Assignment #4
Multithreading
110 Points

Problem

Write a C / C++ program that reads signed integer data from a file, makes decisions upon each integer in the file, and then writes each integer to one or more files. The goal of the assignment is to implement this program using multithreading and mutex locks or semaphores to ensure that the various threads don't interfere with one another.

At minimum, your solution must be divided across two threads that are required to run concurrently:

1. A reader thread that reads data from the file and puts it into a buffer.
2. A processor thread that reads data from the buffer, determines which files to write the data to, then writes the integer to those files.

Your solution should accept the input file as a required command line argument and display an appropriate error message if the argument is not provided or the file does not exist. The command to run your application will look something like this:

Form: assignment_4 <path_to_source_file>
Example: assignment_4 numbers.txt

Output Rules

Your solution must write each integer into one or more output files. The name of each output file and the requirements for each integer written to that file can be found below:

Output File Name	*Rules*	*Example Integers*
even.out	Integers that are classified as even.	…, -4, -2, 0, 2, 4, …
odd.out	Integers that are classified as odd.	…, -3, -1, 1, 3, 5, …
positive.out	Integers that are > 0	1, 2, 3, 4, 5, …
negative.out	Integers that are < 0	…, -5, -4, -3, -2, -1
square.out	Integers that are classified as perfect squares.	0, 1, 4, 9, 16, 25, 46, …
cube.out	Integers that are classified as perfect cubes.	…, -27, -8, -1, 0, 1, 8, 27, …

Input / Output File Specifications

Input File Specification

The input file will adhere to the following specifications:

1. The file will be a plain text file.
2. Each line of the file will contain a single signed 32-bit integer in decimal format.
3. The integer will be represented as a sequence of digits optionally preceded by a negative sign (-).
4. The integer will not be separated by any whitespace or special characters.
5. The file will not contain any headers, footers, or metadata.
6. The file will be encoded using the ASCII standard character encoding.

Output File Specification

Your output files must adhere to the following specifications:

1. The file should be a plain text file.
2. Each line of the file should contain a single signed 32-bit integer in decimal format.
3. The integer should be represented as a sequence of digits optionally preceded by a negative sign (-).
4. The integer should not be separated by any whitespace or special characters.
5. The file should not contain any headers, footers, or metadata.
6. The file should be encoded using the ASCII standard character encoding.

Program Requirements

Your solution should abide by the following rules:

- Your solution will be tested in the HPCC using a bash script to compile your code, run your solution against a test case, and then assign a correctness score to your solution.
- Your solution must be able to compile using GNU v5.4.0.
    - Due to an expected downtime on Quanah, your solution can also be compatible with GNU C/C++ compiler version 10.2.0.
- The grading script will request the following resources for your job, these should be considered your maximum limits when determining how to write your solution:
    - 12-hour maximum runtime
    - 12 cores
    - 64,440 MB of memory
- The final grading will be run against a file containing ~4 billion integers, so your solution should be prepared to withstand a massive influx of data.
- Your output files must preserve the order of the integers as they were in the input file.
    - For example, if the input file contained the integers 1,3,5,9 in that order, then the output file odd.out must contain the 1,3,5,9 in that order.
- Your solution must make use of multithreading and either mutex locks or semaphores.
    - All threads must run concurrently.
    - Serial execution of threads will be treated as failing to meet the requirements of the assignment and likely result in receiving a 0 for correctness.

Helpful Links

- Command Line Arguments in C/C++
- Makefile Tutorial
- HPCC User Guides

Extra Credit

This assignment will grant extra credit for using additional threads to create a more optimized and efficient solution. To be eligible for the base +10%, your solution must create at least three threads instead of the minimum of two and still score 100% correctness as determined by the grader. I would suggest dividing the work as follows:

1. A reader thread that reads data from the file and puts it into a buffer.
2. A processor thread that reads data from the buffer, determines which files to write the data to, and then writes the data to an output buffer.
3. A writer thread that reads data from an output buffer and writes it to the corresponding file(s).

Keep in mind, your solution may use as many threads as you wish. Breaking the work across more threads should help lower your run-time, just be mindful of the resource restrictions listed in "Additional Solution Rules and Hints".

Additional extra credit will be given on a sliding scale based upon wall-clock time compared to other submitted solutions. Once your solution has been executed, the wall-clock time will be printed, and these values will be ranked after all grading is complete. The 30 solutions that achieve the lowest wall-clock times (the fastest executions) while also scoring 100% correctness will then be given extra credit as follows:

Rank	Extra Credit
1 – 3	+30%
4 – 10	+25%

Rank	Extra Credit
11 – 15	+20%
16 – 20	+15%

Rank	Extra Credit
21 – 25	+10%
26 – 30	+5%

What to turn in to BlackBoard

A zip archive (.zip) named <FirstName>_<LastName>_R<#>_Assignment4.zip that contains the following files:

- Your C / C++ source code
  - You can select any names you want, your makefile will dictate the executable name.
- `makefile`
  - The makefile required to compile your application – ensure your final executable is called *assignment_4.*