Project report

Design and implementation of FTP Server

Course Title: *Internet Application*

Name: *Zhang Meng (09212905)*

Zhang Mincong (09212915)

Date: **2012.6.1**

1. Overview

The requirement and target of the project is introduced:

- **Subject:**

FTP Server

- **Requirements**

Understand FTP protocol

Make socket programming in Linux

- **Goals**

Implement an FTP server program

Record and display related information

Limit user's access rights as the rules specified

2. Requirements Analysis

The requirement of the project is analyzed here, including:

- ## Development Environment, such as OS and programming language

We use C++ in Linux, using Ubuntu. We install the Vm virtual machine without graphic interface as the environment with the C++ language in Linux operating system.

Socket is an extension to OS's I/O system, enabling communication between processes and machines. It is a mechanism for exchanging data between processes. These processes can either be on the same machine, or on different machines connected via a network. Once a socket connection is established, data can be sent in both directions until one of the endpoints closes the connection. So we need to construct sockets and create a TCP service.

- ## Functional requirements in details, including task decomposition and analysis

Basic functions (imperative):

Multiple users can login simultaneously.

Support commands such as: PWD, CWD, LIST, MDIR, DELE, RNFR/RNTO.

Provide download and upload services in active mode.

Display user's info such as: username, IP, actions, speed, total traffic.

Advanced functions (optional):

Provide download and upload in passive mode.

Limit download and upload speed as specified.

Limit users in their own home directories with specified access rights.

Analysis:

According to the given template, we find that after constructing the socket, the task is to fulfill the FTP functions, such as PWD, CWD, LIST, MDIR, DELE, RNFR/RNTO. And these functions can be divided into two parts——manipulating files and transferring files. So we decompose the task and Zhang Meng tries to cover the transferring files functions while Zhang Mincong try to cover the manipulating files functions.

As for the connection, first, the server creates a listening socket, and waits for connection attempts from clients. The client creates a socket on its side, and attempts to connect with the server. The server then accepts the connection, and data exchange can begin. Once all data has been passed through the socket connection, either endpoint can close the connection.

3. Preliminary Design

Preliminary design includes:

· Decomposition of functional modules

Apart from the basic module: creating server socket, we list the following FTP functional modules:

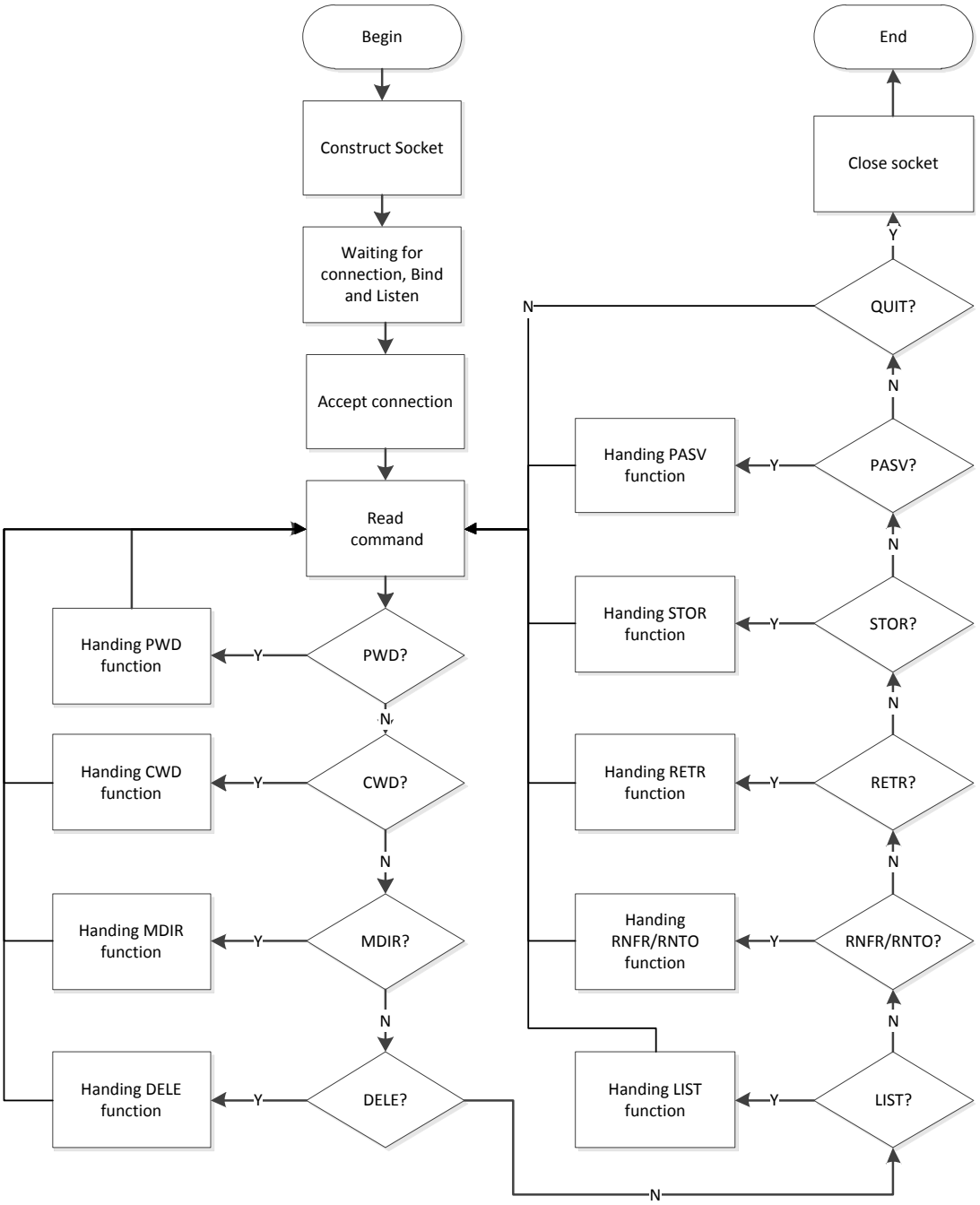
USER PASS PWD CWD CDUP MKD DELE RNFR/RNTO LIST RETR STOR PASV QUIT.

Besides we design the functions such as send_file(), upload_file() and send_list(), which are needed for a better performance.

· Relationship and interface between the modules

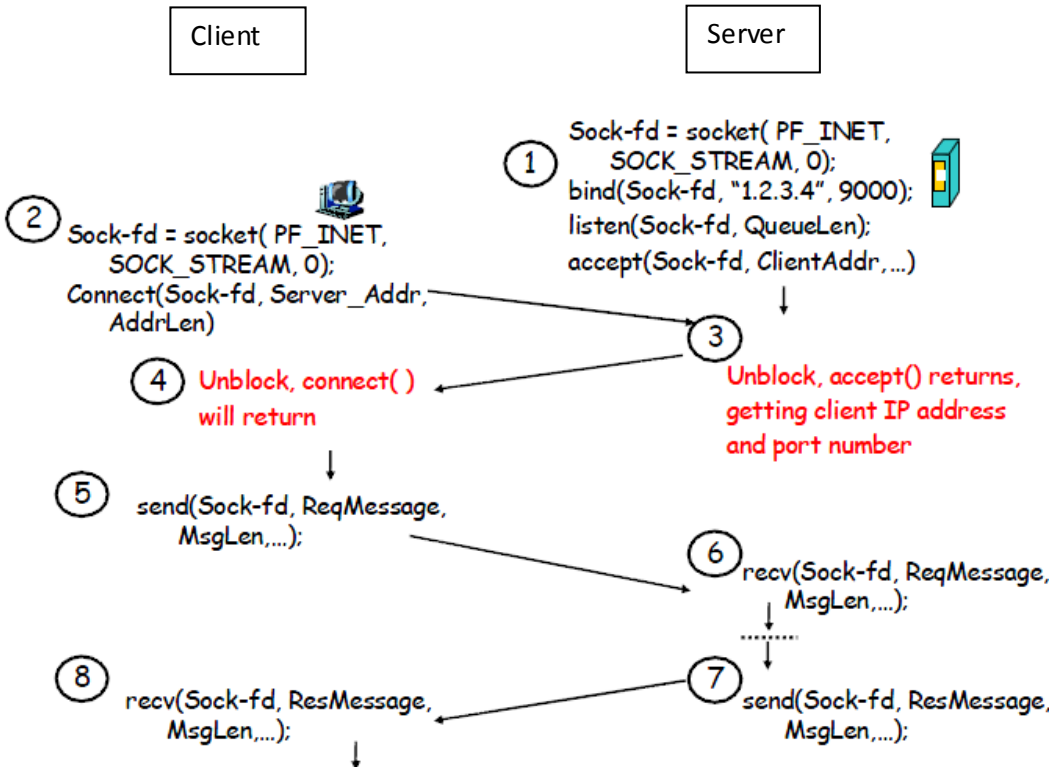
PASS command must behind USER command, after verifying the username. PORT and PASV must be first executed when LIST, RETR, STOR command is needed. Besides, RNFR command must read before RNTO command. And the rest of the FTP functional modules are parallel, which means they don’t affect others.

