MPCS 54001, Winter 2018
Project 2
Due: February 4th at 11:59pm

Acknowledgment: This project is based on a project from Dr. Anthony Nicholson's version of the MPCS 54001 course.

This work must be entirely your own. If you need help, I encourage you to post questions to Piazza and/or see the staff during their office hours. As a reminder, if you post to Piazza, please don't give away the answer!

Task

The goal of this project is for you to implement a simplified web server.

You are allowed to implement your project in C, Java, or Python.

Hypertext Transfer Protocol (HTTP)

RFC 2616 contains the full specification of HTTP/1.1. Mozilla also provides a very nice HTTP overview targeted at developers. Please feel free to take a look at those resources, but also keep in mind that you will only implement a small subset of HTTP for this project as described in the next section.

HTTP Server

The server shall:

- Support `GET` and `HEAD` requests
- Accept server port number as the only command-line argument
- Use TCP as the underlying reliable transport protocol

The server shall not:

- Support any other HTTP methods (e.g., `POST, DELETE`)
- Support HTTPS (TLS sockets)
- Support or care about cookies
- Support persistent connections

You are not allowed to use off-the-shelf HTTP server libraries, for obvious reasons. Please contact the course staff early if you have doubts about whether a library you want to use is appropriate or not.

Example usage:

- C: `web_server <port>`
- Java: `java WebServer <port>`
- Python: `web_server.py <port>`

Serving Fileset Structure

Download the `project2_serving_tree.tar.bz2` tarball here (another copy will be uploaded to the *Resources* section on Piazza).

Copy it to your directory on the lab machines (e.g., to `~/54001/project2`), and unpack it with the following command.

```
tar xvjf project2_serving_tree.tar.bz2
```

This will create the file structure you need to serve. The root of this directory tree is named "www". To run your server, unpack the tarball in the same directory where your server binary lives. Then, all URLs are relative to "www" being the root. For example, assume your server was running on host `linux2` on port 1234, and you start your server from the directory `/home/$USERNAME/54001/project2/`. If a web client requested the URL "http://linux2.cs.uchicago.edu:1234/foo/bar.html", your server should try to return the contents of the following file (if it exists).

```
/home/$USERNAME/54001/project2/www/foo/bar.html
```

Note that your server should work for other sets of files---the serving tree I gave you is just to help you test. If you hardcode your server to only work for those files, you'll have a bad time during grading. When your server starts up, it should inspect the contents of "www" in the current directory and be able to serve 200s for all those files that are found there.

Handling Redirects

There is one special file in the tarball: `www/redirect.defs`. This is not a real file that your web server should serve (if someone tries to fetch it, return `404 Not Found`). Rather, it defines local URL to remote URL mappings for redirects. If anyone tries to

fetch one of the local URLs in that map, instead of returning a file, return a `301 Moved Permanently` redirect to the redirected remote URL. If you've written your code properly, you'll see the web browser or web client you're testing with actually redirect to the remote resource.

Handling Content Types

Because there are hundreds of MIME types, I'm not going to make you handle them all. **However, you should be able to identify the following types of files (based on extension) and set the Content-Type: header appropriately on the response:**

- text/html
- text/plain
- application/pdf
- image/png
- image/jpeg

See this Wikipedia page for a good rundown on common MIME types.

Basic Strategy

If you receive a `GET` or `HEAD` request for a resource whose path exists in the fileset, then return a `200 OK` (and return the file's contents, if it's a `GET` request).

Else, if there is a redirect for that path in the special `redirect.defs` file, then return a `301 Moved Permanently` response, *indicating the redirected URL*.

Else, return `404 Not Found` because the requested resource does not exist.

If you receive a malformed request, then return `400 Bad Request`.

If you receive a request method other than `GET` or `HEAD`, then return `405 Method Not Allowed`.

Don't forget that `HEAD` should work for any valid URL that `GET` works for.

Hints

- You all share the same machines (`linux1`, `linux2`, and `linux3`) so if you start your server and get an error about a non-reserved port being in use, just try

another port number greater than 1024.

- In addition to the Mozilla HTTP overview, the textbook actually has some great examples w.r.t. the format of HTTP messages.
- Use a Linux command-line client (such as **`curl`**) to test your code. Read the man pages and find out all the options. You can request a `HEAD` request, rather than `GET`, for instance.
- Don't forget that at the end of each line, you can't just terminate with a "\n" but need to use "\r\n" (both a "normal" newline and a line feed character).
- Don't forget you need a blank line (just "\r\n") in between the header and body of the response to a `GET` request.
- Don't forget to set the "`Content-Type:`" header field in a response, when returning data, so that the client can properly interpret the response.

Deliverables

Create a directory named `project2` in your individual Phoenixforge Subversion repository. Upload your source code (server source files and a Makefile if you created one) to that directory. **Please do not upload the serving tree fileset.**

Grading

This project will be worth 45% of your total project grade for the course.

We will compile your submissions and grade them against an automated web client. We also might test some URLs in a browser (you should do the same).

General grading rubric is as follows:

- Compiles, starts, binds to a TCP port: **10 points**
- `GET` requests return `200` + the data for all existing files in tree: **10 points**
- `GET` requests return `301` for all paths specified as redirects in redirects.defs *and actually redirects the web client*: **10 points**
- `GET` requests return `404` for all non-existent paths and `www/redirect.defs`: **10 points**
- `HEAD` requests return `200` (but no data) for all existing files: **10 points**
- `HEAD` requests return `301`, `404` for all paths in exactly the same way as for `GET` requests above: **10 points**
- Any `POST` request or other unknown method returns a `405`: **10 points**
- Any malformed request returns a `400`: **10 points**

- Server can handle multiple requests in succession without restarting (loops around and accepts the connection again): **5 points**
- Server either spawns new thread, forks new process, or asynchronously handles web requests (i.e., no serial servers): **5 points**
- Style points, grader discretion (efficiency, code correctness, or other considerations e.g., you didn't just hardcode responses to the files in the tarball, but actually read the files: **10 points**

for a total of 100 points.

Testing

You can perform local testing on the same machine by using the `curl` utility to talk to your web server implementation. Some example `curl` commands are included below to help guide your testing, but please keep in mind that the following test cases are not necessarily comprehensive and the course staff might use their own additional tests for grading. You are responsible for coming up with your own test cases for comprehensive test coverage of your web server functionality.

In addition to testing with `curl`, you should also point a web browser (e.g., Chrome, Firefox) at the URLs in the following test cases and manually verify that the browser reacts to the response as expected. The following test cases assume that your server is running on `localhost` on port 8080, so you can adjust the URL as needed for your testing.

```
# GET returns 200 and file contents for existing file path
curl -v -s http://localhost:8080/index.html

# GET returns 301 for redirect path in redirects.def (and redirects web client)
curl -v -s http://localhost:8080/uchicago/cs

# GET returns 404 for non-existent file path
curl -v -s http://localhost:8080/doesnotexist.html

# HEAD returns 200 and no file contents for existing file path
curl -v -s -I http://localhost:8080/index.html

# HEAD returns 301 for redirect path in redirects.def
curl -v -s -I http://localhost:8080/uchicago/cs
```

```
# HEAD returns 404 for non-existent file path
curl -v -s -I http://localhost:8080/doesnotexist.html

# Unknown method returns 405
curl -v -s -X OPTIONS http://localhost:8080/index.html

# Malformed request returns 400
nc -v localhost 8080
<type ‘badrequest’, then press enter>
```