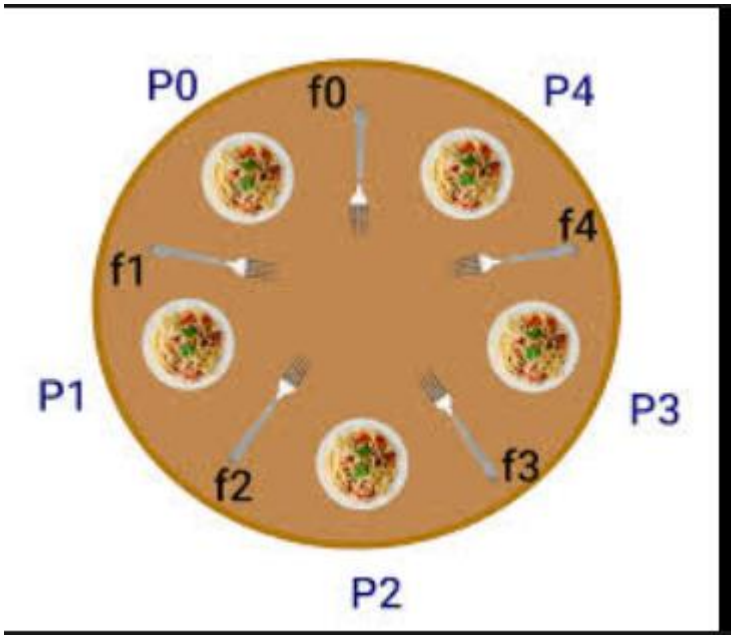

Original Image

Why deadlocks can occur in the problem setup:

Deadlocks can occur in the dining philosophers problem due to the potential for circular waiting. Each philosopher needs two forks to eat, and if all philosophers pick up their left fork simultaneously, a circular dependency is created, and they will be waiting indefinitely for the right fork to be released.

How your proposed solution avoids deadlock:

My solution attempts to avoid deadlock by implementing a deadlock detection mechanism. The *deadlockCheckThread* periodically checks for deadlock by attempting to acquire each fork. If it successfully acquires any fork, it assumes that the system is not in deadlock. If it fails to acquire any fork, it assumes a deadlock and releases all forks.

This solution breaks the circular wait condition by releasing all forks if a deadlock is detected, allowing the philosophers to pick up any available fork and continue their execution.

Fairness of the solution:

The fairness of the solution is not guaranteed. The choice of which philosopher gets to eat next depends on the availability of both forks and bowls. The code uses pthread_mutex_trylock to attempt to acquire locks without blocking. This means that a philosopher may not be able to acquire both forks and a bowl simultaneously, leading to some philosophers being stuck in thinking state for extended periods.The fairness can be improved by implementing a more sophisticated algorithm for resource allocation or by introducing a priority system to ensure that all philosophers get a chance to eat in a reasonable amount of time.

Rough estimate of how often a philosopher is able to eat: The frequency with which a philosopher is able to eat depends on the randomness introduced by the rand() % 3 delays in thinking and eating functions. Without a more deterministic approach or fairness improvement, it's challenging to provide an accurate estimate. However, due to the non-blocking nature of pthread_mutex_trylock, some philosophers may have a higher chance of acquiring resources and eating more frequently than others.

```
Philosopher Summary:
Philosopher 0: Thought 69 times, Ate 64 times.
Philosopher 1: Thought 59 times, Ate 57 times.
Philosopher 2: Thought 65 times, Ate 57 times.
Philosopher 3: Thought 67 times, Ate 64 times.
Philosopher 4: Thought 70 times, Ate 56 times.
```

After approx. 2 min of running the code. I got this as output.

So I think it is a Fair solution , but in these of cases, nothing is guaranteed.