Parallel Applications

Laboratory 4
Deadline: 2 weeks

This assignment will help us learn and evaluate
(i) two ways of writing parallel applications – using threads and using processes,
(ii) two ways of doing inter-process communication – using shared memory and using pipes,
(iii) two ways of doing process synchronization – using atomic operations and using semaphores.

Part I: Baseline application without any parallelism

This part is easy. Use your submission from laboratory 1. Update the makefile as indicated in the instructions below.

Part II

Now suppose you have a processor with three cores. You want to use all of them to make your application finish faster. You can do this by having the first core do *S1*. As pixels get ready, they are passed to the second core (don't wait for the *S1* of the whole image to be completed before communicating to *S2*). The second core does *S2* and passes its results to the third core. The third core does *S3* and the file writing. Do this in the following ways:

1. *S1*, *S2*, and *S3* are performed by 3 different processes that communicate via pipes (or fifos) (further reading on pipes)
2. *S1*, *S2*, and *S3* are performed by 3 different processes that communicate via shared memory. Synchronization is done using atomic operations.
3. *S1*, *S2*, and *S3* are performed by 3 different threads of the same process. They communicate through the process' address space itself. Synchronization is done using semaphores.

- Devise a method to prove in each parallel case that the pixels were received as sent, in the sent order. Describe the method in your report.
- Study the run-time and speed-up of each of the approaches and discuss.
    - It is likely that the file reading and writing times dominate, and so the speed-up obtained by using three cores is negligible. So we will modify our experiment to make it compute-intensive, instead of IO-intensive. Read the image only once, but perform the transformation 1000 times. That is, *S1*, *S2*, and *S3* are done 1000 times each. The results of the first run of *S1* are used by the first run of *S2*, and the results of the first run of *S2* are used by the first run of *S3*. For the

second run of S1, it uses the same input image that has already been read into memory. The results of the second run of *S1* are used by the second run of *S2*, and so on. The results of the 1000th run of *S3* are written to the file.

- Discuss the relative ease/ difficulty of implementing/ debugging each approach.

Submit a single zip file with the source code (organized into multiple folders, one for each question), a makefile, an input *ppm* image, and a report.

- *make part1* should compile the Part I version of the code and run it, creating the file *output_part1.ppm*
- *make part2_1* should compile the multi-process, pipe version of the code and run it, creating the file *output_part2_1.ppm*
- *make part2_2* should compile the multi-process, atomic operation version of the code and run it, creating the file *output_part2_2.ppm*
- *make part2_3* should compile the multi-thread, semaphore version of the code and run it, creating the file *output_part2_3.ppm*