MULTI-THREADED COLLATZ STOPPING TIME GENERATOR

OVERVIEW

This assignment will explore the use of threads in a computational setting and attempt to quantify their usefulness. It will also introduce a common problem encountered in multi-threaded and multi-process environments: a race condition. You will need to solve the concurrency problem as part of this assignment.

THE PROGRAM

When completed, your program (source file named `MTCollatz.java`) should produce a list of Collatz sequences for numbers between 1 and N. The program will have two command-line arguments. The first argument, N, defines the range of numbers for which a Collatz sequence must be computed. The second argument, T, is the number of threads your program must create and launch to compute the Collatz sequences for the given range in parallel. For example, the following execution would use 8 threads to compute Collatz sequences for numbers between 2 and 10,000:

```
java MTCollatz 10000 8
```

Each thread computes a Collatz sequence according to the following formula:

$$f(n) = \begin{cases} \frac{n}{2}, & n\ is\ even \\ 3n+1, & n\ os\ odd \end{cases}$$

$$a_i = \begin{cases} n, & i = 0 \\ f(a_{i-1}), & i > 0 \end{cases}$$

The smallest value of *i* for which $a_i = 1$ is called the stopping time. More details on the problem can be found here: https://en.wikipedia.org/wiki/Collatz_conjecture

The thread counts the stopping time for each Collatz sequence in a global array so that all threads together essentially compute a histogram of Collatz stopping times across a range of numbers. A visualization of a histogram of Collatz stopping times can be found here: https://en.wikipedia.org/wiki/Collatz_conjecture#/media/File:CollatzStatistic100million.png

Stopping times are shown on the X-axis, frequency values on the y-axis. Your program must initialize the array with zeros before it creates the threads and computes the Collatz stopping times for numbers 2 through N. Each thread selects a number in the specified range, computes the stopping time and records the result in the global histogram array before it chooses the next number not yet selected by another thread. This process continues until all threads together have computed stopping times up to the value of *N*. The final histogram should be printed to the standard error (stderr) in the following format:

<k=1,...,1000>, <frequency of Collatz stopping times where *i* = *k*>

Each line contains a value *k* in the range of 1 to 1000 and the corresponding frequency of Collatz stopping times for *i = k*, i.e. Collatz numbers that stop at the value k (a_k=1). The time required to produce the entire histogram must be written to the standard output (stdout). The format of the timing output should be a comma-separated record containing the values *N*, *T*, and the time required to complete the program in seconds and nanoseconds. The following example would be a possible result of the above execution. (The timing values are fabricated and just an example of format).

```
500,8,3.852953000
```

IMPLEMENTATION SUGGESTIONS

In addition to the global histogram array, I suggest keeping a global variable called *COUNTER* that represents the next number for which a Collatz stopping times must be computed. *COUNTER* should start at 2 and end at *N*. As each thread becomes ready for work, it repeatedly increments the value of *COUNTER* and then performs its job of computing a Collatz stopping time and recording the stopping time in the histogram array. Because multiple threads may read the value of COUNTER simultaneously, there is the possibility of a race condition. Therefore, you must introduce thread synchronization using a mutex variable (also known as a lock) to avoid duplication of values in the histogram array. Once threads reach the highest number to test, they terminate. All worker threads must join the main thread before the program computes the elapsed time, prints it, prints out the histogram array and finally terminates.

When using the BASH shell, you have the option of redirecting standard input, standard output, and/or standard error to a file. The `<` and `>` symbols are used to redirect the first two. To redirect `stderr`, you can use `2>`. For example, this command would display `stdout` on the screen but send `stderr` to the file *results.csv*.

```
java MTCollatz 10000 8 2> results.csv
```

You can also append to an output file by using the `>>` and `2>>` symbols.

Experiment with this functionality before trying to save the output of your program executions. It is easy to delete your output files if the redirection command is not entered correctly. Also, experiment with the appropriate system call for time measurement (see below) to compute the elapsed time from the start of the program to the full completion. The elapsed time should approximate the runtime of your program as assessed by considering the system-wide real-time clock.

SUGGESTED LIBRARY FUNCTIONS

Most implementation details are left for you to decide. Below is a list of system calls that you must use in the implementation of your code. You may use other system calls as necessary.

- `java.lang.Thread` – the class to implement Threads
  - `run()` – the run method to run the thread code
  - `join()` – the join method to wait for a child thread
- `java.time.Instant` – the class to measure real-time
- `java.util.concurrent.locks.ReentrantLock` – the class to create and initializes a mutex object
  - `lock()` – the lock method to acquire a lock of a mutex object
  - `unlock()` – the unlock method to release a lock of a mutex object

Hint: Measure elapsed time in milliseconds, not seconds.

EXPERIMENTS

Your program should allow you to perform an experiment involving thread counts from 1 to 30 and a large *N* for the range of Collatz stopping times to be computed. Select a reasonably large value of *N* to force your program in taking measurable time to execute. Create a graph to show the time required for the experiment as the number of threads increases for a fixed *N*. Analyze the results and discuss them in a report. The report should describe the experiment, provide a histogram of the Collatz stopping times, and graph of the performance improvements or degradation that you may observe as you increase the number of threads. As time measurements are influenced by other programs running on your machine, you must run your time measurement tests multiple times for a fixed T and N and taking the average of each run.

DELIVERABLES

Your project submission should follow the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. Follow the submission requirements of your instructor as published in *Canvas* under the first module.
2. You should have at a minimum the following files for this assignment:
    1. `MTCollatz.java`
    2. analysis.doc (the results from an experiment including a table with the data, a graph, & conclusions)
    3. histogram of your Collatz stopping times for a chosen value of N
    4. README (describes any significant problems you had)

Submit a *README* file that describes any significant problems you had. Keep in mind that documentation of source code is an essential part of programming. If you do not include comments in your source code, points will be deducted. If you do not refactor code appropriately, points will be deducted. In addition to the source code and the *README* file, submit a report that includes

- a description of your experiment,
- a visualized histogram of your Collatz stopping time,
- the results from your time experiments, including the graph that visualizes threads vs run-time,
- your analysis on the performance improvements through parallel execution,
- your analysis on the impact of the use of locks on performance,
- your conclusions from the experiment.

TESTING & EVALUATION

Your program will be evaluated on the department's public Linux servers according to the steps shown below. Notice that warnings and errors are not permitted and will make grading quick!

1. Program compilation with JDK.
    - If errors occur during compilation, there will be a substantial deduction. The instructor will not fix your code to get it to compile.
    - If warnings occur during compilation, there will be a deduction. The instructor will test your code.

2. Perform several evaluations runs with input of the grader's own choosing. At a minimum, the test runs address the following questions.

- Are Collatz stopping times computed for numbers in a given range?
- Are multiple threads used to compute the Collatz stopping times?
- Does the use of multiple threads affect the compute time of the program?
- Does the program prevent race conditions

DUE DATE

The project is due as indicated in *Canvas*. Upload your complete solution to the dropbox. I will not accept submissions emailed to me or the grader. Upload ahead of time, as last-minute uploads may fail.

GRADING

This project is worth 100 points total. The rubric used for grading is available on Convas. Keep in mind that there will be deductions if your code does not compile, or poorly documented or organized.