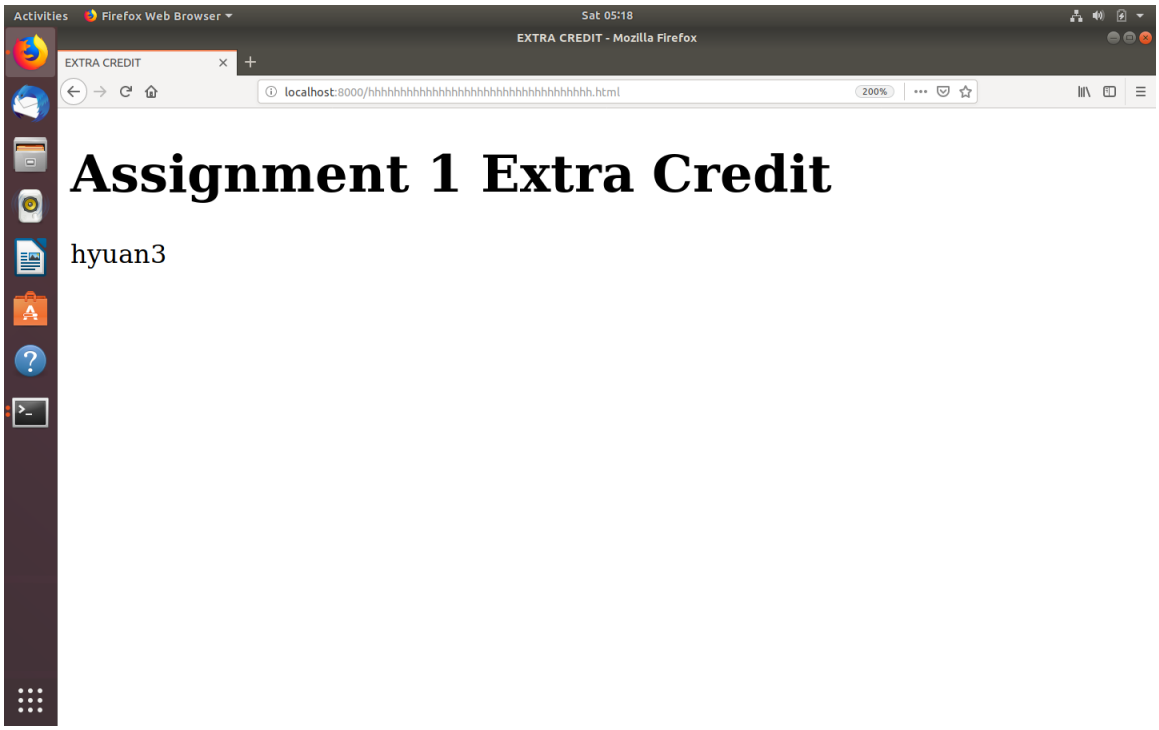
TESTING

- Setting Up Stage - arguments
  - `./httpserver localhost:8000` 200 OK
  - `./httpserver localhost:` 200 OK (with sudo)
  - `./httpserver localhost` FAILED
  - `./httpserver local` FAILED
  - `./httpserver localhost:8000a` FAILED
  - `./httpserver 0.0.0.0:8000` SUCCEED – 200 OK
  - **Same** for `./httpclient`
- Transaction Stage - request
  - `s:file:aaaa-40 characters-aaaa` SUCCEED – 200 OK
  - `s:file:aaaa-40 characters-aaaa (file not exists)` FAILED - 404
  - `s:file:aaaa-40 characters-aaaa not exists` SUCCEED – 201 Created
  - `s:file:aaaa-40 characters-aaaa no permission` FAILED - 403
  - `s:file:aaaa-40 characters-aaaa directory` FAILED - 400
  - `s:` FAILED - 400
  - `s` FAILED - 400
  - `s:file:` FAILED - 400
  - `s:file:bbbbb` FAILED - 400
  - `s:file:cccc-44 characters-cccc` FAILED - 400
  - `s:file:cccc-30 characters-cccc` FAILED - 400
  - `s:file` FAILED - 400
  - `x:file:aaaa-40 characters-aaaa` FAILED - 400
  - **Same** for `r`
- File Transportation - file
  - 0 B file SUCCEED – 200 OK
  - 30 B file SUCCEED – 200 OK
  - 4 KB file SUCCEED – 200 OK
  - 1 MB file SUCCEED – 200 OK
- Other situations
  - With `Content-Length` SUCCEED – 200 OK
  - Without `Content-Length` SUCCEED – 200 OK
  - Sending wrong request to both client and server SUCCEED – 200 OK
  - Work with `SimpleHTTPserver` SUCCEED – 200 OK
  - Work with `curl` SUCCEED – 200 OK
  - Work with web browser SUCCEED – 200 OK
  - Work with others' server and client SUCCEED – 200 OK

EXTRA CREDIT



QUESTION: Briefly discuss (but do not implement) the changes you'd need to make to allow the client to send and receive from stdin and stdout, respectively.

Basically the write() and read() functions will be changed. Instead of open a file to get a file descriptor, we can use the code directly stands for stdin and stdout in read() and write(). Use read(0) to read stdin and write(1) to write stdout.

Beyond that, more situations need to be considered. For example, how to identify the end of stdin. But it shouldn't be too much complicate.

QUESTION: What would happen in your implementation if, during a transfer, the connection was closed, ending the communication early? This extra concern was not present in your implementation of mycat. Why not? Hint: this is an example of complexity being added by an extension of requirements (in this case, data transfer over a network). You do not need to test or implement this, just discuss it.

Regarding of the special cases, like server side shuts down during a transaction, the process should be separated to think of, both on PUT and GET, client and server.

First, if this happens in a **PUT** transaction and the connection shuts down during the data transportation, the **client** side **may not detect this case immediately** and continually send data to server side. Because client side only call receive response at the end of sending data. Thus, even though the

connection shuts down, client will **send out all the data and receive 0 immediately** when client is waiting for server's response. However, it may detect the response is 0 so client may **throw out an error message**, indicating the transaction is not actually finished. Then client finish this transaction.

Server may not detect this disconnection immediately because it always receiving and storing data during a PUT transaction and waiting to give a response. So, the server will receive 0 and stop receiving and storing data. Then it will try to send a response to client with the status code. However, because the connection is over, **server will get an error on sending data**. It will finally detect this connection.

Second, for **GET** transaction. Client will immediately receive a 0 for data, stop receiving and storing data because of disconnection. However, **client will detect this issue finally on double checking Content-Length**. Because the size of data received is not match the Content-Length, client will throw out an error message.

Server may detect this immediately, because server will obtain an error on sending data. However, server cannot do anything but stop sending data and close this transaction.

Mycat is a kind of very simple connection between Terminal and Linux system. This concern isn't present because both Terminal and Linux system are kind of independent. They don't depend on each other to receive any responses. Furthermore, read(2) in mycat can automatically stopped when read_result is 0 which means read the end of file. Basically, read and write functions, which are based on Linux system finish the data transaction. Somehow, they all work in one place(Linux system) and no more worry about the connection. Even the connection is over, that's over. No more effect.

QUESTION: Work with another student in the class and use your client to talk to their server, and their client to talk to your server.

Experience: immediately crushed with unknown incoming request format. It takes long time to discuss and make the format clear. After we figured out what the real HTTP wants and how it works, we adjust our message format for both request and response header. Finally, it works under the standard and identical message format.

Yujia Li (yli302)