Advanced Unix
METU Computer Engineering

Programming Assignment # 2

Instructor: Yusuf Sahillioğlu

Deadline: 14.05.2023 23:59

(20% of the actual grade)

Your code will be tested by Moss against cheating attempts, any cases suspected of plagiarism will result in total loss of grade and might result in further disciplinary actions.

Please submit your C code along with your execution screenshots to ys@ceng.metu.edu.tr

---

You'll work on thread synchronization in Unix using POSIX thread (pthread) library.

Part 1 [50 points]: Synchronize $d \leq m$ threads so they can shift an m x m matrix concurrently. You will repeat the following shifts s many times: first perform a circular shift from left to right, then a circular shift from bottom to top. Here is an example with $m = 4$ and $s = 1$:

7	6	5	3
8	2	4	5
7	3	1	9
2	1	0	8

3	7	6	5
5	8	2	4
9	7	3	1
8	2	1	0

5	8	2	4
9	7	3	1
8	2	1	0
3	7	6	5

Each thread is responsible for m/d consecutive rows (last thread may take more if $d \nmid m$) during row shifting, and then m/d consecutive columns for the column shifting, e.g., with $m = 4$ and $d = 2$, thread 0 shifts rows 0-1, and thread 1 shifts rows 2-3. No thread can start shifting columns until all threads have finished shifting rows. Also, if $s > 1$, no thread can start shifting rows for the current stage until all threads have finished shifting columns in the previous stage.

You will read d and s from keyboard, and the input matrix from input.txt file which begins with m and then one row per line. Main thread prints the input matrix as well as the shifted matrix. It also prints execution time obtained with different d values. Use d=1 (no multithreading), 2, ..

Part 2 [15 points]: Synchronize m threads so they can add 2 m x n matrices concurrently. Each thread is responsible from 1 row. Print the new resulting matrix along with timing: multithread vs. single thread.

Part 3 [35 points]: Multiply m x n and n x k matrices concurrently. As you know one row multiplies one column and accumulates the elementwise multiplication results in a variable. Make that variable global so that x threads update it concurrently and in a mutually exclusive manner (race conditions). You then write the final value of that variable to the corresponding entry in the resulting m x k matrix. Similar to Part 2, let each thread be responsible from 1 row of the m x n matrix (these are different than the x threads above). Print the resulting matrix along with timing: multithread with different x values vs. single thread.