Dining Philosophers' Solution With The " Token Ring"

The strategy used to resolve deadlock involves the use of a token. The token functions as a semaphore, regulating philosophers' access to forks and tokens. Each philosopher receives the token to request access to the fork and releases the token once the transaction is completed. This arrangement symbolizes a mechanism in which the token circulates on a ring.

This strategy prevents two philosophers from eating at the same time through the use of tokens. In order to prevent deadlock, a cyclical order is achieved by eating a single philosopher. The token determines the order in which philosophers use forks and controls this order.

Fork Class

```
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
```

This method is called when an instance of the Fork class is created. This method initializes the fork object. First, it takes the index parameter, which represents a unique ID of the fork, and associates the fork with this ID. Then, a threading.lock object, self.lock, is created to control safe sharing of the fork. The self.picked_up property is initialized to False to indicate that the fork has been picked up, and the self.owner property is initialized to -1 to represent the owner of the fork.

__init__(self, index: int): Allows the creation of a fork. Initializes properties that keep track of the index of the fork, a lock object, whether the fork is currently taken, and who holds the fork

```
def __enter__(self):
    return self
```

__enter__(self): A context manager protocol that allows a fork to be used in a with statement.

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

__call__(self, owner: int): Represents a philosopher picking up a fork. When the lock is acquired, the owner of the fork and the status in which it was acquired are updated.

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

__exit__(self, exc_type, exc_value, traceback): Represents exiting a fork from the with statement. The lock is released and the state of the fork is reset.

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

The purpose of the __str__ method is to convert Fork class objects into strings. This method runs automatically when the str() function is called on a fork object.

Philosopher Class

```
class Philosopher(threading.Thread):
    def __init__(self, index: int, left_fork: Fork, right_fork: Fork, spaghetti: int, token: threading.
    Semaphore):
        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: threading.Semaphore = token
```

init_ Method:

index: An integer specifying the unique ID of the philosopher.

left_fork: A Fork object representing the philosopher's left fork.

right_fork: A Fork object representing the philosopher's right fork.

spaghetti: An integer indicating how many plates of pasta the philosopher can eat.

token: A threading.Semaphore object used to control philosophers' access to tokens.

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

run Method:

The run method is used as the main run method of the Philosopher class. It represents the philosopher's cycle of thinking and eating.

The cycle continues until the philosopher's spaghetti counter is reset.

The think and eat methods simulate the philosopher's thinking and eating actions, respectively.

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

think Method:

The think method simulates the philosopher's thinking process.

```
def eat(self):
    with self.token:
        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
```

eat Method:

The eat method simulates the philosopher's eating process.

Controls fork access via token semaphore.

It symbolizes the philosopher reaching the fork by using the accesses secured in the with block with the left_fork and right_fork forks.

After the meal is eaten, it reduces the philosopher's pasta counter and waits for a period of time (5 + random.random() * 5).

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

The purpose of the __str__ method is to convert Fork class objects into strings. This method runs automatically when the str() function is called on a fork object.