Introduction

Please comply with the following rules:

- Remain polite, courteous, respectful, and constructive throughout the evaluation process. The well-being of the community depends on it.
- Work with the student or group being evaluated to identify potential issues in their project. Take the time to discuss and debate the problems that may have been identified.
- You must consider that there might be some differences in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade them as honestly as possible. Pedagogy is useful only if peer evaluation is conducted seriously.

Guidelines

- Only grade the work that was turned in to the Git repository of the evaluated student or group.
- Double-check that the Git repository belongs to the student(s). Ensure that the project is the one expected. Also, check that git clone is used in an empty folder.
- Carefully check that no malicious aliases were used to deceive you into evaluating content that is not from the official repository.
- To avoid any surprises, and if applicable, review together any scripts used to facilitate the grading (scripts for testing or automation).
- If you have not completed the assignment you are going to evaluate, you have to read the entire subject prior to starting the evaluation process.
- Use the available flags to report an empty repository, a non-functioning program, a Norm error, or instances of cheating.

In these cases, the evaluation process ends, and the final grade is 0, or -42 in case of cheating. However, except for cheating, students are strongly encouraged to review together the work that was turned in, in order to identify any mistakes that should not be repeated in the future.

- Remember that during the defence, no segfault or any other unexpected, premature, or uncontrolled termination of the program is allowed; otherwise, the final grade will be 0. Use the appropriate flag.

- You should never have to edit any file except the configuration file if it exists. If you want to edit a file, take the time to explain the reasons with the evaluated student and make sure both of you are okay with this.
- You must also verify the absence of memory leaks. Any memory allocated on the heap must be properly freed before execution ends.

You are allowed to use any of the different tools available on the computer, such as leaks, valgrind, or e_fence. In case of memory leaks, tick the appropriate flag.

Mandatory Part

Check the code and ask questions

- Install Siege using Homebrew.
- Ask for an explanation about the basics of an HTTP server.
- Ask which event mechanism they use: poll() (or equivalent: select, epoll, kqueue).
- Ask for an explanation of how poll() (or equivalent) works.
- Ask if they use only one poll() (or equivalent) and how they handle server accept() and client read/write operations.
- The poll() (or equivalent) should be in the main loop and must check file descriptors for both reading and writing simultaneously. Otherwise, the grade is 0, and the evaluation process ends immediately.
- After poll() (or equivalent), I/O on event-driven descriptors (e.g., listening/client sockets, CGI pipes/FIFOs) must occur only when readiness is indicated. It is acceptable to read/write until EAGAIN/EWOULDBLOCK, then return to the main loop. Ask the group to show the code path from poll() (or equivalent) to the client's I/O.
- Search for all read/recv/write/send on a socket and check that, if an error is returned, the client is removed.
- Search for all read/recv/write/send and check if the returned value is correctly handled (checking only -1 or only 0 is not enough; both must be handled).
- If errno is checked after read/recv/write/send, the grade is 0 and the evaluation process ends immediately (errno may be logged, not used to drive control flow).

- Writing or reading event-driven descriptors (sockets, pipes/FIFOs, TTY, etc.) without going through poll() (or equivalent) is strictly **FORBIDDEN**. Regular disk files are exempt.
- Confirm that synchronous I/O on regular disk files (e.g., reading the configuration or small static files) does not stall the event loop.
- The project must compile without any relinking issues. If not, use the “Invalid compilation” flag.
- If any point is unclear or incorrect, the evaluation stops.

Configuration

In the configuration file, check whether you can do the following and test the result:

- Search for the HTTP response status codes list on the internet. During this evaluation, if any status codes are incorrect, do not award any points related to them.
- Set up multiple websites on different interfaces and ports.
- Set up a default error page (try modifying the 404 error page).
- Limit the size of the client request body (use: curl -X POST -H "Content-Type: plain/text" --data "BODY IS HERE"; write something shorter or longer than the body limit).
- Set up routes on a server to different directories.
- Set up a default file to serve when requesting a directory.
- Set up a list of accepted methods for a specific route (e.g., try DELETE something with and without permission).

Basic Checks

Using telnet, curl, and pre-prepared files, demonstrate that the following features function correctly:

- GET, POST, and DELETE requests should work.
- UNKNOWN requests should not result in a crash.
- For every test, you should receive the appropriate status code.
- Upload some files to the server and retrieve them.

Check CGI

Pay attention to the following:

- The server should function correctly with CGI.
- The CGI should be run in the correct directory for relative path file access.
- With the help of the student(s), you should check that everything is working properly. You must test the CGI with the GET and POST methods.
- You must test with CGI files containing errors to ensure that the server's error handling works correctly. You can use a script containing an infinite loop or an error — you are free to do whatever test you want within the limits of acceptability, at your discretion. The group being evaluated should help you with this.

The server should never crash, and an error should be displayed if an issue occurs.

Check with a Browser

- Use the browser chosen by the team. Open the network tab and try connecting to the server.
- Look at the request header and response header.
- It should be compatible with serving a fully static website.
- Try an incorrect URL on the server.
- Try to list a directory.
- Try a redirected URL.
- Try anything you like.

Port Issues

- In the configuration file, set up multiple interfaces and ports to provide different websites. Use the browser to verify that the configuration works correctly and serves the appropriate website.

- Configure multiple websites on the same interface:port. This should result in an error, unless the team has chosen to implement the virtual host feature. Both approaches are valid, as long as everything works as expected.
- Launch multiple webserv programs at the same time with different configuration files but with common interface:ports. Does it work? If it does, ask why. Ensure that, whatever the group's choice was, the program’s behaviour is coherent and does not crash.

Siege & Stress Test

- Use Siege to run some stress tests.
- Availability should be above 99.5% for a simple GET request on an empty page using Siege with the -b flag.
- Ensure there are no memory leaks (monitor the process memory usage; it should not increase indefinitely).
- Check that there are no hanging connections.
- You should be able to use Siege indefinitely without having to restart the server (take a look at Siege with the -b flag).
- When conducting load tests using the Siege command, be careful — it depends on your OS. It is crucial to limit the number of connections per second by specifying options such as -c (number of clients), -d (maximum wait time before a client reconnects), and -r (number of attempts). The choice of these parameters is at the evaluator's discretion. However, it is imperative to reach an agreement with the person being evaluated to ensure a fair and transparent assessment of the web server's performance.
