hw4

Due Feb 17, 2016 by 11:59pm **Points** 10 **Submitting** a file upload **File Types** zip, tgz, tar.gz, and bz2

Due: 11:59 PM on Wednesday, Feb. 17.

You should complete this homework with your chosen partner. You may change partners for this homework if you wish, but you don't need to. Please register your partnership by joining a HW 4 group on Canvas. (If you don't know what this means, please ask in class.)

For this homework, you will implement a rudimentary web server. The purpose is for you to begin taking advantage of concurrency in Rust.

The deliverable

The purpose of web_server is to respond to the single command of HTTP 0.9, the GET method, which has the following shape:

```
GET /path/to/file HTTP
```

That is, it is the literal world GET, followed by a blank space, followed by a Unix-style absolute path to a file, followed by another blank space and the literal token HTTP. The following line is a blank line. For forward compatibility, you should also accept newer HTTP versions, which will end their request with a token that includes the version, *e.g.,* HTTP/1.1. And you should skip over any header lines following the request but preceding the blank line.
In return to a valid GET request, the web server spawns a thread that retrieves the request, records it to a log file, and generates a response. For this assignment, the following four response statuses are appropriate:

- 200 OK, which starts a reply that serves the specified file;
- 400 Bad Request, which indicates that the command is not a properly formatted GET command;
- 403 Forbidden, which rejects a command because it specifies a file that is off-limits; and
- 404 Not Found, which informs the client that the specified file does not exist.

Each response is preceded by HTTP/1.0 and blank space.
The complete header of a 200 OK response is formatted as follows:
  HTTP/1.0 200 OK
  {name-of-web-server}
  Content-type: text/***{plain-or-text}***
  Content-length: ***{number-of-bytes-sent}***
including a blank line afterward. To keep things simple, the ***{plain-or-html}*** property is either html for files whose suffix is .html or plain for all others.

The remainder of a 200 OK message is the content of the specified file.

A path specification ***{path-to-file}*** must start with / and is interpreted after concatenating it with the server's root path:

- If the resulting path points to a file, the file is served with a 200 OK response unless its permissions do not allow so.
- If the resulting path points to a directory, it is interpreted as pointing to one of these files: index.html, index.shtml, and index.txt. The first file found is served assuming it is accessible. Otherwise the path triggers a 404-message.
- Otherwise the server responds with an error message.

Your web server should listen on localhost (127.0.0.1), port 8080. To explore its workings, point your web browser to **http://localhost:8080/src/main.rs** (http://localhost:8080/src/main.rs) (assuming you are running it out of the Cargo directory) or use telnet:

```
$ telnet 127.0.0.1 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /src/main.rs HTTP
```

The result should be a response of this shape:

```
HTTP/1.0 200 OK
jat489-web-server/0.1
```

```
Content-type: text/plain
Content-length: 2034

// This module implements a rudimentary web server.

use std::io;
use std::sync::Mutex;
. . .
```

Keep it simple—do not try to handle additional HTTP methods.

List any assumptions that you need to make where this specification is incomplete, and be sure to test thoroughly.

Evaluation

Your grade will be based on:

- correctness (how closely your program adheres to this specification),
- thoroughness of testing,
- completeness of documentation (including assumptions),
- style (not expecting the most idiomatic Rust at this point, but I'll be looking for good factoring—don't put everything in main), and
- efficiency (no need to benchmark or profile, but do choose sensible data structures and avoid needless copying).

How to submit

Please submit a tgz or zip archive of your Cargo project directory here on Canvas. Name your archive *NETID*-*NETID*-hw4.{zip,tgz} (*e.g.,* jat489-kbu590-hw4.tgz).