K.GOWTHAM-192372078

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.

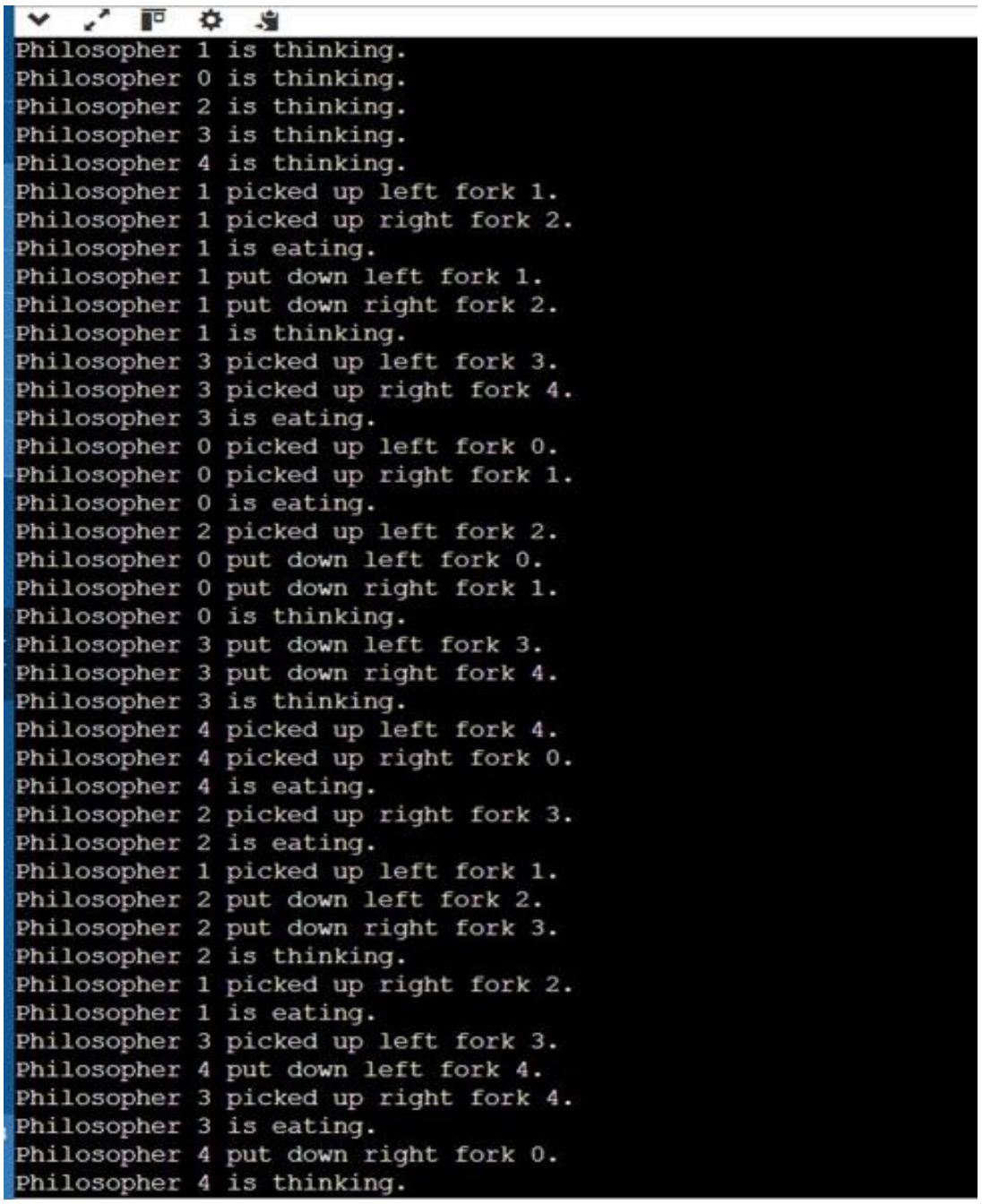
CODE:

```
#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); // Thinking time

    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); // Eating time

    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:


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