Programming Report: Multi-threaded Producer-Consumer Problem

Author: Oscar Lopez
Course: CS433 – Operating Systems
Assignment: 4 – Multi-threaded Programming
Language: C++ with POSIX Threads (Pthreads)
Date: April 10, 2025

Project Overview

This project implements a classic **Producer-Consumer problem** using **multi-threading** and **synchronization mechanisms**. It allows concurrent producer and consumer threads to insert and remove items from a bounded buffer, ensuring **data consistency**, **mutual exclusion**, and **avoidance of race conditions**.

Key Concepts Applied

- **Circular Buffer**
- **POSIX Threads (`pthread`)**
- **Semaphores (`sem_t`)**
- **Mutex Locks**
- **Thread Synchronization**
- **Critical Sections**

Implementation Details

1. Buffer Structure (`buffer.cpp` , `buffer.h`)

- The buffer is a **fixed-size circular array** (`BUFFER_SIZE = 5`).
- `front` and `back` pointers implement the circular nature.
- `count` tracks the number of items in the buffer.

- `insert_item()` and `remove_item()` handle enqueue and dequeue logic.

2. Synchronization (`main.cpp`)

- `mutex` : Ensures **mutual exclusion** when producers/consumers access the buffer.
- `sem_t full` : Counts the number of **filled slots**.
- `sem_t empty` : Counts the number of **empty slots**.

3. Producer Threads

Each producer:

- Sleeps randomly for <1 second to simulate production delay.
- Waits (`sem_wait(&empty)`) for an empty slot.
- Acquires lock (`pthread_mutex_lock`) to enter the critical section.
- Inserts an item and prints the buffer state.
- Releases lock and signals (`sem_post(&full)`) that a new item is available.

4. Consumer Threads

Each consumer:

- Sleeps randomly for <1 second to simulate consumption delay.
- Waits (`sem_wait(&full)`) for a filled slot.