Building a Client/Server System Using Sockets

In this assignment you will extend your parsing and file-listing program from Homework 1 to operate over a network. Specifically, you will split your Homework 1 solution into two programs; one of which will be a client program and the other will be a server program. The server program will behave much as the original Homework 1 program did — it will read lines of input, identify valid HTTP GET requests and attempt to process all valid requests. However, whereas Homework 1 performed all I/O to standard input and standard output, your server program will perform all I/O to a TCP socket. The client program will interact with the user reading a line of input from standard input, send the line to the server program over a socket connection, receive a response line(s) from the server, echo all output from the server to standard output, and then wait for another line of input from the user. A separate new socket should be created for each interaction between the client and the server.

Remember to correct all errors from the HW 1 code before starting this homework — there will be significant grade penalty if any of those errors are not corrected. But copy your HW1 code to the HW3 directory before making any changes (do not modify the files in the HW1 subdirectory).

The figure below illustrates the desired conceptual operation of the client/server system in relation to operation of a prototypical solution to Homework 1. (Note that this is only meant to be an example of a possible solution. It is completely acceptable if the organization of your Homework 1 solution and your final client/server solution differs from mine.)

Your HTTP client program should take one command line argument: the port number of the HTTP server to connect to. As in HW1, your client should accept input requests from a human user using standard input. The
client should echo each user input line to standard output along with the corresponding response line specified in HW1. The client program will terminate when end-of-file is reached on standard input in exactly the same manner as in your solution to Homework 1.

Your HTTP server program requires only a single command line argument: the port number on which it should accept HTTP connections from clients. 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. The outline of the server’s execution is as follows-

```python
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.)

**Notes**

The client and server programs will execute on different computers. The server program should be written to execute on the machine comp431sp19.cs.unc.edu. For testing, you should have your server listen for connections on port number 6000 + the last four 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.

In order for your client/server system be correct, it should be the case that, modulo socket error messages (see below), the output of your client program (i.e., what is written to standard output) is indistinguishable from the output of a (correct) solution to Homework 1 when given the same input(s). Your server program should perform all of its HW1 related I/O to a socket. In particular, it should not interact with the user directly in any way. Specifically, it should not output prompts or other messages (except as noted below) and it should not read commands or other inputs from the keyboard or a file.

Both your client and server programs should be prepared to deal with socket related errors and attempt to recover from the errors when possible. Your program should be able to handle basic socket errors such as one side of the connection closing the connection unexpectedly or a client being unable to connect to a server. If a client or server encounters a socket error then it should print “Connection Error” to standard output. If the client or server is unable to recover from the socket error then it should terminate.

**Testing**

To aid in testing, sample input and output files are provided on the course web page. 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_tests.py script:

```
python3 generate_tests.py
```

Start your server:
python3 -u HTTPServer.py 9000 > 0_my_server_output

Run your client:

python3 HTTPClient.py 9000 < 0_input > 0_my_client_output

Finally kill your server with Ctrl+C

Check the correctness using diff:

diff 0_client_output 0_my_client_output

diff 0_server_output 0_my_server_output

If your program works correctly, the diff commands should produce no output. Perform the same steps using the other test files (e.g., 1_input and 1_output).

**Grading**

Again, remember to correct all errors from the HW 1 code before starting this homework – there will be significant grade penalty if any of those errors are not corrected. Make sure your programs comply with all of the sample test cases.

To submit your program for grading, follow the general submission guidelines distributed with Homework 1 and fill out a Google form that will be provided on the course website. You should have two python files in your ~/comp431/submissions/hw3/ directory. One program should be named HTTPServer.py and the other should be named HTTPClient.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.

**Grading Rubric:**

The grade will have the following distribution:

Client - (40%):

- print HTTP messages from server (Socket I/O) (30%)
- Error Handling 10%

Server - (60%):

- Valid Input Processing (25%)
- Erroneous Input Processing (25%)
- Error Handling 10%