) {
            action = NO;
            sprintf(wbuf, "Usage: rename <file_name> <new_name>\nftp-HiaHia>");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            continue;
        }
        // obtain new name from string
        if ((dir = strtok(NULL, "\r\n")) == NULL) {
            action = NO;
            sprintf(wbuf, "Usage: rename <file_name> <new_name>\nftp-HiaHia>");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            continue;
        }
    }
    // Input error detection
    if (checkName(dir, FILENAME) == 0 && checkName(dir, DIRECTORYNAME) == 0) {
        action = NO;
    }
}
```

```
        sprintf(wbuf, "Error --- Illegal name.\n");
        strcat(wbuf, "Usage: rename <file_name> <new_name>\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    }
    sprintf(wbuf, "RNFR %s\r\n", filename);
    if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
        printf("write error\n");

    sprintf(wbuf, "RNT0 %s\r\n", dir);
    if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
        printf("write error\n");
    continue;
}
} // code here: get - get file from ftp server
else if (strncmp(rbuf, "get", 3) == 0) {
    /* Input error detection */
    if (strlen(rbuf) < 5) {
        sprintf(wbuf, "Usage: get <file_name>\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } else {
        action = GETFILE;
        strtok(rbuf, " ");
        temp = strtok(NULL, "\r\n");
        dir = (char *) malloc(sizeof(char)*100);
        strcpy(dir, temp);
        /* Command varies according to transport type */
        if (mode == PASSIVE) {
            sprintf(wbuf, "%s", "Connecting in Passive mode...\n");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            memset(wbuf, 0, MAXBUF);
            sprintf(wbuf, "%s", "PASV\n");
        } else if (mode == ACTIVE) {
            sprintf(wbuf, "Connecting in Active mode...\n");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            fd = active(sockfd);
        }
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    }
}

} // code here: put - put file upto ftp server
else if (strncmp(rbuf, "put", 3) == 0) {
    //Input error detection
    if (strlen(rbuf) < 5) {
        sprintf(wbuf, "Usage: put <file_name>\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } else {
        // Obtain filename from string
        strtok(rbuf, " ");
        temp = strtok(NULL, "\r\n");
        dir = (char *) malloc(sizeof(char)*100);
        strcpy(dir, temp);
        //Check whether the file exists.
        if (access(dir, R_OK) != 0) {
            sprintf(wbuf, "Error --- Failed to read the file.\nftp-HiaHia>");
        }
    }
}
```

```

        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } else {
        //Command varies according to transport mode
        action = PUTFILE;
        memset(wbuf, 0, MAXBUF);
        if (mode == PASSIVE) {
            sprintf(wbuf, "%s", "Connecting in Passive mode...\n");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            memset(wbuf, 0, MAXBUF);
            sprintf(wbuf, "%s", "PASV\n");
        } else if (mode == ACTIVE) {
            sprintf(wbuf, "Connecting in Active mode...\n");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            fd = active(sockfd);
        }
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    }
}

/* dl - view or change the download speed limitation*/
else if (strncmp(rbuf, "dl", 2) == 0) {
    memset(wbuf, 0, MAXBUF);
    //Input error detection
    if (strlen(rbuf) < 4 || strtok(rbuf, " ") == NULL || (temp = strtok(NULL, "\r\n")) == NULL)
    {
        sprintf(wbuf, "Inform --- Download limitation is %d Kb/s.\nftp-HiaHia>", dl);
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } else {
        if ((setting = atoi(temp)) == 0 || setting < 1 || setting > 9999) {
            sprintf(wbuf, "Usage: dl <download_limitation> (1~9999)\nftp-HiaHia>");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            continue;
        } else {
            dl = setting;
            sprintf(wbuf, "Inform --- Download limitation has changed to %d
Kb/s.\nftp-HiaHia>", dl);
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            continue;
        }
    }
}

/* ul - view or change the upload speed limitation*/
else if (strncmp(rbuf, "ul", 2) == 0) {
    memset(wbuf, 0, MAXBUF);
    //Input error detection
    if (strlen(rbuf) < 4 || strtok(rbuf, " ") == NULL || (temp = strtok(NULL, "\r\n")) == NULL)
    {
        sprintf(wbuf, "Inform --- Upload limitation is %d Kb/s.\nftp-HiaHia>", ul);
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
    } else {
        if ((setting = atoi(temp)) == 0 || setting < 20 || setting > 2048) {
            sprintf(wbuf, "Usage: ul <upload_limitation> (20~2048)\nftp-HiaHia>");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            continue;
        } else {
            ul = setting;

```

```

        sprintf(wbuf, "Inform --- Upload limitation has changed to %d
Kb/s.\nftp-HiaHia>", ul);
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    }

