LogFileProcessingSystem

Qinghao Wu

**Introduction:**

I implemented three solutions for the given problem. The first solution is a single thread solution, and the second and third solutions are multithreading solutions.

**Design:**

Single Thread Solution:

Single thread solution is quite straightforward. The basic idea is read and rewrite the files one by one. I use one global variable to record the current line number and update the current line number after finish writing one file. The single thread solution is easy but slow, because the CPU wastes lots of time waiting for the data to be loaded from disk.

Multithreading Solution 1:

The basic idea of this solution is using multi threads to read and rewrite simultaneously. Each thread will repeatedly read a new file and rewrite the same file. Each thread will need to finish the whole process for one file before getting a new file to process.

Since there are more than one threads, the thread need to know which file to process next and the start line number for that file. I solve this problem by sharing resources. I use thread synchronization to avoid data corruption.

For this solution, different threads can read and write file simultaneously, but the thread need to make sure all the previous files have been read before getting the start line number of the file it is processing. The processing includes getting previous line number and update the line number. And the line number information is added at write step.

Suppose we have five threads, it should work like this:



The only waiting time exists in processing step. Since processing is much faster than read and write, the waiting time should be quite small compared with read and write time. Once the thread finish reading, processing, and writing for one file, it will repeat its previous work until all files have been read.

Multithreading Solution 2:

## I use Producer and Consumer Pattern for this solution. There is one storehouse shared by all threads. Some threads read file repeatedly and put them to the storehouse, and some threads get file repeatedly from storehouse and rewrite it with line number information.

The threads responsible for reading need to make sure all previous files of the file it is currently processing on have been put into the storehouse. When one thread is putting one file to the storehouse, it will also contain the file number and line number information with it. And also update the shared line number and file number information. The thread will check how many files available for writing in storehouse before reading new file, and sleep when there are more than 1000 files available in storehouse. We can make sure our program will not run out of memory using this check. The threads for writing can get file from storehouse in order, and rewrite them directly.

Suppose we have five threads for reading and five threads for writing, it should work like this:



The only waiting time exists in processing step. The thread for writing can get a file and write if there are file available in storehouse. Since processing is much faster than reading and writing, the waiting time should be quite small compared with read and write time. The thread for reading and processing will repeat its previous work when it finishes reading, processing a file, and put it into storehouse. The threads for writing will repeatedly get a file from storehouse and rewrite it.

All solution will check whether the file name follows the given format using regular expression, and ignore the files that not follow the given format.

**Implementation:**

I implemented all my solutions in a java project, and exported runnable jar files for my solutions. I added some tests using JUnit, and run tests automatically after building using a continuous testing plugin for Eclipse called Infinitest. You can find all the source code at src folder. You can find all project resources and some larger dataset at following GitHub repository:

<https://github.com/wuqinghao0102/LogFileProcessingSystem>

The most recent branch is Version2.0, and you can check all other branches to find my development process.

**Instructions:**

All the three runnable jar files and some sample datasets are in the demo folder. You can test the jar file using provided dataset or create your own dataset. Here are some commands and explanations for running the jar files.

**LogFileProcessingSystem\_s.jar**

This jar file is the single thread version of my solutions. You can use the following command to run this jar file:

**java –jar LogFileProcessingSystem\_s.jar YourFileFolder**

"YourFileFolder" is the your corresponding directory that contains your log files.

**LogFileProcessingSystem\_m.jar, LogFileProcessingSystem\_m2.jar**

These two jar files are the multithreading versions of my solutions. You can use the following command to run this jar file:

**java -jar LogFileProcessingSystem\_m.jar YourFileFolder 8**

**java -jar LogFileProcessingSystem\_m2.jar YourFileFolder 8 8**

"YourFileFolder" is the your corresponding directory that contains your log files.

For first one, 8 are the number of threads you want to use; for second one, the first 8 means the number of threads for reading, the second 8 means the number of threads for writing. And it means you have 16 threads in total. You can set them to different numbers, but both of them should be greater than 0.

ATTENTION:

1. The commuter you use to run the jar file should have installed Java Runtime Environment.

2. The name of your log files should follows the format “logest.2014-07-11.log”, the files not follow the format will be ignored.