**192311035**

**M.RAVI SAI VINAY**

# EXPERIMENT-12

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

## AIM:-

To simulate the Dining Philosophers problem in C using multithreading. This problem illustrates synchronization and deadlock handling where philosophers are seated at a table, each needing two chopsticks to eat.

## ALGORITHM:-

1. Create 5 philosopher threads.
2. Each philosopher needs two chopsticks to eat (simulated by mutex locks).
3. The philosophers will pick up chopsticks and eat in a specific order to avoid deadlock.
4. Use mutexes to represent chopsticks and ensure only one philosopher can use a chopstick at a time.
5. Implement a strategy to avoid deadlock (e.g., using resource hierarchy or even/odd strategies).

## CODE:-

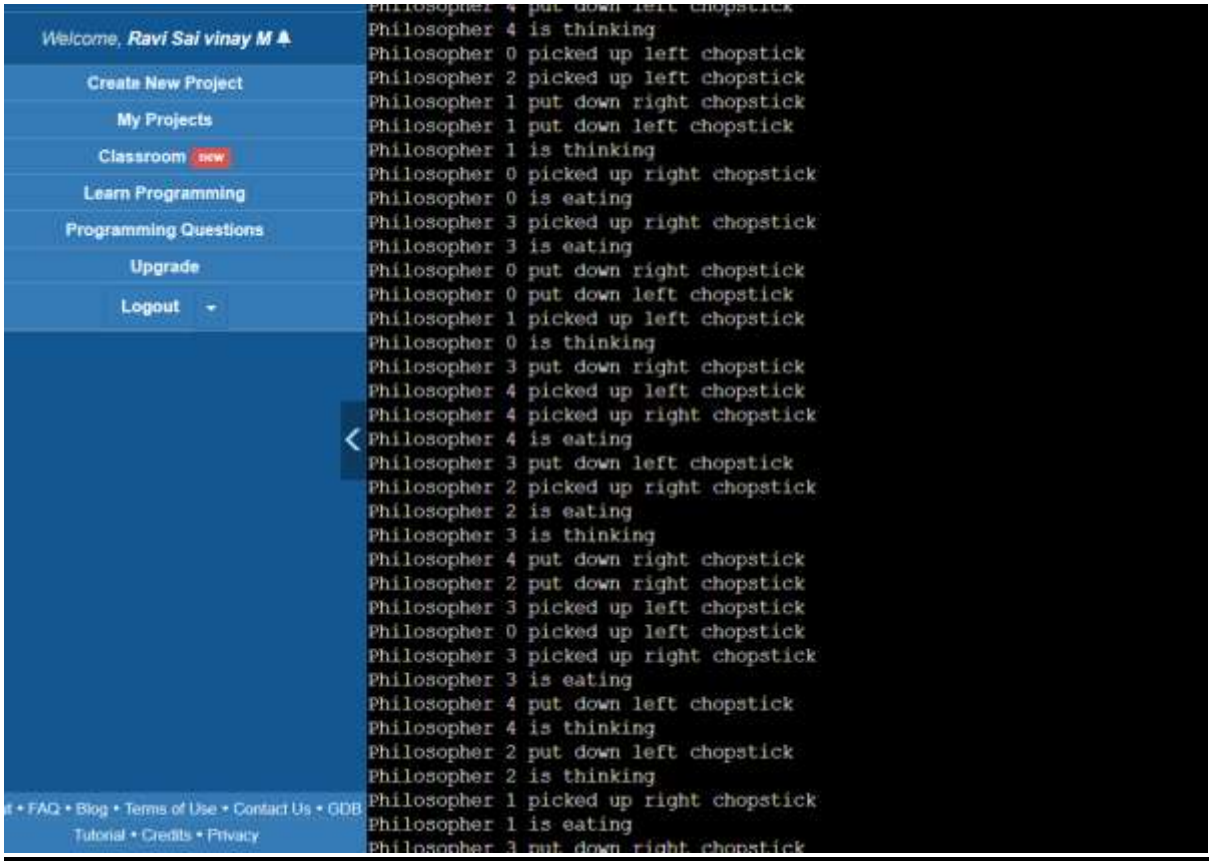
```
#include <stdio.h>

#include <pthread.h>

#include <unistd.h>


#define NUM_PHILOSOPHERS 5


pthread_mutex_t chopsticks[NUM_PHILOSOPHERS];


void* philosopher(void* num) {
```

```
int id = *((int*)num);
int left = id;
int right = (id + 1) % NUM_PHILOSOPHERS;

while (1) {
    printf("Philosopher %d is thinking\n", id);

    // Simulate thinking time
    usleep(1000);

    // Pick up the left chopstick
    pthread_mutex_lock(&chopsticks[left]);
    printf("Philosopher %d picked up left chopstick\n", id);

    // Pick up the right chopstick
    pthread_mutex_lock(&chopsticks[right]);
    printf("Philosopher %d picked up right chopstick\n", id);

    // Simulate eating
    printf("Philosopher %d is eating\n", id);
    usleep(1000);

    // Put down the right chopstick
    pthread_mutex_unlock(&chopsticks[right]);
```

```
        printf("Philosopher %d put down right chopstick\n", id);

        // Put down the left chopstick
        pthread_mutex_unlock(&chopsticks[left]);
        printf("Philosopher %d put down left chopstick\n", id);
    }
}

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

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

    // Create philosopher threads
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        philosopher_ids[i] = i;
        pthread_create(&philosophers[i], NULL, philosopher, &philosopher_ids[i]);
    }

    // Wait for philosopher threads to finish (this won't happen as philosophers are always eating or thinking)
```

```
    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(&chopsticks[i]);
    }

    return 0;
}
```

## OUTPUT:-



## RESULT:-

The program simulates the Dining Philosophers problem with five philosophers. Each philosopher alternates between thinking and eating, while properly managing access to the chopsticks using mutexes. This program avoids deadlock by ensuring that philosophers pick up the chopsticks in a defined order (left first, then right).