Project report

Final Project-FTP Proxy

Course Title: *Internet Applications*

Name: Wang Yian *(2013213414)*

Lyu Mengyu*(2013213338)*

Date: *June 16th, 2016*

1. Overview

This final project is about FTP proxy program based on Linux command.
The goals for this final project are:

- Deeply understand the related knowledge of FTP (File Transfer Protocol).
- Complete a FTP proxy program based on Linux command line terminal.

What we need to realize in this project are：

1. Enter user name and password in the user interface. Then we connect control connection.After that we connect data connection.
2. FTP server first connect to Proxy, send FTP request. After Proxy get the request it will send that to server. Proxy will also send the response of server back to client.
3. FTP Proxy can use control connection to get command request from client, including : PWD, CWD, LIST/MLSD, MDIR, DELE, RNFR/RNTO, RETR, and STOR.
4 . FTP Proxy can get response from the server, can analyze and change PASV parameter if necessary and then change it (In our program only when Proxy get 227 will change it into proxy's data connection listening port number.).
5. FTP Proxy need cache（PDF and picture). If client need to download the file in cache, Proxy will not need to download it from the server.
6. data connection need to show passive mode and active mode

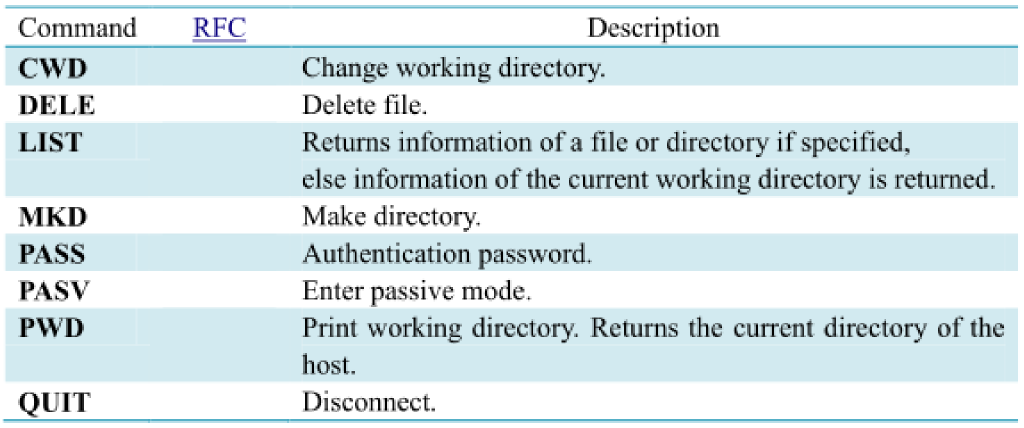
2. Requirements Analysis

In this project we also need to define our development environment and details functional requirements to realize a perfect ftp client's request.

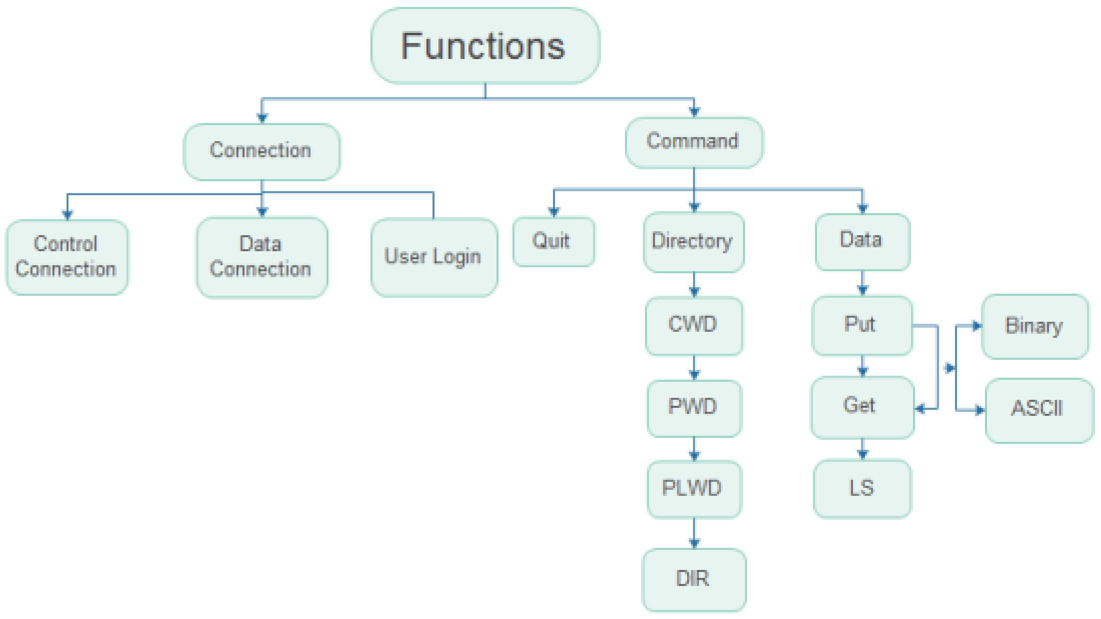
2.1 Environment

Developing Environment:Ubuntu System, Linux
Operating system: Ubuntu system
Language:C
Development environment:C free
Running tools: command line interface (CLI) as input and output.

Table 1 FTP commands



Command	RFC	Description
CWD		Change working directory.
DELE		Delete file.
LIST		Returns information of a file or directory if specified, else information of the current working directory is returned.
MKD		Make directory.
PASS		Authentication password.
PASV		Enter passive mode.
PWD		Print working directory. Returns the current directory of the host.
QUIT		Disconnect.



2.2 Basic Function

The requirements for this project are:

1. The FTP server can be set up using the existing software, for instance,FileZilla server. For the FTP client, students can use one general FTP client tool software, for example FileZilla client.
2. The FTP proxy can performs as both FTP client and FTP server. FTP client will connect FTP proxy first and send the FTP requests. FTP proxy is able to receive the requests, and then forward the requests to FTP server. After that, it can receive the replies from the FTP server, and then forward the replies to the FTP client.
3. FTP proxy has to listen at port 21 for control connection and we use port 25601 for data connection.
4. FTP proxy is able to set up separate control connections with FTP client and FTP server separately. res
   a) FTP proxy is able to receive the commands from FTP client using control connection. The commands include: PWD, CWD, LIST/MLSD, MDIR, DELE, RNFR/RNTO, RETR, and STOR. And it can analyze the commands and change if necessary such as in our program when we get 227 (i.e. modify the parameters for PORT), and then forward the commands to the FTP server using control connection.
   b) FTP proxy is able to receive the FTP replies from FTP serverusing control connection. And it can resolve the replies and modify if necessary (i.e. modify the parameters in the replies for PASV), and then forward the replies to the FTP client.
5. FTP proxy is able to set up data connections if required with FTP client and FTP server separately. In this project, cache mechanism will be used in FTP proxy (for pdf file and picture). If the file that FTP client wants to download is already exists in the cache, FTP proxy will not download the file from FTP server but send the file from the cache to FTP client. And no data connection will be set up between FTP proxy and FTP server.

Otherwise, data connection will be set up between FTP proxy and FTP server.

a) If FTP client wants to upload a file, FTP proxy can receive the file from FTP client and then upload the file to FTP server.

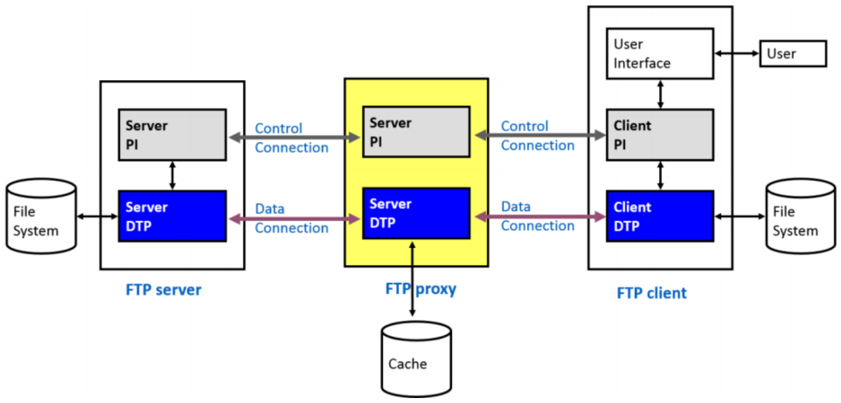
b) If FTP client wants to download a file, FTP proxy can receive the file from FTP server and forward the file to FTP client. Meanwhile, the file will store in its cache.

