Programming Assignment 1: Multi-threaded HTTP Server

Due: October 1st, 2020 23:59:59pm EDT

In this assignment, you will implement a simple multi-threaded HTTP server that only accepts HTTP GET requests and returns the desired content to the client.

The goal of this assignment is to get you familiar with i) the remote.cs.binghamton.edu environment, ii) using git for managing software development, and iii) socket programming and multi-threading.

This assignment is worth **13.3%** of your total score in this course. You **must** work **by yourself** and use **Python (preferred), C/C++, or Java** to implement this assignment.

1 Background

HTTP GET requests are used to request data from the specified source. For example, if we want to access a webpage at `http://www.foo.com/bar.html`, our browser will send something similar to the following HTTP GET request.

```
GET /bar.html HTTP/1.1
Host: www.foo.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

In the above example, the Firefox browser is the client, and `Host: www.foo.com` is the server host name. The client is requesting server resource located at `/bar.html`. Usually, HTTP servers run on port 80. So, by default, the server port number is 80.

If, for example, the HTTP server runs on port 8080, then the URL would be `http://www.foo.com:8080/bar.html`, and the `Host` field in the HTTP request will also contain the port number: `www.foo.com:8080`.

The HTTP server replies with HTTP response. A valid HTTP response includes: (1) the status line, (2) the response header, and (3) the message body. For example, the text below shows an HTTP response. Note that there is an empty line between the response header and the message body. This empty line indicates the end of header fields[1].

```
HTTP/1.1 200 OK
Date: Thu, 02 Apr 2015 01:51:49 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Tue, 10 Feb 2015 17:56:15 GMT
Accept-Ranges: bytes
Content-Length: 2693
Content-Type: text/html

... ... <content of the bar.html>
```

[1] `https://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html`

2 Basic HTTP Server

Your HTTP server should support a subset of the HTTP standard. It only needs to serve HTTP GET requests and should only support basic portions of the standard. That is, you do not need to support persistent connections, request pipelining or other, advanced portions of the standard. Your server must only correctly handle HTTP requests where a single request/response pair is sent over a single TCP connection. Specifically, your server should operate as follows:

- Your HTTP server should not take any command line arguments. It should create a TCP server socket to listen for incoming TCP connections on an unused port, and output the host name and port number the HTTP server is running on. **Note: You must use sockets in this assignment. Implementation using existing libraries such as Python's `SimpleHTTPServer` will receive 0 points. If you are unsure about certain library, please be sure to ask the instructor first.**
- It should look for a directory called `www`, **located in the same directory as your HTTP server executable**. The `www` directory should contain resources you want your HTTP server to serve. If this directory does not exist, the HTTP server should output an error message and quit.
- When a new request comes in, the HTTP server accepts the connection, establishing a TCP connection with the client.
- The HTTP server parses the HTTP request from the client and prepares an HTTP response.
- The HTTP server looks up the requested resource from its local `www` directory.
  - If the requested resource is located by the HTTP server, prepare a response with status code "200 OK", followed by response header and the content of the requested resource. The response header should include the following fields:
    * `Date` The date and time the response is originated in the format defined by RFC 7231 Date/Time Formats [2].
    * `Server` A name for your HTTP server. You can choose a server name you prefer.
    * `Last-Modified` Last modified time and date of the requested resource, also in RFC 7231 Date/-Time Formats.
    * `Content-Type` The MIME type of this content. You can use the mime.types file located at `/etc/mime.types` to determine the correct MIME type given a filename extension. If a filename extension is not found in the mime.types file, you can use `application/octet-stream`.
    * `Content-Length` The length of the requested resource in bytes.
  - If the requested resource does not exist, reply with status code "404 Not Found". You can be creative and write a fancy custom error page to return to the client.
- After finishing sending the response, the HTTP server closes the socket connection to the client.

2.1 Command Line Output

Upon successfully serving a request, your HTTP server must write to standard output (stdout) the following items:

requested resource

client IP client's IP address in dotted decimal representation

[2] `https://tools.ietf.org/html/rfc7231#section-7.1.1.1`

client port client's port number

access count the number of times this resource has been requested since the start of the HTTP server

These four items of a same request should be printed to a same line and seperated by the "|" character. Below is an example:

```
/bar.html|128.226.118.20|4759|1
/foo.jpg|128.226.118.20|6001|1
/bar.html|128.226.118.26|6550|2
/bar.html|128.226.118.21|10039|3
/foo.jpg|128.226.118.21|5300|2
/foo.jpg|128.226.118.26|36500|3
```

3 Multi-threaded HTTP Server

To handle multiple client requests, **different requests should be handled in different threads, with a new thread created for each client request**.

When handling each request in a separate thread, it is important to protect the access of shared objects, for example, the `access count` of resources.

4 How to Test Your Implementation

You **must** test your HTTP server on the remote nodes of CS department computers: `remote.cs.binghamton.edu`. When accessing `remote.cs.binghamton.edu`, you are actually redirected to one of the 8 `REMOTE` machines ({remote00, remote01, ..., remote06, remote07}.cs.binghamton.edu) using DNS redirection. So to test your implementation, you need to run your HTTP server on one of the `REMOTE` nodes and set your web client, e.g., `wget`[3], to use the correct server host name: "remoteXX.cs.binghamton.edu", instead of "remote.cs.binghamton.edu".

For example, suppose your HTTP server is started on `remote02.cs.binghamton.edu` port `47590`. You can run the following command in an **EMPTY** directory on a different `REMOTE` computer to download the resource:

