**Ex. No: 12**

**Date:**

**SOLVING DINING PHILOSOPHER'S PROBLEM USING SEMAPHORES**

**Problem Statement:**

To solve dining philosopher’s problem using semaphores

**Problem Description:**

The Dining Philosophers Problem is a classic concurrency challenge where philosophers sit around a circular table, alternating between thinking and eating. To eat, a philosopher must use two adjacent chopsticks. The problem highlights the need for synchronization to prevent conflicts and deadlocks. Philosophers must follow rules: picking up available chopsticks and releasing them when finished. The objective is to devise a solution using semaphores or mutexes to enable philosophers to dine without contention or deadlock, emphasizing the importance of synchronization in concurrent systems.

**Algorithm:**

1. Initialize Semaphore class for chopstick management.
2. Create a list of chopsticks, each represented as a Semaphore.
3. Initialize philosopher\_status list for tracking philosopher states.
4. Define philosopher function to handle philosopher behavior.
5. Implement think function to simulate thinking.
6. Define dine function to manage dining behavior.
7. Use wait and signal to control access to chopsticks.
8. Print philosopher status using print\_status function.
9. In the main section, create philosopher threads and start them.
10. Join all philosopher threads to ensure completion of their tasks.

**Code:**

import threading

import time

class Semaphore:

    def \_\_init\_\_(self, initial\_value=1):

        self.value = initial\_value

        self.lock = threading.Lock()

    def wait(self):

        with self.lock:

            while self.value <= 0:

                pass

            self.value -= 1

    def signal(self):

        with self.lock:

            self.value += 1

NUM\_PHILOSOPHERS = 5

chopsticks = [Semaphore(1) for \_ in range(NUM\_PHILOSOPHERS)]

philosopher\_status = ["thinking"] \* NUM\_PHILOSOPHERS

cycles\_completed = 0

def philosopher(id):

    global cycles\_completed

    while cycles\_completed < 10:

        think(id)

        dine(id)

        cycles\_completed += 1

def think(id):

    philosopher\_status[id] = "thinking"

    print\_status()

    time.sleep(2)

def dine(id):

    left\_chopstick = id

    right\_chopstick = (id + 1) % NUM\_PHILOSOPHERS

    if chopsticks[left\_chopstick].value > 0 and chopsticks[right\_chopstick].value > 0:

        chopsticks[left\_chopstick].wait()

        chopsticks[right\_chopstick].wait()

        philosopher\_status[id] = "eating"

        print\_status()

        time.sleep(1)

        chopsticks[left\_chopstick].signal()

        chopsticks[right\_chopstick].signal()

    else:

        philosopher\_status[id] = "hungry"

        print\_status()

def print\_status():

    for i in range(NUM\_PHILOSOPHERS):

        print(f'Philosopher {i} is {philosopher\_status[i]}')

    print()

if \_\_name\_\_ == "\_\_main\_\_":

    philosophers = []

    for i in range(NUM\_PHILOSOPHERS):

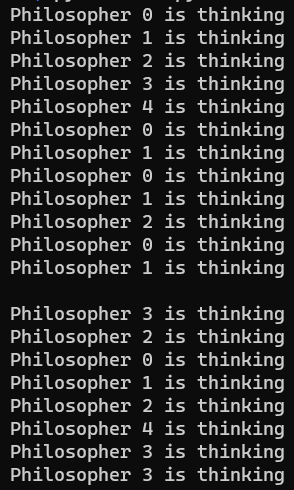
        philosopher\_thread = threading.Thread(target=philosopher, args=(i,))

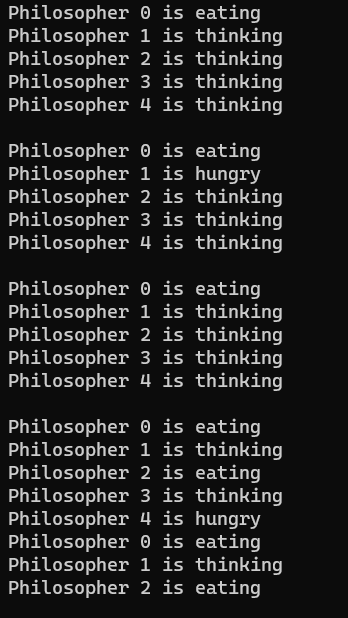
        philosopher\_thread.start()

        philosophers.append(philosopher\_thread)

    for philosopher\_thread in philosophers:

        philosopher\_thread.join()

**Output:**

****

**Result:**

Thus, the dining philosopher problem using semaphores has been implemented successfully.