6. For data connection, it is required to implement*both passive mode and active mode*.

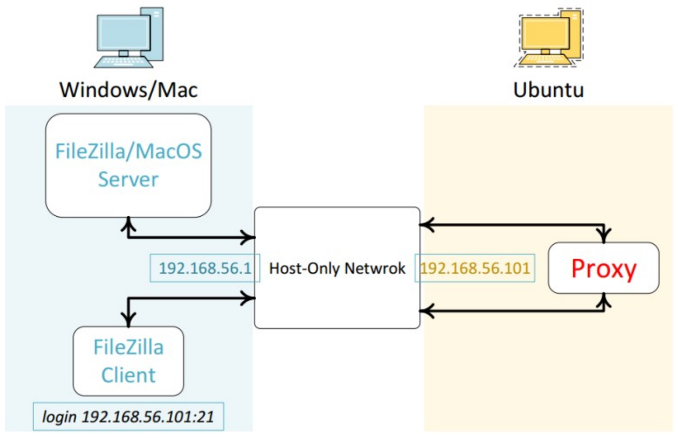
3. Preliminary Design

3.1 FTP System Design

The structure between FTP server, FTP Proxy and FTP client is shown as below:



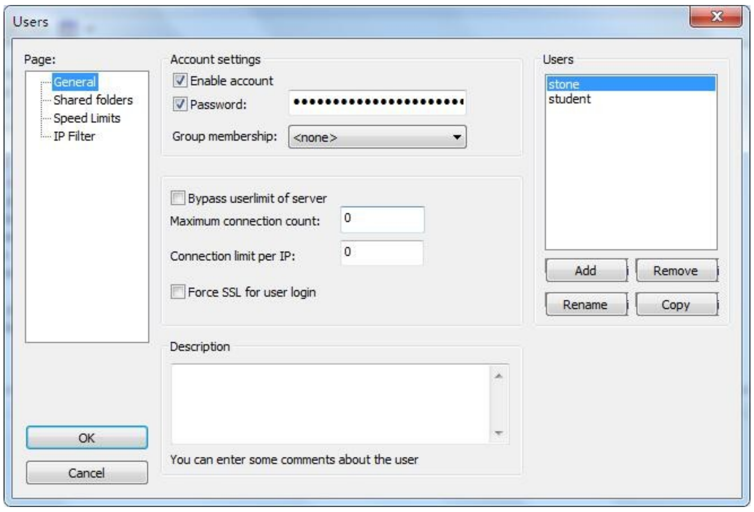
There are control connection and data connection between both FTP client and Proxy and Proxy and FTP server. The communication is using a virtual Network Card, host-only network, this is used between our Windows and Ubuntu.



This project need FTP server and client. These two will use tools to configure, only Proxy need to write in the code.

FileZilla Client:

Add→user Configure user. Let User name be Stone and password be 123456



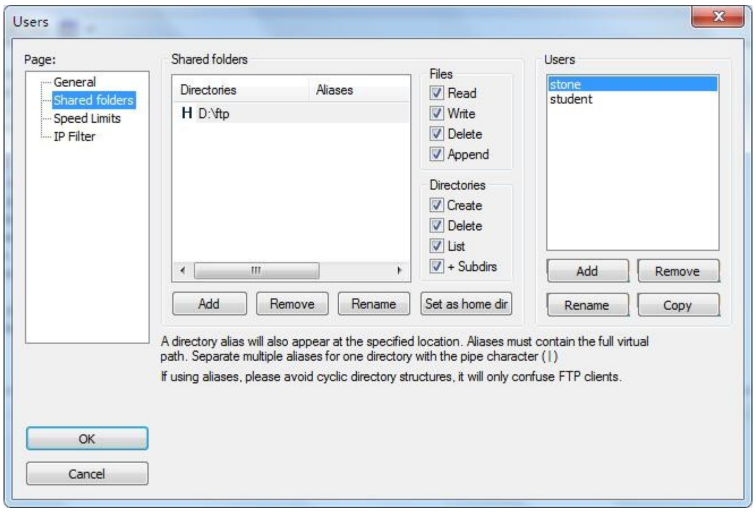
Account settings
☑ Enable account
☑ Password: ••••••••••••••••••••••
Group membership: <none>

☐ Bypass userlimit of server
Maximum connection count: 0
Connection limit per IP: 0
☐ Force SSL for user login

Description
You can enter some comments about the user

Users
stone
student

Choose the authority for the user, can read write and so on:



Shared folders

Directories	Aliases
H D:\ftp	

Files
☑ Read
☑ Write
☑ Delete
☑ Append

Directories
☑ Create
☑ Delete
☑ List
☑ + Subdirs

Users
stone
student

A directory alias will also appear at the specified location. Aliases must contain the full virtual path. Separate multiple aliases for one directory with the pipe character (|)
If using aliases, please avoid cyclic directory structures, it will only confuse FTP clients.

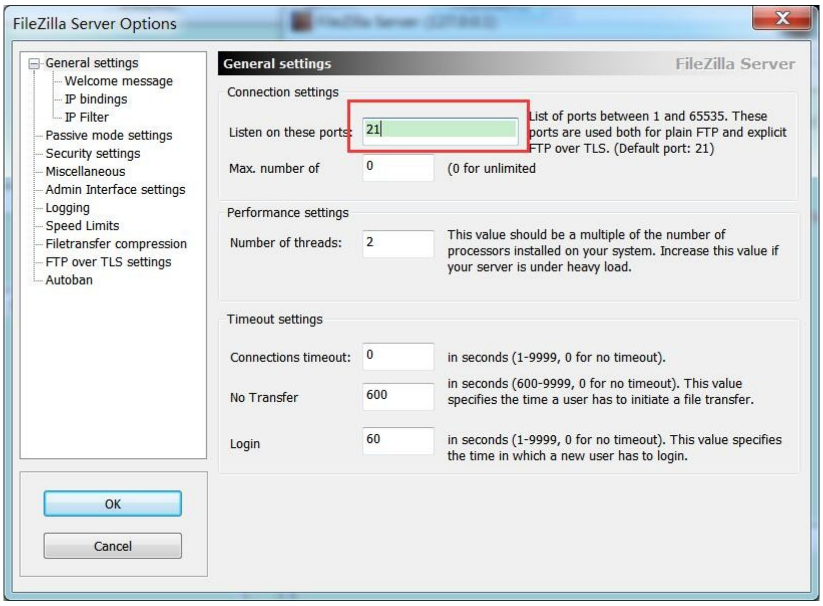
FileZilla Server:

Server we use FileZilla, initial configure we the host: 192.168.56.1, the opposite port number will be set as the same port number (which is read by proxy) as the client using.

Server will listen the port 21 for waiting for client’s control connection.

```c
j = strrchr(buff, ',');
p1 = atoi(j+1);
*j = '\0';
j = strrchr(buff, ',');
p2 = atoi(j+1);
sprintf(buff, "PORT 192,168,56,101,%d,%d\r\n", p1, p2);
nread = strlen(buff);
dataport = p1 * 256 + p2;
  //bind() and listen() for proxy_data_socket
  struct sockaddr_in addr_pdata;
  memset(&addr_pdata, 0, sizeof(addr_pdata));
  addr_pdata.sin_family = AF_INET;
  addr_pdata.sin_addr.s_addr = htonl(INADDR_ANY);
  addr_pdata.sin_port = htons(dataport);

  if ((bind(proxy_data_socket, (struct sockaddr *) &addr_pdata, sizeof(addr_pdata))) < 0) {
      printf("p_d_s bind() failed.\n");
      exit(0);
  }
  if (listen(proxy_data_socket, 10) < 0) {
      printf("listen error.\n");
      exit(0);
  }
```



General settings

Connection settings

Listen on these ports: 21 — List of ports between 1 and 65535. These ports are used both for plain FTP and explicit FTP over TLS. (Default port: 21)

Max. number of: 0 — (0 for unlimited

Performance settings

Number of threads: 2 — This value should be a multiple of the number of processors installed on your system. Increase this value if your server is under heavy load.

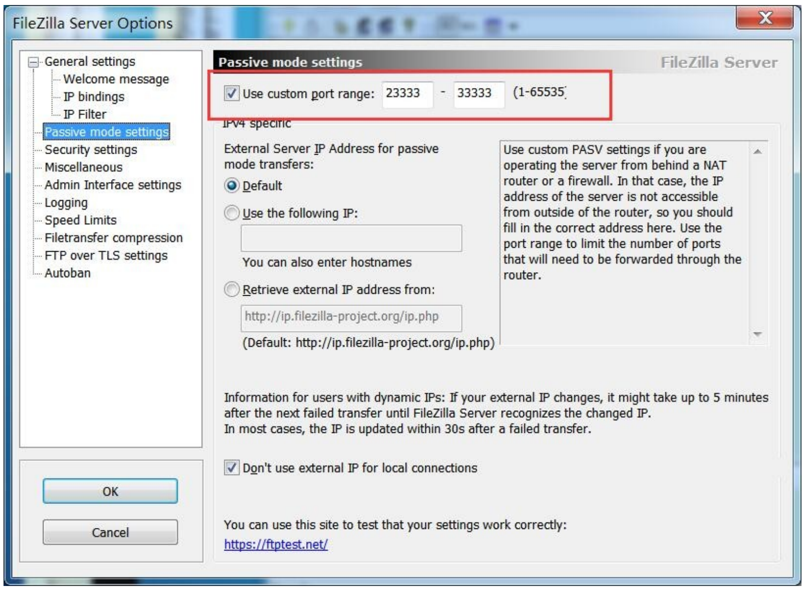
Timeout settings

Connections timeout: 0 — in seconds (1-9999, 0 for no timeout).

No Transfer: 600 — in seconds (600-9999, 0 for no timeout). This value specifies the time a user has to initiate a file transfer.

Login: 60 — in seconds (1-9999, 0 for no timeout). This value specifies the time in which a new user has to login.

Passive mode settings:


FileZilla Server Options

Passive mode settings

☑ Use custom port range: 23333 - 33333 (1-65535)

IPv4 specific

External Server IP Address for passive mode transfers:

● Default

○ Use the following IP:

You can also enter hostnames

○ Retrieve external IP address from:

http://ip.filezilla-project.org/ip.php

(Default: http://ip.filezilla-project.org/ip.php)

Use custom PASV settings if you are operating the server from behind a NAT router or a firewall. In that case, the IP address of the server is not accessible from outside of the router, so you should fill in the correct address here. Use the port range to limit the number of ports that will need to be forwarded through the router.

