Programming Assignment 3: Organization

COP 4520, Spring 2025, Juan Parra

Due: March 14th before 11:59 PM

Table of Contents

Objective **2**

1 Motivation **3**

2 Requirements **3**

3 Adopting a Growth Mindset **3**

4 Assignment Breakdown **4**

5 Special Restrictions **6**
5.1 Performance Expectations 6

6 General Tips for Working on Programming Assignments **7**

7 Deliverables **7**

8 Grading **8**

Objective

Develop and implement two concurrent systems that demonstrate robust handling of shared data and synchronization across multiple threads.

1. Create a thread-safe linked-list that does the following operations
    - Insertion
    - Deletion
    - Search
2. Design a multi-threaded module

1 Motivation

The challenges presented in these problems highlight the critical importance of designing robust, concurrent systems capable of handling complex, real-world scenarios. The first problem simulates a scenario where multiple agents must concurrently manipulate a shared data structure—an ordered linked-list—to ensure efficiency. This task is emblematic of many real-world applications, from database management systems to network packet processing, where ensuring data consistency and system responsiveness in the face of concurrent modifications is paramount.

Similarly, the second problem emphasizes the need for real-time, concurrent data processing in a resource-constrained environment, such as a Mars Rover. With multiple sensors collecting data simultaneously, the system must reliably record, store, and analyze temperature readings without missing critical intervals. The ability to efficiently compile reports—identifying temperature extremes and fluctuations—mirrors the challenges encountered in real-time monitoring and control systems in aerospace, industrial automation, and environmental monitoring.

Both scenarios motivate the development of effective synchronization techniques and thread-safe data structures. They not only reinforce fundamental concepts in concurrent programming but also push the boundaries of system design to ensure correctness, performance, and progress under high-concurrency conditions. This makes them ideal case studies for exploring innovative solutions that can be applied to a wide range of high-stakes, data-driven applications.

2 Requirements

This assignment is individual. You will also be required to submit a single source file with your solution. It is imperative you keep your solution inside **src** as shown during assignment 0.

You can use a programming language of your choice as well for this assignment (limited to C++, Java, and Rust). If you do not have a preference for a programming language, I would recommend **C++**.

The name of your file will be **main** for C++ and Rust, whereas for java, it will be **Main**. The extension type will be based on what language you code in:

- Rust: extension is **rs** and you will need a **Cargo.toml**
- Java: extension is **java**
- C++: extension is **cpp**

3 Adopting a Growth Mindset

Excerpt from Dr. Szumlanski:

A word of advice before we dive in to the details: When faced with an assignment like this, which has many different facets, some of which might appear foreign and/or challenging, it's important not to look at it as an instrument that is being used to measure how much you already know (whether that's knowledge of programming, operating systems, or even what some might call "natural intellectual capability"). Rather, it's important to view this assignment as a chance to learn something new, grow your skill set, and embrace a new challenge. It's also important to view intellectual capability as something that can grow and change over one's lifetime. Adopting that mindset will allow you to reap greater benefits from this assignment (and from all your college-level coursework) regardless of the grades you earn.

4 Assignment Breakdown

In the source file you submit, **main**, you must implement the following logic.

Input Parsing

- Read a single value from a file
- Value 1: Refers to problem 1

Problem 1 Statement

The Minotaur's birthday party was a success. The Minotaur received a lot of presents from his guests. The next day he decided to sort all of his presents and start writing "Thank you" cards. Every present had a tag with a unique number that was associated with the guest who gave it. Initially all of the presents were thrown into a large bag with no particular order. The Minotaur wanted to take the presents from this unordered bag and create a chain of presents hooked to each other with special links (similar to storing elements in a linked-list). In this chain (linked-list) all of the presents had to be ordered according to their tag numbers in increasing order.

The Minotaur asked 4 of his servants to help him with creating the chain of presents and writing the cards to his guests. Each servant would do one of three actions in no particular order:

1. Take a present from the unordered bag and add it to the chain in the correct location by hooking it to the predecessor's link. The servant also had to make sure that the newly added present is also linked with the next present in the chain.
2. Write a "Thank you" card to a guest and remove the present from the chain. To do so, a servant had to unlink the gift from its predecessor and make sure to connect the predecessor's link with the next gift in the chain.

3. Per the Minotaur's request, check whether a gift with a particular tag was present in the chain or not; without adding or removing a new gift, a servant would scan through the chain and check whether a gift with a particular tag is already added to the ordered chain of gifts or not.

As the Minotaur was impatient to get this task done quickly, he instructed his servants not to wait until all of the presents from the unordered bag are placed in the chain of linked and ordered presents. Instead, every servant was asked to alternate adding gifts to the ordered chain and writing "Thank you" cards. The servants were asked not to stop or even take a break until the task of writing cards to all of the Minotaur's guests was complete.

After spending an entire day on this task the bag of unordered presents and the chain of ordered presents were both finally empty! Unfortunately, the servants realized at the end of the day that they had more presents than "Thank you" notes.