    }
    /* clear - clean the screen */
    else if (strncmp(rbuf, "clear", 5) == 0) {
        system("clear");
        sprintf(wbuf, "ftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } /* help - show help message */
    else if (strncmp(rbuf, "help", 4) == 0) {
        memset(wbuf, 0, MAXBUF);
        strcat(wbuf,
Menu*****\n");
        strcat(wbuf, " cd <directory>          --- Move to a specific directory.\n");
        strcat(wbuf, " pwd                      --- Return the name of current
directory.\n");
        strcat(wbuf, " delete <file>          --- Delete a file.\n");
        strcat(wbuf, " rename <file> <new name>      --- Rename a file.\n");
        strcat(wbuf, " mkdir <directory>        --- Set up a new directory under current
directory.\n");
        strcat(wbuf, " ls                      --- View file list in current directory.\n");
        strcat(wbuf, " ls -a                  --- In detail view file list in current
directory.\n");
        strcat(wbuf, " ls -t <type>          --- Highlight file of type <type> in a list\n");
        strcat(wbuf, " get <file>          --- Download a file from the server.\n");
        strcat(wbuf, " put <file>          --- Upload a file onto the server.\n");
        strcat(wbuf, " mode                  --- View current connection mode.\n");
        strcat(wbuf, " mode <mode>          --- Change connection
mode(1-passive;0-active)\n");
        strcat(wbuf, " type                  --- View current data type.\n");
        strcat(wbuf, " type <mode>          --- Change data
type(a-ASCII;i-BINARY)\n");
        strcat(wbuf, " ul                      --- View the upload speed limitation.\n");
        strcat(wbuf, " ul <speed>          --- Limit upload speed under a specific
value.\n");
        strcat(wbuf, " dl                      --- View the download speed
limitation.\n");
        strcat(wbuf, " dl <speed>          --- Limit download speed under a specific
value.\n");
        strcat(wbuf, " help                  --- View help information.\n");
        strcat(wbuf, " quit                  --- Log out and exit.\n");
        strcat(wbuf,
*****\n\n");
        strcat(wbuf, "ftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    } /* mode - view or change data transport mode */
    else if (strncmp(rbuf, "mode", 4) == 0) {
        memset(wbuf, 0, MAXBUF);
        // View data transport mode
        if (strlen(rbuf) < 6 || strtok(rbuf, " ") == NULL || (temp = strtok(NULL, "\r\n")) == NULL)
        {
            if (mode == 1)
                sprintf(wbuf, "Inform --- Data transport is in PASSIVE mode.\nftp-HiaHia>");
            else

```



```

        sprintf(wbuf, "Inform --- Data transport is in ACTIVE mode.\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
    } else {
        // Input error detection
        if (!((setting = atoi(temp)) == 0 || setting == 1)) {
            sprintf(wbuf, "Usage: mode <mode> (1-passive;0-active)\nftp-HiaHia>");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            continue;
        } else {
            // Mode setting
            mode = setting;
            if (mode == 1)
                sprintf(wbuf, "Inform --- Data transport changed to PASSIVE
mode.\nftp-HiaHia>");
            else
                sprintf(wbuf, "Inform --- Data transport changed to ACTIVE
mode.\nftp-HiaHia>");
            write(STDOUT_FILENO, wbuf, strlen(wbuf));
            continue;
        }
    }
}
/* type - view or change data type */
else if (strcmp(rbuf, "type", 4) == 0) {
    memset(wbuf, 0, MAXBUF);
    // View current data type
    if (strlen(rbuf) < 6) {
        if (type == BINARY)
            sprintf(wbuf, "Inform --- Data type is BINARY.\nftp-HiaHia>");
        else if (type == ASCII)
            sprintf(wbuf, "Inform --- Data type is ASCII.\nftp-HiaHia>");
        else if (type == UNKNOWN)
            sprintf(wbuf, "Inform --- Data type is unknown,please set a
type.\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
    }
    // Set data type to ASCII
    else if (strcmp(rbuf, "type a", 6) == 0) {
        sprintf(wbuf, "%s\n", "TYPE A");
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        sprintf(wbuf, "Inform --- Setting type in ASCII.\n");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        type = ASCII;
        action = TYPE;
        // Set data type tp BINARY
    } else if (strcmp(rbuf, "type i", 6) == 0) {
        sprintf(wbuf, "%s\n", "TYPE I");
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        sprintf(wbuf, "Inform --- Setting type in BINARY.\n");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        type = ASCII;
        action = TYPE;
    } else {
        // Input error detection
        sprintf(wbuf, "Usage: type <type> (1-ASCII;0-BINARY)\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
    }
} else {
    // Input error detection
    memset(wbuf, 0, MAXBUF);

```

```

        sprintf(wbuf, "Error --- Invalid command. Type in \"help\" when you need
help.\nftp-HiaHia>");
        write(STDOUT_FILENO, wbuf, strlen(wbuf));
        continue;
    }
}

/* data to read from socket */
if (FD_ISSET(sockfd, &rset)) {
    memset(rbuf, 0, MAXBUF);
    memset(wbuf, 0, MAXBUF);
    if ((nread = recv(sockfd, rbuf, MAXBUF, 0)) < 0)
        printf("recv error\n");
    else if (nread == 0)
        break;

    /* Open data connection */
    if (strncmp(rbuf, "150", 3) == 0) {
        if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
            printf("write error to stdout\n");
        // Data connection for ls function
        if (action == LS) {
            if (mode == PASSIVE) {
                ftp_list(fd, listStatus);
                continue;
            } else if (mode == ACTIVE) {
                set = sizeof (local_host);
                for (i = 0; i < 3; i++) {
                    if ((new_sock = accept(fd, (struct sockaddr *) &local_host, (socklen_t *) &set))
== -1) {

