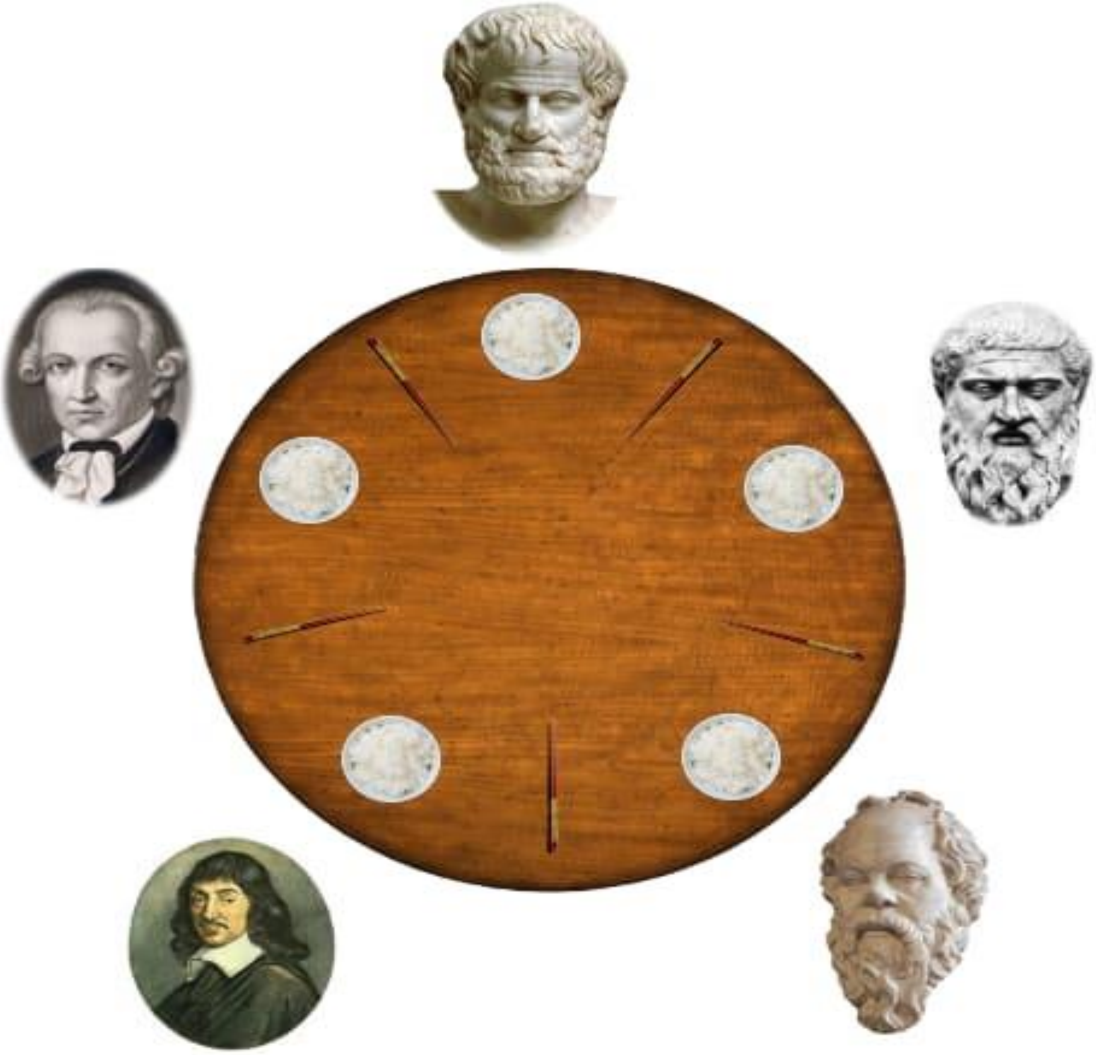


Condition-Based Solution

12/25/23

Parallel Programming

[Author]

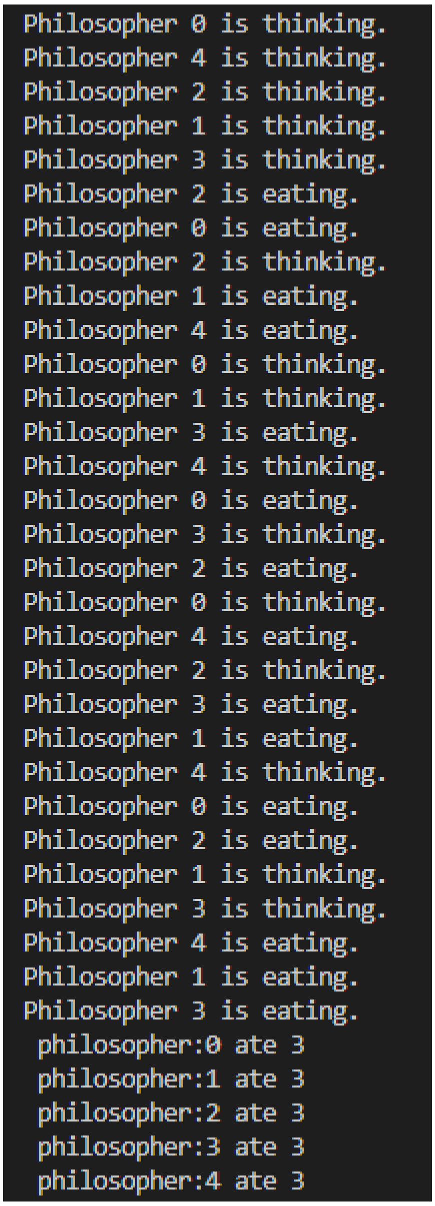
A Condition-Based solution includes some inter-communication thread specifications, such as using wait, notify, notify_all, etc. Each philosopher leaves their own signature on forks, for example, a personal identification. To use a condition, the Forks object must first be demonstrated; the Forks object is a shared resource among all philosophers. If a fork is not available (locked), the philosopher must wait until it is free. At this point, thanks to the wait() method, condition objects wait until they are freed. The function making the object free is notified by notify and notify_all.

The pickFork() method locks the first fork that comes from the thread (to avoid a race condition) and adds the philosopher's marker, "-1," indicating that nobody has taken the fork object so far. If someone quickly seizes the chance with their own ID, it means the fork is not free and is currently in use by another philosopher. Other incoming philosophers have to wait for a while.

After eating, the putFork() method must unlock the condition object to allow other philosophers to use it. However, some philosophers may be in the waiting process, so the owner of the fork needs to wake up all waiting philosophers with the notify_all() method.

All philosophers communicate over condition objects; if the object is not available, the philosopher waits until it is free. After eating, the philosopher changes the status of the fork and wakes up waiting philosophers.

OUTPUT



```
Philosopher 0 is thinking.
Philosopher 4 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 0 is eating.
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 4 is eating.
Philosopher 0 is thinking.
Philosopher 1 is thinking.
Philosopher 3 is eating.
Philosopher 4 is thinking.
Philosopher 0 is eating.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 0 is thinking.
Philosopher 4 is eating.
Philosopher 2 is thinking.
Philosopher 3 is eating.
Philosopher 1 is eating.
Philosopher 4 is thinking.
Philosopher 0 is eating.
Philosopher 2 is eating.
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is eating.
Philosopher 1 is eating.
Philosopher 3 is eating.
 philosopher:0 ate 3
 philosopher:1 ate 3
 philosopher:2 ate 3
 philosopher:3 ate 3
 philosopher:4 ate 3
```