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

Leave a comment on this evaluation

Finish evaluation