· Overall flow chart

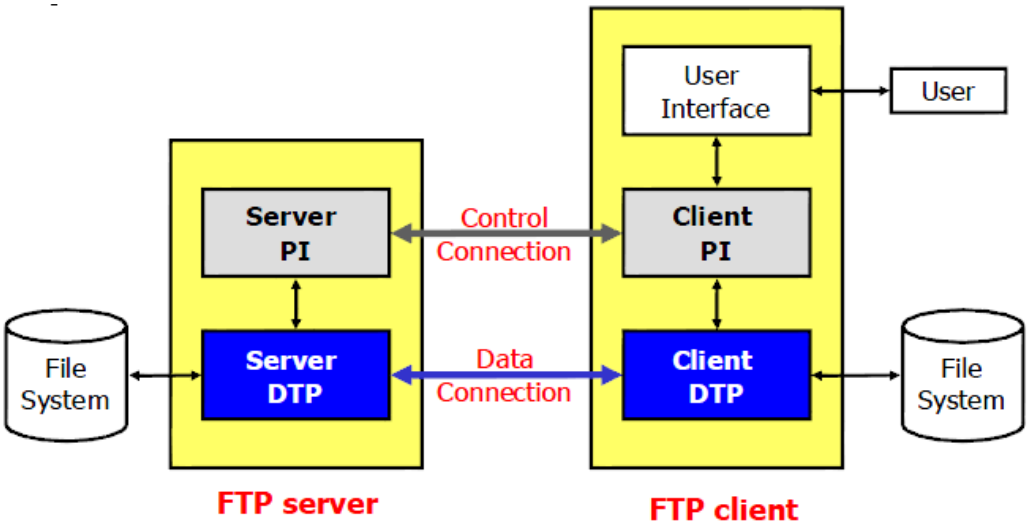


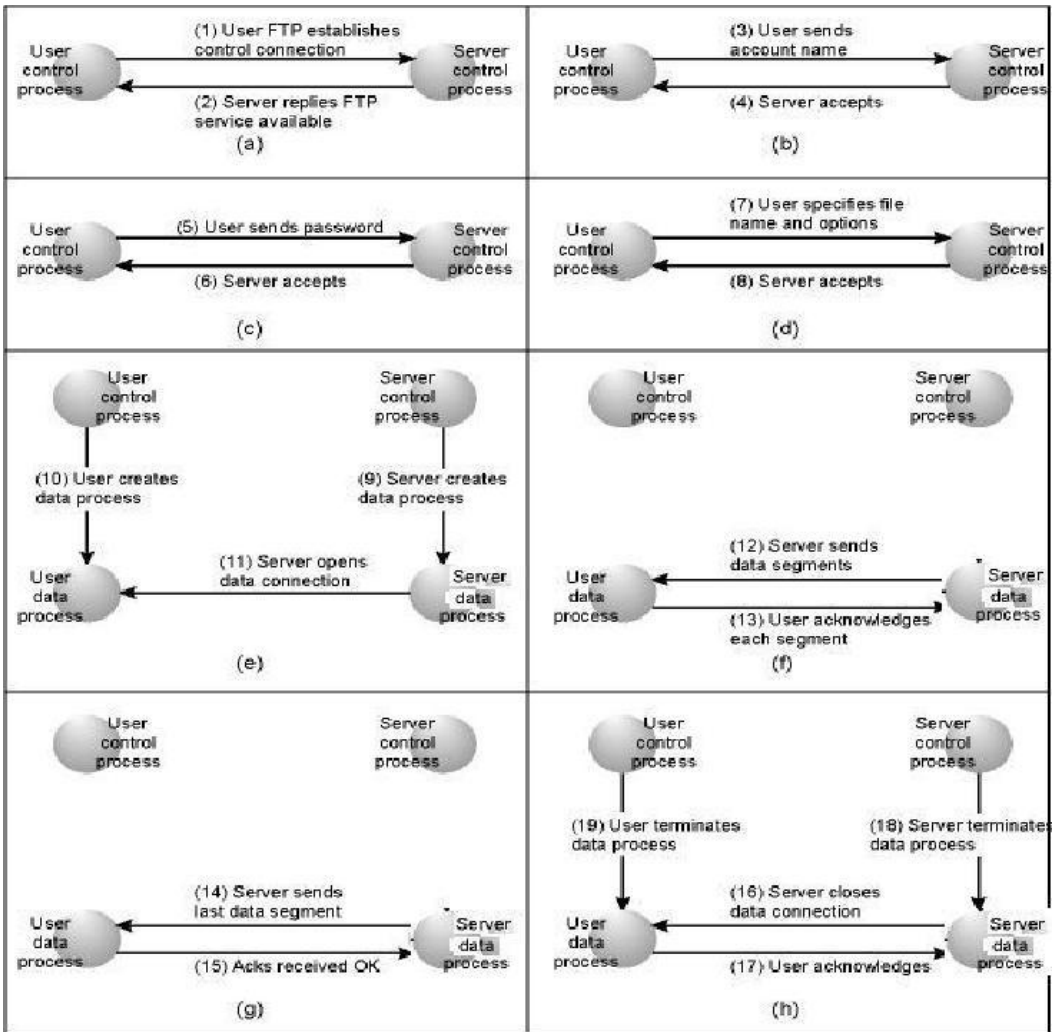
· Design of data structures

First we will introduce the structure of the TCP connection, which is used in FTP for reliable transfers.



And the FTP Model is as follow:





.

4. Detailed Design

Detailed design includes

· Design analysis of each module

For the FTP function, after using sscanf(), we can store the input data into a buffer and judge them and find if they fit the following commands.

USER: We can use the strcmp() to judge whether the input is fit with the users database.

PASS: We can use the strcmp() to judge whether the input is fit with the passwords database.

PWD: getcwd() function can get the current working directory.

CWD: After searching the linux API we find that chdir() fit this command, after using this, getcwd() can get the current working directory.

CDUP: We find that this command is similar to CWD and we store ".." in the buffer and chdir() can make it.

MKD: mkdir() function can fit this command.

DELE: remove()function can fit this command.

RNFR/RNTO: We store the name in the tempBuf and it can be easily done.

LIST: This command involves active or passive connect mode and send_list() function, which we will fulfill later.

RETR: For retrieving (getting) files, we set a send_file() function to make it.

STOR: For storing (putting) files, we set a upload_file() function to make it.

PASV: this command can make server listens to a specific port and client should access that port.

QUIT: close() is used and fd should be clear.

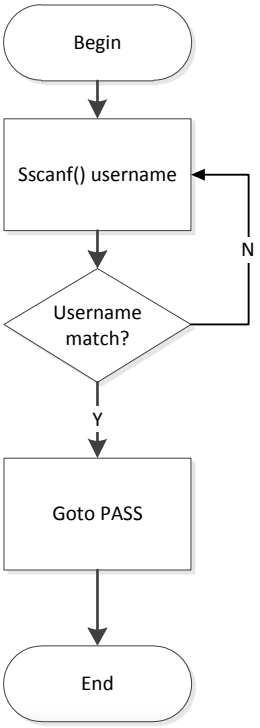
send_file():After connection(passive or active), we can open the very file and read it into the buffer, which can be sent using socket.

upload_file():After connection(passive or active), we can open the very file and read it into the buffer, which can be uploaded using socket.

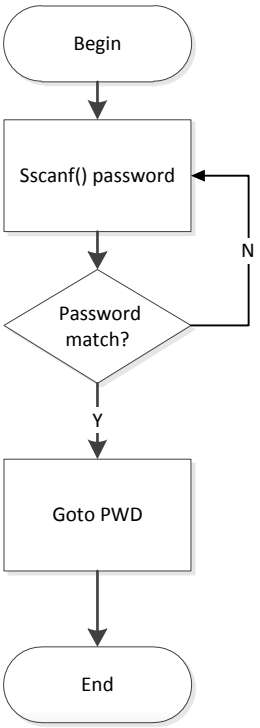
send_list():Current directory is read by using readdir(), which can be sent to client after data connection.

- **Flow chart of each module**

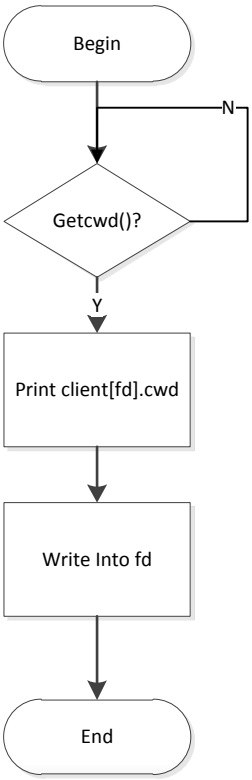
USER:



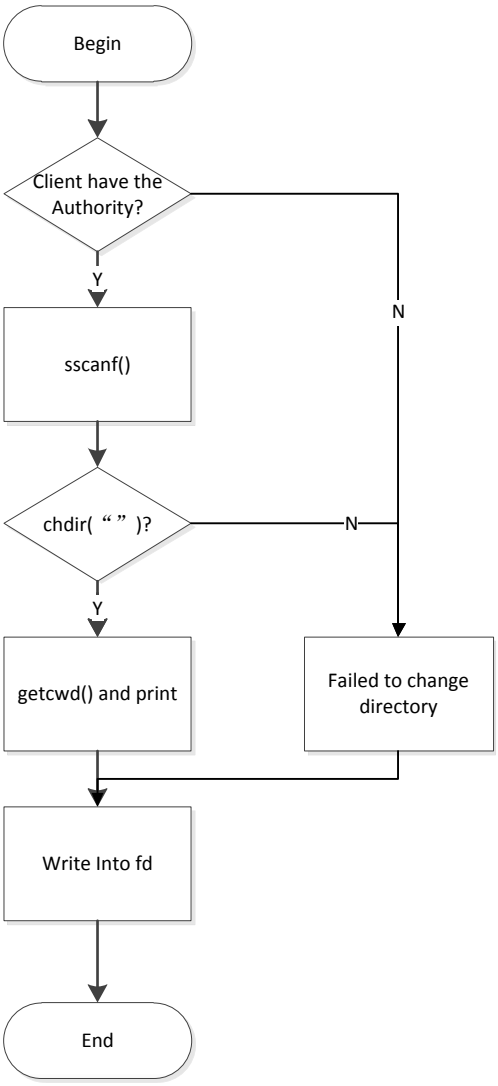
PASS:



