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 (/PROJECTS/42CURSUS-PHILOSOPHERS)

You should evaluate 1 student in this team

★

Git repository

`git@vogsphere.msk.21-school.ru:vogsphere/intra-uuid-e82a928`

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 were 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 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/24616/en.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

Philo code

- Check the code of Philo for the following things and ask for an explanation.
- Check if there is one thread per philosopher.
- Check there's 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 output should never produce 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 test

- Do not test with more than 200 philosophers
- Do not test with time_to_die or time_to_eat or time_to_sleep under 60 ms
- Test with 1 800 200 200, the philosopher should not eat and should die!
- 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 with your values to check all the rules. Check if a philosopher dies at the right time if they don't steal forks, etc.

☑ Yes ☒ No

Bonus Part

Philo_bonus code

- Check the code of philo_bonus for the following things and ask for an 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 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 starts eating at the same time.

☑ Yes ☒ No

Philo_bonus test

- Do not test with more than 200 philosophers
- 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 with your 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

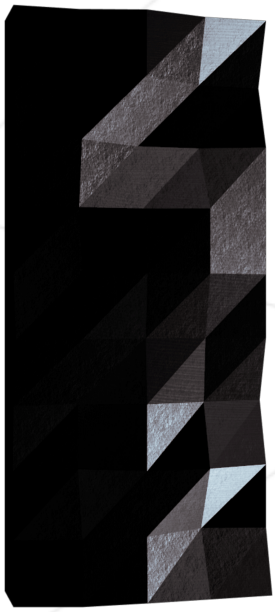
Empty work | No author file | Invalid compilation | Norme | Cheat | Crash

Leaks | Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation



Philosophers

I've never thought philosophy would be so deadly.

Summary: In this project, you will learn the basics of threading a process. You will learn how to make threads. You will discover the mutex.

Contents

Chapter I

Introduction

Philosophy (from Greek, philosophia, literally "love of wisdom") is the study of general and fundamental questions about existence, knowledge, values, reason, mind, and language. Such questions are often posed as problems to be studied or resolved. The term was probably coined by Pythagoras (c. 570 – 495 BCE). Philosophical methods include questioning, critical discussion, rational argument, and systematic presentation. Classic philosophical questions include: Is it possible to know anything and to prove it? What is most real? Philosophers also pose more practical and concrete questions such as: Is there a best way to live? Is it better to be just or unjust (if one can get away with it)? Do humans have free will?

Historically, "philosophy" encompassed any body of knowledge. From the time of Ancient Greek philosopher Aristotle to the 19th century, "natural philosophy" encompassed astronomy, medicine, and physics. For example, Newton's 1687 Mathematical Principles of Natural Philosophy later became classified as a book of physics. In the 19th century, the growth of modern research universities led academic philosophy and other disciplines to professionalize and specialize. In the modern era, some investigations that were traditionally part of philosophy became separate academic disciplines, including psychology, sociology, linguistics, and economics.

Other investigations closely related to art, science, politics, or other pursuits remained part of philosophy. For example, is beauty objective or subjective? Are there many scientific methods or just one? Is political utopia a hopeful dream or hopeless fantasy? Major sub-fields of academic philosophy include metaphysics ("concerned with the fundamental nature of reality and being"), epistemology (about the "nature and grounds of knowledge [and]...its limits and validity"), ethics, aesthetics, political philosophy, logic and philosophy of science.

Chapter II

Mandatory part

You will have to write one program for the mandatory part and one for the bonus part but they will have the same basic rules:

- This project is to be coded in C, following the Norm. Any leak, crash, undefined behavior, or norm error means 0 to the project.
- One or more philosophers are sitting at a round table doing one of three things: eating, thinking, or sleeping.
- While eating, they are not thinking or sleeping, while sleeping, they are not eating or thinking and of course, while thinking, they are not eating or sleeping.
- The philosophers sit at a circular table with a large bowl of spaghetti in the center.
- There are some forks on the table.
- As spaghetti is difficult to serve and eat with a single fork, it is assumed that a philosopher must eat with two forks, one for each hand.
- The philosophers must never be starving.
- Every philosopher needs to eat.
- Philosophers don't speak with each other.
- Philosophers don't know when another philosopher is about to die.
- Each time a philosopher has finished eating, he will drop his forks and start sleeping.
- When a philosopher is done sleeping, he will start thinking.
- The simulation stops when a philosopher dies.
- Each program should have the same options: number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]
    - number_of_philosophers: is the number of philosophers and also the number of forks
    - time_to_die: is in milliseconds, if a philosopher doesn't start eating 'time_to_die' milliseconds after starting his last meal or the beginning of the simulation, it dies

  - time_to_eat: is in milliseconds and is the time it takes for a philosopher to eat. During that time he will need to keep the two forks.
  - time_to_sleep: is in milliseconds and is the time the philosopher will spend sleeping.
  - number_of_times_each_philosopher_must_eat: argument is optional, if all philosophers eat at least 'number_of_times_each_philosopher_must_eat' the simulation will stop. If not specified, the simulation will stop only at the death of a philosopher.

- Each philosopher should be given a number from 1 to 'number_of_philosophers'.
- Philosopher number 1 is next to philosopher number 'number_of_philosophers'. Any other philosopher with the number N is seated between philosopher N - 1 and philosopher N + 1
- Any change of status of a philosopher must be written as follows (with X replaced with the philosopher number and timestamp_in_ms the current timestamp in milliseconds)
  - timestamp_in_ms X has taken a fork
  - timestamp_in_ms X is eating
  - timestamp_in_ms X is sleeping
  - timestamp_in_ms X is thinking
  - timestamp_in_ms X died
- The status printed should not be scrambled or intertwined with another philosopher's status.
- You can't have more than 10 ms between the death of a philosopher and when it will print its death.
- Again, philosophers should avoid dying!

Program name	`philo`
Turn in files	`philo/`
Makefile	`Yes`
Arguments	`number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
External functs.	`memset, printf, malloc, free, write, usleep, gettimeofday, pthread_create, pthread_detach, pthread_join, pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_unlock`
Libft authorized	`No`
Description	`philosopher with threads and mutex`

In this version the specific rules are:

- One fork between each philosopher, therefore if they are multiple philosophers, there will be a fork at the right and the left of each philosopher.
- To avoid philosophers duplicating forks, you should protect the forks state with a mutex for each of them.
- Each philosopher should be a thread.

Chapter III

Bonus

Program name	`philo_bonus`
Turn in files	`philo_bonus/`
Makefile	`Yes`
Arguments	`number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
External functs.	`memset, printf, malloc, free, write, fork, kill, exit, pthread_create, pthread_detach, pthread_join, usleep, gettimeofday, waitpid, sem_open, sem_close, sem_post, sem_wait, sem_unlink`
Libft authorized	`No`
Description	`philosopher with processes and semaphore`

In this version the specific rules are:

- All the forks are in the middle of the table.
- They have no states in memory but the number of available forks is represented by a semaphore.
- Each philosopher should be a process and the main process should not be a philosopher.