./philosophers
author: nvasilev

Dining philosophers problem

Problem statement

Five philosophers dine together at the same table. Each philosopher has their own place at the table. Their philosophical problem in this instance is that the dish served is a kind of spaghetti that has to be eaten with two forks.

There is a fork between each plate. Each philosopher can only alternately think and eat. Moreover, a philosopher can only eat their spaghetti when they have both a left and right fork. Thus two forks will only be available when their two nearest neighbors are thinking, not eating. After an individual philosopher finishes eating, they will put down both forks. The problem is how to design a regimen (a concurrent algorithm) such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think (an issue of incomplete information).

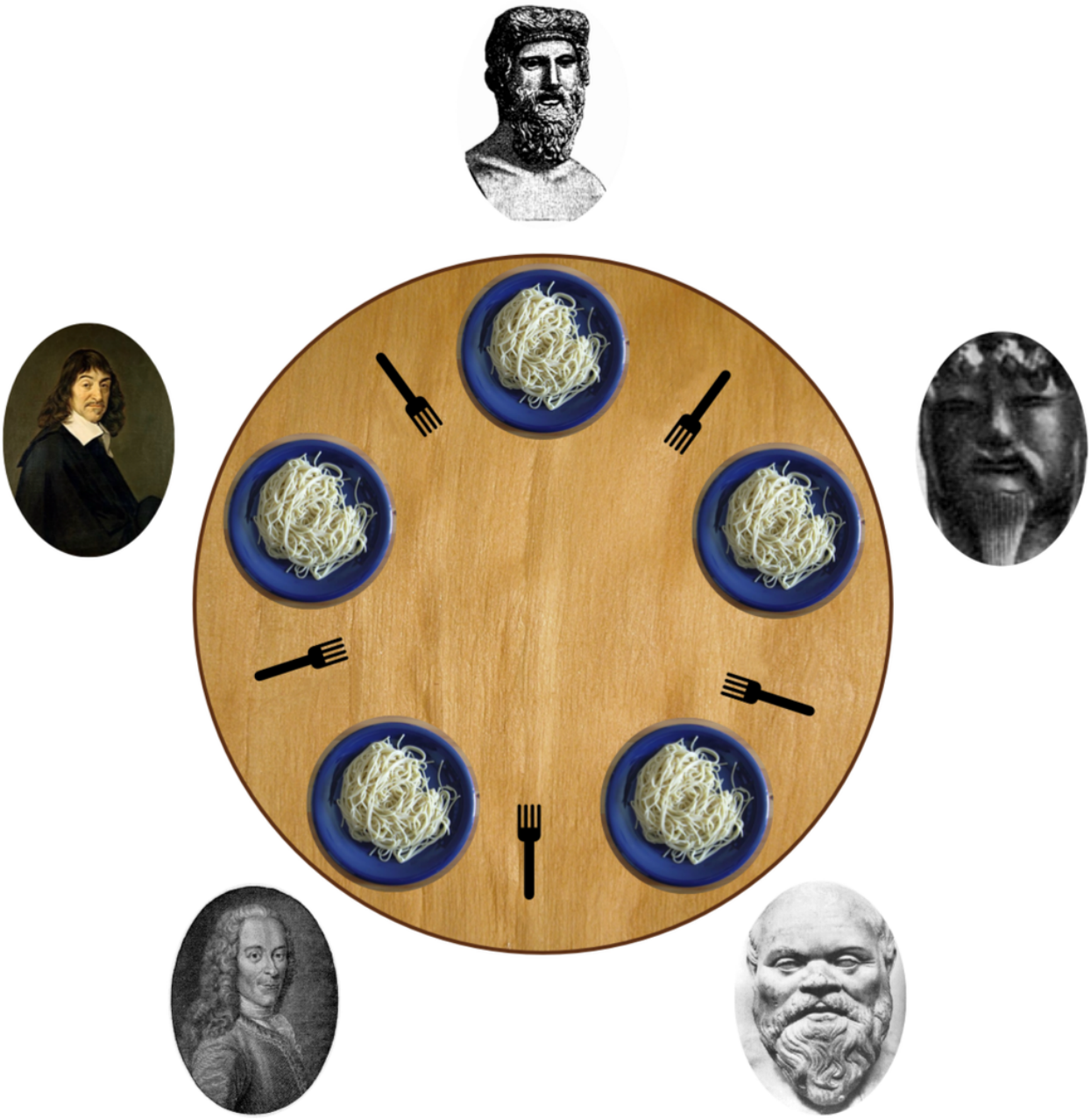
Problems

The problem was designed to illustrate the challenges of avoiding deadlock, a system state in which no progress is possible. To see that a proper solution to this problem is not obvious, consider a proposal in which each philosopher is instructed to behave as follows:

think until the left fork is available; when it is, pick it up;
think until the right fork is available; when it is, pick it up;
when both forks are held, eat for a fixed amount of time;
put the left fork down;
put the right fork down;
repeat from the beginning.
However, they each will think for an undetermined amount of time. And may end up holding a left fork thinking, staring at the right side of the plate, unable to eat because there is no right fork until they starve.

Resource starvation, mutual exclusion, and livelock are other types of sequence and access problems.[source: Wikipedia]

Illustration of the dining philosophers problem



Rules

Allowed functions (mandatory)

- **Memory:** memset, malloc, free
- **Printing:** printf, write
- **Time:** usleep, gettimeofday
- **Threads:** pthread_create, pthread_detach, pthread_join, pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_unlock

Allowed functions (bonus)

- **Memory:** memset, malloc, free
- **Printing:** printf, write
- **Time:** usleep, gettimeofday
- **Processes:** fork, kill, exit, waitpid
- **Threads:** pthread_create, pthread_detach, pthread_join
- **Semaphores:** sem_open, sem_close, sem_post, sem_wait, sem_unlink

Subjects

- **EN:** https://cdn.intra.42.fr/pdf/pdf/58800/en.subject.pdf
- **FR:** https://cdn.intra.42.fr/pdf/pdf/58801/fr.subject.pdf

Documentation

Dining philosophers problem

- https://www.javatpoint.com/os-dining-philosophers-problem
- The Dining Philosophers Problem
- https://medium.com/science-journal/the-dining-philosophers-problem-fded861c37ed
- https://en.wikipedia.org/wiki/Dining_philosophers_problem

Threads

- https://www.geeksforgeeks.org/multithreading-c-2/
- Introduction To Threads (pthreads) | C Programming Tutorial

Mutexes

- https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/
- Thread synchronization with mutexes in C

Semaphores

- https://www.tutorialspoint.com/mutex-vs-semaphore
- https://www.geeksforgeeks.org/mutex-vs-semaphore/

CodeVault -> videos on procs, threads, mutexes and semaphores

- https://www.youtube.com/c/CodeVault

Visualizer

- https://github.com/nafuka11/philosophers-visualizer