CMP-310 Operating Systems Mini Project Fall 2025

Due Date: November 10 at 11:55 PM (via iLearn) **(100 points)**

Mini-Project: Multithreaded Producer-Consumer Application

Project Overview:

In this project, you will develop a **multithreaded Producer-Consumer application** that utilizes a circular **bounded buffer** to manage data exchange between threads. The primary goal is to implement synchronization mechanisms using **semaphores** and **mutexes** to ensure data integrity and prevent race conditions.

Learning Objectives:

1. Understand the concepts of **multithreading** and **synchronization** in operating systems.
2. Apply **semaphores** and **mutexes** to solve concurrency problems.
3. Manage **bounded buffers** and implement inter-thread communication.

Project Requirements:

1. **Bounded Buffer Implementation:**
   - Create a shared buffer with a fixed size (e.g., 10 slots). Must use a circular queue.
   - Use semaphores to manage the number of available slots and filled slots.
   - Use a mutex to ensure mutual exclusion when accessing the buffer.
2. **Producer Threads:**
   - Each producer thread will generate random data (e.g., integers) and add them to the buffer.
   - The number of producers should be configurable (e.g., through command-line arguments).
   - Implement a mechanism to pause if the buffer is full.
3. **Consumer Threads:**
   - Each consumer thread will remove data from the buffer and process it (e.g., print it to the console or write it to a file).
   - The number of consumers should be configurable.
   - Implement a mechanism to pause if the buffer is empty.
4. **Synchronization and Safety:**
   - Use semaphores to track buffer capacity and prevent overflows/underflows.
   - Ensure no **race conditions** occur when multiple threads access the buffer.
   - Avoid **deadlocks** and ensure efficient resource utilization.
   - Do not busy-wait or use sleeps for synchronization. Use semaphores and mutexes to block.
5. **Graceful Termination:**
   - Implement a mechanism for cleanly terminating producer and consumer threads after a specific condition is met (e.g., after producing/consuming a fixed number of items).
   - Ensure all resources (mutexes, semaphores) are appropriately released.
   - To ensure that consumer threads exit correctly after all producers have finished producing, use the Poison Pill technique. After all producer threads complete their

work, the main thread should insert one special value called a poison pill into the buffer for each consumer. When a consumer thread removes this value from the buffer, it knows that no more items are coming and terminates its loop gracefully.

6. **Input Validation and Error Handling:**
    - Validate command-line inputs (e.g., number of producers/consumers, buffer size).
    - Handle errors gracefully (e.g., thread creation failure).

Bonus Features (Optional, up to +10% total):

- **Priority handling (5%)**
    - Add an item priority field (0 = normal, 1 = urgent). Consumers must always take urgent items first while keeping FIFO within each priority. Include a short test where at least 25% of items are urgent.
- **Throughput and latency metrics (5%)**
    - Record enqueue timestamp (when a producer inserts an item) and dequeue timestamp (when a consumer removes it).
    - Report ***average latency*** (mean(dequeue_time − enqueue_time) across all real items) and overall ***throughput*** (items per second) for two runs:
        - small buffer (e.g., 2) with many threads.
        - larger buffer (e.g., 32).
    - Add a small results table in the report with a 2–3 sentence interpretation.

Deliverables:

1. **Source Code:** Well-documented and organized code with explicit comments explaining synchronization logic.
2. **README File:** Instructions on how to compile, run, and test the application. Include examples of input parameters and expected outputs.
3. **Report:** A brief report (1-2 pages) covering:
    - Design decisions (e.g., why specific synchronization mechanisms were chosen).
    - Challenges faced and how they were addressed.
4. **Demonstration Video:** A short video (2–5 minutes) showcasing the application in action and explaining key features. The video must include at least **two group members** presenting or narrating. You should walk through how the application works, highlight your synchronization mechanisms, and briefly explain any major challenges you overcame. The video can be recorded using screen recording tools (e.g., Zoom) or even a phone.

Grading Criteria:

1. **Correctness (50%):** The application meets the functional requirements and handles synchronization correctly.
2. **Code Quality (20%):** Code is clean, modular, and well-commented.
3. **Robustness (15%):** Proper error handling and input validation.
4. **Documentation (15%):** A clear README and report that explains the project.

5. **Bonus Features (10%):** An extra credit of up to +10% for implementing the additional features.

Submission Deadline:

- Submit all deliverables by **November 16** via **iLearn**.

Group Work Guidelines:

- Each group will consist of up to **4 students**. You can self-join your group on iLearn.
- Document individual contributions clearly in the report.
- Ensure equal participation and collaboration.

Additional Notes:

- **Language Requirement:** Students must implement the project in C using the POSIX pthread library. Make sure to include the appropriate headers (e.g., pthread.h, semaphore.h).
- **Termination Condition:**
  - Each producer thread must generate a fixed number of items (e.g., 20), so the total number of items equals the number of producers × 20.
  - After all producers finish and are joined, the main thread enqueues **one** POISON_PILL **per consumer**.
  - Poison pills are not counted as items.
  - The program ends when:
    - real_items_consumed = total_real_items, and
    - each consumer has received a POISON_PILL and exited.
- **Test Case Suggestion:** Include in your README a test case such as:
  ```
  ./producer_consumer 3 2 10
  ```
  This runs the program with 3 producer threads, 2 consumer threads, and a circular buffer size of 10. Clearly specify the number of items each producer generates and provide an example of the expected consumer output.
- **Compilation and Sample Output:**
  Compile your code using:
  ```
  gcc -o producer_consumer producer_consumer.c -pthread
  ```

  Example usage:
  ```
  ./producer_consumer 3 2 10
  ```

  This runs the program with **3 producer threads**, **2 consumer threads**, and a **buffer size of 10**.

  Sample output:
  ```
  [Producer-1] Produced item: 42
  [Producer-2] Produced item: 17
  [Consumer-1] Consumed item: 42
  [Producer-3] Produced item: 88
  ```

```
[Consumer-2] Consumed item: 17
...
[Producer-1] Finished producing 20 items.
[Consumer-1] Finished consuming.
```

You may also choose to display buffer indices if they are helpful. Ensure that your output is **clear and consistent**.

- **Group Formation Deadline:** All groups must be formed on iLearn by **October 16**. After this date, group changes will not be allowed.

Good luck!