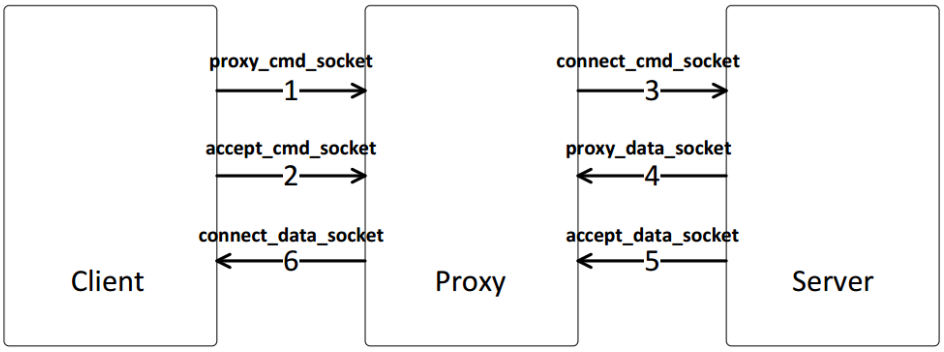
Information for users with dynamic IPs: If your external IP changes, it might take up to 5 minutes after the next failed transfer until FileZilla Server recognizes the changed IP.
In most cases, the IP is updated within 30s after a failed transfer.

☑ Don't use external IP for local connections

You can use this site to test that your settings work correctly:
https://ftptest.net/

It means connection is correct among client, proxy and server.

3.2 Modular Decomposition



Module1:

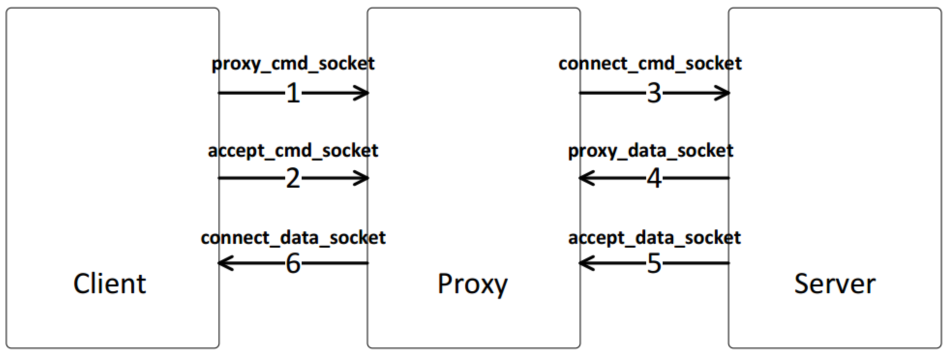
System design configuration for client and server.

Module 2:

Establish TCP connection among client, proxy and server

First we are going to establish a communication path among client, proxy and server for the exchange of commands and replies.

The connection process use socket structure. Among their communication, all the types' data transmission use socket type with required information, like IP address or port number.



Module 3:

Transmit command connection and reply code

We operate under command connection and for each commands, it will generate at least one reply 3-digit code followed by delimiter and text message and we can distinguish it to enter different operations.

Module 4:

Establish data connection

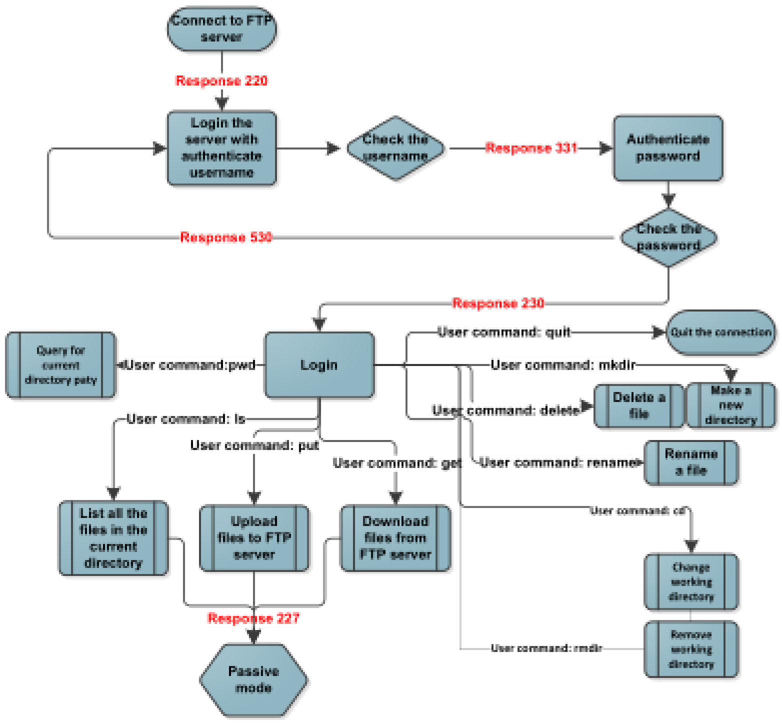
We set a full duplex connection over which data is transferred, in a specified mode and type. The data transferred among server, proxy and client. And the data connection can be passive or active. Client can upload or download files from server through proxy.

Module 5:

Establish cache structure

We set a structure which can judge whether proxy has this file or not. For proxy, it can check whether I have saved this file or not. If yes, with client's request, Proxy will transmit file to client. If not, it will download the file from server, and then transmit it to client. At the same time, Proxy will save it in its own content.

3.3 Relationships of Modules



4. Detailed Design

4.1 Establish Proxy Communication

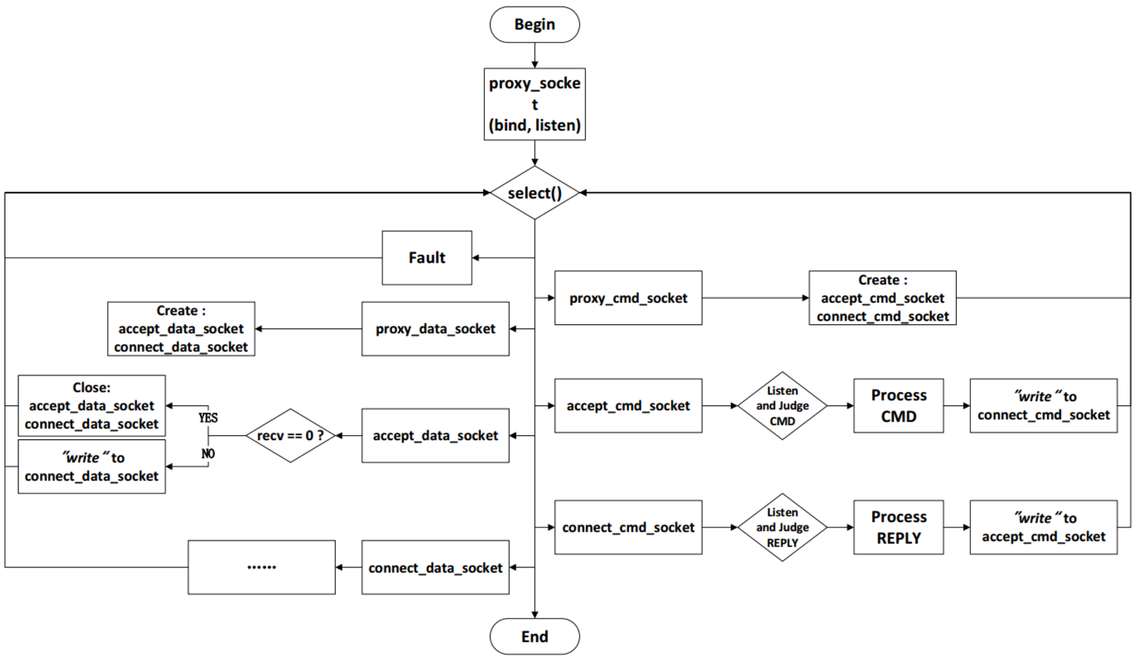
Connection Steps：

1. Use FileZilla
2. Proxy is buld on Ubuntu system
3. Open proxy_cmd_socket、bind（）、listen
4. Open proxy_data_socket、bind（）、listen
5. Put proxy_cmd_socket into master_set
6. Put proxy_data_socket into master_set
7. Select will listen to read operation（working_set is using）
8. Decide selectResult. If selectResult > 0 , we will decide which is the socket
9. Decide whether it is in the working_set, if in, accept, and connect proxy and client control connection.
10. Connectproxy and server’s control connection.

4.2 Establish Active and Passive Mode

Active Model：

1. Client send 6 number to sever 192，168，56，1， 100，1,first four number are the IP, last two are relevant to port number and the port number is 100*256+1
2. Proxy get the request from client, analyze and then send its own IP and port number and see itself as a client.
3. Server respond to proxy
4. Proxy respond to client
5. After the control connection and the data connection, the transmission will go on.



Begin
proxy_socket (bind, listen)
select()
Fault
proxy_cmd_socket
Create : accept_cmd_socket connect_cmd_socket
Create : accept_data_socket connect_data_socket
proxy_data_socket
Close: accept_data_socket connect_data_socket
YES
recv == 0 ?
accept_data_socket
NO
"write" to connect_data_socket
accept_cmd_socket
Listen and Judge CMD
Process CMD
"write" to connect_cmd_socket
connect_cmd_socket
Listen and Judge REPLY
Process REPLY
"write" to accept_cmd_socket
......
connect_data_socket
End

Passive Model：

1. Server send IP and port number to proxy
2. Proxy analyze the request and store the the IP and port number.
3. Proxy will expose its IP and port number and send message to client.
4. Client will send message to the specific IP and port number.
5. Proxy will transfer the message to the stored IP and port number.
6. The passive model data transmission channel is established.

