COMP 431 — INTERNET SERVICES & PROTOCOLS

Spring 2025

Homework 3, February 7

Due: February 20, 12:15 PM

Building an FTP Client/Server System Using Sockets

In this assignment you will extend your FTP "server" program from Homework 1, and an incomplete FTP "client" program that we provide to interoperate over a network 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 file I/O with socket I/O. Further, the file I/O to read and write a file that was entirely contained in the HW1 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.

You may find it helpful to review slide-set 3 and its corresponding video lecture. Both are linked on the course webpage.

FTP Session

An FTP Session represents an interaction between an FTP Client and FTP Server over a network. Before an FTP session can be initiated, the server must be *listening* on an established port for client requests. Now, an FTP session is initiated by the client by connecting to the server's listening port. This connection is the ***FTP-control*** connection. The FTP-control connection is used by the client to send FTP commands and used by the server to send FTP replies. This connection persists for the entire session, i.e. until the client quits, or the client enters a new valid CONNECT command.

Before the client requests a file from the server, the client opens a port for listening. When the server receives the request, it connects to the client's listening port. This connection is the ***FTP-data*** connection. The FTP-data connection is used by the server to send the requested files to the client. A new FTP-data connection is initiated for each PORT command the client sends.

So, each FTP session utilizes two distinct connections.

FTP Client Program

We provide an FTP client service that implements command parsing in the form of two Python scripts. One half of this assignment is to complete the implementation of the client in FTP_Client.py (will be released soon; we will make an announcement on piazza when it is ready). If you haven't already, read the FTP client description on the course webpage before proceeding further.

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 HW1. 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 HW1) 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 specified file. As in HW1, 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 (i.e. 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 on the FTP-control connection:

```
425 Can not open data connection.
```

Note that this is a new reply that was not used in HW1.

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 with reply-code 250 to the client over the FTP-control connection. Note that all server replies are sent on the FTP-control connection. Only file data is sent on the FTP-data connection.

Finally, your server program should also echo all commands received from the client, as well as all responses sent to the client to standard output. Specifically, when the server receives a command from the client, it first echoes the command, and then sends its reply to the client.

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

Here are some points denoting the changes between the FTP server in this homework in contrast to HW1:

- A valid QUIT command in this homework should report *"221 Goodbye."* instead of *"200 Command OK."*
- Successful RETR commands 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 the FTP-data connection. The server should no longer copy the requested file to a retr_files directory.

Running your client and server

To test your server, run it using the following command:

```
python3 -u FTPServer.py <PORT_NUMBER>
```

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

For this assignment the client and server programs will execute on *different* computers. We provide two course machines (CS login) – *comp431-1sp25.cs.unc.edu* and *comp431-1sp25b.cs.unc.edu*. You should run the server on one machine, and client on the other. **The port number you should use is 9000 + the last 4 digits of your PID.** 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. 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.** The client was run with the command "`python3 -u FTP_Client.py 8080`".

```
-CONNECT comp431-1sp21.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. The server was run with the command "`python3 -u FTP_Server.py 9000`".

```
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. Testing in this assignment will be similar to Homework 1 except there is no input for the server since the server receives its input from the client.

First, generate the client's input, and expected output on the client machine:

```
python3 ClientInput1.py > ClientInput1
```

```
python3 ClientOutput1.py > ClientOutput1
```

Next, generate the server's expected output on the server machine:

```
python3 ServerOutput1.py > ServerOutput1
```

Start your server on one machine. Replace `PORT_NO` with the port number computed using your PID as described above.

```
python3 -u FTP_Server.py PORT_NO > MyServerOutput1
```

Run your client on the second machine:

```
python3 -u FTP_Client.py PORT_NO < ClientInput1 > MyClientOutput1
```

Finally kill your server with Ctrl+C.

Check the correctness using diff.

```
diff ServerOutput1 MyServerOutput1
```

```
diff ClientOutput1 MyClientOutput1
```

If your program works correctly, the diff commands will produce no output. Perform the same steps using the other sample tests provided, and your own tests. Please note that these sample tests are not comprehensive (i.e., you should test your program much more thoroughly than these test files). 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.

Please remember from Homework1 the various requirements, in particular, you should ensure your code passes the autograder for Homework1 as similar test cases will be used in this assignment. The Homework1 Autograder is available for testing your code right now; so, please make use of this if you did not get 100% on Homework1.

PLEASE NOTE THAT GETTING A 100% ON THE AUTOGRADER FOR HW1 DOES NOT MEAN YOUR CODE IS FLAWLESS. THE AUTOGRADER FOR HW1 IS NOT INTENDED TO BE COMPREHENSIVE.

Common mistakes made in Homework1:

A valid USER command followed by some other valid command followed by a PASS command.

Many students allowed the USER command to still accept the PASS command, even though it was separated by the other valid command. This is not correct, as both the other valid command and the PASS command should return a 530 response. Any command that is not a QUIT command, a PASS command, or a USER command should return a 530 response after USER.

A valid PORT command followed by an INVALID RETR command should not close the PORT.

Many students allowed the PORT to close after an invalid RETR command occurred, which should not happen. The PORT should remain open unless a valid RETR command occurs, another valid PORT command overrides the previous PORT, or a valid QUIT command occurs.

An invalid QUIT command should not close out the connection. In line with this, once the connection is closed by the QUIT command, it should still process future inputs unless the connection gets reestablished.

Many students missed test cases where the connection was either closed by an invalid QUIT or the connection was left alive after a valid QUIT. If a valid QUIT occurs, the following commands should be ignored until the connection is reestablished. If an invalid QUIT occurs, the connection should stay alive and process the following commands as if nothing happened.

If you have any questions regarding the assignment, please come to TA office hours or post to Piazza!

Submission

The same process and policy of submission from Homework 1 will be used in this assignment with one small change: we will not be grading you on the quality of your code for this assignment. The correctness of your program will be graded *only* based on its performance on the class servers.

Grading Rubric

1. Process valid CONNECT requests (15%):
   a. **Client :** Generate correct response message and USER, PASS, SYST, TYPE commands.
   b. **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.
   c. **Client :** Read/process the server replies.
2. Handle the case when the client is unable to connect to the server (10%)
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) (15%)
4. Process valid GET requests and retrieve files (40%):
   a. **Client :** generate valid PORT and RETR commands.
   b. **Server:** Parse the PORT and RETR commands and generate proper responses. Handle errors.
   c. Transfer the file from server to client.
   d. Handle GET after new CONNECT request

5. Handle the case when the requested file does not exist on the server or other errors (10%)
6. Process a valid QUIT command (10%)