COMP 431 — INTERNET SERVICES & PROTOCOLS

Spring 2018

Programming Homework 5, Mar 9

Due: March 29, 8:30 AM

Building an FTP Client/Server System Using Sockets

You can now put all the pieces together and create a working FTP client and server. In this assignment you will extend your FTP "server" program (HW2) and your FTP "client" program (HW3) to interoperate over a network by using TCP sockets. To do this you will need to implement all elements of the FTP protocol and replace some `stdin`/`stdout` I/O and parts of your file I/O with socket I/O. Further, the file I/O to read and write a file that was entirely contained in the HW2 program will be split between the client and server – the server will read all the bytes in a requested file, write those bytes to the client over an ***FTP-data*** connection, and the client will read these bytes from the connection and write them to a local file.

FTP Client Program

Your FTP client program should take one command line argument: an initial port number for a "welcoming" socket that the client will use to allow the server to make an FTP-data connection. As in HW3, your client should accept input requests from a human user using standard input. The specifications of the three types of allowed input lines are given in HW3. The client should echo each user input line to standard output along with the corresponding response line specified in HW3.

When a valid CONNECT request is accepted from the user, the client should attempt to establish a TCP socket connection to the server program which it expects to be running on the host and port specified in the user's input. This connection is the ***FTP-control*** connection and it should not be closed until the user enters another valid CONNECT request (which will initiate a new FTP-control connection). When your client successfully connects to the server it must be prepared to receive the server's greeting reply ("220 COMP 431 FTP server ready.") on the FTP-control connection. *When your client program receives any reply from the server on the FTP-control connection, it should display the output specified for the FTP Reply Parser program in HW3*. Thus, if the user input line is :

CONNECT classroom.cs.unc.edu 9000

The following lines would be displayed on standard output.

CONNECT classroom.cs.unc.edu 9000
CONNECT accepted for FTP server at host classroom.cs.unc.edu and port 9000
FTP reply 220 accepted. Text is: COMP 431 FTP server ready.

If the client is unable to connect to the server, it should write "CONNECT failed" to standard output and prepare to read the next input from the user. After the FTP-control connection has been established and the initial server reply received, the client should send the four-command sequence specified in HW3 (USER, PASS, SYST, and TYPE) to the server using the FTP-control connection. Before your client program sends a command to the server on the FTP-control connection, it should first echo that command to standard output. After each command in this sequence is sent to the server, the client should read and process the corresponding reply (as specified in HW2 and HW3) received from the server on the FTP-control connection before sending the next one.

If the user enters additional valid CONNECT requests the existing FTP-control connection is closed and a new one established using the host and port specified in the CONNECT.

When a valid GET request is accepted from the user, the client should send the two-command sequence specified in HW3 (PORT, RETR) to the server on the FTP-control connection and process the server's reply to each command before proceeding to the next. The PORT command's parameter should specify the IP address of the host where the client program is running and the port number specified as a command line argument to the client. Each time a new PORT command is sent to the server, the port number should be incremented by 1 (NOTE: the client will be using a different "welcoming" socket and associated port for each FTP-data connection). Before sending the PORT command to the server, the client should create a "welcoming" socket specifying the port number used in the PORT command to be sure that port is ready for the server's FTP-data connection. If the "welcoming" socket cannot be created, the client should write "GET failed, FTP-data port not allocated." to standard output and read the next user input line.

After the RETR command has been sent to the server, the client should accept the server's FTP-data connection on the "welcoming" socket and then read the bytes for the requested file from the new socket created for that connection. The client should continue to read data bytes from the server until the End-of-File (EOF) is indicated when the server closes the FTP-data connection (the client should also close the FTP-data connection at EOF). Note that the server replies to the RETR command are received on the FTP-control connection. Only file data is received on the FTP-data connection.

The file data read from the FTP-data connection should be written to a new file in a directory named `retr_files` in the client's current working directory (be sure to create this directory using the UNIX "mkdir" command before you try to execute your client). The file's name in the `retr_files` directory should have the form "filexxx" where xxx is the number of valid RETR commands your client has recognized so far during this execution of the program. For example, the first valid RETR command will result in writing the data to the file named `file1` in the `retr_files` directory; the second valid RETR command will result in a file named `file2`, etc. The file transferred with a RETR command may have arbitrary content and should be written as a stream of bytes (using file I/O).