4.3 Establish Get File

Before data transported, system will create data transport connection, active or passive. Client send “get filename.extension ”(which means a filename with extension) to ask for downloading and this command will be transferred into “RETR filename.extension” by PI. When proxy received request from client, proxy will remember the IP address and port number of client. Then, proxy will check whether the file is in the cache of not. If not, proxy will send request for downloading the same file to server. Server received the downloading request, it will send data through the data connection to proxy, and proxy will send it to client. And, if the extension of file is .pdf or .jpg, the file will be add to cache.

4.4 Establish Put File

Before data transported, system will create data transport connection, active or passive. Client send "put filename.extension "(which means a filename with extension) to ask for uploading and this command will be transferred into "STOR filename.extension" by PI. When proxy received request from client, proxy will remember the IP address and port number of client. Then, proxy will send request for uploading the same file to server and send "150" control information to client. Server received the uploading request, it will send "150" control response. Then client send data through the data connection to proxy, and proxy will send it to server.

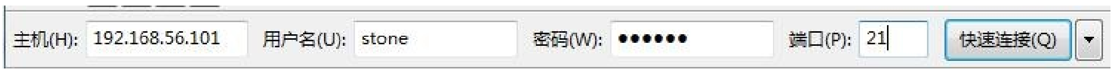
4.5 Establish Cache

1. Client send RETR control command to proxy, and proxy will send to server.
2. When proxy received RETR control command, proxy will try to open the file, if file can be open it means file is in the cache. Or file is not in the cache.
3. If file is in the cache, proxy will send the file to client directly. And do not set up data connection to server.
4. If file is not in the cache, proxy will send the RETR request to server. And download the file from server. If the extension of file is .pdf or .jpg, proxy will cache the file. Then sendfile data to client through data connection.

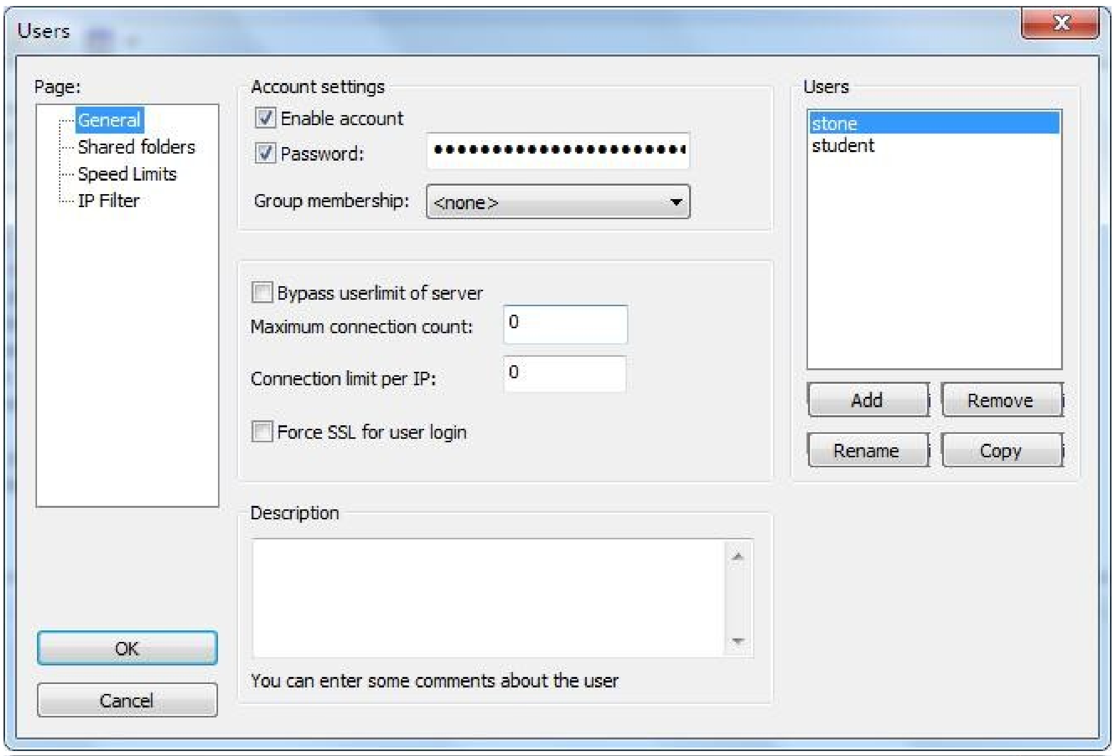
5. Results

5.1 Login

In put username and password into client：


主机(H): 192.168.56.101 用户名(U): stone 密码(W): •••••• 端口(P): 21

Set the username and password of FTP account and jurisdiction of file in server:


Users

Page:
- General
- Shared folders
- Speed Limits
- IP Filter

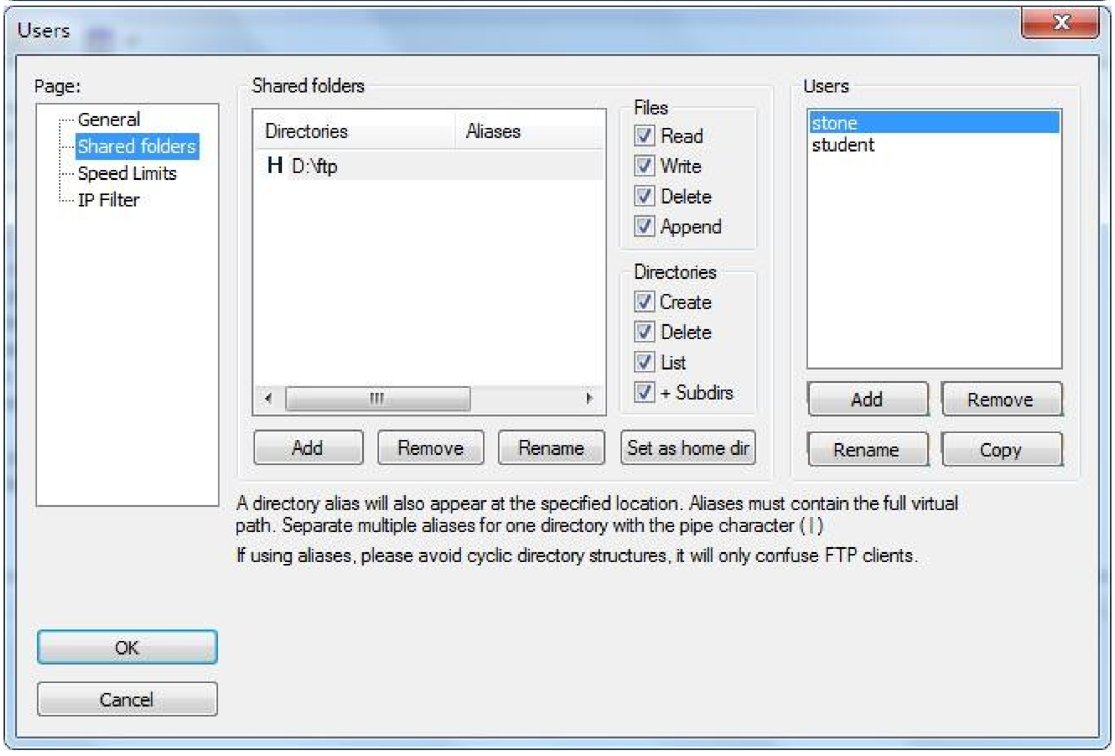
Account settings
- [✓] Enable account
- [✓] Password: ••••••••••••••••••••••
- Group membership: <none>

- [ ] Bypass userlimit of server
- Maximum connection count: 0
- Connection limit per IP: 0
- [ ] Force SSL for user login

Description

You can enter some comments about the user

Users
- stone
- student


Users

Page:
- General
- Shared folders
- Speed Limits
- IP Filter

Shared folders

Directories	Aliases
H D:\ftp	

Files
- [✓] Read
- [✓] Write
- [✓] Delete
- [✓] Append

Directories
- [✓] Create
- [✓] Delete
- [✓] List
- [✓] + Subdirs

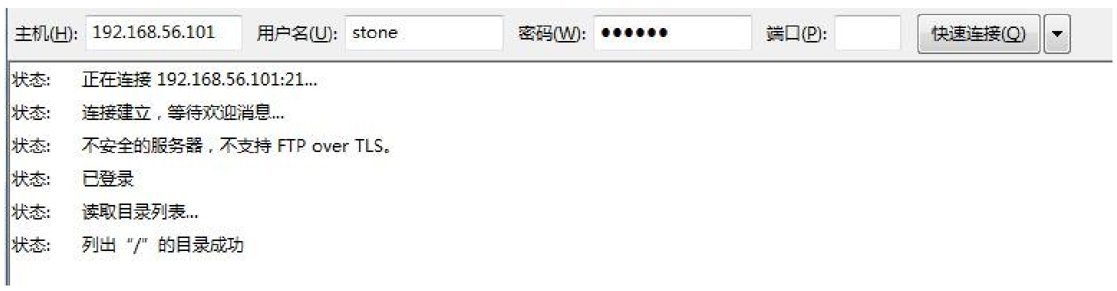
Users
- stone
- student

A directory alias will also appear at the specified location. Aliases must contain the full virtual path. Separate multiple aliases for one directory with the pipe character (|)

If using aliases, please avoid cyclic directory structures, it will only confuse FTP clients.

