CS433 – Programming Assignment 4 Report

Names: Aidan DePew and Usman Ali
Course: CS433 – Operating Systems
Date: April 18 2025

Problem Description

In this assignment, we implemented a solution to the classic Producer-Consumer problem using multiple threads and a bounded buffer. The main challenge was to properly synchronize access to a shared buffer to ensure that producers don't insert items into a full buffer, consumers don't remove items from an empty buffer, and only one thread accesses the buffer at a time.

Our program creates multiple producer and consumer threads. Each producer inserts its unique ID into the buffer, and each consumer removes items. A random sleep interval simulates production and consumption delays. The output logs each action and the buffer's state after each operation.

Program Design

We designed the program with modularity in mind, separating the logic into three source files. The file, buffer.h, defines the Buffer class and its public methods. The file, buffer.cpp, contains the logic for the bounded buffer using synchronization. The file, main.cpp, parses arguments and creates producer and consumer threads.

We used a circular queue implemented with a vector to maintain first in first out order. Synchronization was implemented using Pthreads, specifically a mutex and condition variables. This approach matched what we studied in class and in the textbook, making it a reliable and familiar choice.

System Implementation

Each producer thread continuously generates its own thread ID as the item, sleeps for a short, random time, and inserts into the buffer. Each consumer thread sleeps for a random short time, then removes and processes an item.

pthread_mutex_t ensures mutual exclusion. pthread_cond_t not_full blocks producers when the buffer is full. pthread_cond_t not_empty blocks consumers when the buffer is empty.

Implementation Challenges:

As a team, we ran into issues making sure the condition variable logic didn't result in lost wake-ups. We also had to debug buffer overflow issues early on due to incorrect

index handling. We overcame this by carefully reviewing the circular buffer logic and rechecking textbook synchronization examples.

Results

We compiled and ran the program with:

g++ main.cpp buffer.cpp -lpthread -o prog4

Then tested it using:

./prog4 10 3 2

This created 3 producers and 2 consumers that ran for 10 seconds.

Sample Output:

Producer 1: Inserted item 1

Buffer: [1]

Producer 2: Inserted item 2

Buffer: [1, 2]

Consumer Removed item 1

Buffer: [2]

The buffer behavior matched the expected FIFO order, and the output was consistent with the provided example.

Features We Implemented:

- A thread-safe, FIFO bounded buffer.
- Multiple producers and consumers working in parallel.
- Sleep delays to simulate real-world timing.

References:

- Textbook (Chapter 7 – Producer-Consumer Problem)
- POSIX threading documentation (pthread_mutex, pthread_cond, usleep)

Conclusion

As a team, we successfully implemented a solution that meets the requirements of the Producer-Consumer problem. The buffer behaved correctly under multiple concurrent threads, with no race conditions or deadlocks.

What the Program Does:

- Provides synchronized insert and remove operations.
- Maintains a fixed-size FIFO buffer.
- Uses condition variables to manage thread blocking.

What We Learned:

- How to apply monitor-style synchronization with condition variables.
- The importance of consistent locking and unlocking of shared resources.