If the client receives an unexpected reply from the server or receives a reply that indicates an error condition (4xx or 5xx reply-code), the current sequence of commands to the server should be ended and a new user input line read from standard input. A user's QUIT request should be handled as specified in HW3 with the additional requirements that the client should send the FTP QUIT command to the server, receive the reply shown below, close the FTP-control connection and then exit. The server reply to QUIT is (Note: this reply was not used in HW2 or HW3).

221 Goodbye.

FTP Server Program

Your FTP server program requires only a single command line argument: the port number on which it should accept FTP-control connections from clients.

The outline of the server's execution is as follows-

```
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind(("", <PORT_NUMBER>))
server_socket.listen(1)  # Allow only one connection at a time
while True:
    <Accept a client control connection>
    <Communicate with the client>
```

(`server_socket.setsockopt()` is called before bind to allow reusing the same port. Otherwise the same port number cannot be used for a while after terminating the server.)

When your server accepts an FTP-control connection from a client it should immediately send the server "greeting" reply on that connection:

220 COMP 431 FTP server ready.

After sending this message the server must receive valid USER and PASS commands as specified in HW2. Once a valid PASS command has been received, the server enters its command processing loop processing commands received on the FTP-control connection and sending the appropriate reply (as specified in HW2) to the client over the FTP-control connection. This continues until either the client closes the FTP-control connection or the QUIT command is received. In either case the server closes the FTP-control connection, resets any internal data to its initial state, and accepts the next connection from a client.

When a valid RETR command is received, the server should read the file. As in HW2, the parameter of the RETR command specifies the name of a file located in the current working directory from which your server was executed or it specifies the complete pathname for a file located within the space of subdirectories below (rooted in) the server's working directory. If an error occurs (the file does not exist or access is denied), a reply with a reply-code of 550 is sent to the client and no further processing of that command is done. If there is no error, a reply with a reply-code of 150 is sent to the client and the server attempts to make a TCP connection to the host and port specified in the most recent valid PORT command (to establish an FTP-data connection). If the connection attempt fails, the server should reply:

425 Can not open data connection.

(Note: this is a new reply that was not used in HW2 or HW3)

If establishing the FTP-data connection is successful, the server then reads the bytes from the requested file and writes them to the client on the FTP-data connection. When all the bytes from the file have been sent to the client, the server closes the FTP-data connection and sends a reply of reply-code 250 to the client over FTP-control connection. Note that the server replies to the RETR command are sent on the FTP-control connection. Only file data is sent on the FTP-data connection.

Finally, your server program should also output the following to standard output. The server echoes all commands received from the client, as well as all responses sent to the client. Specifically, when the

server receives a command from the client, it first echoes the command, and then output its reply to the client (example provided below).

The server program, like most servers, will conceptually never terminate. It will be terminated by some external means such typing *control-C* in the shell.

Some points denoting the changes between the FTP server in this homework in contrast to the 2nd homework-

- A valid QUIT command in this homework should report *"221 Goodbye."* instead of *"200 Command OK."*
- Successful RETR commands in this homework should try to establish a new FTP data connection with the client and send the file data through that connection. Remember to handle the case of failure to establish connection.
- The grading of homework-2 was lenient due to errors on the reference implementation. You have to fix these issues in your implementation of server for this homework.
    - Between error 503 and 530, error 530 has a higher priority when applicable.

      530 should be the response only for:
        - Any command that appears before a successfully executed USER-PASS pair.

      Error 503 should be reported for other such bad sequence cases.

    - Reporting error 500 or error 501 are not interchangeable for this homework. For errors in command, the server should report error 500. For errors in the parameter or for invalid termination, the server should respond with error 501.

When you are done developing the solution for server, run it with the following command-

python3 -u FTPServer.py <PORT_NUMBER>

(The -u option is to ensure that stdout is unbuffered.)

Notes

For this assignment the client and server programs will execute on *different* computers. You should run your servers and clients on *classroom.cs.unc.edu*, *quintet.cs.unc.edu, or swan5.cs.unc.edu*. The port number you should use is 9000 + the last 4 digits of your social security number. This will minimize the possibilities of a port conflict but it will not guarantee that port conflicts do not occur. Thus your programs *must* be prepared to deal with errors that occur when trying to create a socket on a port number that is in use by someone else.

Your client program should use standard input and output for interaction with a human user as specified above but use socket I/O for all communication with the server and file I/O only to write files that are fetched from the server. The server program shouldn't interact with the user but should interact with the client using socket I/O and output necessary information to standard output (echo the commands and replies) as specified above .

Here is an example showing how your client program's output should look with a correct sequence of valid user requests. **NOTE:** in this example the user's input lines that have been echoed are marked with a "-" to make the example clear. Do NOT include the "-" in your program output).