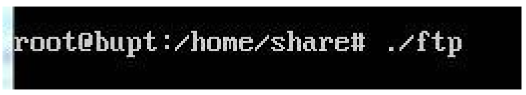
Login:

Client:



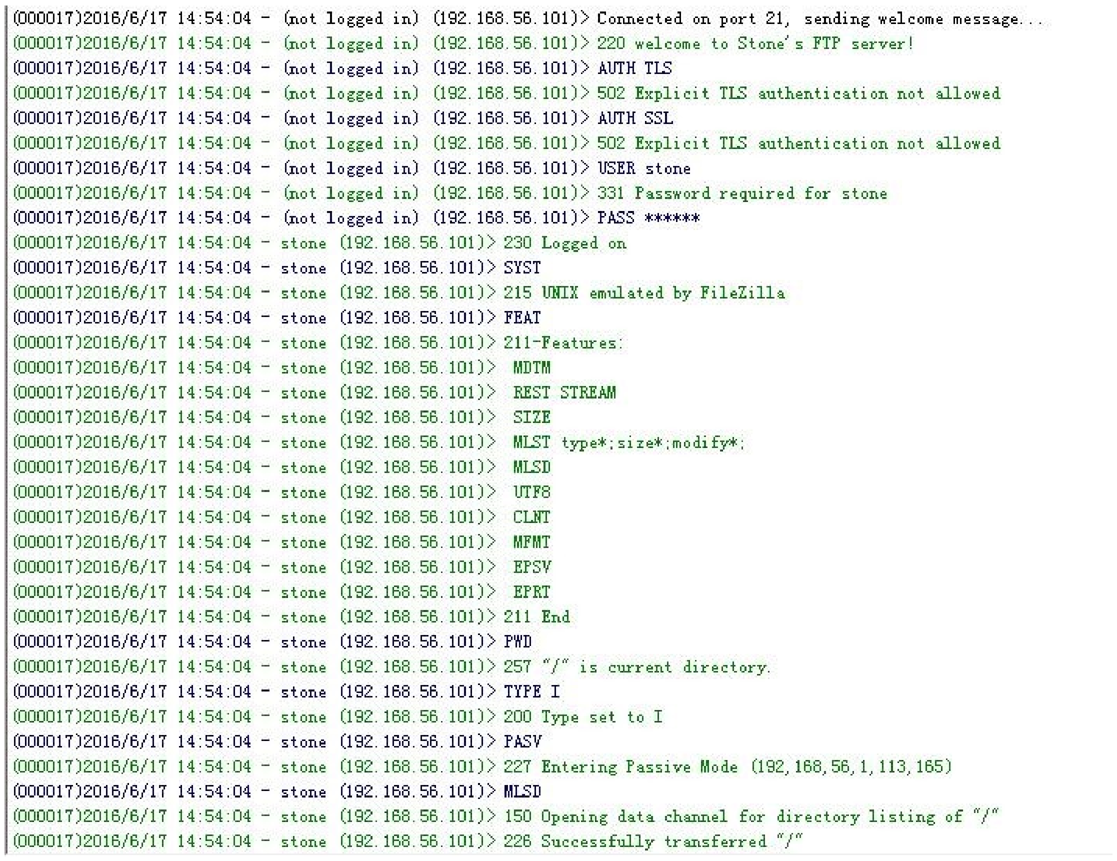
```
主机(H): 192.168.56.101   用户名(U): stone   密码(W): ••••••   端口(P):   快速连接(Q)
状态:	正在连接 192.168.56.101:21...
状态:	连接建立，等待欢迎消息...
状态:	不安全的服务器，不支持 FTP over TLS。
状态:	已登录
状态:	读取目录列表...
状态:	列出 “/” 的目录成功
```

Proxy:



```
root@bupt:/home/share# ./ftp
```

Server:



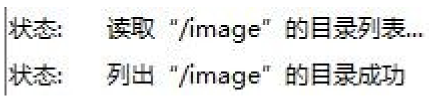
```
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> Connected on port 21, sending welcome message...
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> 220 welcome to Stone's FTP server!
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> AUTH TLS
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> 502 Explicit TLS authentication not allowed
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> AUTH SSL
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> 502 Explicit TLS authentication not allowed
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> USER stone
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> 331 Password required for stone
(000017)2016/6/17 14:54:04 - (not logged in) (192.168.56.101)> PASS ******
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 230 Logged on
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> SYST
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 215 UNIX emulated by FileZilla
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> FEAT
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 211-Features:
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  MDTM
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  REST STREAM
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  SIZE
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  MLST type*;size*;modify*;
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  MLSD
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  UTF8
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  CLNT
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  MFMT
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  EPSV
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)>  EPRT
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 211 End
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> PWD
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 257 "/" is current directory.
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> TYPE I
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 200 Type set to I
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> PASV
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 227 Entering Passive Mode (192,168,56,1,113,165)
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> MLSD
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 150 Opening data channel for directory listing of "/"
(000017)2016/6/17 14:54:04 - stone (192.168.56.101)> 226 Successfully transferred "/"
```

5.2 CWD

Chang work direction:

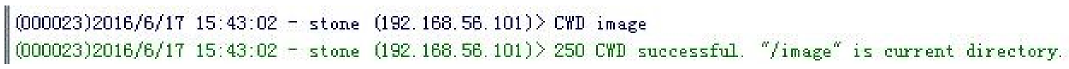
Client:

Into image direction



```
状态:	读取 “/image” 的目录列表...
状态:	列出 “/image” 的目录成功
```

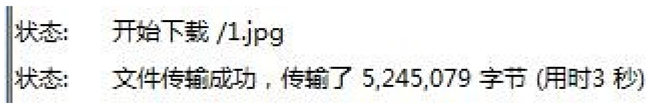
Server:



```
(000023)2016/6/17 15:43:02 - stone (192.168.56.101)> CWD image
(000023)2016/6/17 15:43:02 - stone (192.168.56.101)> 250 CWD successful. "/image" is current directory.
```

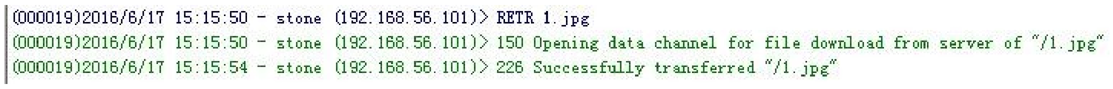
5.3 RETR

Client:



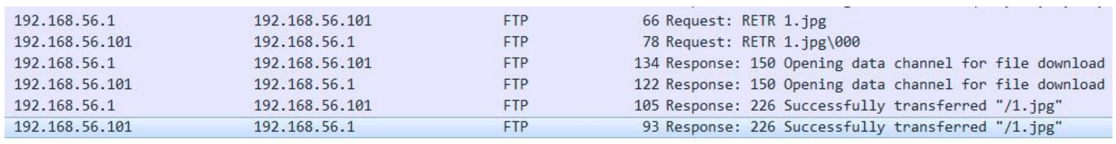
```
状态:    开始下载 /1.jpg
状态:    文件传输成功，传输了 5,245,079 字节 (用时3 秒)
```

Server:



```
(000019)2016/6/17 15:15:50 - stone (192.168.56.101)> RETR 1.jpg
(000019)2016/6/17 15:15:50 - stone (192.168.56.101)> 150 Opening data channel for file download from server of "/1.jpg"
(000019)2016/6/17 15:15:54 - stone (192.168.56.101)> 226 Successfully transferred "/1.jpg"
```

Wireshark:



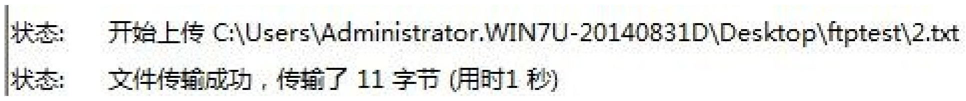
```
192.168.56.1        192.168.56.101      FTP     66 Request: RETR 1.jpg
192.168.56.101      192.168.56.1        FTP     78 Request: RETR 1.jpg\000
192.168.56.1        192.168.56.101      FTP    134 Response: 150 Opening data channel for file download
192.168.56.101      192.168.56.1        FTP    122 Response: 150 Opening data channel for file download
192.168.56.1        192.168.56.101      FTP    105 Response: 226 Successfully transferred "/1.jpg"
192.168.56.101      192.168.56.1        FTP     93 Response: 226 Successfully transferred "/1.jpg"
```

5.4 STOR

Client and proxy:

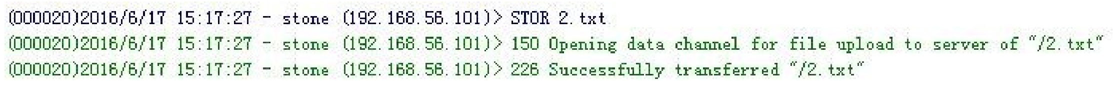
Client request to upload 2.txt. when proxy received "STOR" command, data connection will be set up in active mode or passive mode between client and proxy and proxy and server. Then client send file data through data connection to proxy, proxy received and send the data to server.