                        printf("accept return:%s errno: %d\n", strerror(errno), errno);
                        write(STDOUT_FILENO, "Warning --- accept() error.\n", 29);
                        continue;
                    } else break;
                }
                ftp_list(new_sock, listStatus);
                continue;
            }
        } // Data connection for getfile function
        else if (action == GETFILE || action == GETFILE1) {
            if (mode == PASSIVE) {
                ftp_get(fd, dir);
                continue;
            } else if (mode == ACTIVE) {
                set = sizeof (local_host);
                for (i = 0; i < 3; i++) {
                    if ((new_sock = accept(fd, (struct sockaddr *) &local_host, (socklen_t *) &set))
== -1) {

                        printf("accept return:%s errno: %d\n", strerror(errno), errno);
                        write(STDOUT_FILENO, "Warning --- accept() error.\n", 29);
                        continue;
                    } else break;
                }
                ftp_get(new_sock, dir);
                continue;
            }
        } // Data connection for putfile function
        else if (action == PUTFILE || action == PUTFILE1) {
            if (mode == PASSIVE) {

```

```

        ftp_put(fd, dir);
        continue;
    } else if (mode == ACTIVE) {
        set = sizeof (local_host);
        for (i = 0; i < 3; i++) {
            if ((new_sock = accept(fd, (struct sockaddr *) &local_host, (socklen_t *) &set))
== -1) {

                printf("accept return:%s errno: %d\n", strerror(errno), errno);
                write(STDOUT_FILENO, "Warning --- accept() error.\n", 29);
                continue;
            } else break;
        }
        ftp_put(new_sock, dir);
    }
}

/* Active mode reply and data type setting reply*/
if (strncmp(rbuf, "200", 3) == 0) {
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    if (action == LS) {
        memset(wbuf, 0, MAXBUF);
        sprintf(wbuf, "%s", "LIST -al\n");
        if (write(sockfd, wbuf, 9) != 9)
            printf("write error\n");
        continue;
    } else if (action == GETFILE) {
        memset(wbuf, 0, MAXBUF);
        sprintf(wbuf, "SIZE %s\n", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    } else if (action == PUTFILE) {
        memset(wbuf, 0, MAXBUF);
        sprintf(wbuf, "SIZE %s\n", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    } else if (action == TYPE) {
        action = NO;
        replycode = ACTIONOK;
        memset(rbuf, 0, MAXBUF);
        sprintf(rbuf, "%s", logo);
        if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
            printf("write error to stdout\n");
        continue;
    }
}

/* Return of size of file on the server */
if (strncmp(rbuf, "213", 3) == 0) {
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    if (action == GETFILE) {
        strtok(rbuf, " ");
        size = atoi(strtok(NULL, "\r\n"));
        if (access(dir, F_OK) != 0) {
            sprintf(wbuf, "RETR %s\n", dir);
            if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
                printf("write error\n");
        }
    }
}

```

```
        continue;
    } else {
        file1 = open(dir, O_RDONLY);
        cursize = lseek(file1, 0, SEEK_END);
        if (cursize < size) {
            sprintf(wbuf, "REST %d\n", cursize);
            if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
                printf("write error\n");
            close(file1);
            action = GETFILE1;
            continue;
        } else {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, "WARNING --- File already downloaded.\n");
            strcat(rbuf, logo);
            nread = strlen(rbuf);
            action = NO;
            replycode = ACTIONOK;
            close(file1);
        }
    }
}

if (action == PUTFILE) {
    strtok(rbuf, " ");
    if ((temp = strtok(NULL, "\r\n")) == NULL) {
        sprintf(wbuf, "STOR %s\n", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    }
    size = atoi(temp);
    file1 = open(dir, O_RDONLY);
    cursize = lseek(file1, 0, SEEK_END);
    if (cursize > size) { //xuchuan
        sprintf(wbuf, "APPE %s\n", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        close(file1);
        action = PUTFILE1;
        continue;
    } else {
        memset(rbuf, 0, MAXBUF);
        strcat(rbuf, "WARNING --- File already uploaded.\n");
        strcat(rbuf, logo);
        nread = strlen(rbuf);
        action = NO;
        replycode = ACTIONOK;
        close(file1);
    }
}

}

/* set replycode and wait for user's input */
if (strcmp(rbuf, "220", 3) == 0) {
    memset(rbuf, 0, MAXBUF);
    if (init == 0) {
        strcat(rbuf, "Username: ");
        nread = 10;
        init = 1;
    }
}
```

```
        replycode = USERNAME;
    }

    /* Quit. */
    if (strncmp(rbuf, "221", 3) == 0) {
        break;
    }

    /* Data transport complete and waiting for user input. */
    if (strncmp(rbuf, "226", 3) == 0) {
        printf("\n");
        if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
            printf("write error to stdout\n");
        if (action == LS) {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, logo);
            nread = strlen(rbuf);
            replycode = ACTIONOK;
            action = NO;
        } else if (action == GETFILE || action == GETFILE1) {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, "Success --- File stored to local directory.\n");
            strcat(rbuf, logo);
            nread = strlen(rbuf);
            replycode = ACTIONOK;
            action = NO;
        } else if (action == PUTFILE || PUTFILE1) {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, "Success --- File sent to the server.\n");
            strcat(rbuf, logo);
            nread = strlen(rbuf);
            replycode = ACTIONOK;
            action = NO;
        }
    }

