D -

**Batch: D - 1 Roll No.: 16010122096**

**Experiment No. 06**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE: Implementation of Dining Philosophers problem using mutexes and semaphores.** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Process synchronization algorithms using mutexes and semaphore – Dining Philosopher problem

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 2.** To understand the concept of process, thread and resource management.

**CO 3.** To understand the concepts of process synchronization and deadlock.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems”, McGraw Hill Third Edition.**
3. **Sumitabha Das “ UNIX Concepts & Applications”, McGraw Hill Second**

**Edition.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Knowledge of Concurrency, Mutual Exclusion, Synchronization, Deadlock, Starvation, threads.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Description of the chosen process synchronization algorithm:

### Synchronization Approach:

* **Mutexes**: Each fork is protected by a mutex to ensure exclusive access.
* **Preventing Deadlock**: Implement a strict order for picking up forks to avoid deadlock.

### Implementation Steps:

1. Initialize mutexes for each fork.
2. Create philosopher threads that:
   * Think.
   * Pick up the left and right forks.
   * Eat (critical section).
   * Release the forks and repeat.

**Implementation details:**

#include<bits/stdc++.h>

using namespace std;

const int num\_philosophers = 6;

vector<mutex> forks(num\_philosophers);

void philosopher(int index) {

    mutex& left\_fork = forks[index];

    mutex& right\_fork = forks[(index + 1) % num\_philosophers];

    while (true) {

        cout << "Philosopher " << index << " is thinking" << endl;

        this\_thread::sleep\_for(chrono::seconds(1));

        lock(left\_fork, right\_fork); // Lock both forks

        lock\_guard<mutex> left\_lock(left\_fork, adopt\_lock);

        lock\_guard<mutex> right\_lock(right\_fork, adopt\_lock);

        cout << "Philosopher " << index << " is eating" << endl;

        this\_thread::sleep\_for(chrono::seconds(1));

    }

}

int main() {

    vector<thread> threads;

    for (int i = 0; i < num\_philosophers; ++i) {

        threads.emplace\_back(philosopher, i);

    }

    for (auto& thread : threads) {

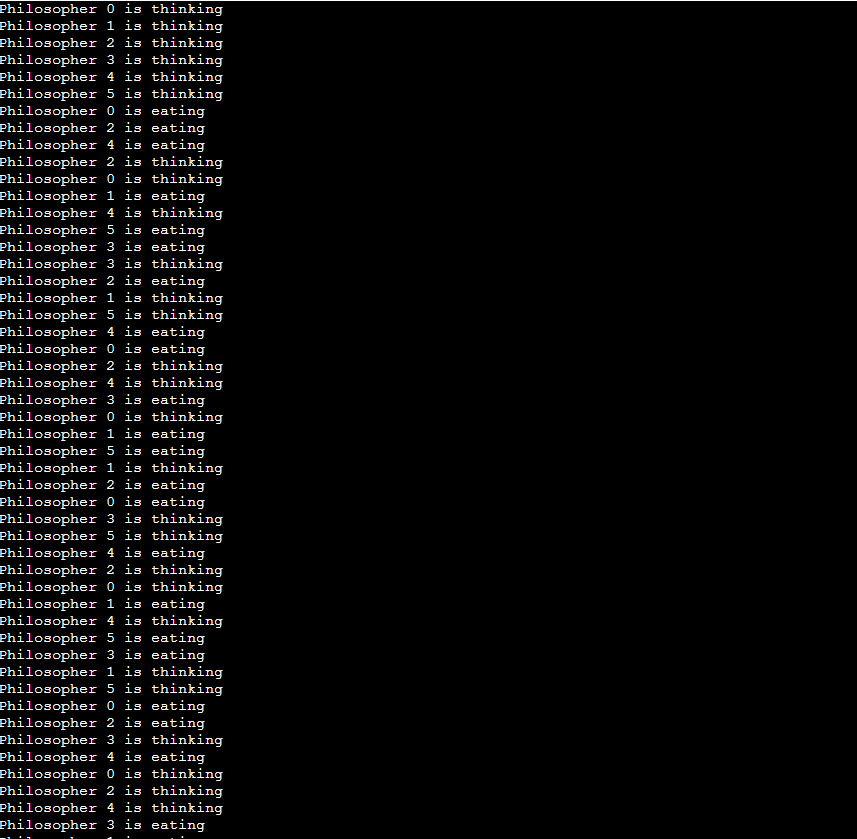
        thread.join();

    }

    return 0;

}

**Output:**



**Conclusion:**

The implementation illustrates process synchronization, emphasizing resource management to prevent deadlocks in concurrent programming through the Dining Philosophers problem.

**Post Lab Descriptive Questions**

1. Differentiate between a monitor, semaphore and a binary semaphore?

 **Monitor**: A higher-level synchronization construct that encapsulates shared variables and synchronization mechanisms, allowing threads to access shared data safely.

 **Semaphore**: A synchronization primitive that uses counters to control access to shared resources, allowing multiple threads to wait and signal.

 **Binary Semaphore**: A type of semaphore that can only take values 0 or 1, functioning like a mutex but without ownership.

1. Identify the scenarios in the dining-philosophers problem that leads to the deadlock situations?

Deadlock can occur in the Dining Philosophers problem when:

* All philosophers pick up their left fork simultaneously, waiting indefinitely for the right fork.

1. Which of the following can be used to avoid deadlock in the Dining Philosophers Problem?
   1. Using a semaphore initialized to the number of philosophers.
   2. Using a semaphore initialized to one less than the number of philosophers.
   3. Using a mutex for each philosopher.
   4. Using a monitor for each fork

**b. Using a semaphore initialized to one less than the number of philosophers.**  
(This prevents all philosophers from holding forks simultaneously, avoiding deadlock.)

1. Which synchronization construct encapsulates shared variables, synchronization primitives, and operations on shared variables?
   1. Semaphore
   2. Binary Semaphore
   3. Monitor
   4. Mutex

**c. Monitor**  
(A monitor encapsulates shared variables, synchronization primitives, and operations on shared data.)

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**