Problem 1 Requirement

Design and implement a concurrent linked-list that can help the Minotaur's 4 servants with this task. In your test, simulate this concurrent "Thank you" card writing scenario by dedicating 1 thread per servant and assuming that the Minotaur received, at most, 500,000 presents from his guests.

Problem 2 Statement

You are tasked with the design of the module responsible for measuring the atmospheric temperature of the next generation Mars Rover, equipped with a multi core CPU and 8 temperature sensors. The sensors are responsible for collecting temperature readings at regular intervals and storing them in shared memory space. The atmospheric temperature module has to compile a report at the end of every hour, comprising the top 5 highest temperatures recorded for that hour, the top 5 lowest temperatures recorded for that hour, and the 10-minute interval of time when the largest temperature difference was observed. The data storage and retrieval of the shared memory region must be carefully handled, as we do not want to delay a sensor and miss the interval of time when it is supposed to conduct temperature reading. Design and implement a solution using 8 threads that will offer a solution for this task. Assume that the temperature readings are taken every 1 minute.

Problem 2 Requirement

Simulate the operation of the temperature reading sensor by generating a random number from -100F to 70F at every reading. In your report, discuss the efficiency, correctness, and progress guarantee of your program.

Example

- Input:

```
250000
```

- Output:

```
Solution 1:
Thread Count = 250000
Duration = 8 ms

Solution 2:
Largest Temperature Delta = 147 Degrees
```

Performance Evaluation

- Measure and report:
  - Total execution time.
  - Number of threads used.
  - Speedup achieved compared to single-threaded execution (optional).

5 Special Restrictions

You must adhere to the following special restrictions when writing this assignment.

5.1 Performance Expectations

The performance thresholds for **each** problem in Rust, C++, and Java are as follows:

- **Rust (Low Overhead, High Optimization)**
  - **Low (Good Performance)**: ≤ 6 seconds
  - **Medium (Acceptable Performance)**: Between 6 and 10 seconds
  - **High (Poor Performance)**: ≥ 10 seconds
- **C++ (Highly Optimized, Compiled Language)**
  - **Low (Good Performance)**: ≤ 6 seconds
  - **Medium (Acceptable Performance)**: Between 6 and 11 seconds
  - **High (Poor Performance)**: ≥ 11 seconds
- **Java (JVM Overhead, Garbage Collection Impact)**

- **Low (Good Performance)**: ≤ 8 seconds
- **Medium (Acceptable Performance)**: Between 8 and 13 seconds
- **High (Poor Performance)**: ≥ 13 seconds

Notes on Comparison

- **Rust**: Known for low-level optimizations and zero-cost abstractions, Rust performs exceptionally well in multi-threaded scenarios due to its safe concurrency model.
- **C++**: With highly optimized libraries like `std::thread` or OpenMP, C++ is competitive for performance-critical applications and performs similarly to Rust.
- **Java**: While Java has slightly more overhead due to the JVM and garbage collection, good practices with tools like `ForkJoinPool` can result in competitive performance.

6 General Tips for Working on Programming Assignments

Here's some general advice that might serve you well in this and future assignments in any course:

1. In general, it is ideal to code in short bursts and review your changes early before waiting too long. Example, you check after the completion of a short function or after a few lines in a complex function. The idea is to make sure you are progressively succeeding rather than progressively failing.

7 Deliverables

Submit your repo's URL via Webcourses. The repo should contain the minimum files for compilation (i.e., the source file).

The source file should contain all the required functions and any helper functions you deem necessary.

Be sure to include your name and UCFID as a comment at the top of your source file. Please ensure you have not modified the protected files (such as **test_cases**, **scripts**, etc) and only modified **classroom.yml** with a one word change describing the language you will be using.

Please make sure to submit your collaboration log as well.

8 Grading

The grading for this assignment is divided into the following categories:

- **Correctness (50%)**
  - Solutions must ensure that data structures remain accurate and consistent, with operations like insertions, deletions, and searches yielding correct results.
  - Implementation should exhibit robust thread safety by using appropriate synchronization techniques.
  - Manage boundary conditions and sustain performance under extreme or unusual scenarios
- **Efficiency (30%)**
  - Maintain continuous operation without undue delays
  - Ensuring that sensor readings or report generation occur on schedule
  - Demonstrated use of multi-threading to improve performance.
- **Documentation (20%)**
  - Clear and concise explanation of the design choices.
  - Description of the correctness and efficiency of the solution.
  - Summary of performance evaluation, including execution time and thread usage.

Your grade will be based largely on your program's ability to compile and produce the exact output expected. Even minor changes (such as capitalization or punctuation errors) in your output will cause your program's output to be marked as incorrect, resulting in point deductions. Please be sure to follow all requirements carefully and test your program thoroughly.

As a general reminder, please do not wait till the last second to read this or work on the code. It's easy for life to get in the way and especially important to get ahead when given the resources. You got this, good luck!