COMP30023 Project 2
Serving the Web

Out date: 3 May, 2022
Due date: No later than 5pm Friday 20 May, 2022 AEST
Weight: 15% of the final mark

1 Project Overview

The aim of this project is to familiarize you with socket programming, multi-threading, and the HTTP protocol. Your task is to write a basic HTTP server that responds correctly to a limited set of GET requests. The HTTP server must return valid response headers for a range of files and paths.

Your HTTP Server must be written in C. Submissions that do not compile and run on a cloud VM, like the one you have been provided with, may receive zero marks. You must write your own HTTP handling code; you may not use existing HTTP libraries.

2 Project Details

Your task is to write a simple HTTP Server. There is only a limited range of content that needs to be served, and it only needs to respond to GET requests for static files. You can assume that GET requests are no longer than 2 kB in size; note that this is larger than a typical MTU, and so you must be able to read in a multi-packet request.

Example content that needs to be served is:

Content	File Extension
HTML	.html
JPEG	.jpg
CSS	.css
JavaScript	.js

For these extensions, the correct MIME type should be reported (see `https://mimetype.io/all-types`). For all other extensions, or if there is no extension like `Makefile`, you can return a MIME type of `application/octet-stream`. You can assume that the requests are plain ASCII; you do not need to deal with Internationalized Resource Identifiers [`https://en.wikipedia.org/wiki/Internationalized_Resource_Identifier`] and you do not need to deal with %-encoding [https://en.wikipedia.org/wiki/Percent-encoding].

The minimum requirement is that the server implements HTTP 1.0, and as such, your server does not have to support pipelining or persistent connections. Your server should be able to handle multiple incoming requests by making use of Pthreads (or similar concurrent programming technique) to process and respond to each incoming request.

Your server program must use the following command line arguments:

- protocol number: 4 for IPv4 and 6 for IPv6
- port number
- string path to root web directory

The port number will be the port the server will listen for connections on. The string path to the root web directory points to the directory that contains the content that is to be served. For example:

```
/home/comp30023/website
```

All paths requested by a client should be treated as relative to the path specified on the command line. For example, if a request was received with the path `/css/style.css` it would be expected to be found at:

```
/home/comp30023/website/css/style.css
```

The server must support responding with either a 200 response containing the requested file, or a 404 response if the requested file is not found. You do not need to handle requests with invalid headers (but the program should never crash), but paths to non-existent files may be requested, and if they are a 404 should be returned. Response headers should be valid and must included at a minimum:

- Http Status
- Content-type

2.1 Program execution / command line arguments

To run your server program on your VM prompt, type:

```
./server [protocol number] [port number] [path to web root]
```

where:

- `[protocol number]` is 4 for IPv4 or 6 for IPv6
- `[port number]` is a valid port number (e.g., 8080), and
- `[path to web root]` is a valid absolute path (e.g., /home/comp30023/website)

3 Marking Criteria

The marks are broken down as follows

Task # and description	Marks
1. Server runs and sends valid responses to IPv4 requests	4
2. Server MIME types and paths work for IPv4 requests	4
3. Parts 1 and 2 also work for IPv6	1
4. Build quality	1
5. Quality of software practices	2
Extension. Parallel downloads work	3

Code that does not compile and run on cloud VM will be awarded zero marks for parts 1–5. Your submission will be tested and marked with the following criteria:

Task 1. Server runs and sends valid responses Code must run on the marking VM without crashing (e.g., seg faulting) **regardless of the inputs**. Any code that crashes for any reason may be allocated a score of 0 overall (not just for this component). However, if your code crashes at the submission deadline, still submit it because some marks *may* be awarded.

Server sends a valid HTTP 200 response in reply to a GET request for an HTML file located in web root directory (not a sub-directory) (2 marks)

Server sends a valid HTTP 404 response in reply to a GET request for a file in the web root directory that does not exist or cannot be opened for reading (2 marks)

(If you are a purist, then if the file exists but cannot be opened for reading, then the program can alternatively return a 403 (Forbidden Error). No extra marks are allocated for that.)

GET requests using path components `../` should return a 404 error. (Otherwise a query like `../secret` would break out of the web root, unless handled specifically.)

Task 2. Server MIME types and paths Server sends a valid HTTP 200 response with the correct MIME type in reply to a GET request for a file located in web root directory (2 marks)

Server sends a valid HTTP 200 response with the correct MIME type in reply to a GET request with a path **below** the web root for any of the specified file types (e.g. `GET /css/style.css HTTP/1.0`) (2 marks)

Task 3. IPv6 (The internet is shifting from IPv4 to IPv6, and we must too.)

The server can accept connections and perform Tasks 1 and 2 using IPv6 (1 mark)

Task 4. Build quality Running `make clean && make -B && ./server <command line arguments>` should execute the submission. If this fails for any reason, you will be told the reason, and be allowed to resubmit (with the usual late penalty). If it still fails, you will get 0 for Tasks 1–4 and the extension. Test this by committing regularly, and checking the CI feedback. (If you need help, ask on the forum.)

A 0.5 mark penalty will be applied if compiling using "`-Wall`" yields a warning or if your final commit contains `server`, any other executable or `.o` files (see Practical 2).

The server need not exit. The automated test script will kill it at the end of the tests.