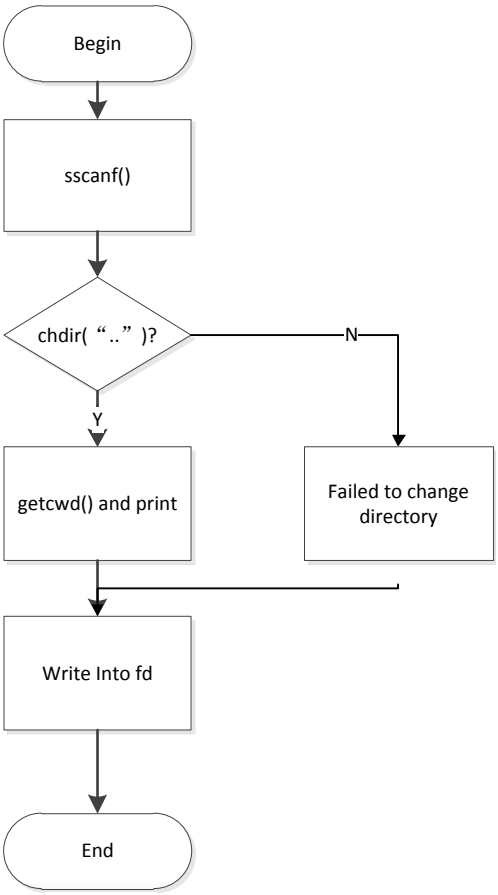
PWD:



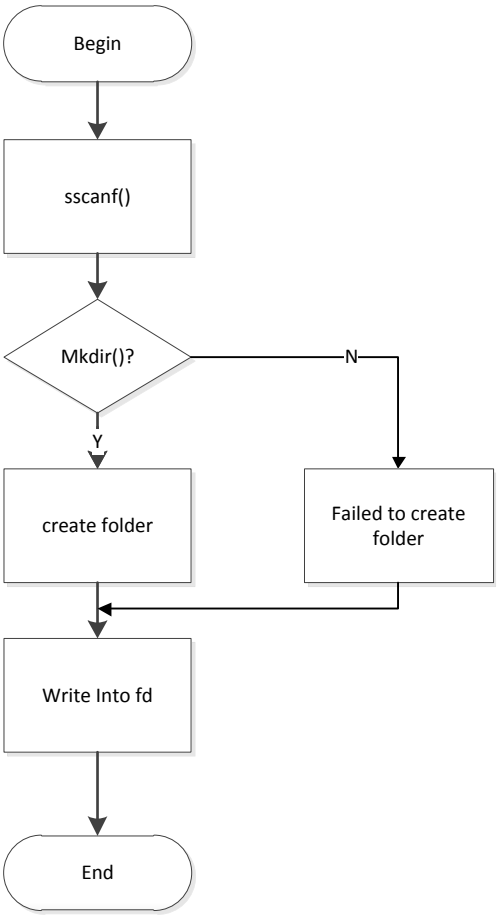
CWD:


Begin
Client have the Authority?
Y
N
sscanf()
chdir(“ ”)?
N
Y
getcwd() and print
Failed to change directory
Write Into fd
End

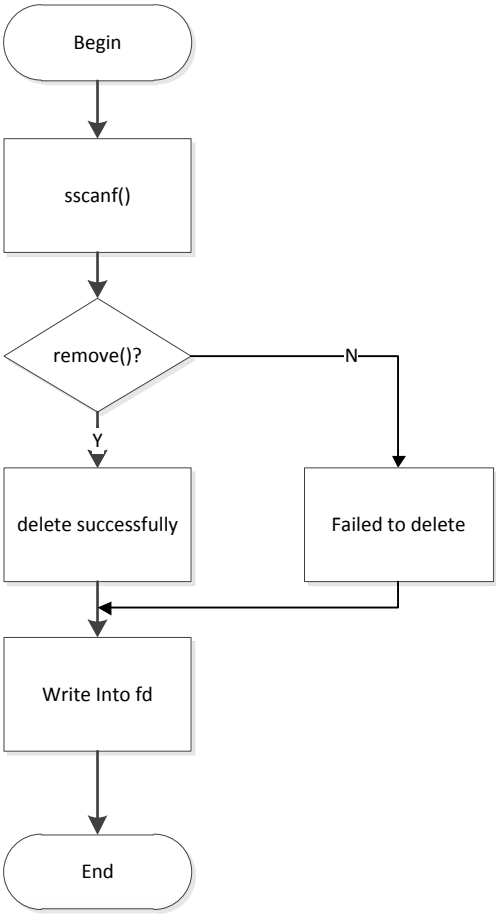
CDUP:


Begin
sscanf()
chdir(“..”)?
N
Y
getcwd() and print
Failed to change directory
Write Into fd
End

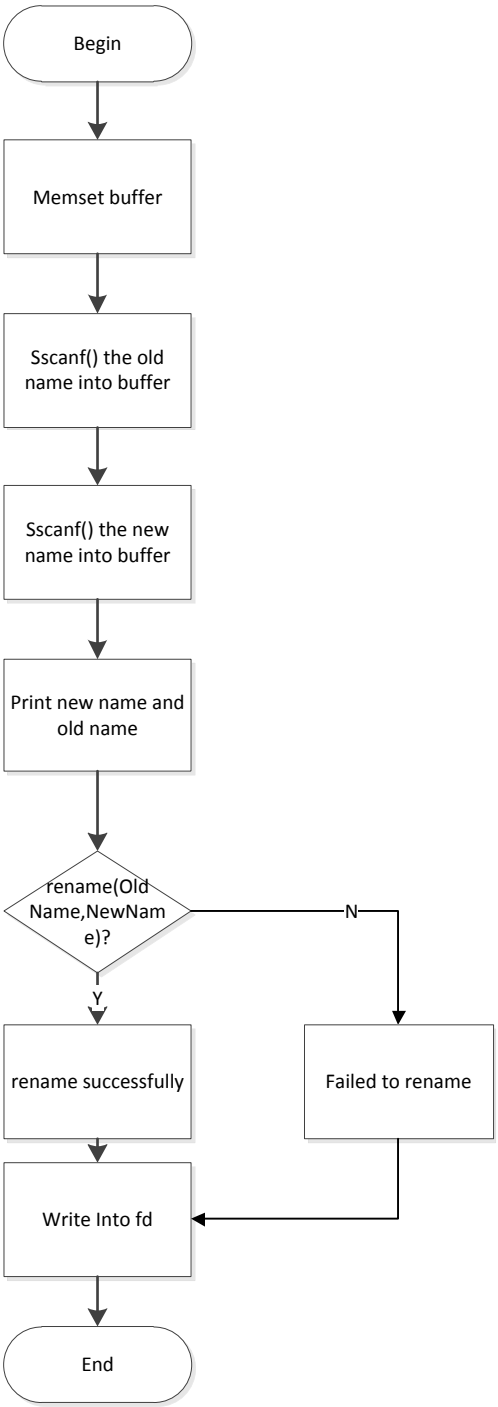
MKD:


Begin
sscanf()
Mkdir()?
N
Y
create folder
Failed to create folder
Write Into fd
End

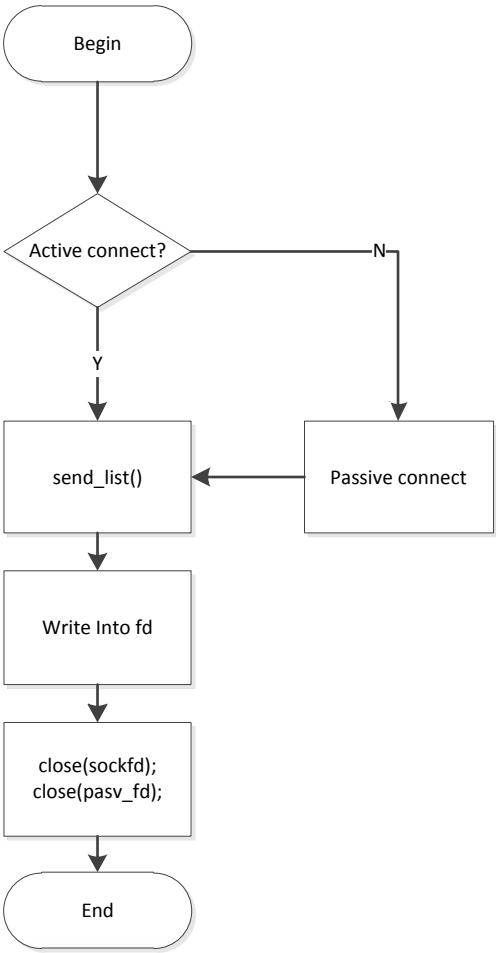
DELE:



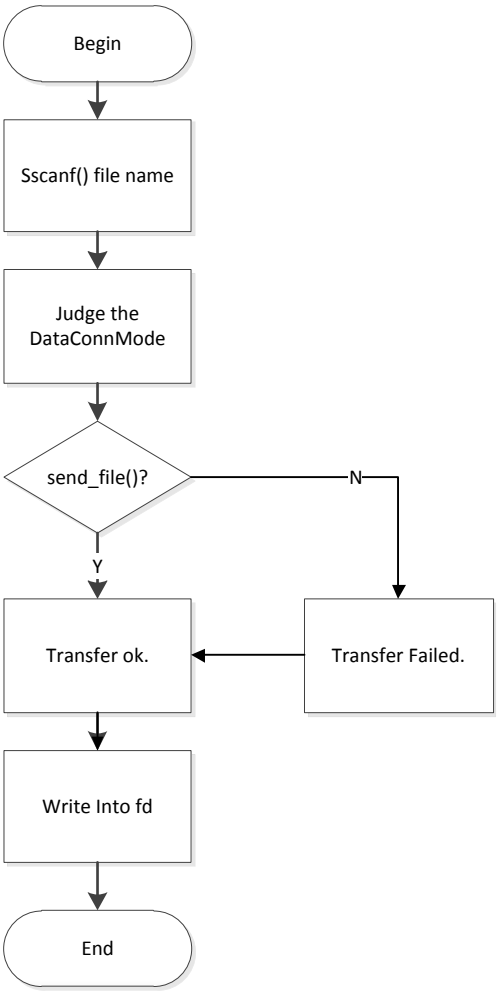
RNFR/RNTO:


Begin
Memset buffer
Sscanf() the old name into buffer
Sscanf() the new name into buffer
Print new name and old name
rename(Old Name,NewNam e)?
N
Y
rename successfully
Failed to rename
Write Into fd
End

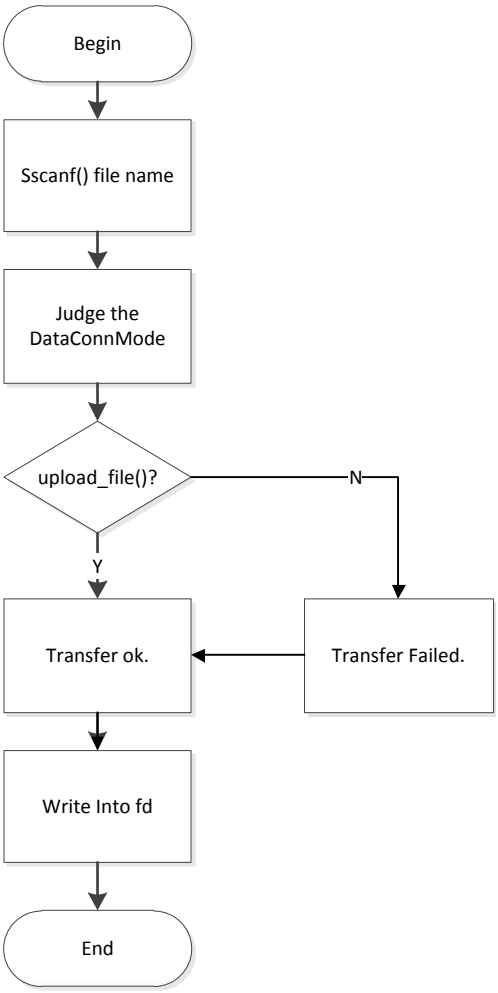
LIST:


Begin
Active connect?
N
Y
send_list()
Passive connect
Write Into fd
close(sockfd);
close(pasv_fd);
End

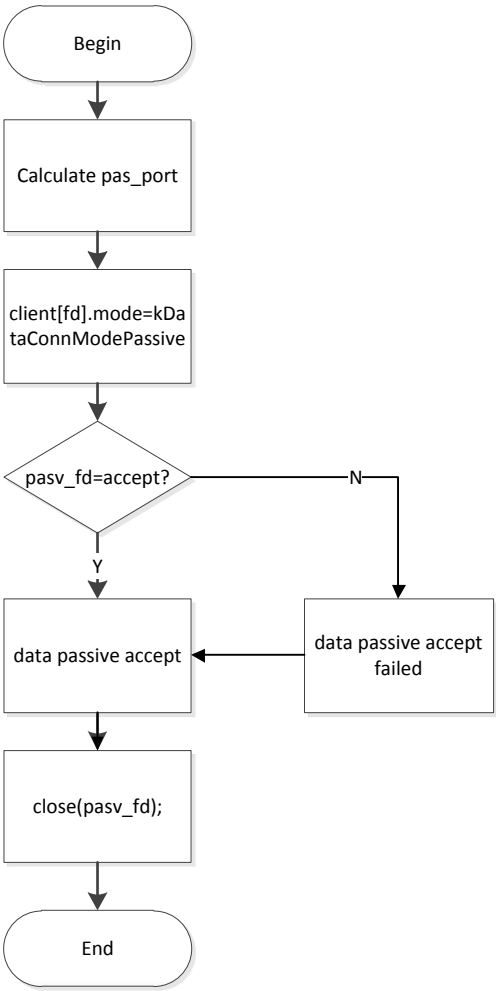
RETR:


Begin
Sscanf() file name
Judge the DataConnMode
send_file()?
N
Y
Transfer ok.
Transfer Failed.
Write Into fd
End

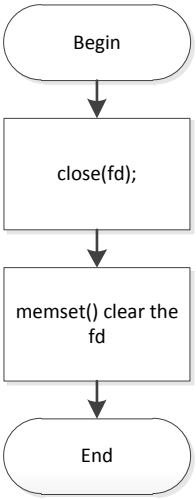
STOR:


Begin
Sscanf() file name
Judge the DataConnMode
upload_file()?
N
Y
Transfer ok.
Transfer Failed.
Write Into fd
End

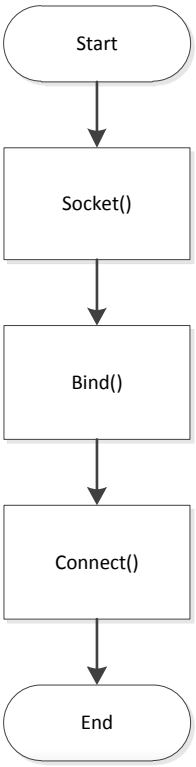
PASV:



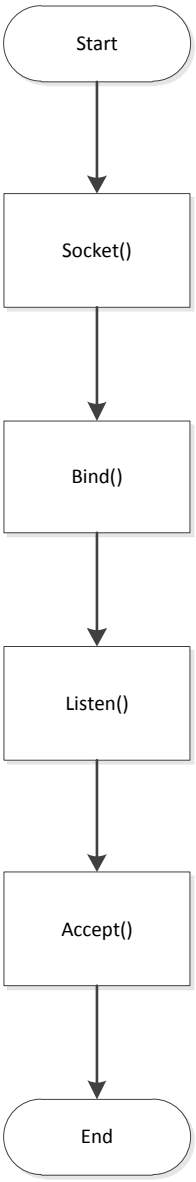
QUIT:



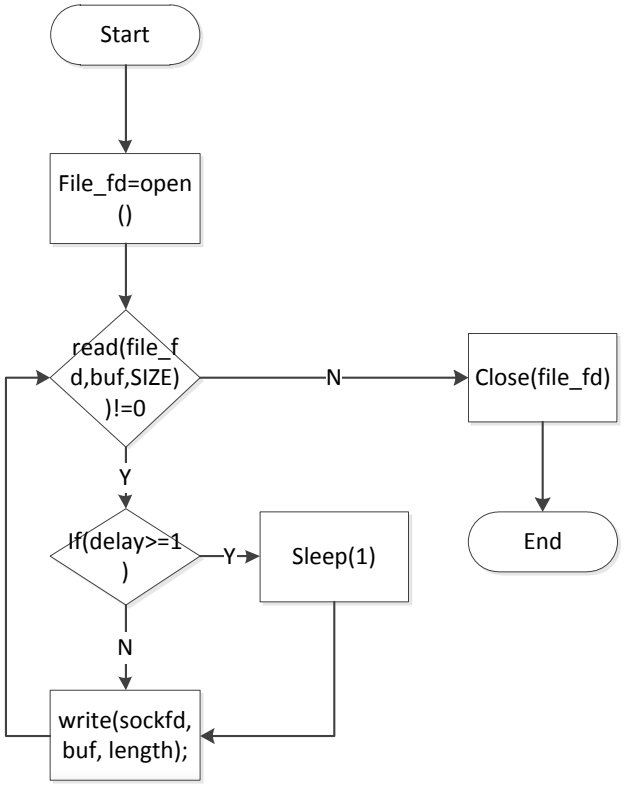
data_active_connection():



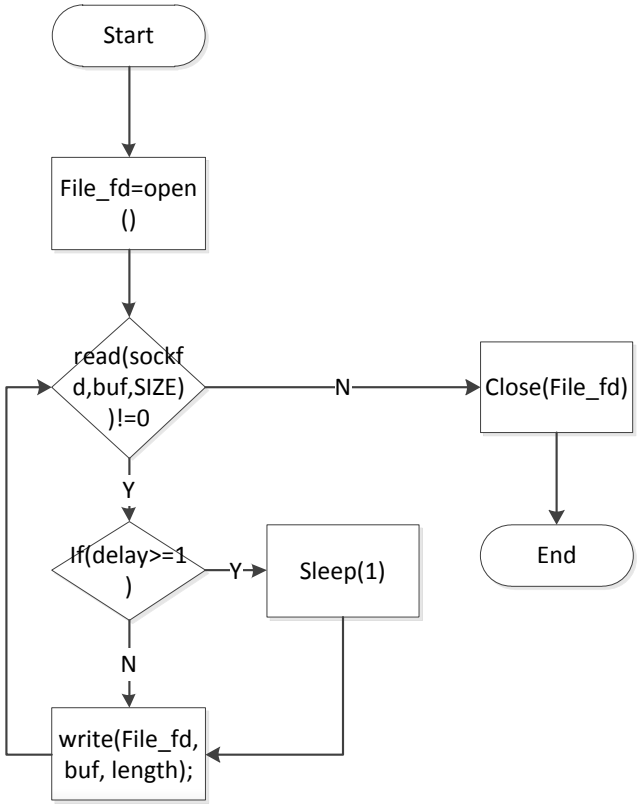
data_passive_connection():



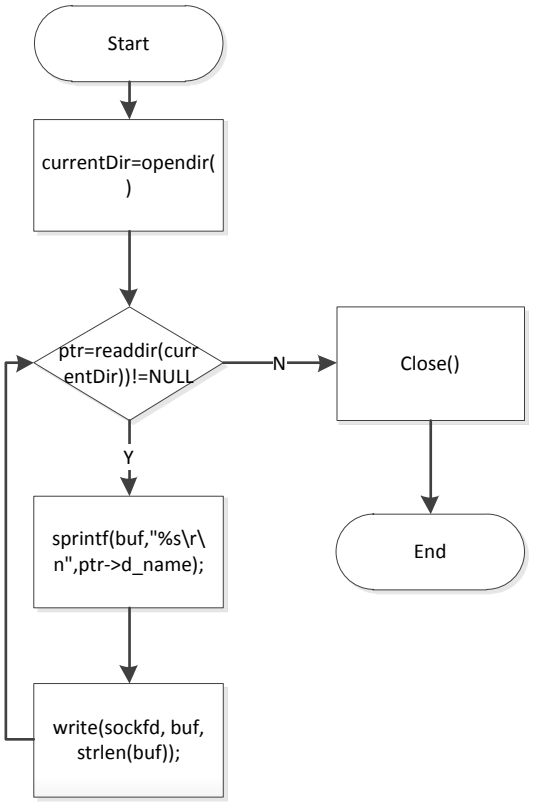
send_file():



upload_file():



send_list():



5. Results

Show the execution results with Screen Shot.

- **Basic functions**

1. Start server

```
root@IA:/mnt/hgfs/share-win# gcc demo6.c -o demo6
root@IA:/mnt/hgfs/share-win# ./demo6 2000
The max speed is 2000 byte/s
server waiting
Handling client: 36.179.123.183
```

2. A client connects to server

Client:

```
is09@IA:/mnt/hgfs/share-win/zmc$ ftp 127.0.0.1
Connected to 127.0.0.1.
220 (wxftp 1.0)
Name (127.0.0.1:is09): is09
331 Password required for is09.
Password:
230 Login successfully
Remote system type is Login.
ftp> _
```

Server:

```
adding client on fd 4
server waiting
Handling client: 127.0.0.1

serving client on fd 4: USER is09

server waiting
Handling client: 127.0.0.1

serving client on fd 4: PASS bupt

server waiting
Handling client: 127.0.0.1

serving client on fd 4: SYST

server waiting
Handling client: 127.0.0.1
```

3. Multiple users can login simultaneously

Client:

```
Name (127.0.0.1:is09): is09
331 Password required for is09.
Password:
230 Login successfully
Remote system type is Login.
ftp>
```

```
is09@IA:/mnt/hgfs/share-win/zmc$ ftp 127.0.0.1
Connected to 127.0.0.1.
220 (wxftp 1.0)
Name (127.0.0.1:is09): admin
331 Password required for admin.
Password:
230 Login successfully
Remote system type is Login.
```

Server:

```
serving client on fd 5: SYST

Handling client: 127.0.0.1

The client is is09.

server waiting


serving client on fd 4: SYST

Handling client: 127.0.0.1

The client is admin.

server waiting
```

4. A client using PWD

Client:

```
ftp> pwd
257 /mnt/hgfs/share-win
```

Server:

```
serving client on fd 4: PWD

/mnt/hgfs/share-win

server waiting
Handling client: 127.0.0.1

_
```

5. A client using CWD

Client

```
ftp> cd ..
   250 Directory successfully changed. /mnt/hgfs/share-win
ftp> cd zmc
   250 Directory successfully changed. /mnt/hgfs/share-win/zmc
ftp> _
```

Server

```
serving client on fd 4: CWD zmc

/mnt/hgfs/share-win/zmc

server waiting
Handling client: 127.0.0.1
```

6. A client using CDUP

Client

```
ftp> pwd
257 /mnt/hgfs/share-win/test
ftp> cdup
250 /mnt/hgfs/share-win
ftp> _
```

Server

```
serving client on fd 4: PWD

/mnt/hgfs/share-win/test

server waiting
Handling client: 127.0.0.1

serving client on fd 4: CDUP

/mnt/hgfs/share-win

server waiting
Handling client: 127.0.0.1
```

7. A client using LIST

Client

```
ftp> ls
200 Port command successful.
150 Opening data connection for directory list.
a
c.txt
Test.txt
226 Transfer ok.
```

Server

```
serving client on fd 4: PORT 127,0,0,1,134,155

port is executed.
LOG:     client PORT output:127,0,0,1,134,155
server waiting
Handling client: 127.0.0.1

serving client on fd 4: LIST

data_conn_active is executed.
send_list is executed.
server waiting
Handling client: 127.0.0.1
```

8. A client using MKD

Client

```
ftp> mkd
(directory-name) newfolder
250 create folder newfolder successfully!
ftp> pwd
257 /mnt/hgfs/share-win/test
ftp> ls
200 Port command successful.
150 Opening data connection for directory list.
a
c.txt
newfolder
Test.txt
226 Transfer ok.
ftp> _
```

Server

```
serving client on fd 4: MKD newfolder

server waiting
Handling client: 127.0.0.1

_
```

9. A client using RETR

Client

```
ftp> get test2.txt
local: test2.txt remote: test2.txt
200 Port command successful.
150 Opening data connection for directory list.
226 Transfer ok.
4707 bytes received in 3.04 secs (1.5 kB/s)
ftp> _
```

Server

```
serving client on fd 4: RETR test2.txt

RETR is executed by 4.
data_conn_active is executed.
send_file is exexuted.
test2.txt is sent to the fd 5
total transmission: 4707 bytes
transmission time: 3.000000 second(s)
speed: 1569.000000 byte(s)/second(s)
server waiting
Handling client: 127.0.0.1
```

10. A client using STOR

Client

```
ftp> put test3.txt
local: test3.txt remote: test3.txt
200 Port command successful.
150 Opening data connection for directory list.
226 Transfer ok.
4707 bytes sent in 0.09 secs (53.8 kB/s)
ftp> _
```

Server

```
serving client on fd 4: PORT 127,0,0,1,165,21

port is executed.
LOG:     client PORT output:127,0,0,1,165,21
server waiting
Handling client: 127.0.0.1

serving client on fd 4: STOR test3.txt

data_conn_active is executed.
upload_file is exexuted.
test3.txt has been stored on the server from 5 fd
total transmission: 4707 bytes
transmission time: 0.000000 second(s)
speed: inf byte(s)/second(s)
server waiting
Handling client: 127.0.0.1
```

11. A client using RNFR&RNTO

Client

```
ftp> rename
(from-name) test4.txt
(to-name) test5.txt
350 Ready for RNTO.
250 rename successfully!
ftp> _
```

Server

```
serving client on fd 4: RNFR test4.txt

test4.txt
server waiting
Handling client: 127.0.0.1

serving client on fd 4: RNTO test5.txt

test4.txt
test5.txt
server waiting
Handling client: 127.0.0.1
```

12. A client using DELE

Client

```
ftp> dele
(remote-file) test2.txt
250 delete test2.txt successfully!
ftp> _
```

Server

```
serving client on fd 4: DELE test2.txt

server waiting
Handling client: 127.0.0.1
```

13. A client disconnect
Client

```
ftp> close
221 Goodbye.
```

```
ftp> quit
221 Goodbye.
```

Server

```
serving client on fd 4: QUIT

removing client on fd 4
server waiting
Handling client: 127.0.0.1
```

· Advanced functions

1. Provide download and upload in passive mode.

A client using RETR
Client

```
ftp> passive
Passive mode on.
ftp> get 120525.c
local: 120525.c remote: 120525.c
227 Entering Passive Mode (192,168,116,128,23,113).
150 Opening data connection for directory list.
226 Transfer ok.
20546 bytes received in 21.05 secs (1.0 kB/s)
ftp> _
```

Server

```
PASV is executed.
data_coon_pasive is exexuted.
pasv_fd is 5
server waiting



serving client on fd 4: RETR 120525.c

Handling client: 127.0.0.1

The client is admin.

RETR is executed by 4.
send_file is exexuted.
120525.c is sent to the fd 6
total transmission: 20546 bytes
transmission time: 21.000000 second(s)
speed: 978.380952 byte(s)/second(s)
server waiting
```

A client using STOR
Client

```
ftp> put 120525.txt
local: 120525.txt remote: 120525.txt
227 Entering Passive Mode (192,168,116,128,23,114).
150 Opening data connection for directory list.
226 Transfer ok.
22415 bytes sent in 0.08 secs (277.0 kB/s)
ftp> _
```

Server

```
The client is admin.

PASV is executed.
data_coon_pasive is exexuted.
pasv_fd is 6
server waiting



serving client on fd 4: STOR 120525.txt

Handling client: 127.0.0.1

The client is admin.

upload_file is exexuted.
120525.txt has been stored on the server from 7 fd
total transmission: 22415 bytes
transmission time: 23.000000 second(s)
speed: 974.565217 byte(s)/second(s)
server waiting
```

2. Limit download and upload speed as specified.
Set up the limited speed at Server

```
root@IA:/mnt/hgfs/share-win# ./demo6 2000
The max speed is 2000 byte/s
server waiting
Handling client: 36.147.114.183
```

A client using RETR
Client

```
ftp> get test2.txt
local: test2.txt remote: test2.txt
200 Port command successful.
150 Opening data connection for directory list.
226 Transfer ok.
4707 bytes received in 3.04 secs (1.5 kB/s)
ftp> _
```

Server

```
serving client on fd 4: RETR test2.txt

RETR is executed by 4.
data_conn_active is executed.
send_file is exexuted.
test2.txt is sent to the fd 5
total transmission: 4707 bytes
transmission time: 3.000000 second(s)
speed: 1569.000000 byte(s)/second(s)
server waiting
Handling client: 127.0.0.1
```

A client using STOR
Client

```
ftp> put test3.txt
local: test3.txt remote: test3.txt
200 Port command successful.
150 Opening data connection for directory list.
226 Transfer ok.
4707 bytes sent in 0.09 secs (53.8 kB/s)
ftp> _
```

Server

```
serving client on fd 4: PORT 127,0,0,1,165,21

port is executed.
LOG:     client PORT output:127,0,0,1,165,21
server waiting
Handling client: 127.0.0.1

serving client on fd 4: STOR test3.txt

data_conn_active is executed.
upload_file is exexuted.
test3.txt has been stored on the server from 5 fd
total transmission: 4707 bytes
transmission time: 0.000000 second(s)
speed: inf byte(s)/second(s)
server waiting
Handling client: 127.0.0.1
```

3. Limit users in their own home directories with specified access rights.

Limit: (We judge that clients can only be in the main directory and cannot enter other directory, so if a client wants to change the directory, failure will occur.)