    /* open data connection in PASSIVE mode*/
    if (strncmp(rbuf, "227", 3) == 0) {
        if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
            printf("write error to stdout\n");
        if (action == LS) {
            strtosrv(rbuf, host, port);
            fd = cliopen(host, port);
            memset(wbuf, 0, MAXBUF);
            sprintf(wbuf, "%s", "LIST -al\n");
            if (write(sockfd, wbuf, 9) != 9)
                printf("write error\n");
            continue;
        } else if (action == GETFILE) {
            strtosrv(rbuf, host, port);
            fd = cliopen(host, port);
            sprintf(wbuf, "SIZE %s\n", dir);
            if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
                printf("write error\n");
            continue;
        } else if (action == PUTFILE) {
            strtosrv(rbuf, host, port);
            fd = cliopen(host, port);
            sprintf(wbuf, "SIZE %s\n", dir);
            if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
```

```

        printf("write error\n");
        continue;
    } else
        printf("227");
}

/* LOGIN successfully. */
if (strcmp(rbuf, "230", 3) == 0) {
    memset(rbuf, 0, MAXBUF);
    connected = 1;
    system("clear");
    strcat(rbuf,
Successful!*****\n");
    strcat(rbuf, logo);
    nread = strlen(rbuf);
    replycode = LOGIN;
    tcsetattr(0, TCSANOW, &stored_settings);
}

/*file action complete*/
if (strcmp(rbuf, "250", 3) == 0) {
    strcat(rbuf, logo);
    nread = strlen(rbuf);
    replycode = ACTIONOK;
    action = NO;
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    continue;
}

/* Directory successfully established. */
if (strcmp(rbuf, "257", 3) == 0) {
    strcat(rbuf, logo);
    replycode = ACTIONOK;
    nread = strlen(rbuf);
    action = NO;
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    continue;
}

/* enter password */
if (strcmp(rbuf, "331", 3) == 0) {
    memset(rbuf, 0, MAXBUF);
    strcat(rbuf, "your password: ");
    nread = 16;
    replycode = PASSWORD;
    tcsetattr(0, &stored_settings);
    new_settings = stored_settings;
    new_settings.c_lflag &= (~ECHO);
    tcsetattr(0, TCSANOW, &new_settings);
}

/* Pause when doing file actions. Used in breakpoint resume and rename. */
if (strcmp(rbuf, "350", 3) == 0) {
    if (action == RENAME) {
        if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
            printf("write error to stdout\n");
        continue;
    }
    if (action == GETFILE1) {

```

```
        if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
            printf("write error to stdout\n");
        sprintf(wbuf, "RETR %s\n", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    }
}

/* Service closedpwd */
if (strncmp(rbuf, "421", 3) == 0) {
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    replycode = LOGIN;
    action = NO;
    continue;
}

/* Unable to open data connection */
if (strncmp(rbuf, "425", 3) == 0) {
    strcat(rbuf, logo);
    nread = strlen(rbuf);
    replycode = ACTIONOK;
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    continue;
}

/* File unavailable */
if (strncmp(rbuf, "450", 3) == 0) {
    strcat(rbuf, logo);
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    continue;
}

/* Invalid command */
if (strncmp(rbuf, "500", 3) == 0) {
    strcat(rbuf, logo);
    nread = strlen(rbuf);
    replycode = ACTIONOK;
    action = NO;
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    continue;
}

/* Invalid parameter */
if (strncmp(rbuf, "501", 3) == 0) {
    strcat(rbuf, logo);
    nread = strlen(rbuf);
    replycode = ACTIONOK;
    action = NO;
    if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
        printf("write error to stdout\n");
    continue;
}

/* Command not excuted */
if (strncmp(rbuf, "502", 3) == 0) {
    strcat(rbuf, logo);
    nread = strlen(rbuf);
    replycode = ACTIONOK;
```

```
        action = NO;
        if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
            printf("write error to stdout\n");
        continue;
    }
    /* Command in invalid order */
    if (strncmp(rbuf, "503", 3) == 0) {

        strcat(rbuf, logo);
        nread = strlen(rbuf);
        replycode = ACTIONOK;
        action = NO;
        if (write(STDOUT_FILENO, rbuf, strlen(rbuf)) != strlen(rbuf))
            printf("write error to stdout\n");
        continue;
    }
    /* Not logged onto server. Username or password error. */
    if (strncmp(rbuf, "530", 3) == 0) {
        memset(rbuf, 0, MAXBUF);
        strcat(rbuf, "\n\n*****Login failed. Incorrect username or
password.*****\n");
        strcat(rbuf, "Username: ");
        nread = strlen(rbuf);
        replycode = USERNAME;
        tcsetattr(0, TCSANOW, &stored_settings);
    }

    /* Unable to open data connection */
    if (strncmp(rbuf, "550", 3) == 0) {
        if (action == CD) {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, "Failed --- Directory not found.\n");
            strcat(rbuf, logo);
            nread = strlen(rbuf);
            replycode = LOGIN;
            action = NO;
        } else if (action == MKDIR) {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, "Failed --- Directory name already exists. Please choose another.\n");
            strcat(rbuf, logo);
            nread = strlen(rbuf);
            replycode = LOGIN;
            action = NO;
        } else if (action == DELETE) {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, "Failed --- File not found. Please check the file name.\n");
            strcat(rbuf, logo);
            nread = strlen(rbuf);
            replycode = ACTIONOK;
            action = NO;
        } else if (action == RENAME) {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, "Failed --- File or directory not found. Please check the filename.\n");
            strcat(rbuf, logo);
            nread = strlen(rbuf);
            replycode = LOGIN;
            action = NO;
        } else if (action == GETFILE || action == GETFILE1) {
            memset(rbuf, 0, MAXBUF);
            strcat(rbuf, "Failed --- File or directory not found. Please check the filename.\n");
```