Client:



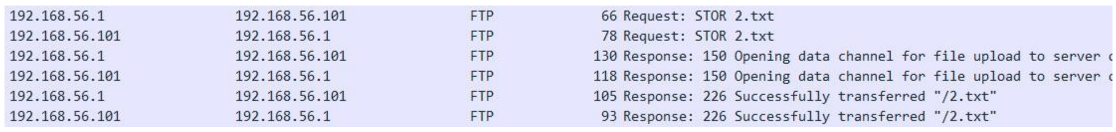
```
状态:    开始上传 C:\Users\Administrator.WIN7U-20140831D\Desktop\ftptest\2.txt
状态:    文件传输成功，传输了 11 字节 (用时1 秒)
```

Server:



```
(000020)2016/6/17 15:17:27 - stone (192.168.56.101)> STOR 2.txt
(000020)2016/6/17 15:17:27 - stone (192.168.56.101)> 150 Opening data channel for file upload to server of "/2.txt"
(000020)2016/6/17 15:17:27 - stone (192.168.56.101)> 226 Successfully transferred "/2.txt"
```

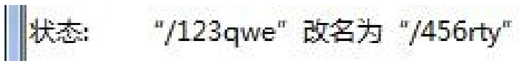
Wireshark:



```
192.168.56.1        192.168.56.101      FTP     66 Request: STOR 2.txt
192.168.56.101      192.168.56.1        FTP     78 Request: STOR 2.txt
192.168.56.1        192.168.56.101      FTP    130 Response: 150 Opening data channel for file upload to server o
192.168.56.101      192.168.56.1        FTP    118 Response: 150 Opening data channel for file upload to server o
192.168.56.1        192.168.56.101      FTP    105 Response: 226 Successfully transferred "/2.txt"
192.168.56.101      192.168.56.1        FTP     93 Response: 226 Successfully transferred "/2.txt"
```

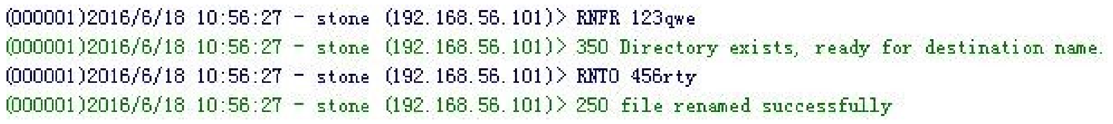
5.5 RNFR and RNTO

Client:



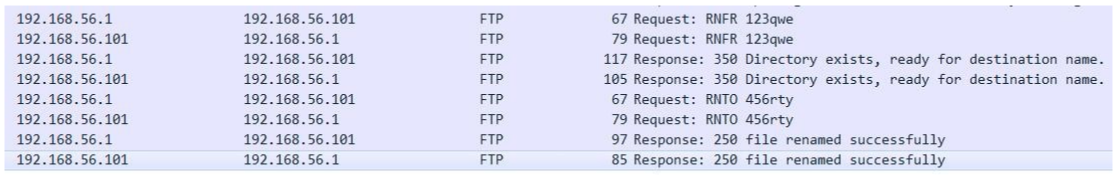
```
状态:    "/123qwe" 改名为 "/456rty"
```

Server:



```
(000001)2016/6/18 10:56:27 - stone (192.168.56.101)> RNFR 123qwe
(000001)2016/6/18 10:56:27 - stone (192.168.56.101)> 350 Directory exists, ready for destination name.
(000001)2016/6/18 10:56:27 - stone (192.168.56.101)> RNTO 456rty
(000001)2016/6/18 10:56:27 - stone (192.168.56.101)> 250 file renamed successfully
```

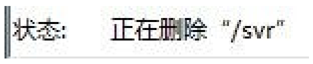
Wireshark:



```
192.168.56.1      192.168.56.101   FTP    67 Request: RNFR 123qwe
192.168.56.101    192.168.56.1     FTP    79 Request: RNFR 123qwe
192.168.56.1      192.168.56.101   FTP   117 Response: 350 Directory exists, ready for destination name.
192.168.56.101    192.168.56.1     FTP   105 Response: 350 Directory exists, ready for destination name.
192.168.56.1      192.168.56.101   FTP    67 Request: RNTO 456rty
192.168.56.101    192.168.56.1     FTP    79 Request: RNTO 456rty
192.168.56.1      192.168.56.101   FTP    97 Response: 250 file renamed successfully
192.168.56.101    192.168.56.1     FTP    85 Response: 250 file renamed successfully
```

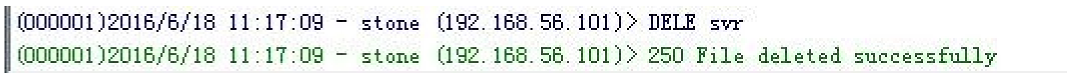
5.6 DELE

Client:



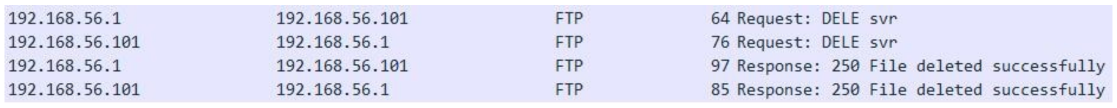
状态: 正在删除 “/svr”

Server:



```
(000001)2016/6/18 11:17:09 - stone (192.168.56.101)> DELE svr
(000001)2016/6/18 11:17:09 - stone (192.168.56.101)> 250 File deleted successfully
```

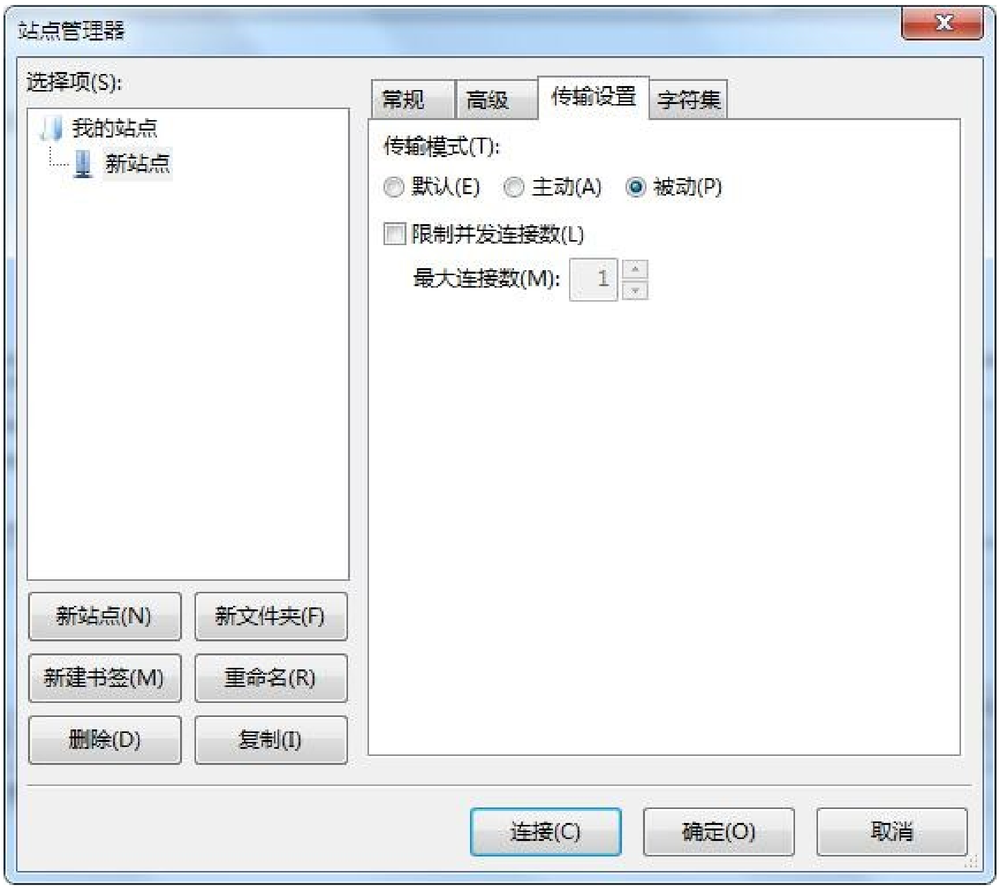
Wireshark:



```
192.168.56.1      192.168.56.101   FTP   64 Request: DELE svr
192.168.56.101    192.168.56.1     FTP   76 Request: DELE svr
192.168.56.1      192.168.56.101   FTP   97 Response: 250 File deleted successfully
192.168.56.101    192.168.56.1     FTP   85 Response: 250 File deleted successfully
```

5.7 Passive Mode

Client:



传输模式(T):
默认(E) 主动(A) [✓ 被动(P)]
☐ 限制并发连接数(L)
最大连接数(M): 1

Server:

```
(000003)2016/6/18 11:35:55 - stone (192.168.56.101)> PASV
(000003)2016/6/18 11:35:55 - stone (192.168.56.101)> 227 Entering Passive Mode (192,168,56,1,126,247)
```

Wiresharek:

```
192.168.56.1          192.168.56.101        FTP        60 Request: PASV
192.168.56.101        192.168.56.1          FTP        72 Request: PASV
192.168.56.1          192.168.56.101        FTP       116 Response: 227 Entering Passive Mode (192,168,56,1,126,247)
192.168.56.101        192.168.56.1          FTP       106 Response: 227 Entering Passive Mode (192,168,56,101,126,247)
```

Appendix: Source Codes

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <string.h>
#include <fcntl.h>


