Introduction to Java
CS9053 Section I
Thursday 6:00 PM – 8:30 PM
Prof. Dean Christakos
Assignment 8
March 30th, 2024
Due: April 5, 2024 11:59 PM

Part I: Calculate Pi, multithreadedly

One of the methods to calculate π is the Leibniz formula, which represents π as an infinite series:

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots\right)$$

This series converges very slowly, but it provides an interesting problem for applying multithreading to speed up the computation.

Your task is to implement a Java program that uses multithreading to calculate π using the Leibniz formula. Your program should divide the computation among multiple threads, with each thread calculating a portion of the series. The main goal is to compare the performance and accuracy of the multithreaded approach with varying numbers of threads.

To get this to within 10^{-9} of π, you will have to do 1 billion steps, which takes a lot of time. How much faster can you calculate this with multithreading?

Requirements

1. **Division of Work**: Partition the series among several threads. For instance, if using 4 threads to calculate the first 1,000 terms, each thread should compute 250 terms.

2. **Thread Synchronization**: Ensure that the partial results from each thread are correctly combined to produce the final result. Use appropriate synchronization mechanisms to avoid race conditions when accumulating the results.

3. **Parameterization**: Allow the user to specify the number of terms in the series to calculate and the number of threads to use

4. **Accuracy and Performance Evaluation**: After computing π, your program should print out the calculated value, the actual value of π (from **Math.PI** for comparison), the absolute error, and the execution time. Compare the performance (execution time) and accuracy (difference from **Math.PI**) with different numbers of threads (e.g., 1, 2, 4, 8, 16). Compare the performance improvement with number of threads to the number of processing cores you have (via `Runtime.getRuntime().availableProcessors()`)

- Consider using the **ExecutorService** framework to manage a pool of threads efficiently.
- Make sure to test your program with a high number of terms in the series to better observe the performance impact of multithreading.

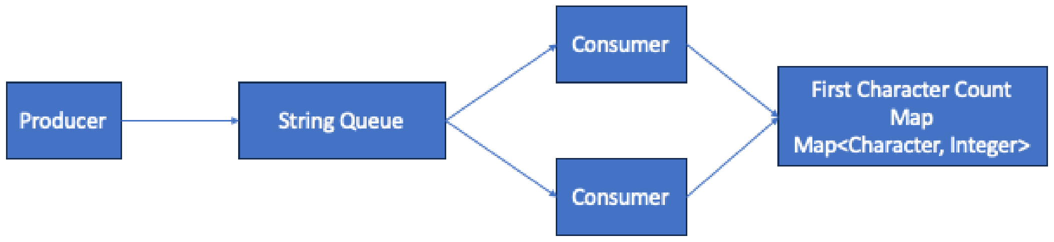
Part II: Understanding Synchronization

Here you are going to create a bunch of threads and make sure everything is properly synchronized.

At the start, you have a `Producer`, puts words on a queue found in `words.10000.txt`. We've already read them in and shuffled the words list. The `Producer` puts words on the queue if the queue has space. Then we have two `Consumer` threads that read in the String objects off of the queue put there by the `Producer` and signals to the `Producer` that there is space available on the queue.

The `Consumer` objects update a Map<Character, Integer> that is counts the starting characters of the words in the word list. So if the words were "amber, ardous, zebra, cat, create", the resulting `letterCount` map would be "a: 2, c: 2, z: 1"

The system looks like this:



This is all put together in the class `MonitorQueues` which connects everything together and starts the `Producer` and `Consumer` threads.

The skeleton code does not in any way handle concurrency at all. You can modify the code and data types to make sure there are no race conditions.

Your tasks:

- Create an accurate count of the first characters for all 10,000 words. There is a single correct answer here. You may want to write some code that figures out what that correct answer is to compare it against the results you get in the multi-threaded system above

- Make sure that all of the characters are put on the queue by the Producer. If the queue is full, wait until it has space. (`wait()` and `notify()` are your friends, here)

- Everything should be properly synchronized. No one should be taking data off of empty queues. If you're doing this right, you will end up with an accurate count

- There are different ways of doing this and getting the right answers. The only things you can't do is change the delay constants or maximum queue size or disable one of the Consumers.