Client

```
ftp> pwd
257 /mnt/hgfs/share-win
ftp> cd zmc
550 Failed to change directory.
ftp> _
```

Server

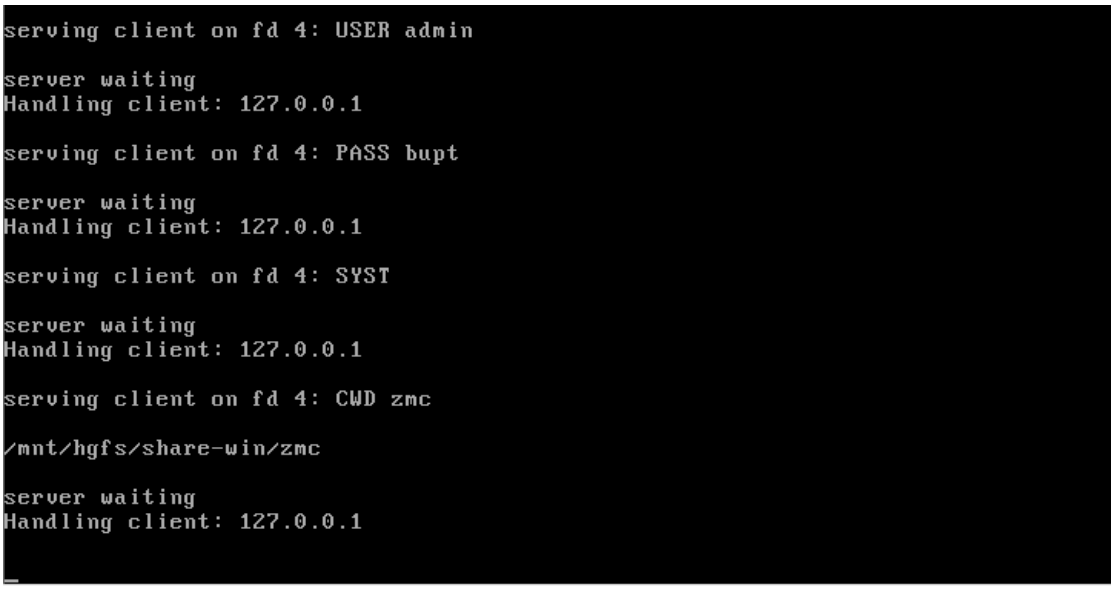
```
serving client on fd 4: CWD zmc

server waiting
Handling client: 127.0.0.1
```

Unlimit: (We judge that admin can go to any directory he/she wants.)

Client

```
ftp> user
(username) admin
331 Password required for admin.
Password:
230 Login successfully
Remote system type is Login.
ftp> cd zmc
250 Directory successfully changed. /mnt/hgfs/share-win/zmc
```

Server



6. Summary and Conclusion

Specify how you divide jobs with your partner and your jobs in the project. Self-evaluation of your work is given here, including future improvement.

divide jobs:

Zhang Meng covered the transferring files functions while Zhang Mincong covered the manipulating files functions. We work together to finish the target.

Self-evaluation:

Zhang Mincong:

At the first glace of the given template, I found it difficult to understand some of its functions, because that we are not familiar with the network programming. Although we have just finished the sockets programming, finishing the FTP is still a tough work for us. So we divide our jobs and I went to search the Internet for some solution, finding that although there are lots of code for FTP, few of them is suitable for our learning of FTP——they may too difficult to understand or not suitable for our learning.

But after searching, I can understand the template well. So I borrow a book named API of C in Linux from Zhang Meng and try to cover the PWD and CWD, at the meantime CDUP is easy to solve. The API made a huge contribution for our coding, for the reason that we find it provide great number of function which we can directly use in the coding. However, some days after that the TA told us it is not a standard way to construct the FTP, because a standard FTP involves lots of processes and using functions like chdir() will affect the server directories. It is really a shock. Struggling for a couple of days we decide to continue the job——finding a new way is tough and we have no time to cover a new method (our course Software Engineering show us that it is not good to try a new way since risks will often happen). We try to remedy the problem and the solution is to add chdir(topDir) at the bottom of the loop. For the solution my teammate contributes a lot. He try every method to overcome the problem, besides that he finished the Advanced functions.

Of course I learnt a lot in Internet Application, finding that it is still a long way for me to have a better performance in the future.

Zhang Meng:

Time flies, during the last four weeks, we worked together for the same goal. It is a valuable experience for me to improve myself and learn from my teammate. As for me, I worked very hard to finish the FTP server.

At first, I spent much time review the course contents and read the materials for Linux programming. Since then, I always finished my work before the time and happy to help my teammate.

There are some contingencies during our process. For example, when we finish out the part of CWD, we found that the used of chdir() would affect other user's current path, then we discussed together and decided to change the structure of our code, finally we solved it.

From this lab, I have learnt a lot of knowledge about Internet Application. It helps me understand the socket programming and the structure and flow chart of FTP. I appreciate the help my teammate, the lecturers and teaching assistants.

Future improvement:

1. Multiple processes available
2. Code structure is more reasonable
3. Dedicated functions are collected into specific library
4. More functions will be supported

Appendix: Source Codes

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <dirent.h>
#include <time.h>

// Maximum number of clients
#define MAX_CLIENT 128

//Root account shall be used for the following 2 ports
#define SERVER_CMD_PORT 21
#define SERVER_DAT_PORT 20
#define PAS_PORT 6000

#define SIZE 128

//Mode to set up data connections, Active or Passive  枚举
enum data_conn_mode {
        kDataConnModeActive, kDataConnModePassive
};
int pas_port=PAS_PORT;
int max_speed;

struct sockaddr_in servAddr; /* Local Server address */

//Structure to store client information: name, IP address, ...
struct Client {
        char user[128];
        char pass[128];
        unsigned short data_port;
        struct in_addr sin_addr;  //IP address
        enum data_conn_mode mode; //mode
        char cwd[128];
        int power;
        double traffic;
        double speed;
        //Other information
};

char username[]="is09";
char admin[]="admin";
char password[]="bupt";
char topDir[SIZE];
//Used to listen for command connection;
int server_sockfd;
//Used to store the socket-fd returned by accept();
int client_sockfd;

/*Array of client structure, client_socketfd is used as index.
For example, if client_socketfd of a client is 4, then the structure of client[4] stores his information. */
struct Client client[MAX_CLIENT];

//Creat a socket for server, server_sockfd is given a value
int create_server_socket();

// Set up data connection in Active mode
```

```
int data_conn_active(int fd, struct in_addr *sin_addr, unsigned short port);

// Reaction of command LIST or NLST: Sending directory and file list;
// dir is an absolute path
int send_list(int sockfd, char *dir);

//Reaction of command RETR
int send_file(int sockfd, char *fileName);

//Reaction of command STOR
int upload_file(int sockfd, char *fileName);
//Set up data connection in Passive mode
int data_conn_passive(unsigned short port);

int main(int argc, char *argv[])
{
	int client_len;
	if(argc==2){
		max_speed=atoi(argv[1]);
		printf("The max speed is %d byte/s\n",max_speed);
	}
	else max_speed=1000;
	int pasv_fd;
	struct sockaddr_in client_address; //此数据结构用做 bind、connect、recvfrom、sendto 等函数的参数，指明地址信息。
	int result;
	char tempBuf[SIZE]="";
	fd_set readfds, testfds; // a set of sockets to be monitored

	//create a socket and server_sockfd will be set.
	create_server_socket();

	FD_ZERO(&readfds);
	FD_SET(server_sockfd, &readfds); // add server_sockfd to fd_set

	/*
   FD_ZERO(fd_set   *fdset);将指定的文件描述符集清空，在对文件描述符集合进行设置前，必须对其进行初始化，如果不清空，由于在系统分配内存空间后，通常并不作清空处理，所以结果是不可知的。
	FD_SET(fd_set   *fdset);用于在文件描述符集合中增加一个新的文件描述符。
	FD_CLR(fd_set   *fdset);用于在文件描述符集合中删除一个文件描述符。
	FD_ISSET(int   fd,fd_set   *fdset);用于测试指定的文件描述符是否在该集合中。
   */

	char currentDir[SIZE]="";
	getcwd(topDir,sizeof(topDir));
	while(1) {
		char buf[SIZE];
		int fd;
		int nread;//receiving

		testfds = readfds; //used to store readfds temporarily
		printf("server waiting\n\n\n\n");



		/* select(int maxfd, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timeval *timeout) is used to monitor events of
		a set of sockets. If timeout sets to NULL, wait forever and return only when one of the descriptors is ready for I/O.
		A return value <0 indicates an error, 0 indicates a timeout. */
		result = select(FD_SETSIZE, &testfds, (fd_set *)0, (fd_set *)0, (struct timeval *)0); //??????
		if(result< 0) {
			perror("ftp_server");
			exit(1);
		}

		for(fd = 0; fd < FD_SETSIZE; fd++) {
			// checking which socket triggers a read event.
```

