BLG 312E Operating Systems Homework 2 Report

Özkan Gezmiş 150200033

May 10, 2024

1 Introduction

Multi-threading and multi-processing are two common techniques used for concurrent programming in C. In this report, we will introduce and compare two implementations of a program that simulates customer ordering products from a store. One implementation uses multi-threading with pthreads, while the other uses multi-processing with fork.

2 Multi-threaded Implementation

The multi-threaded implementation utilizes pthreads to create threads for each customer. Each thread randomly orders products based on predefined scenarios. The code employs mutex locks to ensure thread safety when accessing shared data structures.

In a multithreaded program, all threads within the same process share the same memory space. This means that all threads have access to the same variables and data structures within the process's memory.

3 Multi-process Implementation

The multi-process implementation utilizes fork to create child processes, each representing a customer. Each child process randomly orders products based on predefined scenarios. The code ensures synchronization among processes using shared memory and mutex locks.

Unlike multithreading, different processes cannot directly access the same data because they have their own data. In the context of the shopping system implementation, shared memory is utilized to enable communication between the parent process and its child processes created using fork. The mmap system call is commonly used to establish shared memory regions between processes. By mapping a shared memory segment into the address space of multiple processes, they can access and modify shared data structures.

4 Comparison

We compare the performance of both implementations based on their execution time, resource utilization, and scalability.

4.1 Scalability with High Customer Count

Multiprocessing: When the number of active customers is high, multiprocessing may incur significant overhead due to the creation and management of multiple independent processes. Each process requires its own memory space and resources, leading to increased system resource usage and potentially diminished performance as the number of processes grows.

Multithreading: In contrast, multithreading can handle a high number of customers more efficiently since threads within the same process share the same memory space. With multithreading, the overhead of creating and managing threads is typically lower compared to processes, making it a more scalable solution for a large number of concurrent operations.

4.2 Performance with High Product Count

Multiprocessing: In scenarios where the number of products is high, multiprocessing may offer better performance compared to multithreading. This is because multiprocessing allows independent processes to execute concurrently, leveraging the capabilities of multiple CPU cores effectively. Each process can handle a subset of products independently, leading to potentially better parallelization and overall faster execution.

Multithreading: With multithreading, the performance benefits may be limited in situations with a high number of products. While threads within the same process can execute concurrently, they may contend for shared resources such as CPU time and memory bandwidth. As a result, the performance gains from parallel execution may be constrained by resource contention and synchronization overhead, particularly when accessing shared data structures or performing computationally intensive tasks.

5 Which One Is Faster

I explained the theoretical part in the former part.

I measured the time taken by the algorithms, excluding the initialization part, and the results are below:

- 10 Customers - 10 Products, Multithreading: 0.008637s Multiprocessing: 0.001048s
- 100 Customers - 10 Products, Multithreading: 0.048268s Multiprocessing: 0.019775s

- 200 Customers - 10 Products, Multithreading: 0.083533s Multiprocessing: 0.033543s
- 100 Customers - 100 Products, Multithreading: 0.099018s Multiprocessing: 0.013012s
- 100 Customers - 1000 Products, Multithreading: 0.639803s Multiprocessing: 1.0992s
- 100 Customers - 10000 Products, Multithreading: 5.195306s Multiprocessing: 1.142164s

In summary, as I explained before multiprocessing is better when the product number is high and multithreading is better when the customer number is high. Since, creating new process for each customer is high overhead.

6 10000 Active Customers Case

Multiprocessing involves running multiple processes concurrently, each with its own memory space. It is typically more suitable for CPU-bound tasks where processes are independent and do not share much data. In contrast, multithreading involves running multiple threads within the same process, sharing the same memory space. It is more suitable for I/O-bound tasks or tasks that involve sharing data between threads.

But since customer number is high, multithreading would be a better choice.