ELE340 Coursework
Producer–Consumer Model Report

David Cracknell
Reg. No.: 220139519

2026

1 Model Architecture & Implementation

1.1 Architectural Overview

The code implements a priority-based producer–consumer model using multithreading and a spin lock. It runs from a single runtime structure responsible for storing configuration settings, shared message-queue synchronisation primitives, and logging runtime behaviour. As shown in Figure 1, producer threads generate messages with an attached priority and value before inserting them into the FIFO queue. The consumer then removes the highest-priority messages and processes them, with priority also influenced by the age of each message in the queue.

1.2 Synchronisation & Concurrency Control: Thread Behaviour

Concurrency is enforced using two semaphores and one spin lock. `slots_sem` counts free FIFO slots and blocks producers when the queue is full, while `items_sem` counts queued messages and blocks consumers when the queue is empty. Shared state is guarded by `state_lock` (a spin lock), providing mutual exclusion for a short, high-frequency critical section; this avoids mutex sleep/wake overhead when hold times are small, at the cost of CPU busy-waiting under contention. Message priority increases with time spent in the queue (aging), improving fairness and reducing starvation. During shutdown, any acquired semaphore token is returned to avoid leaking capacity and to prevent threads blocking indefinitely.

1.3 Error Handling and Expected Input/Output

Error handling is performed by wrapper functions that check all pthread/semaphore calls and input arguments, printing a clear error message and terminating immediately if an unrecoverable error occurs. This prevents deadlocks or data corruption. CLI flags configure the run; adding `-d` enables timestamped debug logs for every push/pop, plus summary throughput and blocking statistics for diagnosis. (2 logs required for are in the logs file in the zip folder already submitted.)

