Go to homepage

SCALE FOR PROJECT PHILOSOPHERS

You should evaluate 1 student in this team

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.

- Identify with the student or group whose work is evaluated the possible dysfunctions 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. The pedagogy is useful only and only if the peer-evaluation is done seriously.

Guidelines

- Only grade the work that was turned in 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.

- Check carefully that no malicious aliases was used to fool you and make you evaluate something that is not the content of 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, cheating, and so forth.
In these cases, the evaluation process ends and the final grade is 0, or -42 in case of cheating. However, except for cheating, student are strongly encouraged to review together the work that was turned in, in order to identify any mistakes that shouldn't be repeated 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 verify the absence of data races.
You are allowed to use any of the different tools available on the computer, such as valgrind with "--tool=helgrind" and "--tool=drd". In case of any data-race, the evaluation stops here.

- 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

Mandatory Part

Error Handling

This project is to be coded in C, following the Norm.
Any crash, undefined behavior, memory leak, or norm error means 0 to the project.
On some slow hardware, the project might not work properly. If some tests don't work on your machine try to discuss it honestly before counting it as false.

Yes | No

Global variables

Check if there is any global variable which is used to manage the shared resources among the philosophers.
If you find such a nasty thing, the evaluation stops here. You can go on and check the code, but do not grade the exercises.

Yes | No

philo code

- Ensure the code of philo complies with the following requirements and ask for explanations.
- Check there is one thread per philosopher.
- Check there is only one fork per philosopher.
- Check if there is a mutex per fork and that it's used to check the fork value and/or change it.
- Check the outputs are never mixed up.
- Check how the death of a philosopher is verified and if there is a mutex to prevent a philosopher from dying and starting eating at the same time.

Yes | No

philo testing

- Do not test with more than 200 philosophers.
- Do not test with time_to_die or time_to_eat or time_to_sleep set to values lower than 60 ms.
- Test 1 800 200 200. The philosopher should not eat and should die.
- Test 5 800 200 200. No philosopher should die.
- Test 5 800 200 200 7. No philosopher should die and the simulation should stop when every philosopher has eaten at least 7 times.
- Test 4 410 200 200. No philosopher should die.
- Test 4 310 200 100. One philosopher should die.
- Test with 2 philosophers and check the different times: a death delayed by more than 10 ms is unacceptable.
- Test with any values of your choice to verify all the requirements. Ensure philosophers die at the right time, that they don't steal forks, and so forth.

Yes | No

Bonus part

philo_bonus code

- Ensure the code of philo_bonus complies with the following requirements and ask for explanations.
- Check that there is one process per philosopher and that the main process is not a philosopher.
- Check that there are no orphan processes at the end of the execution of this program.
- Check if there is a single semaphore that represents the number of forks.
- Check the output is protected against multiple access. To avoid a scrambled display.
- Check how the death of a philosopher is verified and if there is a semaphore to prevent a philosopher from dying and starting eating at the same time.

Yes | No

philo_bonus testing

- Do not test with more than 200 philosophers.
- Do not test with time_to_die or time_to_eat or time_to_sleep set to values lower than 60 ms.
- Test 5 800 200 200. No philosopher should die.
- Test 5 800 200 200 7. No philosopher should die and the simulation should stop when every philosopher has eaten at least 7 times.
- Test 4 410 200 200. No philosopher should die.
- Test 4 310 200 100. One philosopher should die.
- Test with 2 philosophers and check the different times: a death delayed by more than 10 ms is unacceptable.
- Test with any values of your choice to verify all the requirements. Ensure philosophers die at the right time, that they don't steal forks, and so forth.

Yes | No

Ratings

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

Ok | Outstanding project

Empty work | Incomplete work | Invalid compilation | Norme | Cheat

Crash | Concerning situation | Leaks | Forbidden function

Can't support / explain code

Conclusion

Give this repository a star.