Lab 5: Sockets

Due Date: Wed December 4.

Early Submission: Wed November 27 (Do not need to fill a form, we will automatically apply this).

URL: https://classroom.github.com/a/17gR8bRK

In this lab you will build a very simple client server program using Python’s version of asynchronous sockets. The main thing you need to worry about in this lab is getting data from the client to the server, and then printing the information appropriately at the server.

Protocol.

When clients connect to the server the first message they send identifies them. This message must be of the form

+CLIENT <client name>

Please use the format_client_string function to produce such a string at the client, and the parse_client_string function to parse it at the server. Note that it is very important that you use these functions, since we will be testing your server and client against versions not written by you.

Once the client has sent the client identifier, it can send any text it wants. On receiving the text the server should print it out as

<client name>: <string>

Please use the print_server_log function to produce these output strings.

Code and Libraries

In this project we are using Python’s asyncio library (https://docs.python.org/3/library/asyncio.html), which allows our server to handle multiple simultaneous connections. It is
strongly recommended that you read the following documentation pages in order to understand this code:

- https://docs.python.org/3/library/asyncio.html
- https://docs.python.org/3/library/asyncio-stream.html

The project code sets up a basic server socket for you in the `run_server` function, this is equivalent to creating a new `socket`, calling `bind`, and then calling `accept`. When a new client arrives, we call the `handle_connection` function which you are responsible for writing.

The client code should similarly be written in the `run_client` function.

Handling Errors and Failures

- First, please do not use prints in any of the code you write. Our tests rely on reading input, and using print is a problem. Instead use `logging.info` as is done in `handle_connection` (line 69).

- Second, should you have a client connect but not supply a client IDentifier (i.e., if `parse_client_string` returns None) then you should close the connection. You should look at the documentation for `writer.close` and `writer.wait_closed` to see how you can do this.

Running and Testing the Program

You can run a server as follows:

```
python3 main.py -l -p 8080
```

The `-l` indicates that you want a server, the `-p 8080` indicates the port you want to connect to.

You can run a client that connects to this server as follows:

```
python3 main.py -c Moo 127.0.0.1 8080
```

In this case the `-c Moo` indicates that the client ID is Moo, the `127.0.0.1` provides an address for the current machine, and `8080` is the port to connect to.

We do not have test scripts for this project, but instead suggest you test by sending data between your client and your server. Being able to run the following on two terminals is a good sign that things are working:

On terminal 1

```
$ python3 main.py -l -p 8080
moo: Hello # This should print out only after you start the client below and send the first string.
moo: Bloop
INFO:root:Closed # This should print once you hit ctrl-c on terminal 2.
```
On terminal 2

$ python3 main.py -c moo 127.0.0.1 8080
> Hello
> Bloop
> <ctrl-c>

Two things to note about these examples: first, you might have additional log outputs (things which look like INFO:...), that is completely fine; second, this does not test the case where a client does not send an identifier. We leave it up to you to construct a case to test this.