```
wget http://remote02.cs.binghamton.edu:47590/bar.html
```

Note:

- You must replace `bar.html` with a valid resource in your HTTP server directory.
- Since this is only an HTTP server, not an HTTPS server, your URL must use HTTP, not HTTPS.
- You may also run your HTTP server on your own machine and use Wireshark[4] for debugging. However, you must test your implementation on `REMOTE` computers before submission.
- We do not recommend using a web browser for debugging and testing.

After `wget` successfully downloads the requested resource, use the `diff`[5] command to check if the downloaded resource matches the resource located in the `www` directory.

[3] `http://linux.die.net/man/1/wget`
[4] `https://www.wireshark.org/`
[5] `http://linux.die.net/man/1/diff`

`wget` has a few options that can help you debug your implementation. For example, the `--limit-rate` option allows you limit the download speed to a pre-specified amount. This can be helpful if you want to debug your multi-threaded implementation. If you need to inspect if your HTTP server constructs the HTTP response header correctly, you can take advantage of the `--save-headers` option. More information can be found at the manual page of wget.

5 Github classroom

To access this assignment, first log into your Github account created with your BU email. Then go to: `https://classroom.github.com/a/nzJJbm49`. After clicking this link, Github classroom will ask you to join the class roster by selecting your email from a list. Github classroom will automatically create a repository (e.g., if your Github username is jdoe, the repository will be named cs457-557-fall2020-pa1-jdoe). This is the repository you will push your code to.

This repository is a private repository. Only you, course instructor, and the TA are able to see this repository. Follow the instruction on the Github page to create a new repository on your local directory and link it to the Github repository. Note that `git` is already installed on CS department computers. To use it on your own computer, you must install it first.

To add a file to the next commit, use the `git add` command. To commit your code, use the `git commit` command. Be sure to also push your commit to the Github repository after every commit using the `git push` command (e.g., `git push origin master`).

We expect each repository to have **at least five commits**, with the first one and the last one **more than 72 hours apart**. It is thus very important that you frequently push your latest code to this repository.

If you have never used `git` before, you can start with a cheatsheet prepared by Github[6]. More detailed references are also available online[7].

6 How to submit

To submit, make a final commit with commit message "Final commit" and push your latest code to the private Github repository Github classroom created. Your repository should contain the following files:

1. All of your source code that implements the HTTP server. Note that your server executable should not take any command line arguments and must output the host name and port number it is running on.

2. A `Makefile` to compile your source code into one executable, which should be named `server`. If you used Python, then the Makefile is not required.

3. A `README` file describing how to compile and run your code on `remote.cs.binghamton.edu` computers, a brief description of your implementation, and sample input/output.

4. A `STATEMENT` file, containing the following statement followed by the student's full name:

   "I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of **0** for the involved assignment and my grade will be reduced by one level (e.g., from A to A- or from B+ to B) for my first offense, and that I will receive a grade of **"F" for the course** for any additional offense of any kind."

[6] `https://education.github.com/git-cheat-sheet-education.pdf`
[7] `https://git-scm.com/docs`

After pushing your final commit to the Github repository, please let us know by **submitting your commit hash to myCourses**. The commit hash will be 40 characters long, e.g., "77469e1dada9586004520e34f0819ea246b6240b". There are two ways to locate your commit hash: You can type `git log` in your local repository, which will output the commit history. Locate the commit you want to use as your final submission, record the hash associated with the commit. Alternatively, you can also go to the Github page of your repository and locate it on the webpage.

It is important that you submit your commit hash to myCourses. This helps us know your submission is ready for grading and which of your commits we should grade. We will not grade your assignment unless you have submitted the commit hash to myCourses before the assignment submission deadline.

Your assignment will be graded on the CS Department computers `remote.cs.binghamton.edu`. It is your responsibility to make sure that your code compiles and runs correctly on these remoteXX computers.

Your assignment must be your original work. We will use MOSS[8] to detect plagiarism in the projects.

Appendix

To help you test your HTTP server implementation, you can use the files available in a `.zip` folder on myCourses: `https://mycourses.binghamton.edu/bbcswebdav/courses/3T202090/objects.zip`. There are 5 files in this zipped file. You can put these files into your local `www` directory and have your HTTP server serve them. Please also feel free to use other objects for testing.

[8] `https://theory.stanford.edu/~aiken/moss/`