**EX 12: Design a C program to simulate the concept of Dining-Philosophers problem.**

**Aim:**

To design a C program that simulates the Dining Philosophers Problem using multithreading and synchronization to avoid deadlock and starvation.

**Algorithm:**

1. **Start.**
2. **Initialize Resources:**
   * Use an array of mutexes to represent forks.
   * Create threads for each philosopher.
3. **Define the philosopher's activities:**
   * Each philosopher alternates between thinking and eating.
   * Ensure proper locking and unlocking of forks using mutexes to avoid deadlocks.
4. **Prevent Deadlocks:**
   * Use resource ordering or allow one philosopher to pick up the forks in a reversed order**.**
5. **Simulate Execution:**
   * Run the threads, showing philosophers' actions.
6. **Join Threads:**
   * Wait for all philosopher threads to complete.
7. **Terminate.**

**Program:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_PHILOSOPHERS 5

sem\_t forks[NUM\_PHILOSOPHERS];

void\* philosopher(void\* num) {

int id = \*(int\*)num;

while (1) {

printf("Philosopher %d is thinking.\n", id);

sleep(1);

sem\_wait(&forks[id]);

sem\_wait(&forks[(id + 1) % NUM\_PHILOSOPHERS]);

printf("Philosopher %d is eating.\n", id);

sleep(1);

sem\_post(&forks[id]);

sem\_post(&forks[(id + 1) % NUM\_PHILOSOPHERS]);

}

}

int main() {

pthread\_t philosophers[NUM\_PHILOSOPHERS];

int philosopher\_ids[NUM\_PHILOSOPHERS];

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

sem\_init(&forks[i], 0, 1);

philosopher\_ids[i] = i;

pthread\_create(&philosophers[i], NULL, philosopher, &philosopher\_ids[i]);

}

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

pthread\_join(philosophers[i], NULL);

}

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) {

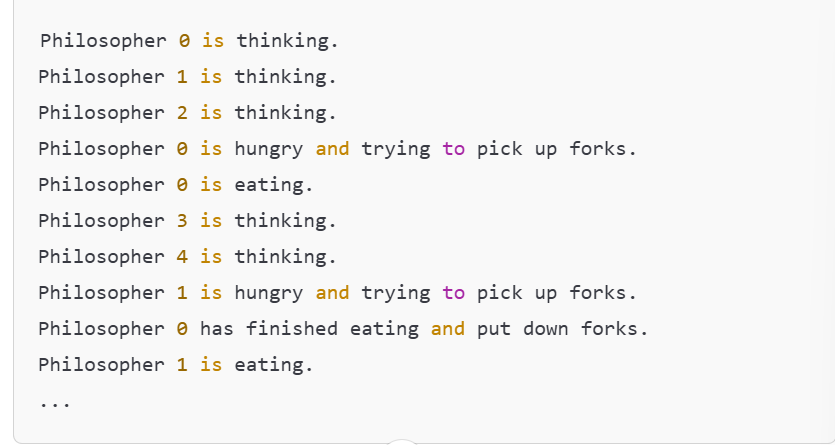
sem\_destroy(&forks[i]);

}

return 0;

}

**OUTPUT:**

****