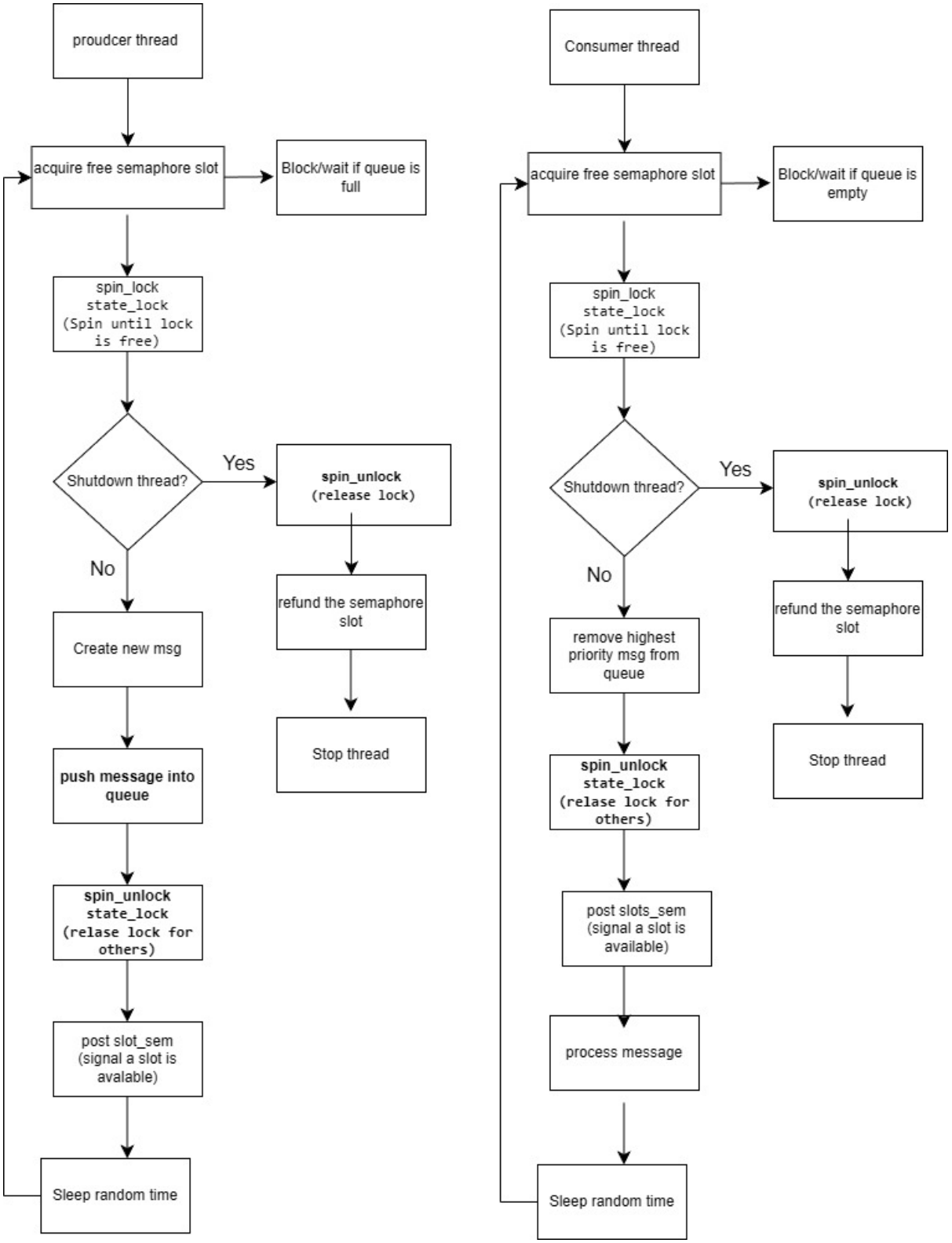

Figure 1: Flow chart of the producer and consumer work flow

2 Model Performance Analysis

2.1 Optimisation of Queue Size and Parameters

Figure 3 shows how the queue size evolves over time for different producer–consumer configurations with $c_{\max} = 4$ s and $p_{\max} = 2$ s. Under these settings, the producers are generally faster than the consumer, so the queue spends much of the time close to its maximum capacity. This also explains Figure 2: producer threads are frequently blocked because they attempt to add items when the buffer is already full, while the consumer is rarely blocked because it almost always has work available.

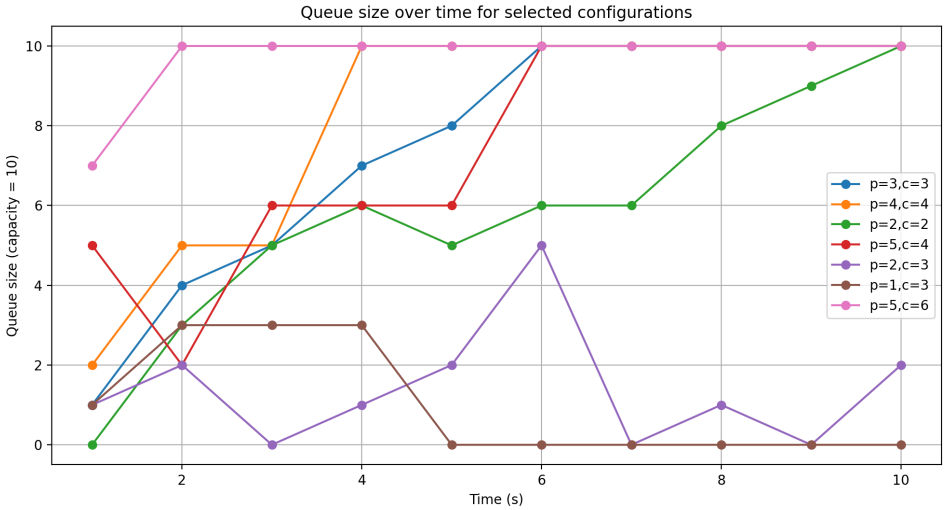

Figure 2: Queue size over time for selected parameter sets (`cmax`=4, `pmax`=2, $Q = 10$).

