May 11th, 2022

CSE344 – System Programming - Homework #5 - v1
Threads synchronization

Objective

This is your final homework for this semester. The idea is to use POSIX threads to parallelize a couple of simple mathematical tasks. Let’s see how much we can accelerate via threads.

```
Example:./hw5 -i filePath1 -j filePath2 -o output -n 4 -m 2
```

Input

The two input files, provided as either relative or absolute paths, will be regular ASCII files of arbitrary content and length.

The process

The process will start by reading the two files into memory. It will read sequentially (2^n)x(2^n) characters (with n provided as a command-line argument, n > 2) from each file. Each character will be converted to its corresponding integer ASCII code equivalent. Every consecutive 2^n characters that it reads will be considered as one matrix row. Therefore, this will lead to reading 2 square matrices A and B of size (2^n)x(2^n) each. If any of the two files has insufficient content, this will be considered a fatal error.

The process will then create m (provided as a command-line argument, m >= 2k, k>=1) POSIX threads. Each thread will have 2 sequential tasks to complete.

a) First, they will calculate C=AxB in a parallel fashion. This means that each thread will be responsible for calculating (2^n)/m columns of C. For example if m=2, then the first thread will calculate the left half of C, and the second thread will calculate the right half of C. This could make the calculation of C twice as fast (with respect to using a single thread)...at least in theory.

b) Once C has been calculated, then the threads will switch to the second task and calculate the 2D Discrete Fourier Transform of C, the same way as before (i.e. (2^n)/m columns calculated by each thread).

Once they have all finished, the process will collect the outputs of each thread and write them to the output file (relative or absolute path), in CSV format (one matrix row per file line). You will report the total time spent, from the moment the files were read into memory, until the calculations were completed. **Try your program for various values of n and m, and report your findings in terms of acceleration, if any. Take into account the number of cores on your CPU.**

Careful

The threads must wait for each other and not advance to the second part of the calculations before ALL have finished the first part. This is a synchronization ******* (starts with b and ends with arrier). Why? Because in order to calculate each DFT coefficient, you need to access ALL the matrix elements.

Output examples (the numbers are made up – **include a timestamp before each output line**):

```
Two matrices of size 1024x1024 have been read. The number of threads is 4
Thread 3 has reached the rendezvous point in 0.0123 seconds.
Thread 4 has reached the rendezvous point in 0.0121 seconds.
Thread 2 has reached the rendezvous point in 0.0119 seconds.
Thread 1 has reached the rendezvous point in 0.0124 seconds.
Thread 1 is advancing to the second part
Thread 4 is advancing to the second part
Thread 3 is advancing to the second part
Thread 2 is advancing to the second part
Thread 1 has has finished the second part in 0.1321 seconds.
Thread 3 has has finished the second part in 0.1212 seconds.
Thread 2 has has finished the second part in 0.1175 seconds.
Thread 4 has has finished the second part in 0.1327 seconds.
The process has written the output file. The total time spent is 0.173 seconds.
```

Restrictions and requirements
- All threads must run concurrently.
- Solve all synchronization problems only with mutexes and condition variables.
- In case of SIGINT, the process and the threads will terminate gracefully, closing all open files and freeing all allocated resources.
- Calculations will be correct down to 3 decimals.

Evaluation
Your submission will be evaluated with various input files n, m values.

Grading
1) Compilation error: grade set to 1; if the error is resolved during the demo, then evaluation continues.
2) Compilation warning (with respect to the -Wall flag); -1 for every warning until 10. -20 points if there are more than 10 warnings; no chance of correction at demo.
3) No makefile, makefile without -Wall, makefile without “make clean”: -30
4) No pdf report submitted (or submitted but insufficient, e.g. 3 lines of text with no design explanation, etc), file formats other than pdf: -20
5) The program crashes/locks or produces wrong output with valid input: -100
6) Poor synchronization (e.g., sleeps, busy waiting, timed waits, trylocks, etc): -100
7) The program doesn’t satisfy a requirement or violates a restriction: -100
8) Presence of memory leak (regardless of amount – checked with valgrind) -30
9) No late submissions.
10) In case of an arbitrary and fatal error, exit by printing to STDERR a nicely formatted informative message. Otherwise: -10
11) If you don’t cleanup after children processes and/or leave zombies: -50

Is my homework submission valid?
It is valid if given correct input files, your program makes the multi-threaded calculations, even if the results are wrong.

Submission rules:
- Your source files, your makefile and a report; place them all in a directory with your student number as its name, and zip the directory.
- Your report must contain: how you solved this problem, your design decisions, which requirements you achieved and which you have failed.
- The report must be in English.
- Your makefile must only compile the program, not run it!
- Do not submit any binary executable files. The TAs will compile them on their own boxes.
- Your code will be compared against online sources; you are free to be inspired, but you are also expected to write your own code.
- Homeworks are individual tasks. Proven cases of plagiarism will be punished to the full extent.

Hints:
- Plan/design carefully.

- Check for memory leaks with valgrind before submitting.
- Compile and run your program on more than one POSIX systems to ensure portability, if you can.
- Control whether you satisfy each and every one of the requirements prior to submission.
- Test your programs with arbitrary inputs and put it through a stress-test.

Good luck.