```
if(FD_ISSET(fd, &testfds)) {  //测试指定的文件描述符是否在该集合中。

        if(fd == server_sockfd) {   //fd 是 server 的时候
                // Processing new connection requests.
                client_len = sizeof(client_address);
                client_sockfd = accept(server_sockfd, (struct sockaddr
*)&client_address,&client_len);
                // add server_sockfd to fd_set
                FD_SET(client_sockfd, &readfds);
                client[client_sockfd].sin_addr.s_addr = client_address.sin_addr.s_addr;
                printf("adding client on fd %d\n", client_sockfd);
                sprintf(buf, "220 (wxftp 1.0)\r\n");
                write(client_sockfd, buf, strlen(buf));
        } //然后 fd+1 ， 然后就给 client commands
        else {
                //Processing commands from client
                // check if anything received from socket fd;
                // nread  stores number of bytes received.
                ioctl(fd, FIONREAD, &nread);                    //
                if(nread == 0) {
CLOSE:
                        /* no data received, indicating the client has closed FTP command
connection. */
                        close(fd);
                        memset(&client[fd], 0, sizeof(struct Client));
                        FD_CLR(fd, &readfds); //clear fd
                        printf("removing client on fd %d\n", fd);
                }
                else {
                        //read commands from socket fd and process;
                        read(fd, buf, nread);//read fd
                        buf[nread] = '\0';
                        printf("serving client on fd %d: %s\n", fd, buf);
                       printf("Handling client: %s \r\n\n", inet_ntoa(client_address.sin_addr));


                        if(strncmp(buf, "USER", 4) == 0) {   //比较两个字符串前 4 个字
符，一样就返回 0
                                // Receiving user name;
                                // You need to check if it is a valid user;
                                sscanf(&buf[4], "%s", client[fd].user); //从 buf 前四个里
读进 user 用户名中；sscanf 从一个字符串中读进与指定格式相符的数据

        if((strcmp(client[fd].user,username)==0)||(strcmp(client[fd].user,admin)==0)){
                                        if(strcmp(client[fd].user,username)==0){
                                                client[fd].power=0;
                                        }
                                        else client[fd].power=1;
                                        sprintf(buf, "331 Password required
for %s.\r\n", client[fd].user); //要求密码
                                        write(fd, buf, strlen(buf));//把用户名写进 fd
里面 ， 然后就跳回开头 server waiting. 写进 fd 的才会在 client 显示出来
                                }
                                else{
                                        printf("This user does not exist.\n");
                                        sprintf(buf, "530 Username incorrect.\r\n");
                                        write(fd, buf, strlen(buf));
                                        goto CLOSE;
                                }
                        }
                        else if(strncmp(buf, "PASS", 4) == 0) {//这时候已经输入完了用户
名
                                sscanf(&buf[4], "%s", client[fd].pass);
                                if(strcmp(client[fd].pass,password)==0){
                                        sprintf(buf, "230 Login successfully \r\n"); //
成功进入
```

```
                                        write(fd, buf, strlen(buf));

        getcwd(client[fd].cwd,sizeof(client[fd].cwd));//这里把 cwd 写进去
                                }
                                else{
                                        printf("This user does not exist.\n");
                                        sprintf(buf, "530 Password incorrect.\r\n");
                                        write(fd, buf, strlen(buf));
                                        goto CLOSE;
                                }//输入密码 adding
                                write(fd, buf, strlen(buf));
                                // Authenticate the user
                        }
                        else if (strncmp(buf, "QUIT", 4) == 0) {
                                //Processing QUIT command

                                sprintf(buf, "221 Goodbye.\r\n");
                                write(fd, buf, strlen(buf));
                                close(fd);
                                memset(&client[fd], 0, sizeof(struct Client));
                                FD_CLR(fd, &readfds);
                                printf("removing client on fd %d\n", fd);
                        }
                        else{
                                chdir(client[fd].cwd);
                                printf("The client is %s.\n\n",client[fd].user);
                                if(strncmp(buf, "PWD", 3) == 0) {

        getcwd(client[fd].cwd,sizeof(client[fd].cwd));//这里把 cwd 写进去了
                                        client[fd].cwd[sizeof(client[fd].cwd)-1]='\0';//
                                        printf("%s  \r\n\r\n",client[fd].cwd);
                                        sprintf(buf,"257 %s  \r\n",client[fd].cwd);
                                        write(fd, buf, strlen(buf));
                                }
                                else if(strncmp(buf, "CWD", 3) == 0) {
                                        if(client[fd].power==1){
                                                sscanf(&buf[4], "%s", buf);
                                                if(chdir(buf)==0){

        getcwd(client[fd].cwd,sizeof(client[fd].cwd));//这里再把当前 cwd 写进去了

        //client[fd].cwd[sizeof(client[fd].cwd)-1]='\0';//
                                                        printf("%s
\r\n\r\n",client[fd].cwd);
                                                        //输出改变后的 pwd
                                                        sprintf(buf,"250
Directory successfully changed to %s \r\n  ",client[fd].cwd);
                                                }
                                                else{
                                                        sprintf(buf,"550 Failed
to change directory.\r\n");
                                                }
                                        }
                                        else{
                                                sprintf(buf,"550 Failed to change
directory.\r\n");
                                        }
                                        write(fd, buf, strlen(buf));
                                }
                                else if(strncmp(buf, "CDUP", 4) == 0) {//change to parent
directory  回到父目录
                                        //Processing CDUP command
                                        chdir("..");
                                        //输出改变后的 pwd

        getcwd(client[fd].cwd,sizeof(client[fd].cwd));//这里再把当前 cwd 写进去了
                                        client[fd].cwd[sizeof(client[fd].cwd)-1]='\0';//
                                        printf("%s\r\n\r\n",client[fd].cwd);
```

```
                sprintf(buf,"250 %s\r\n",client[fd].cwd);
                write(fd, buf, strlen(buf));
        }
        else if(strncmp(buf, "PORT", 4) == 0) {
                //Processing PORT command;
                /* store IP address of the client in
client[fd].sin_addr; store port number following PORT command in client[fd].data_port; set client[fd].mode as kDataConnModeActive;... */
                sscanf(&buf[4], "%s", tempBuf);
                if(port(fd,tempBuf)==0)
                        sprintf(buf, "200 Port command
successful.\r\n");
                else
                        sprintf(buf, "421 Port failed.\r\n");
                write(fd, buf, strlen(buf));
        }
        else if(strncmp(buf, "LIST", 4) == 0 || strncmp(buf,
"NLST", 4) == 0) {
                //Processing LIST or NLST command
                int sockfd;
                if(client[fd].mode == kDataConnModeActive)
{
                        sockfd = data_conn_active(fd,
&client[fd].sin_addr, client[fd].data_port);
                }
                else if (client[fd].mode ==
kDataConnModePassive) {
                        sockfd = pasv_fd;
                }
                int result = 0;
                if (sockfd != -1) {
                        sprintf(buf,"150 Opening data
connection for directory list.\r\n");
                        write(fd, buf, strlen(buf));
                        if(send_list(sockfd, client[fd].cwd)
== 0) {
                                sprintf(buf, "226
Transfer ok.\r\n");
                        }
                        else {
                                sprintf(buf, "550 Error
encountered.\r\n");
                        }
                }
                else {
                        sprintf(buf, "550 Error
encountered.\r\n");
                }
                write(fd, buf, strlen(buf));
                close(sockfd);
                close(pasv_fd);
        }
        //make new dir
        else if((strncmp(buf, "MKD", 3) == 0))
        {
                char filename[SIZE]="";
                sscanf(&buf[3], "%s", filename);
                {
                        if(mkdir(filename,0) ==0){
                                sprintf(buf,"250 create
folder %s successfully!\n", filename);
                        }
                        else {
                                sprintf(buf,"550 create
folder failed.\n");
                        }
                }
                write(fd, buf, strlen(buf));
        }
        //delete file
```

```
successfully!\n", filename);

successfully!\n");

{

&client[fd].sin_addr, client[fd].data_port);

kDataConnModePassive) {

connection for directory list.\r\n");

0) {

Transfer ok.\r\n");

encountered.\r\n");

connection.\r\n");
```

```
else if(strncmp(buf, "DELE", 4) == 0){
        char filename[SIZE]="";
        sscanf(&buf[4], "%s", filename);
        if(remove(filename) ==0){
                sprintf(buf,"250 delete %s

        }
        else{
                sprintf(buf,"550 delete failed.\n");
        }
        write(fd, buf, strlen(buf));
}
//Rename from
else if((strncmp(buf, "RNFR", 4) == 0)){
        memset(&tempBuf, 0, SIZE*sizeof(char));
        sscanf(&buf[4], "%s", tempBuf);
        printf("%s\n",tempBuf);
        sprintf(buf,"350 Ready for RNTO.\r\n");
        write(fd, buf, strlen(buf));
}
//Rename to
else if ((strncmp(buf, "RNTO", 4) == 0)){
        char newName[SIZE]="";
        sscanf(&buf[4], "%s", newName);
        printf("%s\n",tempBuf);
        printf("%s\n",newName);
        if(rename(tempBuf,newName) ==0){
                sprintf(buf,"250 rename

        }
        else{
                sprintf(buf,"550 rename failed.\n");
        }
        write(fd, buf, strlen(buf));
}
else if((strncmp(buf, "RETR", 4) == 0)){
        //Processing RETR command
        printf("RETR is executed by %d.\n",fd);
        int sockfd;
        char fileName[SIZE];
        sscanf(&buf[4], "%s", fileName);
        if(client[fd].mode == kDataConnModeActive)

                sockfd = data_conn_active(fd,

        }
        else if (client[fd].mode ==

                sockfd = pasv_fd;
        }
        int result = 0;
        if (sockfd != -1) {
                sprintf(buf,"150 Opening data

                write(fd, buf, strlen(buf));
                if(send_file(sockfd, fileName) ==

                        sprintf(buf, "226

                }
                else {
                        sprintf(buf, "550 Error

                }
        }
        else{
                sprintf(buf, "425 Can't open data

        }
```

```
                write(fd, buf, strlen(buf));
                close(sockfd);
        }
        else if((strncmp(buf, "STOR", 4) == 0)){
                //Processing STOR command
                int sockfd;
                char fileName[SIZE];
                sscanf(&buf[4], "%s", fileName);
                if(client[fd].mode == kDataConnModeActive)
                        sockfd = data_conn_active(fd,
                }
                else if (client[fd].mode ==
                        sockfd = pasv_fd;
                }
                int result = 0;
                if (sockfd != -1) {
                        sprintf(buf,"150 Opening data
                        write(fd, buf, strlen(buf));
                        if(upload_file(sockfd, fileName)
                                sprintf(buf, "226
                        }
                        else {
                                sprintf(buf, "550 Error
                        }
                }
                else{
                        sprintf(buf, "425 Can't open data
                }
                write(fd, buf, strlen(buf));
                close(sockfd);
                close(pasv_fd);
        }
        else if(strncmp(buf, "PASV", 4) == 0){
                printf("PASV is executed.\n");
                int p1,p2,sockfd;
                int data_fd;
                pas_port++;
                struct sockaddr_in clinAddr;
                int len;
                pasv_fd= data_conn_passive(pas_port);
                p2=pas_port%256;
                p1=pas_port/256;
                client[fd].mode=kDataConnModePassive;
                sprintf(buf, "227 Entering Passive Mode
                write(fd, buf, strlen(buf));
                printf("pasv_fd is %d\n",pasv_fd);
                getsockname(pasv_fd,(struct
                if((pasv_fd=accept(pasv_fd,(struct
                        printf("pasv_fd is %d\n",pasv_fd);
                        printf("data passive accept
                        close(pasv_fd);
                }
        }
        else{
```