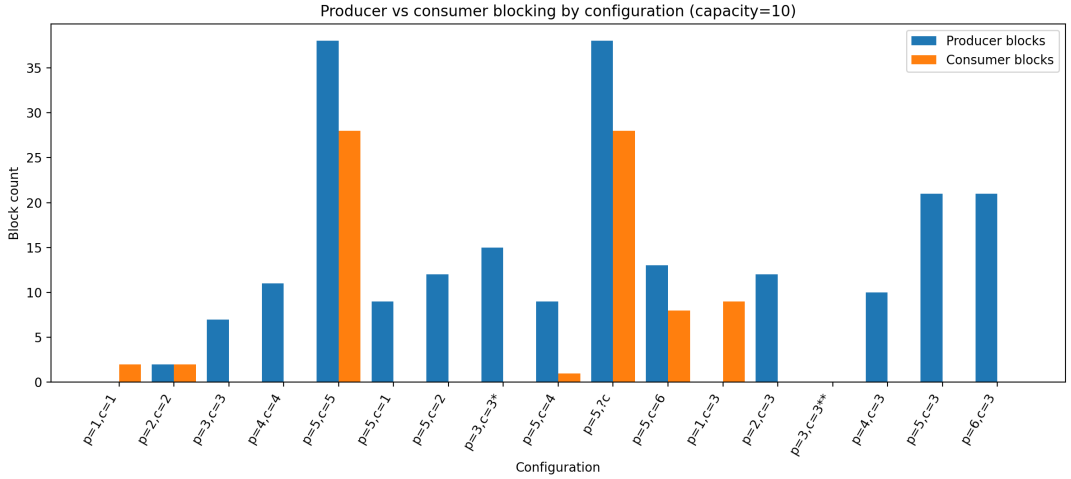

Figure 3: Producer vs Consumer Blocking with Different Parameters for no of consumers and producers. (`cmax`=4, `pmax`=2, $Q = 10$)

2.2 Optimisation of Queue Size and Parameters

Using Figure 3, the FIFO capacity that best balances utilisation and resource cost appears to be $Q = 3$–4. Smaller capacities cause the queue to reach its limit quickly, increasing producer blocking without improving throughput. Larger capacities do not meaningfully reduce blocking or latency; they mainly reserve extra memory for messages that rarely need buffering. In contrast, $Q = 3$–4 keeps the queue highly utilised without wasting memory, which limits the time messages spend waiting in the buffer. This aligns with the parameter sweep in Figure 4: more balanced $c_{\max}$ and $p_{\max}$ values produce smoother queue dynamics and fewer extreme bottlenecks.

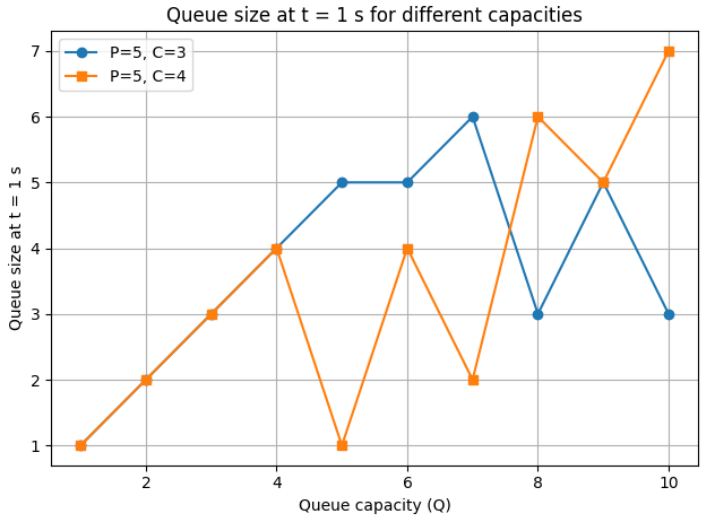

Figure 4: Line graph of the queue at T=1 size vs total capacity

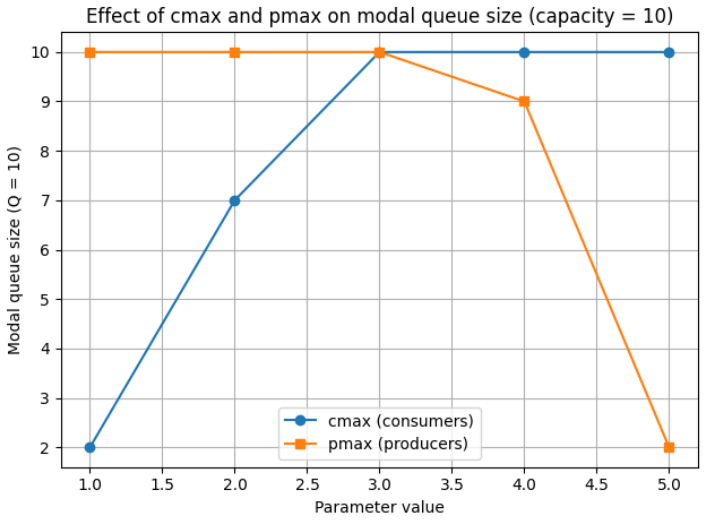

Figure 5: graph of the change in performance when cmax and pmax are varied. (`consumers`=3, `producers`=5, $Q = 10$)