NAME: G. GANESH

REG NO: 192373008

EXERICSE-12

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

Aim:

To simulate the concept of the Dining Philosophers problem using a C program.

Algorithm:

1. **Problem Description:**

   - ➢ There are N philosophers sitting at a round table. Each philosopher thinks and alternately eats food.
   - ➢ To eat, a philosopher needs two forks. The forks are placed between each pair of philosophers.
   - ➢ Each philosopher picks up two forks (one at a time) and eats. After eating, they put down the forks.

2. **Steps:**

   - ➢ Create a semaphore or mutex for each fork to ensure that only one philosopher can use a fork at a time.
   - ➢ Each philosopher will try to pick up the forks in a certain order (e.g., left fork first, then right fork).
   - ➢ If both forks are available, the philosopher eats. After eating, they put down both forks.
   - ➢ Use mutual exclusion (mutex) and synchronization to avoid deadlock and resource contention.

Procedure:

1. Use pthread library to create multiple threads (philosophers).
2. Use mutexes or semaphores to ensure mutual exclusion when accessing shared resources (forks).
3. Each philosopher will wait for forks, eat, and then put the forks down.
4. Synchronize the program to avoid deadlock and ensure proper eating order.

Code:

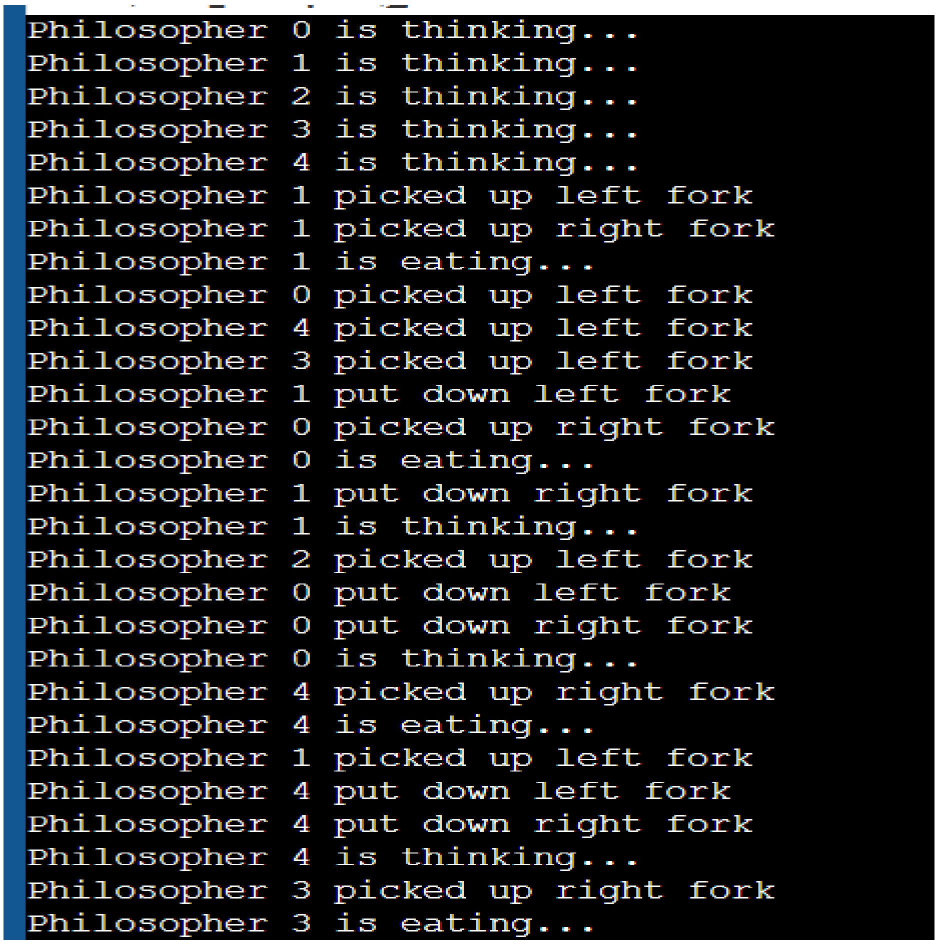
```
#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>
```

```
#include <unistd.h>
#define NUM_PHILOSOPHERS 5
pthread_mutex_t forks[NUM_PHILOSOPHERS]; // Mutex for each fork
pthread_t philosophers[NUM_PHILOSOPHERS];
void *philosopher(void *num) {
    int philosopher_num = *(int *)num;
    while (1) {
        printf("Philosopher %d is thinking...\n", philosopher_num);
        sleep(rand() % 2);
        pthread_mutex_lock(&forks[philosopher_num]);
        printf("Philosopher %d picked up left fork\n", philosopher_num);
        pthread_mutex_lock(&forks[(philosopher_num + 1) % NUM_PHILOSOPHERS]);
        printf("Philosopher %d picked up right fork\n", philosopher_num);
        printf("Philosopher %d is eating...\n", philosopher_num);
        sleep(rand() % 3);  // Philosopher is eating
        pthread_mutex_unlock(&forks[philosopher_num]);
        printf("Philosopher %d put down left fork\n", philosopher_num);
  pthread_mutex_unlock(&forks[(philosopher_num + 1) % NUM_PHILOSOPHERS]);
        printf("Philosopher %d put down right fork\n", philosopher_num);
    }
    return NULL;
}
int main() {
    int philosopher_num[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_num[i] = i;
        pthread_create(&philosophers[i], NULL, philosopher, &philosopher_num[i]);
    }
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(philosophers[i], NULL);
    }
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_destroy(&forks[i]);
    }
    return 0;
}
```

Result:

The Dining Philosophers problem was successfully simulated using the concepts of multithreading and mutual exclusion. Each philosopher alternates between thinking and eating while ensuring proper synchronization to avoid issues like deadlock.

Output:



```
Philosopher 0 is thinking...
Philosopher 1 is thinking...
Philosopher 2 is thinking...
Philosopher 3 is thinking...
Philosopher 4 is thinking...
Philosopher 1 picked up left fork
Philosopher 1 picked up right fork
Philosopher 1 is eating...
Philosopher 0 picked up left fork
Philosopher 4 picked up left fork
Philosopher 3 picked up left fork
Philosopher 1 put down left fork
Philosopher 0 picked up right fork
Philosopher 0 is eating...
Philosopher 1 put down right fork
Philosopher 1 is thinking...
Philosopher 2 picked up left fork
Philosopher 0 put down left fork
Philosopher 0 put down right fork
Philosopher 0 is thinking...
Philosopher 4 picked up right fork
Philosopher 4 is eating...
Philosopher 1 picked up left fork
Philosopher 4 put down left fork
Philosopher 4 put down right fork
Philosopher 4 is thinking...
Philosopher 3 picked up right fork
Philosopher 3 is eating...
```