Parallel Programming Homework Report

Dining Philosophers

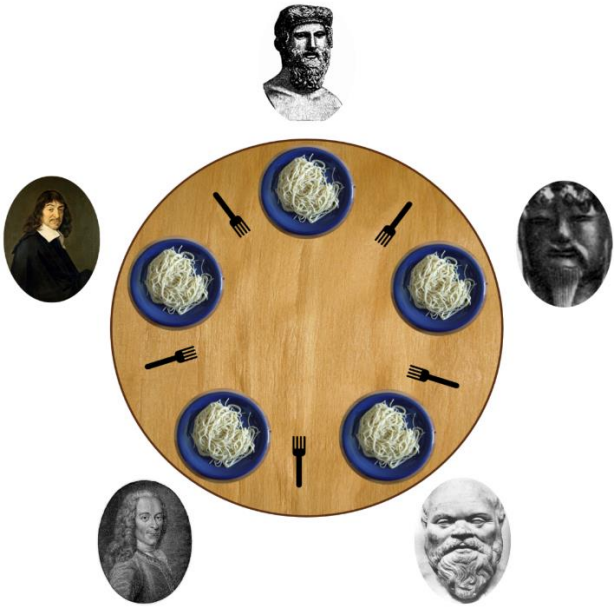
Lehmann and Rabins Solution

190316069 Samet Şengün

210316002 Furkan Soylek

What is Dining Philosophers?

The Dining Philosophers problem is a classic synchronization and concurrency problem often used to illustrate the challenges of designing and implementing parallel algorithms. It was formulated by E.W. Dijkstra in 1965. The problem is framed in terms of a scenario where five philosophers sit around a dining table. Each philosopher thinks deeply and alternates between thinking and eating. A dining philosopher can only eat when he has both the left and right chopsticks. The challenge is to design a solution that avoids deadlock (where no philosopher can finish eating) and ensures that philosophers do not starve (never get a chance to eat).[1]



What is Lehmann and Rabins Solution?

Lehmann and Rabin's approach, mandates that each philosopher selects her forks in a random sequence. In the event that the second fork is not promptly accessible, the philosopher must place down both forks and make another attempt. Although livelock remains a potential concern if all philosophers consistently choose forks in the same order, the introduction of randomization significantly diminishes the likelihood of this occurrence.[2]

Explaining Algorithm:

```
def eat(self):
        rnd=random.randint(0,1)

        if rnd==0:
        # Try to pick up the left fork
            with self.left_fork(self.index):
                # Try to pick up the right fork
                time.sleep(2 + random.random() * 3)
                if self.right_fork.info()==-1:
                    with self.right_fork(self.index):
                        self.process()

        else:
            with self.right_fork(self.index):
                # Try to pick up the right fork
                time.sleep(2 + random.random() * 3)
                if self.left_fork.info()==-1:
                    with self.left_fork(self.index):
                        self.process()
```

rnd = random.randint(0, 1): Generates a random number and assigns it to the variable rnd. This number can be either 0 or 1.

if rnd == 0:: If rnd is 0, it represents a process attempting to pick up the left fork.

a. with self.left_fork(self.index):: Tries to pick up the left fork. Inside the with block, a context manager is used to acquire the left fork and automatically release its lock.

b. time.sleep(2 + random.random() * 3): Waits for a random period. This may be done to simulate a process picking up the fork and prevent another process from taking a fork for a while.

c. if self.right_fork.info() == -1:: Checks the status of the right fork. If the right fork is empty (the info() function returns -1), an nested with block is initiated to attempt to pick up the right fork.

d. with self.right_fork(self.index):: Tries to pick up the right fork.

e. self.process(): After acquiring the fork, calls the process function to perform some actions.

else:: If rnd is not 0, it represents a process attempting to pick up the right fork. The steps are similar to picking up the left fork, but this time, the right fork is prioritized.

a. with self.right_fork(self.index):: Tries to pick up the right fork.

b. time.sleep(2 + random.random() * 3): Waits for a random period.

c. if self.left_fork.info() == -1:: Checks the status of the left fork. If the left fork is empty, initiates a nested with block to attempt to pick up the left fork.

d. with self.left_fork(self.index):: Tries to pick up the left fork.

e. self.process(): After acquiring the fork, calls the process function to perform some actions.a

Authorities

[1] ***https://en.wikipedia.org/wiki/Dining_philosophers_problem***

[2] *Daniel Lehmann and Michael O. Rabin. 1981. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In* Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages *(POPL '81). ACM, New York, NY, USA, 133-138.*