-CONNECT classroom.cs.unc.edu 9000
CONNECT accepted for FTP server at host classroom.cs.unc.edu and port 9000
FTP reply 220 accepted. Text is: COMP 431 FTP server ready.
USER anonymous
FTP reply 331 accepted. Text is: Guest access OK, send password.
PASS guest@
FTP reply 230 accepted. Text is: Guest login OK.
SYST
FTP reply 215 accepted. Text is: UNIX Type: L8.
TYPE I
FTP reply 200 accepted. Text is: Type set to I.
-GET pictures/jasleen.jpg
GET accepted for pictures/jasleen.jpg
PORT 152,2,131,205,31,144
FTP reply 200 accepted. Text is: Port command successful (152.2.131.205,8080).
RETR pictures/jasleen.jpg
FTP reply 150 accepted. Text is: File status okay.
FTP reply 250 accepted. Text is: Requested file action completed.
-QUIT
QUIT accepted, terminating FTP client
QUIT
FTP reply 221 accepted. Text is: Goodbye.

In addition to what is written on standard output, your client should also have copied the content of `pictures/jasleen.jpg` from the server to the file `retr_files/file1` in the working directory where the client is running.

Correspondingly, the server should output the following:

220 COMP 431 FTP server ready.
USER anonymous
331 Guest access OK, send password.
PASS guest@
230 Guest login OK.
SYST
215 UNIX Type: L8.
TYPE I
200 Type set to I.
PORT 152,2,131,205,31,144
200 Port command successful (152.2.131.205,8080).
RETR pictures/jasleen.jpg
150 File status okay.
250 Requested file action completed.
QUIT
221 Goodbye.

Testing

To aid in testing, sample input and output files are provided. Please note that these sample tests are not comprehensive (i.e., you should test your program much more thoroughly than these test files) – and grading will certainly rely on many additional tests. These sample files are provided simply to aid you in initial testing, as well as catching if your program is making basic formatting/syntax mistakes. Use the provided programs to test your code using the following steps:

First, generate the test files using one of the scripts provided on the course webpage. For example, run the generate_test_0_0.py script:

```
python3 generate_test_0_0.py
```

Start your server:

```
python3 -u FTPServer.py 9000 > 0_0_output_my_server
```

Run your client:

```
python3 FTPClient.py 8000 < 0_0_input > 0_0_output_my_client
```

Finally kill your server with Ctrl+C

Check the correctness using diff:

```
diff 0_0_output_server 0_0_output_my_server

diff 0_0_output_client 0_0_output_my_client
```

If your program works correctly, the diff commands should produce no output. Perform the same steps using the other test generation scripts provided, such as generate_test_1_0.py, generate_test_3_0.py and generate_test_3_1.py.

Grading

To submit your programs for grading, follow the general submission guidelines distributed with Homework 3 and notify the TAs using the link provided on the course webpage. You should have exactly two python files (no other files) in your *COMP431/submissions/HW5* directory. The client program should be named exactly *FTPClient.py* and the server should be named exactly *FTPServer.py*. Your name should be in a comment line near the top of your python programs.

As before, your programs should be neatly formatted and well documented. In general, 80% of your grade for a program will be for correctness, 20% for programming style and documentation. Refer to the handout on programming style and documentation for guidelines. Make sure your solutions are well commented and divided up into methods to receive maximum points on this part.

Grading Rubric:

The 80% correctness grade will have the following distribution:

1. Process valid CONNECT requests (30%):
    - **Client :** Generate correct response message and USER, PASS, SYST, TYPE commands.

- **Server :** Generate corresponding response messages. Although the client you design may not introduce many errors, the server must be able to handle a bad client by checking for possible syntax errors and sequence errors and responding properly.
- **Client :** Read/process the server replies.

2. Handle the case when the client is unable to connect to the server (5%)
3. Process a new CONNECT request while there is an existing connection to the server (existing connection is guaranteed to be terminated with a QUIT command before a new CONNECT request) (5%)
4. Process valid GET requests and retrieve files (30%):
    - **Client :** generate valid PORT and RETR commands.
    - **Server:** Parse the PORT and RETR commands and generate proper responses. Handle errors.
    - Transfer the file from server to client.
5. Handle the case when the requested file does not exist on the server or other errors (5%)
6. Process a valid QUIT command (5%)

Note: Since it's not specified, you don't have to handle the case when there is a connection error in the FTP-data connection between server and client. But for the FTP-control connection, you MUST handle the connection errors carefully.