**Batch: C1**

**Roll No.: 16010122221**

**Experiment No. 07**

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

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

**TITLE:** Implementation of Process synchronization algorithms using thread - producer consumer problem , reader-writers problem**.**

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

# Expected Outcome of Experiment:

**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.**

# Achyut S. Godbole , Atul Kahate “Operating Systems”, McGraw Hill Third Edition.

1. **Sumitabha Das “ UNIX Concepts & Applications”, McGraw Hill Second Edition.**

# Pre Lab/ Prior Concepts:

Knowledge of Concurrency, Synchronization, threads.

# Description of the chosen process synchronization algorithm:

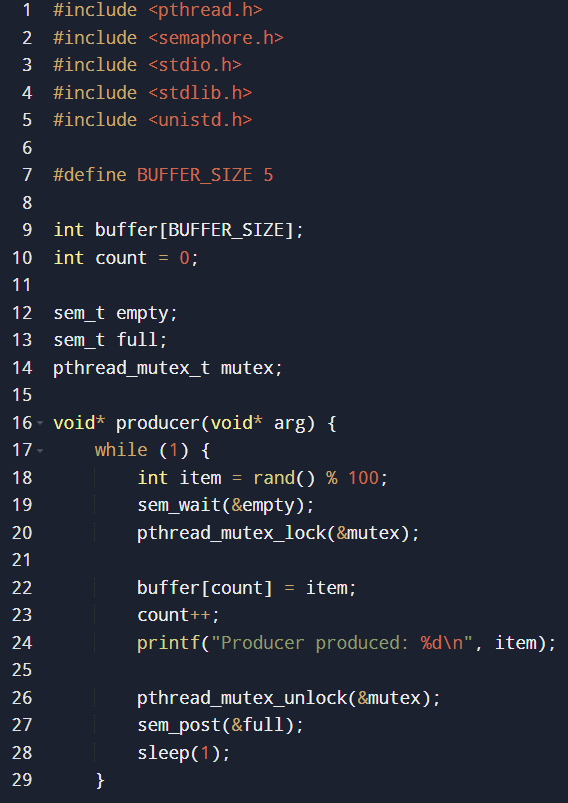
* 1. **Producer-Consumer Problem:** The Producer-Consumer problem is a classic synchronization problem that involves two types of processes: producers and consumers. Producers generate data and place it into a shared buffer, while consumers remove and process the data from the buffer. Synchronization is required to ensure that producers don’t

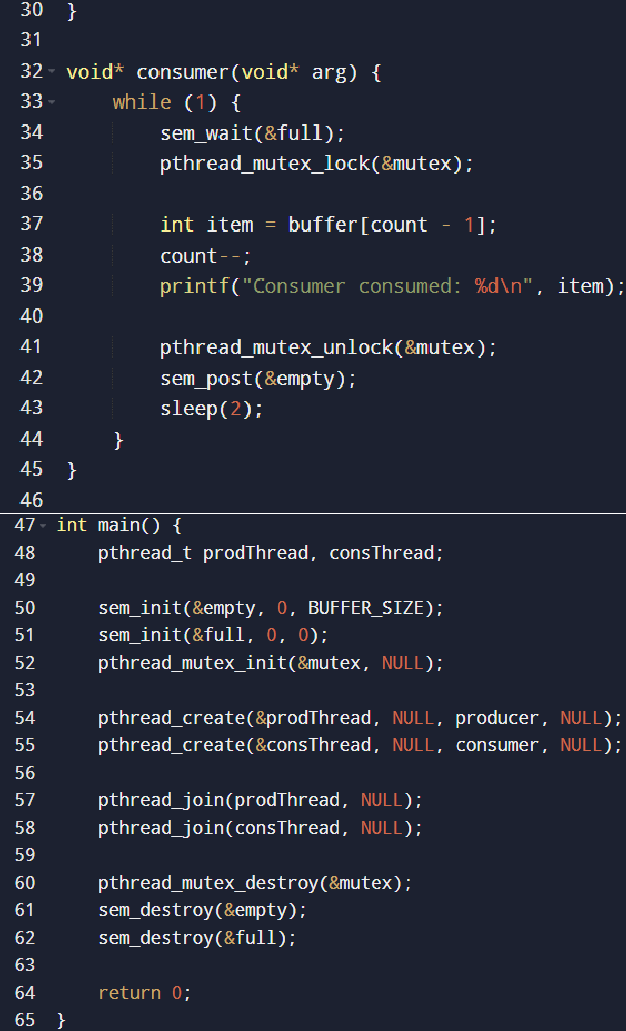
overwrite buffer slots that haven’t been consumed yet, and consumers don’t consume items that haven’t been produced. The problem uses two semaphores: empty (which tracks empty buffer slots) and full (which tracks filled buffer slots). A mutex lock is also used to ensure mutual exclusion when modifying the shared buffer.

