k. NARENDRA-192321162

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

**Aim:**

To simulate the Dining Philosophers problem in C, where multiple philosophers need to share resources (forks) to eat without causing a deadlock or resource contention.

**Algorithm:**

1. Create a set of philosophers and forks.
2. Each philosopher thinks for a random amount of time and then tries to pick up two forks.
3. If a philosopher picks up both forks, they eat for a random time and then release the forks.
4. Ensure that no philosopher holds a fork indefinitely to prevent deadlock.
5. Use mutexes to avoid race conditions when philosophers pick up or release forks.

**Procedure:**

1. Define a philosopher structure, which represents each philosopher.
2. Use mutexes to represent forks, ensuring mutual exclusion.
3. Create a thread for each philosopher using pthread\_create().
4. Simulate thinking, picking up forks, eating, and releasing forks using random delays.
5. Use pthread\_join() to ensure the main thread waits for philosophers to finish.

CODE:

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

#define NUM\_PHILOSOPHERS 5

pthread\_mutex\_t forks[NUM\_PHILOSOPHERS];

void\* philosopher(void\* num) { int id = \*((int\*) num); int left\_fork = id;

int right\_fork = (id + 1) % NUM\_PHILOSOPHERS;

while (1) {

printf("Philosopher %d is thinking.\n", id); usleep(rand() % 1000000);

pthread\_mutex\_lock(&forks[left\_fork]);

printf("Philosopher %d picked up left fork %d.\n", id, left\_fork);

pthread\_mutex\_lock(&forks[right\_fork]);

printf("Philosopher %d picked up right fork %d.\n", id, right\_fork);

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

usleep(rand() % 1000000);

pthread\_mutex\_unlock(&forks[left\_fork]); printf("Philosopher %d put down left fork %d.\n", id, left\_fork);

pthread\_mutex\_unlock(&forks[right\_fork]); printf("Philosopher %d put down right fork %d.\n", id, right\_fork);

}

}

int main() {

pthread\_t threads[NUM\_PHILOSOPHERS]; int philosopher\_ids[NUM\_PHILOSOPHERS];

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) { pthread\_mutex\_init(&forks[i], NULL);

}

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

philosopher\_ids[i] = i;

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

}

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) { pthread\_join(threads[i], NULL);

}

for (int i = 0; i < NUM\_PHILOSOPHERS; i++) { pthread\_mutex\_destroy(&forks[i]);

}

return 0;

}

OUTPUT:

