****

**Assignment on Deadlock**

**Submitted To**

**Faculty Name :** **Prof. Dr. Md. Motaharul Islam,**

**Professor, Department of CSE**

**Submitted By**

**Name : Sheikh Sarafat Hossain**

**Id No : 2022-3-60-109**

**Course Title :** Operating System

**Course Code :** CSE325

**Section :** 07

**Semester :** Fall 24

**Date of Submission:** 12 January 2025

**Deadlock**

**Overview**

This report analyzes the implementation of the Dining Philosophers problem, a classic synchronization problem in computer science. The program uses threads and semaphores to simulate philosophers alternating between thinking and eating while avoiding deadlocks.

**Problem Description**

The Dining Philosophers problem involves five philosophers seated at a circular table, each alternating between two activities: thinking and eating. Each philosopher requires two chopsticks to eat, but there are only five chopsticks available. The primary challenge is to prevent deadlocks while ensuring all philosophers can eat.

**Program Description**

This implementation uses POSIX threads (pthread) and semaphores (sem\_t) to synchronize access to the chopsticks. Each philosopher is represented by a thread, and each chopstick is represented by a semaphore.

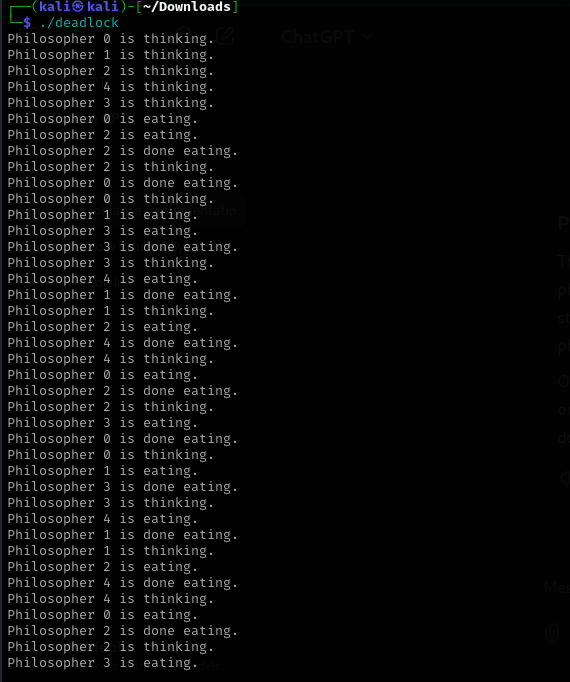
**Key Features**

1. **Alternating Behavior to Prevent Deadlocks**:
   * Philosophers with even IDs pick up the right chopstick first, then the left chopstick.
   * Philosophers with odd IDs pick up the left chopstick first, then the right chopstick.
   * This alternating behavior prevents cyclic waiting, a common cause of deadlocks.
2. **Use of Semaphores**:
   * Each chopstick is a semaphore initialized to 1, indicating availability.
   * sem\_wait() is used to pick up a chopstick, and sem\_post() is used to put it down.
3. **Infinite Simulation**:
   * Philosophers alternate between thinking (simulated by sleep) and eating indefinitely.

**Program Flow**

1. **Initialization**:
   * Five semaphores (chopsticks) are initialized to 1.
   * Philosopher threads are created, each executing the philosopher function.
2. **Philosopher Function**:
   * Each philosopher alternates between thinking and eating.
   * Chopstick acquisition and release are synchronized using semaphores.
3. **Termination**:
   * After execution, semaphores are destroyed to free resources.

**Output:**

****

**Explanation of the Output**

1. **Thinking Phase**:
   * All philosophers initially enter the thinking state.
2. **Eating Phase**:
   * Philosophers acquire the required chopsticks and enter the eating state.
   * Due to alternating behavior, no two adjacent philosophers eat simultaneously, preventing deadlocks.
3. **Completion**:
   * After eating, philosophers release the chopsticks and return to the thinking state.

**Analysis**

This implementation effectively avoids deadlocks by alternating the order in which philosophers pick up chopsticks. The use of semaphores ensures mutual exclusion, preventing simultaneous access to the same chopstick by multiple philosophers. The program achieves fairness, as all philosophers eventually get a chance to eat.

**Limitations**

1. **Infinite Loop**:
   * The program simulates philosophers indefinitely, which might not be practical for certain use cases. A termination condition could be added for better control.
2. **Starvation Not Addressed**:
   * Although deadlocks are avoided, starvation (where some philosophers might wait indefinitely to eat) is not explicitly prevented.

**Recommendations**

1. **Termination Condition**:
   * Add a termination condition (e.g., after a fixed number of iterations) to make the simulation finite.
2. **Starvation Prevention**:
   * Implement a mechanism to track waiting times and prioritize philosophers who have been waiting longer.

**Conclusion**

The program successfully demonstrates a solution to the Dining Philosophers problem by using semaphores and alternating chopstick acquisition strategies. While the implementation avoids deadlocks, further enhancements could address starvation and provide better control over the simulation's runtime.