- To avoid any surprises, carefully check that both the evaluating
and the evaluated students have reviewed the possible scripts used
to facilitate the grading.

- If the evaluating student has not completed that particular
project yet, it is mandatory for this student to read the
entire subject before starting the defense.

- Use the flags available on this scale to signal an empty repository,
non-functioning program, norm error, cheating etc. In these cases,
the grading is over and the final grade is 0 (or -42 in case of
cheating). However, except for cheating, you are
encouraged to continue to discuss your work (even if you have not
finished it) to identify any issues that may have caused
this failure and avoid repeating the same mistake in the future.

- Remember that for the duration of the defense, no segfault,
no other unexpected, premature, uncontrolled or unexpected
termination of the program, else the final grade is 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 explicit 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 the end of execution.
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.

Attachments

- subject.pdf (https://cdn.intra.42.fr/pdf/pdf/34676/en.subject.pdf)
- tester (/uploads/document/document/5957/tester)
- ubuntu_cgi_tester (/uploads/document/document/5958/ubuntu_cgi_tester)
- cgi_tester (/uploads/document/document/5959/cgi_tester)
- ubuntu_tester (/uploads/document/document/5960/ubuntu_tester)

Mandatory Part

Check the code and ask questions

- launch the installation of siege with homebrew
- Explain the basics of an HTTP server.
- Ask which function they used for I/O Multiplexing.
- Ask to get an explanation of how select (or equivalent) is working.
- Ask if they use only one select (or equivalent) and how they've managed the server accept and the client read/write.
- the select (or equivalent) should be in the main loop and should check fd for read and write AT THE SAME TIME, if not please give a 0 and stop the evaluation.
- There should be only one read or one write per client per select (or equivalent). Ask to show you the code that goes from the select (or equivalent) to the read and write of a client.
- Search for all read/recv/write/send on a socket and check that if an error returned the client is removed
- Search for all read/recv/write/send and check if the returned value is well checked. (checking only -1 or 0 is not good you should check both)
- If a check of errno is done after read/recv/write/send. Please stop the evaluation and put a mark to 0
- Writing or reading ANY file descriptor without going through the select (or equivalent) is strictly FORBIDDEN
- The project must compile without any re-link issue if not use the compile flag.
- If any point is unclear or is not correct use the flag for incomplete work.

Yes No

Configuration

In the configuration file check if you can do the following and test the result:
- look for the HTTP response status codes list on internet and during this evaluation
if any status codes is wrong don't give related points.
- setup multiple servers with different port
- setup multiple servers with different hostname (use something like: curl --resolve example.com:80:127.0.0.1 http://example.com/)
- setup default error page (try to change the error 404)
- limit the client body (use curl -X POST -H "Content-Type: plain/text" --data "BODY IS HERE write something shorter or longer than body limit")
- setup routes in a server to different directories
- setup a default file to search for if you ask for a directory
- setup a list of method accepted for a certain route (ex: try to delete something with and without permission)

Yes No

Basic checks

Using telnet, curl, prepared files demonstrates that the following features work properly:
- GET requests -> should work
- POST requests -> should work
- DELETE requests -> should work
- UNKNOWN requests -> should not produce any crash
- For every test the status code must be good
- upload some file to the server and get it back

Yes No

Check with a browser

- Use the reference browser of the team, open the network part of it and try to connect to the server with it
- Look at the request header and response header
- It should be compatible to serve a fully static website
- Try a wrong URL on the server
- Try to list a directory
- Try a redirected URL
- Try things

Yes No

Port issues

- In the configuration file setup multiple ports and use different websites, use the browser to check that the configuration is working as expected, and show the right website.
- In the configuration try to setup the same port multiple times. It should not work.
- Launch multiple servers at the same time with different configurations but with common ports. Is it working? If it is working, ask why the server should work if one of the configurations isn't working. keep going

Yes No

Siege & stress test

- Use Siege to run some stress tests.
- Availability should be above 99.5% for a simple get on an empty page with a siege -b on that page
- Check if there is no memory leak (monitor the process memory usage it should not go up indefinitely)
- Check if there is no hanging connection
- You should be able to use siege indefinitely without restarting the server (look at siege -b)

Yes No

Bonus Part

Cookies and session

There's a working session and cookies system on the webserver.

Yes No

CGI

There's more than one CGI system.

Yes No

Ratings

Don't forget to check the flag corresponding to the defense

Ok Outstanding project

Empty work No author file Invalid compilation Norme Cheat Crash

Leaks Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation