

## Experiment-12: Design a C program to simulate the concept of Dining-Philosophers problem

Aim:

The aim of this program is to simulate the Dining Philosophers Problem, a classic synchronization problem that demonstrates the issues of concurrency and resource sharing. The goal is to simulate how philosophers (represented by threads) sit at a table, thinking and eating, while sharing limited resources (forks) without causing deadlock or resource contention.

Problem Description:

In the Dining Philosophers Problem:

- There are n philosophers sitting around a circular table.
- Each philosopher has a plate of food and needs two forks to eat (one fork to the left, one to the right).
- Philosophers alternate between thinking and eating.
- Philosophers can pick up a fork when it's available, but they must wait if either fork is unavailable.
- The challenge is to prevent deadlock (where all philosophers are waiting for a fork indefinitely) and ensure there is no starvation (where a philosopher is never able to eat).

Solution Approach:

We can use mutexes or semaphores to synchronize access to the forks. Each philosopher needs to pick up the two adjacent forks (left and right), but this access must be synchronized to avoid race conditions. The program will simulate the dining philosophers problem using mutexes for synchronization.

Procedure:

1. Mutexes: Use mutex locks to ensure that only one philosopher can pick up each fork at a time.
2. Philosophers as threads: Each philosopher is represented by a thread.
3. Thinking & Eating: Philosophers will alternate between thinking and eating. When eating, they will acquire the two adjacent forks (mutex locks). When thinking, they will release the forks and wait.
4. Deadlock Avoidance: Deadlock is avoided by ensuring that each philosopher always picks up the forks in the same order (left first, right second).
- 5.

C Program to Simulate the Dining Philosophers Problem:

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#define NUM_PHILOSOPHERS 5
pthread_mutex_t forks[NUM_PHILOSOPHERS];
void *philosopher(void *num) {
    int id = *(int *)num;
    while (1) {
        printf("Philosopher %d is thinking\n", id);
        usleep(rand() % 1000);
        printf("Philosopher %d is hungry\n", id);
        pthread_mutex_lock(&forks[id]);
        printf("Philosopher %d picked up left fork\n", id);
        pthread_mutex_lock(&forks[(id + 1) % NUM_PHILOSOPHERS]);
        printf("Philosopher %d picked up right fork\n", id);
```

```

        printf("Philosopher %d is eating\n", id);
        usleep(rand() % 1000);
        pthread_mutex_unlock(&forks[(id + 1) % NUM_PHILOSOPHERS]);
        printf("Philosopher %d put down right fork\n", id);
        pthread_mutex_unlock(&forks[id]);
        printf("Philosopher %d put down left fork\n", id);
    }
}

int main() {
    pthread_t philosophers[NUM_PHILOSOPHERS];
    int philosopher_ids[NUM_PHILOSOPHERS];
    int i;
    for (i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_init(&forks[i], NULL);
    }
    for (i = 0; i < NUM_PHILOSOPHERS; i++) {
        philosopher_ids[i] = i;
        if (pthread_create(&philosophers[i], NULL, philosopher, (void *)&philosopher_ids[i]) != 0)
        {
            printf("Error creating thread for philosopher %d\n", i);
            return -1;
        }
    }

    for (i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(philosophers[i], NULL);
    }

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

    return 0;
}

```

Output:

## Output

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

```
Philosopher 2 is hungry  
Philosopher 2 picked up left fork  
Philosopher 0 put down right fork  
Philosopher 0 put down left fork  
Philosopher 0 is thinking  
Philosopher 1 picked up left fork  
Philosopher 4 is thinking  
Philosopher 4 is hungry  
Philosopher 0 is hungry  
Philosopher 0 picked up left fork  
Philosopher 3 put down right fork  
Philosopher 3 put down left fork  
Philosopher 3 is thinking  
Philosopher 2 picked up right fork  
Philosopher 2 is eating  
Philosopher 2 put down right fork  
Philosopher 2 put down left fork  
Philosopher 2 is thinking  
Philosopher 3 is hungry  
Philosopher 3 picked up left fork  
Philosopher 1 picked up right fork  
Philosopher 1 is eating  
Philosopher 2 is hungry  
Philosopher 4 picked up left fork  
192324085|
```

```
=== Execution Halted ===
```