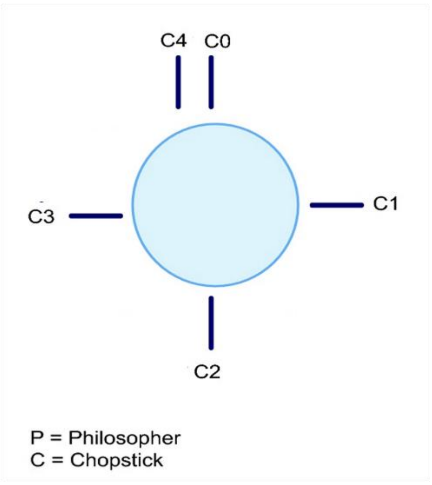
Name : Pratik Kanhaiya Rathod
Roll : 321056
PRN : 22010885
Batch : A2

Assignment No. 6

Aim : Implement Concurrent Dining Philosopher Problem.

Theory:

- **What is dinning philosopher problem?**
  1) The dining philosopher's problem is a classic problem in computer science and concurrent programming. The problem consists of a group of philosophers who are sitting at a round table with a bowl of rice in front of each of them, and a chopstick between each pair of adjacent philosophers.
  2) The goal of each philosopher is to eat the rice, but to do so they must first pick up the chopsticks on either side of them. However, the chopsticks are a shared resource and only one philosopher can hold a chopstick at a time.
  3) The problem arises when all philosophers attempt to pick up the chopsticks at the same time, leading to a deadlock or starvation. If all the philosophers pick up the chopstick on their left simultaneously, then they will be unable to pick up the chopstick on their right, leadingto deadlock. Similarly, if they all try to pick up the chopstick on their right simultaneously, then again, they will be unable to pick up the chopstick on their left, leading to deadlock.



  4) There are several solutions to this problem, including the use of semaphores, monitors, or other synchronization mechanisms. Themost common solution is to introduce a protocol that prevents deadlock, such as the "resource hierarchy solution" or the "Chandy/Misra solution".
  5) In the resource hierarchy solution, the chopsticks are assigned an ordering, and the philosophers must pick up the chopsticks in order

from the lowest numbered chopstick to the highest numbered chopstick. This ensures that at least one philosopher can pick up bothchopsticks and eat, preventing deadlock.

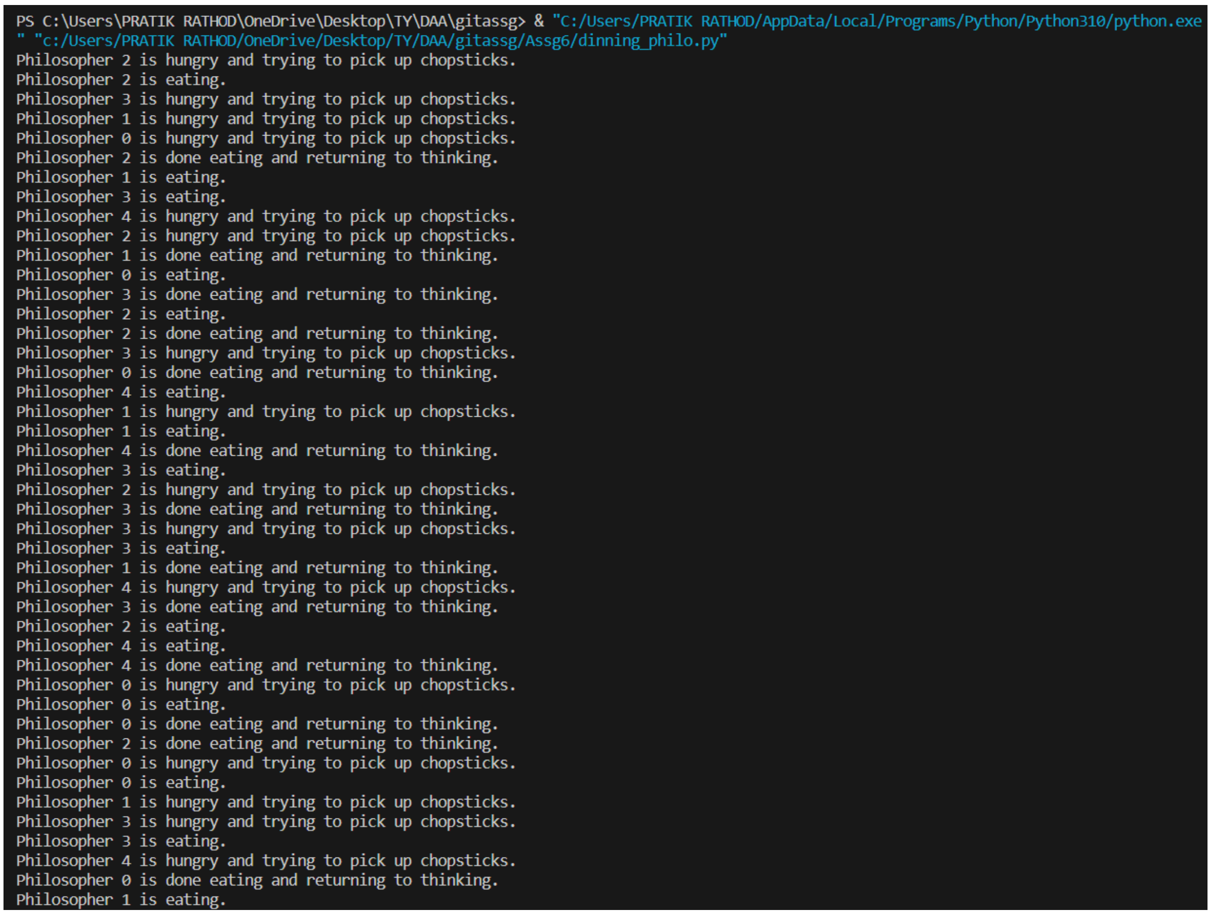
6) In general, the dining philosopher’s problem illustrates the challenges of concurrent programming and the need for careful synchronization mechanisms to prevent problems such as deadlock and starvation.

