B. Siva Shirish - 192324016

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.

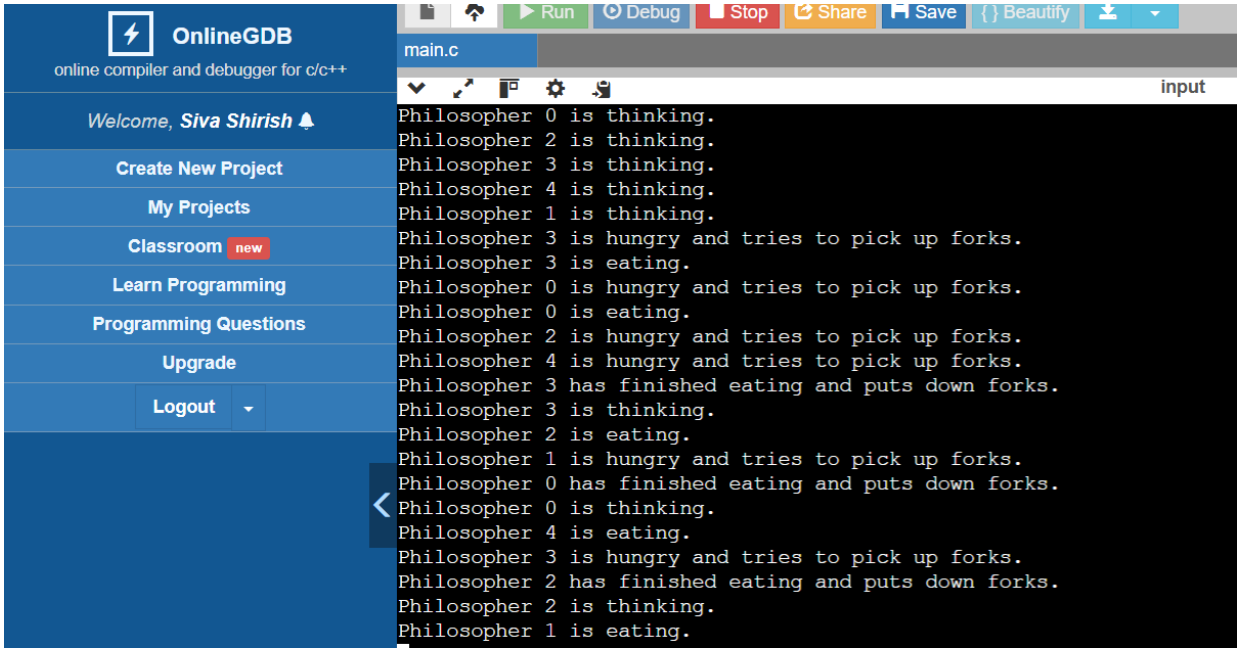
CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>


#define NUM_PHILOSOPHERS 5

pthread_mutex_t forks[NUM_PHILOSOPHERS];
```

```
void *philosopher(void *arg) {
    int id = *(int *)arg;
    int left_fork = id;
    int right_fork = (id + 1) % NUM_PHILOSOPHERS;

    while (1) {

        printf("Philosopher %d is thinking.\n", id);
        sleep(rand() % 3 + 1);


        printf("Philosopher %d is hungry and tries to pick up forks.\n", id);


        if (id % 2 == 0) {
            pthread_mutex_lock(&forks[left_fork]);
            pthread_mutex_lock(&forks[right_fork]);
        } else {
            pthread_mutex_lock(&forks[right_fork]);
            pthread_mutex_lock(&forks[left_fork]);
        }

        printf("Philosopher %d is eating.\n", id);
        sleep(rand() % 3 + 1);


        pthread_mutex_unlock(&forks[left_fork]);
        pthread_mutex_unlock(&forks[right_fork]);
        printf("Philosopher %d has finished eating and puts down forks.\n", id);
    }

    pthread_exit(NULL);
}

int main() {
    pthread_t philosophers[NUM_PHILOSOPHERS];
    int ids[NUM_PHILOSOPHERS];

    // Initialize mutexes for forks
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_init(&forks[i], NULL);
```

```
    }

    // Create philosopher threads
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        ids[i] = i;
        if (pthread_create(&philosophers[i], NULL, philosopher, &ids[i]) != 0) {
            perror("Failed to create thread");
            return 1;
        }
    }

    // Wait for threads to finish (not really necessary in this simulation)
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(philosophers[i], NULL);
    }

    // Destroy mutexes
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_destroy(&forks[i]);
    }

    return 0;
}
```

OUTPUT:


OnlineGDB
online compiler and debugger for c/c++
Welcome, Siva Shirish
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Upgrade
Logout
Run
Debug
Stop
Share
Save
{ } Beautify
main.c
input
Philosopher 0 is thinking.
Philosopher 2 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 1 is thinking.
Philosopher 3 is hungry and tries to pick up forks.
Philosopher 3 is eating.
Philosopher 0 is hungry and tries to pick up forks.
Philosopher 0 is eating.
Philosopher 2 is hungry and tries to pick up forks.
Philosopher 4 is hungry and tries to pick up forks.
Philosopher 3 has finished eating and puts down forks.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 1 is hungry and tries to pick up forks.
Philosopher 0 has finished eating and puts down forks.
Philosopher 0 is thinking.
Philosopher 4 is eating.
Philosopher 3 is hungry and tries to pick up forks.
Philosopher 2 has finished eating and puts down forks.
Philosopher 2 is thinking.
Philosopher 1 is eating.