#define MAXSIZE 1024
```

```
int main(int argc, const char *argv[])
{

    char *host = "192.168.56.1";
    char *wellKnownPort = "21";

    int proxy_cmd_socket    = 0;//proxy listen socket
    int accept_cmd_socket   = 0;//proxy accept socket
    int connect_cmd_socket = 0;//proxy connect socket
    int proxy_data_socket    = 0;//proxy listen socket
    int accept_data_socket   = 0;//proxy accept socket
    int connect_data_socket = 0;//proxy connect socket

    int selectResult = 0;//return value of select
    int select_sd = 10;   //range of file description in select()

    char filename[MAXSIZE];
    int choiceFlag;//choice flag of 0 or 1, 0 is upload
    int sendcache = 0; ;//send cache or not, 0 is not send
    int cachemode = 0; //cached or not, 0 is not cached
    int filefd;//fd for file operation
    int modeflag;//0 is active, 1 is passive
    int dataport;//port number of opposite

    fd_set master_set, working_set;//set of file description
    FD_ZERO(&master_set);//clear master_set


    struct timeval timeout;//structure of time in select()
    bzero(&timeout, sizeof(timeout));//set timeout printer to all 0

    //bind() and listen() for proxy_cmd_socket
    if ((proxy_cmd_socket = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket() failed.\n");
        exit(0);
    }

    struct sockaddr_in IPAddr;
    memset(&IPAddr, 0, sizeof(IPAddr));
    IPAddr.sin_family = AF_INET;
    IPAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    IPAddr.sin_port = htons(21);

   if ((bind(proxy_cmd_socket, (struct sockaddr *) &IPAddr, sizeof(IPAddr))) < 0) {
        printf("bind() failed.\n");
        exit(0);
}

if (listen(proxy_cmd_socket, 10) < 0) {
        printf("listen error.\n");
        exit(0);
}

FD_SET(proxy_cmd_socket, &master_set);


timeout.tv_sec = 120;     //timeput time of select()
timeout.tv_usec = 0;      //ms

while (1) {
        int i;
        FD_ZERO(&working_set);//clear master_set
        memcpy(&working_set, &master_set, sizeof(master_set));

        selectResult = select(select_sd, &working_set, NULL, NULL, &timeout);

        // fail
        if (selectResult < 0) {
                printf("select() failed\n");
                exit(0);
        }

        // timeout
        if (selectResult == 0) {
                printf("select() timed out.\n");
                continue;
        }


        for (i = 0; i < select_sd; i++) {
                if (FD_ISSET(i, &working_set)) {
                        //decide whether the changing socket is in the working_set
                        if (i == proxy_cmd_socket) {

                                struct sockaddr_in addr_pcmd;
                                int addr_pcmd_len;
                                addr_pcmd_len = sizeof(addr_pcmd);
                                accept_cmd_socket   =   accept(proxy_cmd_socket,   (struct   sockaddr
            *)&addr_pcmd, &addr_pcmd_len);

                                if (accept_cmd_socket < 0) {
                                    printf("accept() failed.\n");
                                    exit(0);
                                }

                                if ((connect_cmd_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
            {
                                    printf("socket() failed.\n");
                                    exit(0);
                                }

                                struct sockaddr_in addr_ccmd;
                                memset(&addr_ccmd, 0, sizeof(addr_ccmd));
                                addr_ccmd.sin_family = AF_INET;
                                addr_ccmd.sin_addr.s_addr = inet_addr(host);
                                addr_ccmd.sin_port = htons(atoi(wellKnownPort));

                                if (connect(connect_cmd_socket, (struct sockaddr *)&addr_ccmd,
            sizeof(addr_ccmd)) < 0) {
                                    printf("connect() failed.\n");
                                }

                                printf("connect to server successfully\n");

                                FD_SET(accept_cmd_socket, &master_set);
                                FD_SET(connect_cmd_socket, &master_set);
                        }

                        if (i == accept_cmd_socket) {
                                char buff[MAXSIZE];
                                int nread;
                                sleep(5);
                                if ((nread = read(i, buff, MAXSIZE - 1)) == 0) {
                                    close(i);//if can't get any infomation, then shut down Socket
                                    close(connect_cmd_socket);
                                    //after socket closed, FD_CLR will remove socket out from
            master_set, so that select() will not monitor the closed socket
                                    FD_CLR(i, &master_set);
                                    FD_CLR(connect_cmd_socket, &master_set);
                                }
                                else {
                                    nread = read(i, buff, MAXSIZE - 1);
                                    buff[nread] = '\0';
                                    printf("accept_cmd_socket() successfully.\n");

                                    //PORT
                                    if (strncmp(buff, "PORT", 4) == 0) {
                                        char *j;
                                        int p1, p2;

                                        j = strrchr(buff, ',');
                                        p1 = atoi(j+1);
                                        *j = '\0';
                                        j = strrchr(buff, ',');
                                        p2 = atoi(j+1);
                                        sprintf(buff, "PORT 192,168,56,101,%d,%d\r\n", p1, p2);
                                        nread = strlen(buff);

                                        modeflag = 0;//active

                                        dataport = p1 * 256 + p2;

                                        if (FD_ISSET(proxy_data_socket, &master_set)) {
                                            close(proxy_data_socket);
                                            FD_CLR(proxy_data_socket, &master_set);
                                        }


                                        if ((proxy_data_socket = socket(PF_INET, SOCK_STREAM,
            0)) < 0) {
                                            printf("p_d_s socket() failed.\n");
                                            exit(0);
                                        }

                                        //bind() and listen() for proxy_data_socket
                                        struct sockaddr_in addr_pdata;
                                        memset(&addr_pdata, 0, sizeof(addr_pdata));
                                        addr_pdata.sin_family = AF_INET;
                                        addr_pdata.sin_addr.s_addr = htonl(INADDR_ANY);
                                        addr_pdata.sin_port = htons(dataport);

                                        if ((bind(proxy_data_socket, (struct sockaddr *) &addr_pdata,
            sizeof(addr_pdata))) < 0) {
                                            printf("p_d_s bind() failed.\n");
                                            exit(0);
                                        }
```

```
                if (listen(proxy_data_socket, 10) < 0) {
                    printf("listen error.\n");
                    exit(0);
                }

                FD_SET(proxy_data_socket, &master_set);
            }

            //PASV
            if (strncmp(buff, "PASV", 4) == 0) {
                modeflag = 1;
            }

            //RETR
            if (strncmp(buff, "RETR", 4) == 0) {
                choiceFlag = 1;
                buff[nread] = '\0';
                strcpy(filename, strtok(buff, "\r"));

                int openfd, j;
                openfd = open(filename, O_RDONLY);
                if (openfd < 0) {
                    close(openfd);
                    j = 0;
                }
                else {
                    close(openfd);
                    j = 1;
                }

                if (j == 1) {
                    printf("file cached\n");
                    if (modeflag == 1) {//passive
                        cachemode = 1;      //do not need to cache
                    }
                    if (modeflag == 0) {//active
                        char openDataConnection[50];//info of 150 response
                        char closeDataConnection[50];//info of 226 response

                        sprintf(openDataConnection,"150    Opening    data
channel for \"/%s\"\r\n", filename);
                        write(accept_cmd_socket,        openDataConnection,
strlen(openDataConnection));
```

```
                    if  ((connect_data_socket  =  socket(AF_INET,
SOCK_STREAM, 0)) < 0) {
                        printf("c_d_s socket() failed\n");
                        exit(0);
                    }

                    struct sockaddr_in addr_cdata;

                    memset(&addr_cdata, 0, sizeof(addr_cdata));
                    addr_cdata.sin_family = AF_INET;
                    addr_cdata.sin_addr.s_addr = inet_addr(host);
                    addr_cdata.sin_port = htons(dataport);

                    if  (connect(connect_data_socket,  (struct  sockaddr
*)&addr_cdata, sizeof(addr_cdata)) < 0) {
                        printf("c_d_s connect() failed.\n");
                    }
                    printf("c_d_s connect() successfully.\n");

                    int sendfd;
                    sendfd = open(filename, O_RDONLY);
                    if (sendfd < 0) {
                        printf("open() faild.\n");
                        exit(0);
                    }
                    else {
                        char content[MAXSIZE];
                        int nread1;
                        while  ((nread1  =  read(sendfd,  content,
MAXSIZE - 1)) > 0) {
                            write(connect_data_socket,      content,
nread1);
                        }
                        close(sendfd);
                    }


                    sprintf(closeDataConnection,  "226  Successfully
transferred \"/%s\"\r\n", filename);
                    write(accept_cmd_socket,    closeDataConnection,
strlen(closeDataConnection));
                    close(connect_data_socket);
                }
                continue;
```

```
            }else {
                cachemode = 0;
            }
        }

        //STOR
        if (strncmp(buff, "STOR", 4) == 0) {
            choiceFlag = 0;//upload
        }

    write(connect_cmd_socket, buff, nread);
    buff[nread] = '\0';
    printf("handle client CMD successfully.\n");
}


if (i == connect_cmd_socket) {
    char buff[MAXSIZE];
    int nread2;
    nread2 = read(i, buff, MAXSIZE- 1);
    if (nread2 == 0) {
        close(i);
        close(accept_cmd_socket);

        FD_CLR(i, &master_set);
        FD_CLR(accept_cmd_socket, &master_set);
    }else {
        buff[nread2] = '\0';
        printf("get server CMD successfully.\n");
        //227
        if (strncmp(buff, "227", 3) == 0) {
            char *j;
            int p1, p2;

            j = strrchr(buff, ',');
            p1 = atoi(j+1);
            *j = '\0';
            j = strrchr(buff, ',');
            p2 = atoi(j+1);

            char pasvResponse[100];
            sprintf(pasvResponse, "227 Entering Passive Mode (192,168,56,101,%d,%d)\r\n", p1, p2);
```

