CSE 3237

PARALLEL PROGRAMMING

FALL 2023

DINING PHILOSOPHERS

Token Ring Strategy

Tuğberk Özkara – 210315011

25 December 2023

Introduction

The Dining Philosophers problem is a classic synchronization and concurrency problem that demonstrates the challenges of resource sharing and deadlock avoidance in a multiprocessing environment. The problem tells an imaginary scenario where a group of philosophers is seated around a dining table. Each philosopher alternates between thinking and eating. But to eat they need two forks placed on either side of them. The challenge appears in ensuring that the philosophers do not starve, and the deadlocks are avoided when they attempt to acquire the necessary forks.

About The Dining Philosophers Problem

The core issues of the problem are concurrency, deadlock, and starvation. They share common resources like forks and must coordinate their actions to avoid conflicts. If all philosophers simultaneously pick up one fork and wait for the second, a deadlock can occur where no philosopher can eat. Philosophers must have access to both forks to eat. If they are unable to acquire both forks, they may starve.

About The Token Ring Solution Strategy

One proper solution for the Dining Philosophers problem is the Token Ring strategy. In this approach, a token is passed among the philosophers in a circular manner. A philosopher can only eat when they hold the token, ensuring that only one philosopher at a time is in the critical section like eating. This token could be a helicopter toy or something in real life scenarios.

Token Ring solution has a similar look with The Arbitrator solution as they both give allowance to one philosopher at a time with some kind of access method. They are both approaches to addressing synchronization issues of the problem, but they have distinct differences in their implementations.

In the Arbitrator solution, a centralized entity names arbitrator like a waiter or a moderator is introduced to control access to the shared resources. Philosophers request permission from the arbitrator before attempting to acquire the forks. The arbitrator grants permission based on a set of rules, ensuring that philosophers can only proceed to eat when it is safe to do so.

On the other hand, the Token Ring solution is a decentralized approach. Philosophers are organized in a circular structure, and a unique token is passed among them. A philosopher can only eat when holding the token, which they must pass to the next philosopher after finishing their meal. This approach ensures that only one philosopher at a time is allowed to enter the critical section of eating, avoiding deadlocks.

Components of Token Ring Solution

1. Forks

   Forks are the shared resources of the problem that philosophers need to eat. Each philosopher requires two forks to begin eating. They are modeled as context managers in this problem.

```python
class Fork:
    def __init__(self, index: int):
        self.index: int = index
        self.lock: threading.Lock = threading.Lock()
        self.picked_up: bool = False
        self.owner: int = -1

    def __enter__(self):
        return self

    def __call__(self, owner: int):
        if self.lock.acquire():
            self.owner = owner
            self.picked_up = True
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        self.lock.release()
        self.picked_up = False
        self.owner = -1

    def __str__(self):
        return f"F{self.index:2d} ({self.owner:2d})"
```

2. Philosopher

   The philosopher is modeled as a concurrent thread. They alternate between thinking and attempting to eat. To eat, a philosopher must have two forks.

```
class Philosopher(threading.Thread):
    def __init__(self, index: int, left_fork: Fork, right_fork: Fork,
spaghetti: int, token: Token):
        super().__init__()
        self.index: int = index
        self.left_fork: Fork = left_fork
        self.right_fork: Fork = right_fork
        self.spaghetti: int = spaghetti
        self.eating: bool = False
        self.token: Token = token

    def run(self):
        while self.spaghetti > 0:
            self.think()
            self.eat_with_token()

    def think(self):
        time.sleep(3 + random.random() * 3)

    def eat(self):
        with self.left_fork(self.index):
            time.sleep(5 + random.random() * 5)
            with self.right_fork(self.index):
                self.spaghetti -= 1
                self.eating = True
                time.sleep(5 + random.random() * 5)
                self.eating = False

    def eat_with_token(self):
        self.token.acquire_token(self.index)
        with self.left_fork(self.index):
            time.sleep(5 + random.random() * 5)
            with self.right_fork(self.index):
                self.spaghetti -= 1
                self.eating = True
                time.sleep(5 + random.random() * 5)
                self.eating = False
        self.token.release_token()

    def __str__(self):
        return f"P{self.index:2d} ({self.spaghetti:2d})"
```

Instead of the *eat* method, *eat_with_token* method is implemented. By *eat_with_token method*, a philosopher first needs to acquire the token instead of directly grabbing the forks and starting to eat.

3. Token

   The *Token* class manages the token, a unique decentralized permission that allows a philosopher to eat. The token is passed in a ring among the philosophers.

```
class Token:
    def __init__(self, num_philosophers: int):
        self.lock = threading.Lock()
        self.num_philosophers = num_philosophers
        self.token_holder = 0

    def acquire_token(self, philosopher_index: int):
        with self.lock:
            while self.token_holder != philosopher_index:
                self.lock.release()
                time.sleep(0.1)
                self.lock.acquire()

    def release_token(self):
        with self.lock:
            self.token_holder = (self.token_holder + 1) %
self.num_philosophers
```

The Flow

1. A philosopher acquires the token to enter the critical section of eating. The *Token* class manages the token and provides methods for acquiring and releasing the token.
2. The philosopher attempts to acquire both forks. If both forks are received, they proceed to eat. So, each philosopher in the *Philosopher* class, uses the Token instance to acquire the token before attempting to pick up the forks and eat.
3. After eating, the philosopher releases both forks and passes the token to the next philosopher in the ring.

Solution

1. Avoiding Deadlocks: Since a philosopher can only eat when holding the token, deadlock situations are avoided.
2. Fairness: Unlike the Arbitrator Strategy, each philosopher gets an equal opportunity to eat as the token circulates in a ring.
3. Resource Utilization: The solution ensures that resources are utilized optimally, minimizing the chance of starvation.

Example Steps

Philosopher1 eating;

```
================================================================================
        E           T           T           T           T
F 0 ( 0)      F 1 ( 0)      F 2 (-1)      F 3 (-1)      F 4 (-1)      F 0 ( 0)
      P 0 ( 6)      P 1 ( 7)      P 2 ( 7)      P 3 ( 7)      P 4 ( 7)      :  1
```

Philosopher2 eating;

```
================================================================================
        T           E           T           T           T
F 0 (-1)      F 1 ( 1)      F 2 ( 1)      F 3 (-1)      F 4 (-1)      F 0 (-1)
      P 0 ( 6)      P 1 ( 6)      P 2 ( 7)      P 3 ( 7)      P 4 ( 7)      :  1
```

Philosopher3 eating;

```
================================================================================
        T           T           E           T           T
F 0 (-1)      F 1 (-1)      F 2 ( 2)      F 3 ( 2)      F 4 (-1)      F 0 (-1)
      P 0 ( 6)      P 1 ( 6)      P 2 ( 6)      P 3 ( 7)      P 4 ( 7)      :  1
```

Conclusion

The Token Ring solution provides an effective and suitable way to address the challenges posed by the Dining Philosophers problem. By defining a token that regulates the access to the critical section, this solution ensures both concurrency and deadlock avoidance, creating a proper environment for philosophers for their thinking and eating activities.