Implementation of concurrent dinning philosopher problem in Python:

```
import threading
import time
import random

# Number of philosophers
num_philosophers = 5

# Locks for chopsticks
chopsticks = [threading.Lock() for _ in range(num_philosophers)]

# Function for a philosopher to eat
def philosopher(id):
    left_chopstick = chopsticks[id]
    right_chopstick = chopsticks[(id + 1) % num_philosophers]

    while True:
        # Think for a while
        time.sleep(random.uniform(0, 1))

        # Try to pick up chopsticks
        print(f"Philosopher {id} is hungry and trying to pick up chopsticks.")
        left_chopstick.acquire()
        right_chopstick.acquire()

        # Eat for a while
        print(f"Philosopher {id} is eating.")
        time.sleep(random.uniform(0, 1))

        # Release chopsticks
        left_chopstick.release()
        right_chopstick.release()

        # Return to thinking
        print(f"Philosopher {id} is done eating and returning to thinking.")

# Create philosopher threads
philosophers = [threading.Thread(target=philosopher, args=(i,)) for i in
range(num_philosophers)]
```

```
# Start philosopher threads
for philosopher in philosophers:
    philosopher.start()

# Wait for philosopher threads to finish
for philosopher in philosophers:
    philosopher.join()
```

Output :



```
PS C:\Users\PRATIK RATHOD\OneDrive\Desktop\TY\DAA\gitassg> & "C:/Users/PRATIK RATHOD/AppData/Local/Programs/Python/Python310/python.exe
" "c:/Users/PRATIK RATHOD/OneDrive/Desktop/TY/DAA/gitassg/Assg6/dinning_philo.py"
Philosopher 2 is hungry and trying to pick up chopsticks.
Philosopher 2 is eating.
Philosopher 3 is hungry and trying to pick up chopsticks.
Philosopher 1 is hungry and trying to pick up chopsticks.
Philosopher 0 is hungry and trying to pick up chopsticks.
Philosopher 2 is done eating and returning to thinking.
Philosopher 1 is eating.
Philosopher 3 is eating.
Philosopher 4 is hungry and trying to pick up chopsticks.
Philosopher 2 is hungry and trying to pick up chopsticks.
Philosopher 1 is done eating and returning to thinking.
Philosopher 0 is eating.
Philosopher 3 is done eating and returning to thinking.
Philosopher 2 is eating.
Philosopher 2 is done eating and returning to thinking.
Philosopher 3 is hungry and trying to pick up chopsticks.
Philosopher 0 is done eating and returning to thinking.
Philosopher 4 is eating.
Philosopher 1 is hungry and trying to pick up chopsticks.
Philosopher 1 is eating.
Philosopher 4 is done eating and returning to thinking.
Philosopher 3 is eating.
Philosopher 2 is hungry and trying to pick up chopsticks.
Philosopher 3 is done eating and returning to thinking.
Philosopher 3 is hungry and trying to pick up chopsticks.
Philosopher 3 is eating.
Philosopher 1 is done eating and returning to thinking.
Philosopher 4 is hungry and trying to pick up chopsticks.
Philosopher 3 is done eating and returning to thinking.
Philosopher 2 is eating.
Philosopher 4 is eating.
Philosopher 4 is done eating and returning to thinking.
Philosopher 0 is hungry and trying to pick up chopsticks.
Philosopher 0 is eating.
Philosopher 0 is done eating and returning to thinking.
Philosopher 2 is done eating and returning to thinking.
Philosopher 0 is hungry and trying to pick up chopsticks.
Philosopher 0 is eating.
Philosopher 1 is hungry and trying to pick up chopsticks.
Philosopher 3 is hungry and trying to pick up chopsticks.
Philosopher 3 is eating.
Philosopher 4 is hungry and trying to pick up chopsticks.
Philosopher 0 is done eating and returning to thinking.
Philosopher 1 is eating.
```