```
        strcat(rbuf, logo);
        nread = strlen(rbuf);
        replycode = ACTIONOK;
        action = NO;
    } else if (action == PUTFILE) {
        size = 0;
        sprintf(wbuf, "STOR %s\n", dir);
        if (write(sockfd, wbuf, strlen(wbuf)) != strlen(wbuf))
            printf("write error\n");
        continue;
    }
}

/* Filename rejected */
if (strcmp(rbuf, "553", 3) == 0) {
    if (action == RENAME) {
        memset(rbuf, 0, MAXBUF);
        strcat(rbuf, "Failed --- The new file name has been used. Please choose another.\n");
        strcat(rbuf, logo);
        nread = strlen(rbuf);
        replycode = ACTIONOK;
        action = NO;
    }
}
if (write(STDOUT_FILENO, rbuf, nread) != nread)
    printf("write error to stdout\n");
}
}
if (close(sockfd) < 0)
    printf("close error\n");
else {
    printf("\n*****\n");
    printf("    Thank you for using ftp client.");
    printf("        Authored by ");
    printf(RED"Chao Chen");
    printf(NONE" and ");
    printf(RED"Lina Jin");
    printf(NONE".\n");
    printf("*****\n");
}
}

/* Read and write as data transfer connection */
void
ftp_list(int sockfd, int index) {
    int nread;
    char *temp;
    int i;
    char *record;
    int count = 0;
    int offset;
    char *bigbuf = "\n";

    for (;;) {
        /* data to read from socket */
        if ((nread = recv(sockfd, rbuf1, MAXBUF, 0)) < 0)
            printf("recv error\n");

        else if (nread == 0)
            break;
    }
}
```

```
// Allocate memory for bigbuffer. Add content to memory for each loop.
if ((temp = (char *) malloc(sizeof (char)*(strlen(bigbuf) + MAXBUF))) == NULL)
    printf("Error when allocating memory\n");
memset(temp, 0, strlen(temp));
strcat(temp, bigbuf);
strcat(temp, rbuf1);
bigbuf = temp;
memset(rbuf1, 0, MAXBUF);
}
bigbuf++;

/* Deal with bigbuffer seperatively according to input. Three mode provided*/

if (strlen(bigbuf) == 0) {
    printHint();
    printf(NONE"\nNo file.\n");
} // Print all messages received, Paint each line according to file type.
else if (index == 1) {
    printHint();
    temp = (char *) malloc(sizeof (char)*100);
    memset(temp, 0, 100);
    record = strtok(bigbuf, "\r\n");
    strncpy(temp, record, strlen(record));
    record = record + strlen(record) - 1;
    while (strncmp(record, " ", 1) != 0) {
        record = record - 1;
    }
    record += 1;
    printClr(temp, typeget(record));
    printf(NONE"\n");
    memset(temp, 0, 100);
    count++;
    while ((record = strtok(NULL, "\r\n")) != NULL) {
        strncpy(temp, record, strlen(record));
        record = record + strlen(record) - 1;
        while (strncmp(record, " ", 1) != 0) {
            record = record - 1;
        }
        record += 1;
        printClr(temp, typeget(record));
        printf(NONE"\n");
        memset(temp, 0, 100);
        count++;
        if (count == 23) {
            count = 0;
            printf(NONE"Press Enter to continue.\n");
            getchar();
            system("clear");
            printHint();
        }
    }
    printf(NONE"\n ");
} // Print file name only. Paint in different colors according to input command.
else if (index == 2) {
    printHint();
    record = strtok(bigbuf, "\r\n");
    record = record + strlen(record) - 1;
    while (strncmp(record, " ", 1) != 0) {
        record = record - 1;
    }
```

```

    }
    record += 1;
    printClr(record, typeget(record));
    count++;

    while ((record = strtok(NULL, "\r\n")) != NULL) {
        record = record + strlen(record) - 1;
        while (strncmp(record, " ", 1) != 0) {
            record = record - 1;
        }
        record += 1;
        printClr(record, typeget(record));
        count++;
        if (count % 3 == 2)
            printf("\n");
        if (count == (23 * 3 + 2)) {
            count = 0;
            printf(NONE"\nPress Enter to continue\n");
            getchar();
            system("clear");
            printHint();
        }
    }
    printf(NONE"\n ");
} // Print all message. High light specific line according to specific file type.
else if (index == 3) {
    //fflush(stdout);
    temp = (char *) malloc(sizeof(char)*100);
    memset(temp, 0, 100);
    record = strtok(bigbuf, "\r\n");
    strncpy(temp, record, strlen(record));
    record = record + strlen(record) - 1;
    while (strncmp(record, ".", 1) != 0 && strlen(record) < 7)
        record = record - 1;
    record += 1;
    printf(NONE"\r");
    if (strcmp(record, dir) == 0) {
        printf(MENTION"%s", temp);
        for (i = 0; i < (80 - strlen(temp)); i++) {
            printf(NONE" ");
        }
        printf(NONE"\n");
    } else {
        printf(NONE"%s", temp);
        for (i = 0; i < (80 - strlen(temp)); i++) {
            printf(NONE" ");
            //fflush(stdout);
        }
        printf(NONE"\n");
    }
}
count++;
while ((record = strtok(NULL, "\r\n")) != NULL) {
    memset(temp, 0, 100);
    strncpy(temp, record, strlen(record));
    record = record + strlen(record) - 1;
    while (strncmp(record, ".", 1) != 0 && strlen(record) < 7)
        record = record - 1;
    record += 1;
    if (strcmp(record, dir) == 0) {
        printf(MENTION"%s", temp);

```

