Producer-Consumer System Analysis

Ioannis Michalainas

April 2025

Abstract

This assignment investigates the performance of a multi-threaded producer-consumer system where p producer threads generate computational tasks and q consumer threads process them. The average task waiting time is measured to evaluate the impact of consumer count on performance and to identify optimal thread configurations.

1 Introduction

The implemented system uses a shared FIFO queue to coordinate work between producer and consumer threads. Producers generate computational tasks involving sine evaluations of the form $\sin(0.1k\pi)$ for $k = 1, \ldots, 10$, which are then processed by consumer threads. Key design elements include:

- Bounded queue capacity: 10 tasks
- Thread synchronization via mutexes and condition variables
- Poison pill termination mechanism

2 Methodology

Each producer thread generated 100,000 tasks. Timestamps were recorded at the moment of task enqueueing using `gettimeofday()` and consumers measured the elapsed time upon dequeuing to determine each task's residence time in the queue. For each producer count (1–5), the number of consumer threads was varied from 1 to 6. The results were averaged over five runs per configuration to minimize variability. Visualization was done using Python's `matplotlib`.

3 Results and Discussion

3.1 System Specifications

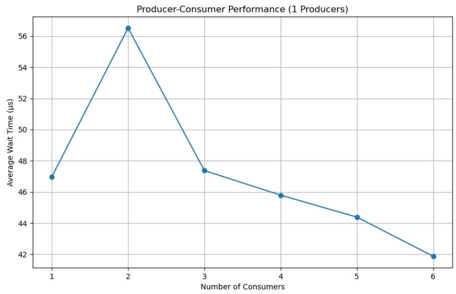
The experiments were conducted on an Arch Linux system with the following hardware and system specifications:

- **Processor:** Intel(R) Pentium(R) Silver N5030 CPU [1.1 GHz - 3.1 GHz]
- **Cores:** 4 physical cores (no hyper-threading)
- **Memory:** 4 GB RAM

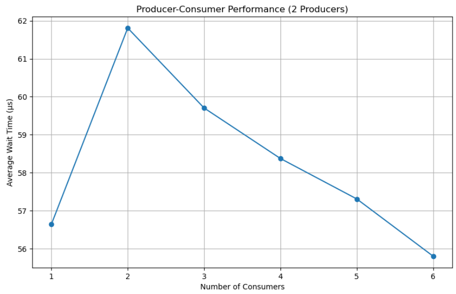
3.2 Waiting Time Trends

Figure 1 shows the average task waiting time versus consumer threads across producer configurations. Key observations include:

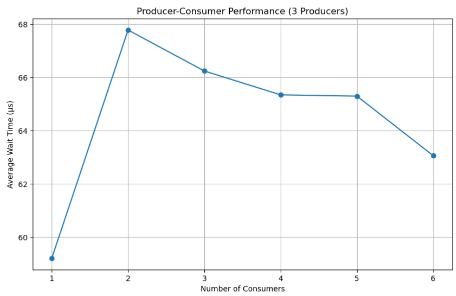
- For 1 producer (Figure 1a), wait time decreased from 46.97 μs (1 consumer) to 41.87 μs (6 consumers).
- For 4 producers (Figure 1d), the optimal configuration was 4 consumers (65.16 μs), aligning with core count.
- For 3 producers (Figure 1c), performance degraded with additional consumers, favoring 1 thread (59.21 μs).



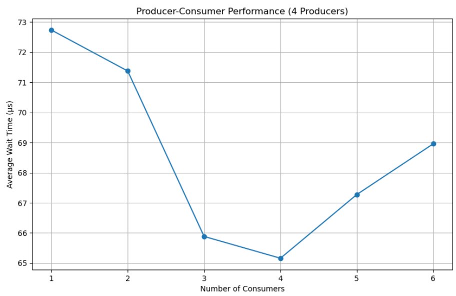
(a) 1 Producer



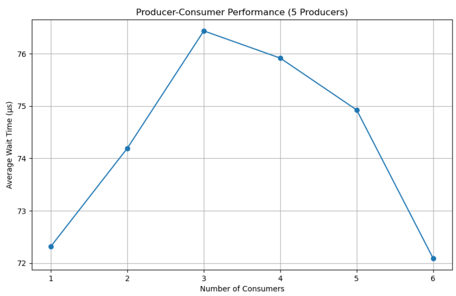
(b) 2 Producers



(c) 3 Producers



(d) 4 Producers



(e) 5 Producers

Figure 1: Average task waiting time versus consumer threads.

3.3 Explanation of Trends

The observed behavior can be attributed to four main factors:

1. **Lightweight Task Complexity:** Each task executed 10 sine computations (1 μs total), which remained insufficient to offset synchronization overheads (1–10 μs). Consumers often idled, allowing more threads to reduce queue residency time.

2. **Queue Contention:** The 10-task queue forced frequent producer blocking. Additional consumers reduced queue dwell time but increased mutex contention for 3+ producers.

3. **Core Utilization:** For 4 producers, 4 consumers maximized parallel execution on the 4-core CPU. Oversubscription (6 consumers) introduced context-switching penalties.

4. **Anomalous 3-Producer Case:** Thread contention outweighed parallelism benefits, favoring fewer consumers. Measurement variability during lengthy runs may also contribute.

3.4 Hardware Limitations

The Intel Pentium N5030 CPU lacks hyper-threading, limiting parallel execution to 4 threads. However, the Linux scheduler efficiently managed oversubscribed threads (e.g., 6 consumers), reducing context-switch penalties for lightweight tasks.