Remember that the quality of the defenses, hence the quality of the of the school on the labor market depends on you. The remote defences during the Covid crisis allows more flexibility so you can progress into your curriculum, but also brings more risks of cheat, injustice, laziness, that will harm everyone's skills development. We do count on your maturity and wisdom during these remote defenses for the benefits of the entire community.

SCALE FOR PROJECT PHILOSOPHERS

You should evaluate 1 student in this team

Git repository

`git@vogsphere-v2-bg.1`

Introduction

Please respect 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 person (or the group) evaluated the eventual dysfunctions of the work. Take the time to discuss and debate the problems you have identified.

- You must consider that there might be some difference in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade him/her as honestly as possible. The pedagogy is valid only and only if peer-evaluation is conducted seriously.

Guidelines

- Only grade the work that is in the student or group's GiT repository.

- Double-check that the GiT repository belongs to the student or the group. Ensure that the work is for the relevant project and 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 other than the content of the official repository.

- 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 prior to starting the defence.

- Use the flags available on this scale to signal an empty repository, non-functioning program, a norm error, cheating etc. In these cases, the grading is over and the final grade is 0 (or -42 in case of cheating). However, with the exception of cheating, you are encouraged to continue to discuss your work (even if you have not finished it) in order 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 defence, 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

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.

Yes | No

Philo_one code

- Check the code of philo_one for the following things and ask for explanation.
- Check if there is one thread 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 if the output is protected against multiple access. To avoid a scrambled view

- Check how the death of a philosopher is checked and if there is a mutex to protect that a philosopher dies and start eating at the same time.

Yes | No

Philo_one test

- Do not test with more than 200 philosphers
- Do not test with time_to_die or time_to_eat or time_to_sleep under 60 ms
- Test with 5 800 200 200, no one should die!
- Test with 5 800 200 200 7, no one should die and the simulation should stop when all the philosopher has eaten at least 7 times each.
- Test with 4 410 200 200, no one should die!
- Test with 4 310 200 100, a philosopher should die!
- Test with 2 philosophers and check the different times (a death delayed by more than 10 ms is unacceptable).
- Test your own values to check all the rules. Check if a philosopher dies at the right time, if they don't steal forks, etc.

Yes | No

Philo_two code

- Check the code of philo_two for the following things and ask for explanation.
- Check if there is one thread per philosopher.
- Check if there is a single semaphore that represents the number of forks.
- Check if the output is protected against multiple access. To avoid a scrambled view.
- Check how the death of a philosopher is checked and if there is a semaphore to protect that a philosopher dies and start eating at the same time.

Yes | No

Philo_two test

- Do not test with more than 200 philosphers
- Do not test with time_to_die or time_to_eat or time_to_sleep under 60 ms
- Test with 5 800 200 200, no one should die!
- Test with 5 800 200 200 7, no one should die and the simulation should stop when all the philosopher has eaten at least 7 times each.
- Test with 4 410 200 200, no one should die!
- Test with 4 310 200 100, a philosopher should die!
- Test with 2 philosophers and check the different times (a death delayed by more than 10 ms is unacceptable).
- Test your own values to check all the rules. Check if a philosopher dies at the right time, if they don't steal forks, etc.

Yes | No

Philo_three code

- Check the code of philo_three for the following things and ask for explanation.
- Check if there will be one process per philosopher and that the first process waits for all of them.
- Check if there is a single semaphore that represent the number of forks.
- Check if the output is protected against multiple access. To avoid a scrambled view.
- Check how the death of a philosopher is checked and if there is a semaphore to protect that a philosopher dies and start eating at the same time.

Yes | No

Philo_three test

- Do not test with more than 200 philosphers
- Do not test with time_to_die or time_to_eat or time_to_sleep under 60 ms
- Test with 5 800 200 200, no one should die!
- Test with 5 800 200 200 7, no one should die and the simulation should stop when all the philosopher has eaten at least 7 times each.
- Test with 4 410 200 200, no one should die!
- Test with 4 310 200 100, a philosopher should die!
- Test with 2 philosophers and check the different times (a death delayed by more than 10 ms is unacceptable).
- Test your own values to check all the rules. Check if a philosopher dies at the right time, if they don't steal forks, etc.

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