```
        for (i = 0; i < (80 - strlen(temp)); i++) {
            printf(NONE, " ");
            //fflush(stdout);
        }
        printf(NONE"\n");
    } else {
        printf(NONE"%s", temp);
        for (i = 0; i < (80 - strlen(temp)); i++) {
            printf(NONE" ");
            //fflush(stdout);
        }
        printf(NONE"\n");
    }
    count++;
    if (count == 23) {
        count = 0;
        printf(NONE"Press Enter to continue.\n");
        getchar();
        system("clear");
    }
}
printf(NONE" \n");
}
if (close(sockfd) < 0)
    printf("close error\n");
}

/* download file from ftp server */
int
ftp_get(int sck, char *pDownloadFileName_s) {
    // code here
    int nread;
    int file;
    struct timeval start;
    struct timeval end;
    struct timeval tv1;
    struct timeval tv2;
    struct timeval tv1a;
    struct timeval tv2a;
    double speed = 0.0;
    double tempSpeed = 0.0;
    double bigsize = 0.0;
    int cursize = 0;
    int percent = 0;
    int secsize = 0;
    int count = 0;
    int totalsize = 0;

    if ((file = open(pDownloadFileName_s, O_RDWR | O_CREAT)) < 0) {
        printf("Error --- Can not create file.\n");

        exit(0);
    }
    // In breakpoint resume mode, write file from the end.
    if (action == GETFILE1) {
        cursize = lseek(file, 0, SEEK_END);
    }
    memset(rbuf1, 0, MAXBUF);
    memset(wbuf, 0, MAXBUF);
    sprintf(wbuf, "Downloading file \"%s\"", pDownloadFileName_s);
```

```

if (action == GETFILE1)
    strcat(wbuf, " in BREAKPOINT RESUME mode.\n");
else
    strcat(wbuf, ".\n");
write(STDOUT_FILENO, wbuf, strlen(wbuf));
gettimeofday(&start, NULL); //Used to calculate total speed.
gettimeofday(&tv1, NULL);
gettimeofday(&tv1a, NULL);
gettimeofday(&tv2, NULL);

for (;;) { // read message
    if ((nread = recv(sck, rbuf1, MAXBUF, 0)) < 0)
        printf("recv error\n");
    else if (nread == 0) {
        gettimeofday(&end, NULL);
        break;
    }
    //write to file
    if (write(file, rbuf1, nread) != nread)
        printf("Error --- Can not send to file.\n");
    gettimeofday(&tv2, NULL);
    cursize += nread;
    totalsize += nread;
    seccsize += nread;
    percent = (int) (100 * (double) cursize / size);
    count++;
    // Calculate transport speed of the latest package.
    tempSpeed = (double) nread / 1024.0 * 1000000.0 / timeDiff(&tv1a, &tv2);
    // If speed is higher than limitation, sleep for a while.
    if (tempSpeed > dl)
        goSleep(1, tempSpeed, nread);
    gettimeofday(&tv1a, NULL);
    // Calculate and display a speed every one second.
    if ((tv1.tv_sec * 1000000 + 500000 + tv1.tv_usec) < (tv2.tv_sec * 1000000 + tv2.tv_usec)) {
        speed = 0.6 * speed + 0.4 * ((double) seccsize / 1024.0 * 1000000.0 / timeDiff(&tv1, &tv2));
        printTransfer(percent, speed, count);
        seccsize = 0;
        gettimeofday(&tv1, NULL);
    }
}

printTransfer(100, speed, count);
memset(wbuf, 0, MAXBUF);
// Calculate average speed during the whole connection.
bigsize = (double) totalsize / 1024;
speed = (double) totalsize / 1024 * 1000000 / timeDiff(&start, &end);
if (bigsize < 1000) {
    sprintf(wbuf, "\nAverage rate: %3.1f kB/s, %5.2f KB received in %6d packages.", speed, bigsize, count);
} else {
    bigsize = bigsize / 1024;
    sprintf(wbuf, "\nAverage rate: %3.1f kB/s, %5.2f MB received in %6d packages.", speed, bigsize,
count);
}
strcat(wbuf, "\n");
write(STDOUT_FILENO, wbuf, strlen(wbuf));
if (close(sck) < 0)
    printf("close error\n");
if (close(file) < 0)
    printf("close file error\n");
}

```