Task 5. Quality of software practices Factors considered include **quality of code**, based on the choice of variable names, comments, indentation, abstraction, modularity, and **proper use of version control**, based on the regularity of commit and push events, their content and associated commit messages (e.g., repositories with a single commit and push, non-informative commit messages will lose 0.5 mark. Profanity or abuse in the commit messages will also lose 0.5 mark; everyone gets frustrated, but commits must remain professional).

Extension. Parallel downloads Server uses Pthreads (or similar concurrent programming technique, or `epoll`) to process incoming requests and sending responses. Do not `fork` multiple processes.

Server can process and respond to at least five HTTP requests concurrently. There is no need to limit this to only five. (3 marks)

4 Submission

All code must be written in C (e.g., it should not be a C-wrapper over non C-code) and cannot use any external libraries, except standard libaries as noted below.

You can reuse the code that *you wrote* for your other *individual* projects if you clearly specify when and for what purpose you have written it (e.g., the code and the name of the subject, project description and the date, that can be verified if needed). You may use standard libraries (e.g., to print, read files, create sockets, manipulate threads etc.). In particular, you may find `sendfile` useful. If you choose to use this, then provide a comment of at least two lines explaining the benefits of sendfile over the alternatives, in terms of both code simplicity and performance. Your code must compile and run on the provided VMs.

The repository must contain a `Makefile` which produces an executable named "`server`", along with all source files required to compile the executable. Place the Makefile at the root of your repository, and ensure that running `make` places the executable there too.

If you have implemented IPv6, include the line

```
#define IMPLEMENTS_IPV6
```

If you have implemented the multithreading extension, include the line

```
#define MULTITHREADED
```

Your server should not shut down by itself. SIGINT (like CTRL-C) will be used to terminate your server between test cases. You may notice that a port and interface which has been bound to a socket sometimes cannot be reused until after a timeout. To make your testing and our marking easier, please override this behaviour by placing the following lines before the bind() call:

```
int enable = 1;
if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0) {
perror("setsockopt");
exit(1);
}
```

Make sure that all source code is committed and pushed. **Executable files** (that is, all files with the executable bit which are in your repository) **will be removed** before marking. Hence, ensure that none of your source files have the executable flag set. (You can verify this by cloning your repo onto your VM, and using `ls -l`.)

For your own protection, it is advisable to commit your code to git at least once per day. Be sure to `push` after you `commit`. The git history may be considered for matters such as special consideration, extensions and potential plagiarism. Your commit messages should be a short-hand chronicle of your implementation progress and will be used for evaluation in the Quality of Software Practices criterion.

You must **submit the full 40-digit SHA1 hash** of your chosen commit to the **Project 1 Assignment** on LMS. You must **also push your submission** to the repository named `comp30023-2022-project-2` in the subgroup with your username of the group `comp30023-2022-projects` on `gitlab.eng.unimelb.edu.au`. You will be allowed to update your chosen commit. However, only the last commit hash submitted to LMS before the deadline will be marked without late penalty.

You should ensure that the commit which you submitted is accessible from a fresh clone of your repository. For example (below `...` are added for clarity to break the line):

```
git clone https://gitlab.eng.unimelb.edu.au/comp30023-2022-projects/<username>/ ...
... comp30023-2022-project-2
cd comp30023-2022-project-2
git checkout <commit-hash-submitted-to-lms>
```

Late submissions will incur a deduction of 2 mark per day (or part thereof).

Extension policy: If you believe you have a valid reason to require an extension, please fill in the form accessible on Project 2 Assignment on LMS. Extensions **will not be** considered otherwise. Requests for extensions are not automatic and are considered on a case by case basis.

5 Testing

You have access to several test cases and their expected outputs. However, these test cases are not exhaustive and will not cover all edge cases. Hence, you are also encouraged to write more tests to further test your own implementation.

Testing Locally: You can clone the sample test cases to test locally, from: `comp30023-2022-projects/project-2`.

Continuous Integration Testing: To provide you with feedback on your progress before the deadline, we will set up a Continuous Integration (CI) pipeline on Gitlab. Though you are strongly encouraged to use this service, the usage of CI is not assessed, i.e., we do not require CI tasks to complete for a submission to be considered for marking.

Note that the test cases which are available on Gitlab are the same as the set described above.

The requisite `.gitlab-ci.yml` file will be provided and placed in your repository, but is also available from the `project-2` repository linked above. Please clone, commit and push to trigger.

6 Getting help

Please see Project 2 Help module on LMS.

7 Collaboration and Plagiarism

You may discuss this project abstractly with your classmates but what gets typed into your program must be individual work, not copied from anyone else. Do **not** share your code and do **not** ask others to give you their programs. The best way to help your friends in this regard is to say a very firm "**no**" if they ask to see your program, point out that your "**no**", and their acceptance of that decision, are the only way to preserve your friendship. See `https://academicintegrity.unimelb.edu.au` for more information.

Note also that solicitation of solutions via posts to online forums, whether or not there is payment involved, is also Academic Misconduct. You should not post your code to any public location (e.g., `github.com`) until final subject marks are released.

If you use a **small** amount of code not written by you, you must attribute that code to the source you got it from (e.g., a book or Stack Exchange).

Do **not** post your code on the subject's discussion board Ed, except in a **Private** thread.

Plagiarism policy: You are reminded that all submitted project work in this subject is to be your own individual work. Automated similarity checking software will be used to compare submissions. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the student concerned.

Using git properly is an important step in the verification of authorship.

Good luck!