* 1. **Reader-Writers Problem:** The Reader-Writers problem involves a shared resource, such as a file or database, that can be read by multiple readers simultaneously but can only be written to by one writer at a time. The goal is to ensure that writers get exclusive access while preventing deadlock or starvation. The implementation uses a semaphore rw\_mutex to ensure exclusive access for writers and a mutex lock to keep track of the number of active readers.

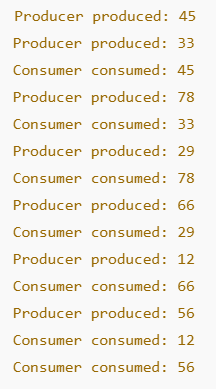
**Algorithm/Pseudocode :**

1. Producer-Consumer Problem: Code:

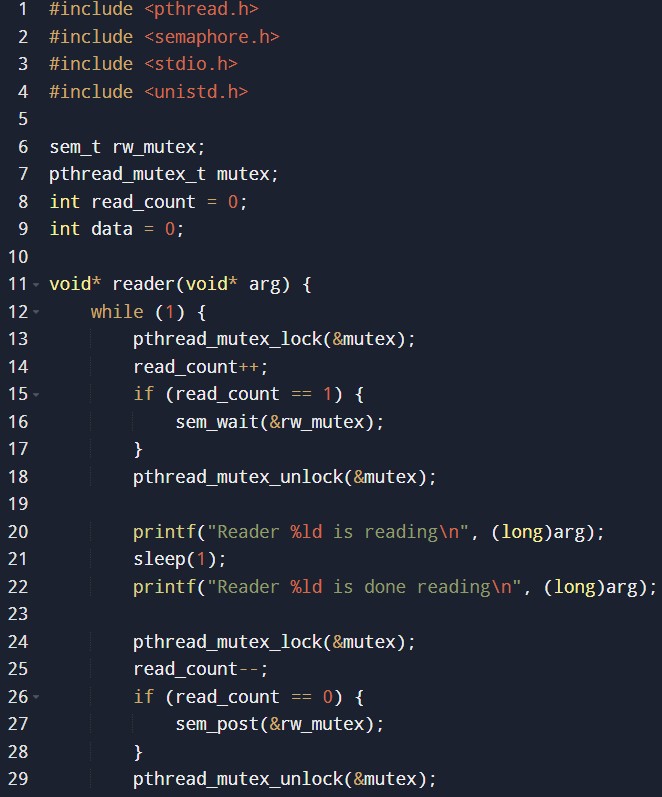


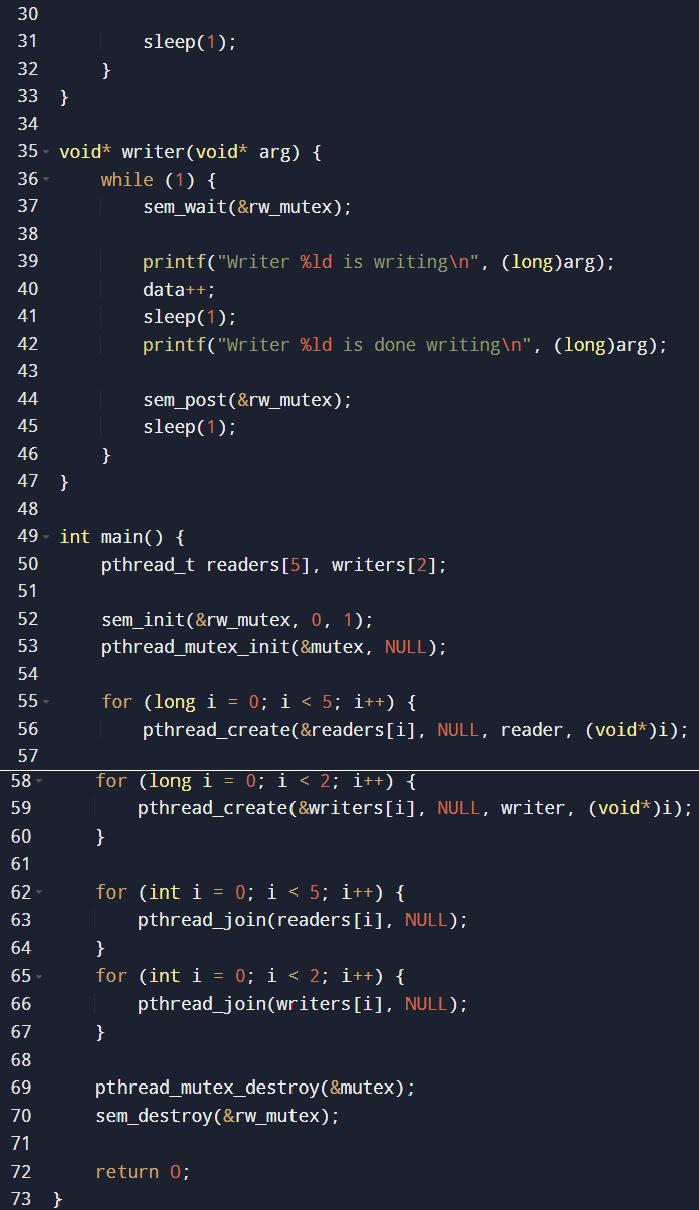


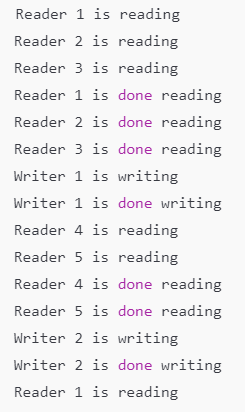
Output:



1. Reader-Writers Problem: Code:





Output:

**Conclusion:**

The **Producer-Consumer** and **Reader-Writers** problems highlight essential process synchronization concepts. Both algorithms prevent race conditions and ensure safe access to shared resources using semaphores and mutexes. The Producer-Consumer problem synchronizes data production and consumption, while the Reader-Writers problem ensures multiple readers can access a shared resource concurrently, but writers get exclusive access. These mechanisms are crucial for maintaining system efficiency and preventing deadlocks in concurrent programming.

**Post Lab Descriptive Questions**

* 1. ***Producer-Consumer Problem:***

# What would happen if the mutex semaphore was not used in the producer-consumer implementation?

If the mutex semaphore was not used, there could be race conditions leading to inconsistent or corrupted data in the buffer, as both producer and consumer could modify the buffer simultaneously.

# How can the buffer size affect the performance of the producer- consumer system?

A larger buffer size reduces the chance of the producer needing to wait for an empty slot and the consumer needing to wait for a filled slot, improving throughput.

# What are the potential issues if the producer and consumer threads are not properly synchronized?

Without proper synchronization, both threads could access the buffer at the same time, leading to race conditions, data inconsistency, or buffer overflows.

* 1. ***Reader-Writers Problem:***

# Explain the importance of the rw\_mutex semaphore in the reader- writers problem.

The rw\_mutex ensures that only one writer accesses the shared resource at a time, preventing race conditions between multiple writers or between readers and writers.

# How does the implementation ensure that writers get exclusive access to the shared resource?

The implementation ensures exclusive access for writers by making writers wait until all readers have finished reading and preventing any new readers from entering while a writer is writing.

# What modifications would you make to prioritize writers over readers?

To prioritize writers over readers, you could modify the algorithm to give preference to writers when both readers and writers are waiting. One approach is to maintain a writer waiting queue or introduce a writer counter.

# Date: Signature of faculty in-charge