```
{
&client[fd].sin_addr, client[fd].data_port);
kDataConnModePassive) {
connection for directory list.\r\n");
== 0) {
Transfer ok.\r\n");
encountered.\r\n");
connection.\r\n");
(192,168,116,128,%d,%d).\r\n",p1,p2);
sockaddr*)&clinAddr,&len);
sockaddr*)&clinAddr,&len))<0){
failed\n");
```

```
                                                                        }
                                                                        chdir(topDir);
                                                                }
                                                        }
                                                }
                                        }
                                }
                        }


                return 0;
}

int create_server_socket()
{
                struct sockaddr_in servAddr; /* Local address */
                //创建 socket
                if((server_sockfd=socket(PF_INET, SOCK_STREAM, 0))<0) {
                                perror("socket");
                }
                int opt=1;

                memset(&servAddr, 0, sizeof(servAddr)); /*ip 地址替换为 0*/
                servAddr.sin_family = PF_INET;
                servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
                servAddr.sin_port =htons(SERVER_CMD_PORT);
                setsockopt(server_sockfd,SOL_SOCKET,SO_REUSEADDR,&opt,sizeof(opt) );
                //绑定 socket
                if ((bind(server_sockfd, (struct sockaddr *) &servAddr,sizeof(servAddr))) < 0) {
                                printf("bind() failed.\n");
                }
                if (listen(server_sockfd, MAX_CLIENT)<0) printf("litsen() failed.\n");

}

int data_conn_active(int fd, struct in_addr *sin_addr, unsigned short port)
{
                printf("data_conn_active is executed.\n");
                /* set up data connection in active mode.
                The input parameters are IP address and port number of the client.
                The return value is socket file descriptor created for data connection. */
                struct sockaddr_in clntAddr; /* Clinet address */

                int sock_fd;/*socket ddesriptor for data connection*/


                memset(&clntAddr,0,sizeof(struct sockaddr_in));
                clntAddr.sin_family=AF_INET;
                clntAddr.sin_addr.s_addr=(*sin_addr).s_addr;
                clntAddr.sin_port=htons(port);

                //创建 data port socket
                if((sock_fd=socket(AF_INET, SOCK_STREAM, 0))<0){
                                close(sock_fd);
                                return -1;
                }
                int opt=1;

                memset(&servAddr, 0, sizeof(struct sockaddr_in)); /*ip 地址替换为 0*/
                servAddr.sin_family = AF_INET;
                servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
                servAddr.sin_port =htons(SERVER_DAT_PORT);
                setsockopt(sock_fd,SOL_SOCKET,SO_REUSEADDR,&opt,sizeof(opt) );
                //绑定 socket
                if ((bind(sock_fd, (struct sockaddr *) &servAddr,sizeof(struct sockaddr_in))) < 0) {
                                close(sock_fd);
```

```
                printf("data_conn_active bind() failed.\n");
                return -1;
        }
        if ((connect(sock_fd,(struct sockaddr *) &clntAddr,sizeof(struct sockaddr_in)))<0){
                close(sock_fd);
                printf("data_conn_active connect() failed.\n");
                return -1;
        }
        return sock_fd;

        /* Create a socket and connect to client whose IP address and port number is given as input parameters. */

}

int send_list(int sockfd, char *dir)
{

        /* Find out directory and file list and send to client.
        For input parameters, sockfd are the socket fd of data connection, and dir is the string including sub-directory and file list under current directory.
        The return value is 0 if sending OK, otherwise -1 is returned. */

       /* To get sub-directory and file list, you can execute shell command by using system( ) or use functions as opendir( ) and readdir( ) */
        printf("send_list is executed.\n");
        DIR *currentDir;
        struct dirent *ptr;
        char buf[SIZE]="";
        if((currentDir=opendir(dir))==NULL){
                return -1;
        }
        ptr=readdir(currentDir);
        ptr=readdir(currentDir);

        while((ptr=readdir(currentDir))!=NULL){
                //printf("%s\n",ptr->d_name);
                sprintf(buf,"%s\r\n",ptr->d_name);
                write(sockfd, buf, strlen(buf));
        }
        closedir(currentDir);
        return 0;
}
int port(int fd,char *tempBuf){
        printf("port is executed.\n");
        //the port comand return something like "127,0,0,1,25,60"
        //input:
        //2 : client feedback string of PORT command
        //return:
        //ftp data socket

        // user to trans digital to string of ipaddress
        char localbuffer[16]={'\0'};

        // sotr ipaddress
        char clientipaddress[128]={'\0'};

        // parse PORT command from client
        unsigned short portinfoparser[6];

        // port number of client
        unsigned short clientport;

        // index
        int index=0;

        printf("LOG:\t client PORT output:%s\n", tempBuf);
```

```
		// start to split "127,0,0,1,25,60"
		char * tok = strtok(tempBuf, ",");

		while(tok != NULL){
			portinfoparser[index++]=(unsigned short)atoi(tok);

			tok=strtok(NULL,",");
		}
		for (index=0; index<4; index++) {
			memset(localbuffer, 0, 16*sizeof(char));
			sprintf(localbuffer, "%d", (int)portinfoparser[index]);
			strcat(clientipaddress, localbuffer);
			if(index!=3)strcat(clientipaddress, ".");
		}


		// get client port
		client[fd].data_port = portinfoparser[4]*256+portinfoparser[5];
		client[fd].sin_addr.s_addr=inet_addr(clientipaddress);
		client[fd].mode=kDataConnModeActive;


		// get client ip address

		return 0;

}
	time_t startT, endT;

	double totalT;

int send_file(int sockfd, char *fileName){
		//fail return -1
		printf("send_file is exexuted.\n");
		char buf[SIZE]="";
		int file_fd;
		int length;
		int tot=0;//total transmission
		double delay;
		if((file_fd=open(fileName,O_RDONLY,0))<0) return -1;
		 startT = time( NULL );//计时
		while((length=read(file_fd,buf,SIZE))!=0){
	tot=tot+length;
			delay=delay+((double)length)/((double)max_speed);
			if(delay>=1)
				sleep((int)delay);
			delay=delay-((double)(int)delay);
			write(sockfd, buf, length);

		}
		if(delay!=0.0) sleep(1);
		close(file_fd);
		endT = time( NULL );
   totalT = difftime(endT,startT);
		printf("%s is sent to the fd %d\r\n",fileName,sockfd);
		printf("total transmission: %d bytes\r\n",tot);
		printf("transmission time: %f second(s)\r\n",totalT);
		printf("speed: %f byte(s)/second(s)\r\n",(tot/totalT));
		return 0;
}
int upload_file(int sockfd, char *fileName){
		//fail return -1
		printf("upload_file is exexuted.\n");
		char buf[SIZE]="";
		int file_fd;
		int length;
```

```
        double delay;
        if((file_fd=open( fileName, O_WRONLY|O_CREAT|O_TRUNC,0644))<0) return -1;
        int tot=0;//total transmission)<0) return -1;
         startT = time( NULL );//计时
        while ((length=read(sockfd, buf, SIZE*(sizeof(char)))) > 0){
    tot=tot+length;
                delay=delay+((double)length)/((double)max_speed);
                if(delay>=1)
                        sleep((int)delay);
                delay=delay-((double)(int)delay);
                write(file_fd, buf, length);
        }
        if(delay!=0.0) sleep(1);
        close(file_fd);
        endT = time( NULL );
  totalT = difftime(endT,startT);
        printf("%s has been stored on the server from %d fd\r\n", fileName,sockfd);
        printf("total transmission: %d bytes\r\n",tot);
        printf("transmission time: %f second(s)\r\n",totalT);
        printf("speed: %f byte(s)/second(s)\r\n",(tot/totalT));

        return 0;
}
int data_conn_passive(unsigned short port){
        printf("data_coon_pasive is exexuted.\n");
        int sock_fd;
        if((sock_fd=socket(PF_INET, SOCK_STREAM, 0))<0) return -1;
        int opt=1;

        memset(&servAddr, 0, sizeof(struct sockaddr_in)); /*ip 地址替换为 0*/
        servAddr.sin_family = PF_INET;
        servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servAddr.sin_port =htons(port);
        setsockopt(sock_fd,SOL_SOCKET,SO_REUSEADDR,&opt,sizeof(opt) );
        if ((bind(sock_fd, (struct sockaddr *) &servAddr,sizeof(struct sockaddr_in))) < 0){
                close(sock_fd);
                printf("passive bind() failed.\n");
                return -1;
        }
        if (listen(sock_fd, MAX_CLIENT)<0){
                close(sock_fd);
                printf("passive litsen() failed.\n");
                return -1;
        }

        return sock_fd;
}
```