Computer Networks 159.334

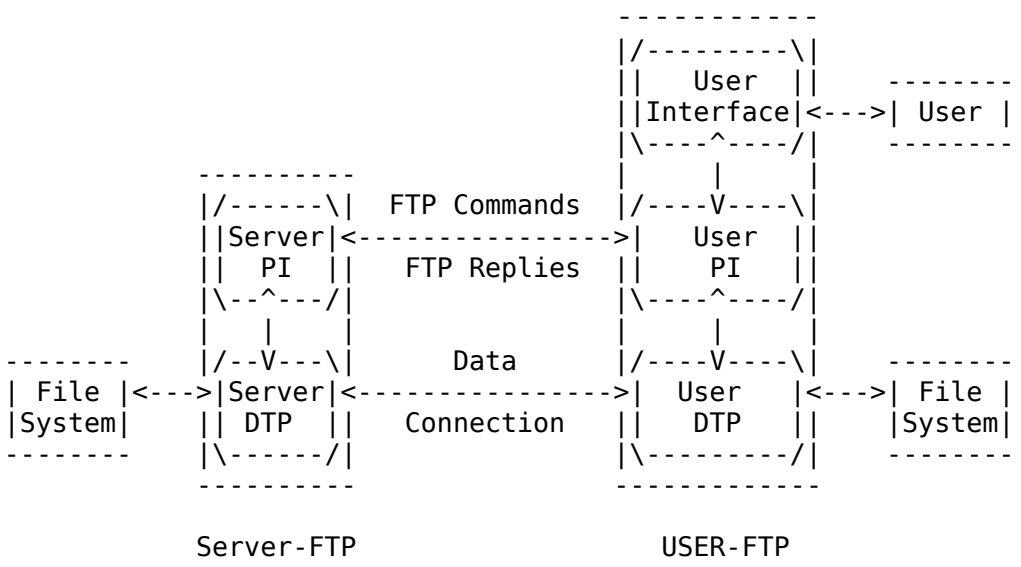
FTP protocol information based on RFC959 **A. L. C. Barczak/2012**

---

1 - Introduction

The RFC959 (http://www.faqs.org/rfcs/rfc959.html) describes the FTP protocol in details. This paper describes the FTP protocol commands and messages to be used in the first assignment. Further details should be found in the RFC itself.

The diagram below shows a general scheme that describes the FTP server and client.



NOTES:
1. The data connection may be used in either direction.
2. The data connection need not exist all of the time.
3. PI = Protocol Interpreter
4. DTP = Data Transfer Process

Diagram 1 – The FTP protocol with two connections.

2 – Minimum Implementation

Normally servers would have to implement at least the following:

TYPE - ASCII Non-print

MODE - Stream
STRUCTURE - File, Record
COMMANDS - USER, QUIT, PORT,
TYPE, MODE, STRU, for the default values
RETR, STOR,
NOOP.

The default values for transfer parameters are:

```
TYPE - ASCII Non-print
```

```
MODE - Stream
STRU – File
```

However, we will use USER, PASS, QUIT, PORT, PASV, LIST (NLST if the client needs it), STOR and RETR in the assignment. The arguments are:

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
QUIT <CRLF>
PORT <SP> <host-port> <CRLF>
PASV <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
LIST [<SP> <pathname>] <CRLF>
NLST [<SP> <pathname>] <CRLF>
```

The server must respond with a 3 digits code. The codes follow a certain logic. For example, the first digit can be 1, 2, 3, 4, or 5. Normally a 1xy code means a preliminary positive reply, 2xy is used for a positive completion reply and 3xy is used for a positive intermediate reply. A 4xy code is a transient negative reply, while a 5xy code indicates a permanent negative completion reply. Follow examples to be used in the assignment:

125 Data connection already open; transfer starting.
150 File status okay; about to open data connection.
200 Command okay.
202 Command not implemented, superfluous at this site.
211 System status, or system help reply.
215 NAME system type.
    Where NAME is an official system name from the list in the
    Assigned Numbers document.
220 Service ready for new user.
221 Service closing control connection.
    Logged out if appropriate.
225 Data connection open; no transfer in progress.
226 Closing data connection.
    Requested file action successful (for example, file transfer or file abort).
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
230 User logged in, proceed.
250 Requested file action okay, completed.
257 "PATHNAME" created.
331 User name okay, need password.
332 Need account for login.
350 Requested file action pending further information.
421 Service not available, closing control connection.
    This may be a reply to any command if the service knows it must shut down.
425 Can't open data connection.
426 Connection closed; transfer aborted.
450 Requested file action not taken.
    File unavailable (e.g., file busy).
451 Requested action aborted: local error in processing.
452 Requested action not taken.
    Insufficient storage space in system.
500 Syntax error, command unrecognized.
    This may include errors such as command line too long.
501 Syntax error in parameters or arguments.
502 Command not implemented.

503 Bad sequence of commands.
504 Command not implemented for that parameter.
530 Not logged in.
532 Need account for storing files.
550 Requested action not taken.
    File unavailable (e.g., file not found, no access).
551 Requested action aborted: page type unknown.
552 Requested file action aborted.
    Exceeded storage allocation (for current directory or dataset).
553 Requested action not taken.
    File name not allowed.

3 – Sequence of command and replies

Usually the server and the client are at a certain state. If the correct command or reply is issued, both programs carry out their work in a certain sequence. Follow the usual sequences for the simple server to be used with standard clients. Each sequence can be used for testing, but there is no guarantee that the client follow this order.
One has to test the server to be sure that the sequence is correct from the client's point of view.

Connection Establishment
- 120
  - 220
- 220
- 421

Login
- USER
  - 230
  - 530
  - 500, 501, 421
  - 331, 332
- PASS
  - 230
  - 202
  - 530
  - 500, 501, 503, 421

Logout
- QUIT
  - 221
  - 500

Transfer parameters
- PORT
  - 200
  - 500, 501, 421, 530
- PASV
  - 227
  - 500, 501, 502, 421, 530

File action commands
- STOR
  - 125, 150
    - (110)
    - 226, 250

425, 426, 451, 551, 552
532, 450, 452, 553
500, 501, 421, 530
RETR
125, 150
(110)
226, 250
425, 426, 451
450, 550
500, 501, 421, 530
LIST
125, 150
226, 250
425, 426, 451
450
500, 501, 502, 421, 530

Informational commands
SYST
215
500, 501, 502, 421

This sequence example does not include all the arguments for the commands and for the replies. For example, the command PORT must carry the IP number and the port number with the appropriate format. Some replies must also carry parameters, such as the response to a PASV command.

4 – Passive X Active connections in FTP protocol

Let's explore the PASV protocol command and the PORT protocol command, and appreciate the differences. For completion, all servers should have both modes implemented because it does not know which one the client might use. For the assignment you can chose to implement either passive or active, or both.

From the client command line application, the mode can be changed (toggled) with the following application command from the ftp prompt:

```
ftp> passive
```

The fundamental difference between passive and active connections is the way the *Data connection* is carried out.

Passive mode:

- the **client** in passive mode will issue PASV
- the **server** will respond with 227 followed by 6 number formatted like this: %d,%d,%d,%d,%d,%d,%d
- the **client** will use the information to **connect** to the **server** on the port informed above
- the Data connection can then be used to send/recv information (usually files/listings)

The IP address is not needed, as you can copy it from the Control connection. But the port number is different. You need to, at least, convert the port number. For compatibility issues, the decimal port number is split into two. The usual format used in TCP is:

```
%d,%d,%d,%d,%d,%d
```

The first 4 integers are part of the IP address, the last 2 are part of the port number. Suppose that the server have the port number 10020 for the data connection, using IP 127.0.0.1. The following lines of C code could be used to send the response to the client:

```
int ptnumber=10020;
sprintf(send_buffer, "227 Passive Mode (%d,%d,%d,%d,%d,
%d)\r\n",127,0,0,1,(ptnumber >> 8),( ptnumber & 0x00FF));
bytes = send(ns, send_buffer, strlen(send_buffer), 0);
```

When the message 227 is received by the client, the Data port on the server is supposed to be opened. Therefore, you have to create two sockets (e.g., `s_data_pasv and ns_data_pasv`), one remote address structure (to copy the IP and Data port to it), and issue `bind()`, `listen()` and `accept()` before the data can be transferred.

Once that is done, the send() and recv() functions to transfer Data should use the socket `ns_data_pasv` (not ns, if ns is being used for the Control connection already).

Active mode:

- the **client** in active mode will issue PORT
- the **client** will send its own IP address and port number, formatted like this: %d,%d,%d,%d,%d,%d,%d
- the **server** will use the information to **connect** to the **client** on the port informed above
- the Data connection can then be used to send/recv information (usually files/listings)

A similar format for the IP and port numbers is used for the PORT command. Assuming that the client is at IP 130.123.246.100 and the client opens its port 10020 for the data connection, the following line will be received by the server:
PORT 130,123,246,100,39,36 *//(equivalent to 130.123.246.100 port 39*256+36=10020)*

The server needs to convert the port number and the IP address if it is going to use it. For debugging purposes, it is convenient to store the port number onto an integer and use the socket libraries for the appropriate conversions. In order to do that, just think of the port number as a 16 bits integer where the two bytes were split into two decimal numbers. To recover the original port:

39 << 8 + 36
or
39*256 + 36 = 10020

In C language, the following lines can be used:

```
unsigned char act_ip[4];
unsigned char act_port[2];
int port_dec;
sscanf(receive_buffer, "PORT %d,%d,%d,%d,%d,
%d",&act_ip[0],&act_ip[1],&act_ip[2],&act_ip[3],&act_port[0],&act_port[1
]);
port_dec=act_port[0];
port_dec=port_dec << 8;
port_dec=port_dec+act_port[1];
data_addr_act.sin_port=htons(port_dec);
```

When the message PORT is sent by the client, the Data port on the client will be available for the server to connect. Therefore, you have to create one socket for the active connection (e.g., `s_data_act`), one remote address structure (to copy the IP and Data port to it), and issue `socket()` and `connect()` before the data can be transferred. Once that is done, the send() and recv() functions should use the socket

`s_data_act` (not ns, if ns is being used for the Control connection already).