CSC 345 – Project #2: Multi-threaded Programming

Due: Mar 10, 2017, before midnight

Objective:
Practice multi-threaded programming and understand scheduling policy and locking mechanism.

Due Date:
Mar 10, 2017, 11:59pm

Project Details:
In this project, you are asked to write a multithreaded application that does some laborious tasks simultaneously. Your goal is to develop your application so that it can complete these tasks as soon as possible. You can assume that you have hardware capable of doing multiple tasks in parallel. Details are described in attached pages.

Program requirements (100 pts):

- [40 pts] 10 points each for correct implementation of each of the four requested tasks. To demonstrate this, your program should offer a command line option such as
  $ ./main single 1
  that runs the application in single-thread mode, and performs the first task. This way, you can show that your implementation of each task is correct, regardless of multi-thread part.
- [10 pts] Your program should be able to yield the same result in multi-thread mode.
- [10 pts] Support different scheduling algorithm for the worker threads. Pthread library supports two options: FIFO and RR. Provide a command line option to demonstrate this such as: **$ ./main RR** or **$ ./main FIFO**. Your program must provide a measure (e.g., total elapsed time), at the end, to see the difference between scheduling schemes.
- [10 pts] Support priority setting for your program. User should be able to set the priority of one task higher or lower than the other. For example,
  $ ./main priority 1 low
  will set the priority of task 1 lower than the other, thus in the end, user should see that task 1 completed after the other three were finished. [high] option for opposite case.
- [10 pts] Devise a statistical experiment that compares RR and FIFO. For example, run 50 independent runs, average out the total run time, and provide analytical conclusion of your experiments. Graphical plots describing the results would be expected.
- [20 pts] Resolve synchronization issue explained in the attached pages.

Formatting Requirements

- Add comments to your code (using /* */)
- In a separate, single PDF document, clearly describe which requirements you implemented. This should also include the experiment report explained above.

What to turn in

- Zip the folder containing your source file(s) **and Makefile**, then submit it through Canvas by the deadline. There will be a penalty for not providing any working Makefile.

Task Description Details

Your program should be implemented in C (not C++) language, run on Linux machines, and should use pthread (POSIX thread) library.

Given 84 data files in CSV format (comma separated data; each line contains a record in the dataset and a record contains multiple attributes of the record separated by comma), your tasks are like following:

- T1: Find total number of unique words among all non-numeric string literals in each file
- T2: Find maximum, minimum, average, and variance of numbers in each file
- T3: Find maximum, minimum, average of the numbers of rows and columns of all files
- T4: Count ratio of missing or zero values in each file

The workloads are not the same. Each task should be handled by separate thread. In the end, your program should report the result of each task in the following format:

=== T1 completed ===
=== T1 report start ===
T1 RESULT: File ###.csv: Total number of unique words: ####
…
T1 RESULT: Total elapsed time: #### seconds
=== T1 report end ===

=== T2 completed ===
=== T2 report start ===
T2 RESULT: File ###.csv: Max = ###, Min = ###, Avg = ###.##, Var = ###.##
…
T2 RESULT: Total elapsed time: #### seconds
=== T2 report end ===

=== T3 completed ===
=== T3 report start ===
T3 RESULT: Rows: Max = ###, Min = ###, Avg = ###.##
T3 RESULT: Cols: Max = ###, Min = ###, Avg = ###.##
T3 RESULT: Total elapsed time: #### seconds
=== T3 report end ===

=== T4 completed ===
=== T4 report start ===
T4 RESULT: File ###.csv: Ratio = #.##%
…
T4 RESULT: Total elapsed time: #### seconds
=== T4 report end ===

=== All Task Completed ===
=== All Task report start ===
RESULT: Total elapsed time: #### seconds
=== All Task report end ===

where #### is the appropriate values your program calculated / processed. Note that the result should be printed AFTER complete processing all files. Of course, the order of completion, as well as the result print out may be mixed up, particularly for T1, T2 and T4. What we want to do, is to print out the result in a synchronized fashion, thus, other tasks must wait until one task finishes reporting the result. Tasks should not be blocked if they did not finish their job. They are only blocked if they finished their own, but another task is currently printing out the result. The report order must be in the order the tasks were completed.

To demonstrate this, we separate the output part into two: (1) printing out completion of task (message "=== Tx completed ===") and (2) printing out the result of task (messages from "=== Tx report start ===" to "=== Tx report end ==="). (1) should not be blocked by anything, and it is allowed to be printed in the middle of other task reporting messages to make sure (a) tasks that have not finished yet are not blocked by the reporting task and (b) reporting is done in the order of task completion.

We want to report result in two ways. One is to the console window. Another is to a file named **output.txt**. Each way is worth 10 pts toward your synchronization part (20 pts total). The user should be given an option from command line to decide which way to test as:

$ ./main file

where the default behavior without option is the console output. **This synchronization, if implemented, should work regardless of other options described in the requirements.** In other words, user can test synchronization by changing priority values or scheduling scheme.

Acknowledgement on Data

analcatdata A collection of data sets used in the book "Analyzing Categorical Data," by Jeffrey S. Simonoff, Springer-Verlag, New York, 2003. The submission consists of a zip file containing two versions of each of 84 data sets, plus this README file. Each data set is given in comma-delimited ASCII (.csv) form, and Microsoft Excel (.xls) form.

NOTICE: These data sets may be used freely for scientific, educational and/or noncommercial purposes, provided suitable acknowledgment is given (by citing the above-named reference).

Further details concerning the book, including information on statistical software (including sample S-PLUS/R and SAS code), are available at the web site

http://www.stern.nyu.edu/~jsimonof/AnalCatData