```
/* upload file to ftp server */
int
ftp_put(int sck, char *pUploadFileName_s) {
    // code here
    int nread;
    int file;
    struct timeval start;
    struct timeval end;
    struct timeval tv1;
    struct timeval tv2;
    struct timeval tv1a;
    struct timeval tv2a;
    double speed = 0.0;
    double tempSpeed = 0.0;
    double bigsize = 0.0;
    int filesize = 0;
    int percent = 0;
    int count = 0;
    int totalsize = 0;
    int secsize = 0;

    if ((file = open(pUploadFileName_s, O_RDWR)) < 0) {
        printf("Error --- Can not open file.\n");
        exit(0);
    }
    filesize = lseek(file, 0, SEEK_END);
    lseek(file, 0, SEEK_SET);
    if (action == PUTFILE1) {
        lseek(file, size, SEEK_SET);
    }
    memset(rbuf1, 0, MAXBUF);
    memset(wbuf, 0, MAXBUF);
    sprintf(wbuf, "Uploading file \"%s\"", pUploadFileName_s);
    if (action == PUTFILE1)
        strcat(wbuf, " in BREAKPOINT RESUME mode.\n");
    else
        strcat(wbuf, ".\n");
    write(STDOUT_FILENO, wbuf, strlen(wbuf));
    memset(wbuf, 0, MAXBUF);
    gettimeofday(&start, NULL);
    gettimeofday(&tv1, NULL);
    gettimeofday(&tv2, NULL);
    for (;;) {
        gettimeofday(&tv1a, NULL);
        //read file
        if ((nread = read(file, rbuf1, MAXBUF)) < 0)
            printf("recv error\n");
        else if (nread == 0) {
            gettimeofday(&end, NULL);
            break;
        }
        //write to socket
        if (write(sck, rbuf1, nread) != nread)
            printf("Error --- Can not send to server.\n");
        gettimeofday(&tv2, NULL);
        size += nread;
        totalsize += nread;
        secsize += nread;
        percent = (int) (100 * (double) size / filesize);
    }
}
```

```

        count++;
        tempSpeed = (double) nread / 1024 * 1000000 / timeDiff(&tv1a, &tv2);
        if (tempSpeed > ul)
            goSleep(2, tempSpeed, nread);
        gettimeofday(&tv1a, NULL);
        if ((tv1.tv_sec * 1000000 + 500000 + tv1.tv_usec) < (tv2.tv_sec * 1000000 + tv2.tv_usec)) {
            speed = 0.6 * speed + 0.4 * ((double) secsize / 1024.0 * 1000000.0 / timeDiff(&tv1, &tv2));
            printTransfer(percent, speed, count);
            secsize = 0;
            gettimeofday(&tv1, NULL);
        }
    }
    printTransfer(100, speed, count);
    fflush(stdout);
    memset(wbuf, 0, MAXBUF);

    bigsize = (double) totalsize / 1024;
    speed = (double) totalsize / 1024 * 1000000 / timeDiff(&start, &end);
    if (bigsize < 1000) {
        sprintf(wbuf, "\nAverage rate: %3.1f kB/s, %5.2f KB sent in %6d packages.", speed, bigsize, count);
    } else {
        bigsize = bigsize / 1024;
        sprintf(wbuf, "\nAverage rate: %3.1f kB/s, %5.2f MB sent in %6d packages.", speed, bigsize, count);
    }
    strcat(wbuf, "\n");
    write(STDOUT_FILENO, wbuf, strlen(wbuf));
    if (close(sck) < 0 || close(file) < 0)
        printf("close error\n");
}

/* get file type index according to input */
int typeget(char *name) {
    char *r;
    r = strstr(name, ".");
    if (r == NULL || strlen(r) == 1) return 0;
    r = r + 1;
    if ((strcmp(r, "c") == 0) || (strcmp(r, "txt") == 0) || (strcmp(r, "dat") == 0) || (strcmp(r, "ini") == 0) ||
        (strcmp(r, "cpp") == 0))
        return 1; //txt
    else if ((strcmp(r, "doc") == 0) || (strcmp(r, "ppt") == 0) || (strcmp(r, "xsl") == 0) || (strcmp(r, "wps") == 0) ||
        (strcmp(r, "pdf") == 0) || (strcmp(r, "docx") == 0) || (strcmp(r, "pptx") == 0) || (strcmp(r, "xlsx") == 0))
        return 2; //office
    else if ((strcmp(r, "png") == 0) || (strcmp(r, "bmp") == 0) || (strcmp(r, "jpg") == 0) || (strcmp(r, "gif") == 0) ||
        (strcmp(r, "tga") == 0) || (strcmp(r, "jpeg") == 0))
        return 3; //picture
    else if ((strcmp(r, "mp3") == 0) || (strcmp(r, "wav") == 0))
        return 4; //sound
    else if ((strcmp(r, "mp4") == 0) || (strcmp(r, "avi") == 0) || (strcmp(r, "wmv") == 0) || (strcmp(r, "mov") == 0) ||
        (strcmp(r, "3gp") == 0))
        return 5; //video
    else if (strcmp(r, "rar") == 0 || strcmp(r, "zip") == 0 || strcmp(r, "7z") == 0 || strcmp(r, "iso") == 0 || strcmp(r,
        "tar") == 0 || strcmp(r, "jar") == 0)
        return 6; //zip
    else
        return 0;
}

/* Return minimum value of the two inputs */
int min(int a, int b) {
    if (a < b) return a;

```





