ELEC 377- Operating Systems

Lab 3: Producer-Consumer Problem

Testing Document

Bradley Stephen & Ben Wyand

Overview

This document explains the testing process and results for the producer-consumer program implemented for Lab 3. The primary objective of testing was to verify that producers and consumers synchronized correctly and that data was handled without any loss or duplication, even in concurrent situations.

Input Files

The program was tested using three input files: t10.dat, t11.dat, and t12.dat. Each file contained a sequence of numbers, one per line. For instance, t10.dat included values from **10** to **180** in increments of **10**. Similar sequences were present in the other files, each with different starting values to differentiate the inputs.

Test Scenarios

We tested the program with different combinations of producers and consumers to evaluate its behavior under various conditions and ensure correct synchronization:

1. **1 Producer, 1 Consumer** (./main 1 1 1):
    - **Expected Outcome**: Since there is only one producer and one consumer, the consumer should receive all the values from the producer in the exact order they were produced, without any issues.
    - **Result**: The output file (out10.dat) contained all the numbers from t10.dat in the correct order. This confirmed that the program works properly in a simple scenario where there is no competition for the buffer.
2. **2 Producers, 1 Consumer** (./main 1 2 1):
    - **Expected Outcome**: The consumer should retrieve all the values produced by both producers, potentially in an interleaved fashion. Each value should be present exactly once, with no duplication or missing data.

- **Result**: The output file (out10.dat) contained all the values from t10.dat and t11.dat, with an interleaved order as expected due to concurrent production. There were no missing or duplicated numbers, demonstrating that the synchronization between producers and the consumer was functioning correctly.

3. **3 Producers, 2 Consumers** (./main 1 3 2):
    - **Expected Outcome**: The workload should be distributed between the two consumers, and all values from the three producers should be accounted for across the output files. Each value should appear once and only once.
    - **Result**: The output files (out10.dat and out11.dat) contained all the values from t10.dat, t11.dat, and t12.dat. Each value appeared exactly once, but the order was mixed, which is expected with concurrent consumers accessing the buffer. This showed that both producers and consumers could handle shared access without issues, and the load was balanced effectively.

Example Outputs

out10.dat:

23 46 76 86 166 116 136 63 176 133 113 133 163 183 30 50 70 90 110 130 150 170

out11.dat:

16 26 13 36 56 66 96 126 33 43 146 156 166 53 73 186 83 93 123 143 153 173 10 20 40 60 80 100 120 140 160 180

Analysis below

Analysis of Results

Completeness: All numbers from the input files appeared in the output files, confirming that no data was lost during production or consumption. Each producer successfully added all of its values to the buffer, and the consumers processed them correctly.

No Duplicates: No value was duplicated in the output files. Each value showed up exactly once, indicating that the consumers did not accidentally pull the same data multiple times.

Interleaving and Randomness: The interleaving of values between the output files was expected due to the concurrent nature of the threads. Additionally, the exact order of the output values might show slight randomness because of how the operating system schedules threads. This randomness is normal and acceptable as long as no data is lost or duplicated. The use of mutexes and condition variables ensured proper buffer management, preventing race conditions.

Conclusion

The testing demonstrated that the producer-consumer program works correctly for various combinations of producers and consumers, even with the slight randomness introduced by concurrent thread execution. The synchronization between threads using pthread mutexes and condition variables was effective, ensuring proper management of the shared buffer. No data was lost or duplicated, and the interleaved order of output values was consistent with the expected behavior of a concurrent program. All test scenarios showed successful coordination between producers and consumers, verifying the robustness of the implementation.