```
                            int responseLen;
                            responseLen = strlen(buff);
                            modeflag = 1;
                            dataport = p1 * 256 + p2;
                            if (FD_ISSET(proxy_data_socket, &master_set)) {
                                close(proxy_data_socket);
                                FD_CLR(proxy_data_socket, &master_set);
                            }

                            if ((proxy_data_socket = socket(PF_INET, SOCK_STREAM,
0)) < 0) {
                                printf("p_d_s socket() failed.\n");
                                exit(0);
                            }

                            struct sockaddr_in addr_pdata;

                            memset(&addr_pdata, 0, sizeof(addr_pdata));
                            addr_pdata.sin_family = AF_INET;
                            addr_pdata.sin_addr.s_addr = htonl(INADDR_ANY);
                            addr_pdata.sin_port = htons(dataport);

                            if ((bind(proxy_data_socket, (struct sockaddr *) &addr_pdata,
sizeof(addr_pdata))) < 0) {
                                printf("p_d_s bind() failed\n");
                                exit(0);
                            }

                            if (listen(proxy_data_socket, 10) < 0) {
                                printf("listen error.\n");
                                exit(0);
                            }

                            FD_SET(proxy_data_socket, &master_set);
                        }
                        }
                        write(accept_cmd_socket, buff, nread2);

                        printf("handle server cmd successfully.\n");
                    }
                }

                if (i == proxy_data_socket) {
                    if (modeflag == 0) {//active
```

```
struct sockaddr_in addr_adata;
int addr_adata_len;
addr_adata_len = sizeof(addr_adata);
accept_data_socket = accept(proxy_data_socket, (struct sockaddr
*)&addr_adata, &addr_adata_len);
if (accept_data_socket < 0) {
	printf("a_d_s accept() failed.\n");
	exit(0);
}


if ((connect_data_socket = socket(AF_INET, SOCK_STREAM, 0))
< 0) {
	printf("c_d_s socket() failed.\n");
	exit(0);
}

struct sockaddr_in addr_cdata2;
memset(&addr_cdata2, 0, sizeof(addr_cdata2));
addr_cdata2.sin_family = AF_INET;
addr_cdata2.sin_addr.s_addr = inet_addr(host);
addr_cdata2.sin_port = htons(dataport);

if (connect(connect_data_socket, (struct sockaddr *)&addr_cdata2,
sizeof(addr_cdata2)) < 0) {
	printf("connect() failed\n");
}

printf("connect to c_d_s successfully.\n");


FD_SET(accept_data_socket, &master_set);
FD_SET(connect_data_socket, &master_set);

//cache
char *extension;
int k,openfd, j;
if (strstr(filename, ".") == NULL)
	k = 0;
if (choiceFlag == 0)//
	k = 0;

strtok(filename, ".");
extension = strtok(NULL, ".");
```

```
                    if (strcmp(extension, "jpg") == 0 || strcmp(extension, "pdf") == 0) {
                        openfd = open(filename, O_RDONLY);

                        if (openfd < 0) {
                            close(openfd);
                            j = 0;
                        }else {
                            close(openfd);
                            j = 1;
                        }
                        if (j == 1) {
                            k = 1;
                        }
                    }

                    if (k == 1) {//do cache
                        sendcache = 1;
                        filefd = creat(filename, 0644);
                        if (filefd < 0) {
                            printf("create() faild.\n");
                            exit(0);
                        }
                        printf("Create cache file successfully.\n");
                    } else {
                        sendcache = 0;
                    }
                }

                if (modeflag == 1) {
                    if (choiceFlag == 0) {//upload
                        struct sockaddr_in addr_adata2;
                        int addr_data_len2;
                        addr_data_len2 = sizeof(addr_adata2);
                        accept_data_socket    =    accept(proxy_data_socket,    (struct
sockaddr *)&addr_adata2, &addr_data_len2);

                        if (accept_data_socket < 0) {
                            printf("a_d_s accept() failed\n");
                            exit(0);
                        }

                        if ((connect_data_socket = socket(AF_INET, SOCK_STREAM,
0)) < 0) {
```

```
                    printf("c_d_s socket() failed\n");
                    exit(0);
                }

                struct sockaddr_in addr_cdata3;
                memset(&addr_cdata3, 0, sizeof(addr_cdata3));
                addr_cdata3.sin_family = AF_INET;
                addr_cdata3.sin_addr.s_addr = inet_addr(host);
                addr_cdata3.sin_port = htons(dataport);

                if (connect(connect_data_socket, (struct sockaddr
*)&addr_cdata3, sizeof(addr_cdata3)) < 0) {
                    printf("connect() failed\n");
                }

                printf("connect to c_d_s.\n");

                FD_SET(accept_data_socket, &master_set);
                FD_SET(connect_data_socket, &master_set);
            }

            if (choiceFlag == 1) {//download
                if (cachemode == 1) {//cached
                    cachemode = 0;
                    char openDataConnection2[50];
                    int sendfd2;

                    sprintf(openDataConnection2, "150 Opening data channel
for \"/%s\"\r\n", filename);

                    struct sockaddr_in addr_adata3;
                    int addr_adata_len2;
                    char content2[MAXSIZE];

                    addr_adata_len2 = sizeof(addr_adata3);
                    accept_data_socket = accept(proxy_data_socket, (struct
sockaddr *)&addr_adata3, &addr_adata_len2);
                    if (accept_data_socket < 0) {
                        printf("a_d_s accept() failed\n");
                        exit(0);
                    }

                    write(accept_cmd_socket, openDataConnection2,
strlen(openDataConnection2));
```

```
                        sendfd2 = open(filename, O_RDONLY);
                        if (sendfd2        < 0) {
                            printf("open() faild.\n");
                            exit(0);
                        }else {
                            int nread3;
                            while ((nread3 = read(sendfd2, content2, MAXSIZE
- 1)) > 0) {

                                write(accept_data_socket, content2, nread3);
                            }
                            close(sendfd2);
                        }

                        char successesResponse[50];
                        sprintf(successesResponse, "226 Successfully transferred
\"/%s\"\r\n", filename);
                        write(accept_cmd_socket,                successesResponse,
strlen(successesResponse));
                        close(accept_data_socket);

                    }else {//not cached
                        struct sockaddr_in addr_adata4;
                        int addr_adata_len4,j,k, openfd2;
                        char *extension2;

                        addr_adata_len4 = sizeof(addr_adata4);
                        accept_data_socket = accept(proxy_data_socket, (struct
sockaddr *)&addr_adata4, &addr_adata_len4);

                        if (accept_data_socket < 0) {
                            printf("a_d_s accept() failed\n");
                            exit(0);
                        }

                        if      ((connect_data_socket       =       socket(AF_INET,
SOCK_STREAM, 0)) < 0) {
                            printf("c_d_s socket() failed\n");
                            exit(0);
                        }

                        struct sockaddr_in addr_cdata4;
                        memset(&addr_cdata4, 0, sizeof(addr_cdata4));
                        addr_cdata4.sin_family = AF_INET;
```

```
                                addr_cdata4.sin_addr.s_addr = inet_addr(host);
                                addr_cdata4.sin_port = htons(dataport);

                                if (connect(connect_data_socket, (struct sockaddr
*)&addr_cdata4, sizeof(addr_cdata4)) < 0) {
                                    printf("c_d_s connect() failed\n");
                                }

                                printf("connect to c_d_s.\n");

                                FD_SET(accept_data_socket, &master_set);
                                FD_SET(connect_data_socket, &master_set);

                                if (strstr(filename, ".") == NULL)
                                    k = 0;
                                if (choiceFlag == 0)
                                    k = 0;

                                strtok(filename, ".");
                                extension2 = strtok(NULL, ".");

                                if (strcmp(extension2, "jpg") == 0 || strcmp(extension2,
"pdf") == 0) {

                                    openfd2 = open(filename, O_RDONLY);
                                    if (openfd2     < 0) {
                                        close(openfd2);
                                        j = 0;
                                    }else {
                                        close(openfd2);
                                        j = 1;
                                    }
                                    if (j = 1) {
                                        k = 1;
                                    }
                                }

                                if (k = 1) {
                                    sendcache = 1;//send cache
                                    filefd = creat(filename, 0644);
                                    if (filefd < 0) {
                                        printf("create() faild.\n");
                                        exit(0);
                                    }
                                    printf("Create file for cache.\n");
```

```
                }else {
                    sendcache = 0;
                }
            }
        }
    }
}

if (i == accept_data_socket) {
    char buff[MAXSIZE];
    int nread4;
    nread4 = read(i, buff, MAXSIZE- 1);

    if (nread4 == 1) {
        write(connect_data_socket, buff, nread4);
        buff[nread4] = '\0';
        if (sendcache) {
            write(filefd, buff, nread4);
        }
    }else {
        close(i);
        close(connect_data_socket);
        FD_CLR(accept_data_socket, &master_set);
        FD_CLR(connect_data_socket, &master_set);
        if (sendcache) {
            close(filefd);
        }
    }
}

if (i == connect_data_socket) {
    char buff[MAXSIZE];
    int nread5;
    nread5 = read(i, buff, MAXSIZE- 1);

    if (nread5 == 1) {
        write(accept_data_socket, buff, nread5);
        buff[nread5] = '\0';
        if (sendcache) {
            write(filefd, buff, nread5);
        }
    }else {
        close(i);
        close(accept_data_socket);
```

```
                                        FD_CLR(accept_data_socket, &master_set);
                                        FD_CLR(connect_data_socket, &master_set);
                                        if (sendcache) {
                                                close(filefd);
                                        }
                                }
                        }
                }
        }
    }
    return 0;
}
```