```

++\n");
printf(YELLOW"++");
printf(RED"Authors: Chao Chen 10213053 2010215110");
printf(YELLOW"++\n");
printf(YELLOW"++");
printf(RED"Lina Jin 10213071 2010215110");
printf(YELLOW"++\n");
printf(YELLOW"++");
++
++\n");
printf(YELLOW"+++++");
+++++\n");

printf(NONE"\n\n");
}

/* Print hints when listing */
void printHint() {
printf(NONE"\r*****\n");
printf(BLUE" (Image file)");
printf(NONE"-");
printf(GREEN"(Text file)");
printf(NONE"-");
printf(CYAN"(Audio file)");
printf(NONE"-");
printf(RED"(Office file)");
printf(NONE"-");
printf(YELLOW"(Video file)");
printf(NONE"-");
printf(BROWN"(Zip file)");
printf(NONE"\n*****\n");
}

/* print statistic messages when downloading or uploading */
void printTransfer(int percent, double speed, int count) {
int num;
int i;
char *a = (char *) malloc(sizeof(char)*40);
//system("clear");
num = (int) (percent * 0.01 * (80 - 35));
memset(rbuf, 0, MAXBUF);
memset(a, 0, 40);
strcat(rbuf, "\r");
if (speed < 1000)
printf(a, "[%3d%% %6.1f KB/s %6d Packages]", percent, speed, count);
else {
speed = speed / 1024;
printf(a, "[%3d%% %6.1f MB/s %6d Packages]", percent, speed, count);
}
strcat(rbuf, a);

for (i = 0; i < num; i++) {
strcat(rbuf, ">");
}
for (i = 0; i < (80 - 35 - num); i++) {
strcat(rbuf, "-");
}
write(STDOUT_FILENO, rbuf, strlen(rbuf));
}

```

```
/* calculate time duration between two time points. */
int timeDiff(struct timeval* tv1, struct timeval* tv2) {
    return (int) (abs(tv2->tv_sec - tv1->tv_sec) * 1000000 + tv2->tv_usec - tv1->tv_usec);
}

/* Delay for a while */
void goSleep(int status, double speed, int size) {
    int s;
    int v = dl;
    if (status == 2)
        v = ul;
    s = (int) (1000000 * ((double) size / 1024 / v - (double) size / 1024 / speed));
    if (s > 0)
        usleep(s);
}

/* Check whether the file name is valid */
int checkName(char *name, int index) {
    if (index == FILENAME && ((strstr(name, ".") == NULL) || (strstr(name, " ") != NULL) || (strstr(name, "\\") != NULL) || (strstr(name, "/") != NULL) || (strstr(name, "|") != NULL) || (strstr(name, "?") != NULL) || (strstr(name, "*") != NULL) || (strstr(name, "<") != NULL) || (strstr(name, ">") != NULL)))
        return 0;
    else if (index == DIRECTORYNAME && ((strstr(name, ".") != NULL) || (strstr(name, " ") != NULL) || (strstr(name, "\\") != NULL) || (strstr(name, "/") != NULL) || (strstr(name, "|") != NULL) || (strstr(name, "?") != NULL) || (strstr(name, "*") != NULL) || (strstr(name, "<") != NULL) || (strstr(name, ">") != NULL)))
        return 0;
    else return 1;
}

//open port for active mode

int active(int sockfd) {
    int client_port, get_sock, set;
    int opt = SO_REUSEADDR;
    struct timeval outtime;
    struct sockaddr_in local;

    //char *local_ip=(char *)malloc(sizeof(char)*24);

    char local_ip[24];
    char *ip_1, *ip_2, *ip_3, *ip_4;

    int addr_len = sizeof (struct sockaddr);
    memset(local_ip, 0, 24);

    srand((unsigned) time(NULL));
    client_port = rand() % 40000 + 1025;
    if ((get_sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\nError --- Failed in socket().\n");
        exit(0);
    }

    outtime.tv_sec = 7;
    outtime.tv_usec = 0;

    if (setsockopt(get_sock, SOL_SOCKET, SO_RCVTIMEO, &outtime, sizeof (outtime)) != 0 || setsockopt(get_sock, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof (opt)) != 0) {
        printf("\nError --- Failed in setsockopt().\n");
        exit(0);
    }
}
```

```
}

bzero(&local_host, sizeof (local_host));
local_host.sin_family = AF_INET;
local_host.sin_port = htons(client_port);
local_host.sin_addr.s_addr = htonl(INADDR_ANY);

bzero(&local, sizeof (struct sockaddr));
while (1) {
    if (bind(get_sock, (struct sockaddr*) &local_host, sizeof (local_host)) != 0 && errno == 11) {
        srand((unsigned) time(NULL));
        client_port = rand() % 40000 + 1025;
        continue;
    }
    break;
}
if (listen(get_sock, 1) != 0) {
    printf("\nError --- Failed in listen().\n");
    exit(0);
}
if (getsockname(sckfd, (struct sockaddr*) &local, (socklen_t *) &addr_len) < 0) {
    printf("\nError --- Failed in getsockname().\n");
    exit(0);
}
snprintf(local_ip, sizeof (local_ip), "%s", inet_ntoa(local.sin_addr));

ip_1 = local_ip;
ip_2 = strchr(local_ip, '.');
*ip_2 = '\0';
ip_2++;
ip_3 = strchr(ip_2, '.');
*ip_3 = '\0';
ip_3++;
ip_4 = strchr(ip_3, '.');
*ip_4 = '\0';
ip_4++;
memset(wbuf, 0, MAXBUF);
sprintf(wbuf, "PORT %s,%s,%s,%s,%d,%d\n", ip_1, ip_2, ip_3, ip_4, client_port >> 8, client_port & 0xff);
return get_sock;
}
```