CS342 Operating Systems - Fall 2019
Project #2
Threads, Processes, and Synchronization

Assigned: Nov 5, 2019, Tuesday.
Due date: Nov 23, 2019, Saturday, 23:55.

- *Submit through Moodle. Make sure you start submitting one day before the deadline. You can overwrite your submission as many times as you wish. Late submissions will not be accepted (no excuse; no email will be accepted).*
- *You can only learn if you think and do yourself.*

1 Assignment

1.1 Part A - Synchronizing Threads

You will now develop a program that will have N *worker threads* created by the *main thread*. The program will have a *shared* binary search tree (pointed by a *global* variable) with size at most K nodes. The tree is shared among all threads. The tree will be implemented with pointers, not using an array. The tree will keep the K largest integers seen so far in the input files. Each worker thread will process an input file of integers (one integer per line). There will be N input files. Each such worker thread will read an input file one integer at a time. This time we will read with fscanf (easier). After reading an integer from the input file, the worker thread will insert it into the binary search tree if necessary, otherwise it will discared the integer. Then it will read the next integer from the file. All N worker threads will work like this concurrently. When all worker threads finish, we will have the K largest integers seen in those N files in the binary tree. Then the main thread will print them out in sorted decreasing order to an output file file. You need to implement the tree youself. You can not use a library or an existing code for this purpose.

We will invoke the program as below:

topk_thread_synchron <K> <N> <infile1> ...<infileN> <outfile>

You will use POSIX mutex and conditions variables (if required) to synchronize the threads.

1.2 Part B - Synchronizing Processes

Now you will write a similar program, but this time worker child processes will be used to process the input files. You will also use shared memory to pass information from children to the parent process. That means you will implement data structure to sit in the shared memory. Shared memory should be large enough. In this part, the structure can be a binary search tree or a heap, or a simple array. It is up to you. But no matter what the structure is, *it must hold the K largest integers seen so far by the child processes in the input files*. That means the content of the shared memory structure should change dynamically while the input files are being processed by the children. You can not use a structure outside shared memory to hold the integers (like an array in a child process).

We will invoke the program as below:

topk_process_synchron <K> <N> <infile1> ...<infileN> <outfile>

You will use POSIX semaphores to synchronize the processes.

1.3 Part C - Experimentation

For both programs, first of all try see what is happening if we don't synchronize the threads or processes. Do we get wrong results some time?

Measure and compare the running of two programs for various values of the following parameters: N, K, integer count in an input file, and try draw conclusions. Write your experimentation experience and results into a report document and convert the document to PDF (report.pdf). The report should include tables or plots and conclusion(s) as well.

2 Submission

Put all your files into a directory named with your Student Id. In a README.txt file write your name, ID, etc. The set of files in the directory will include README.txt, Makefile, report.pdf, topk_thread_synchron.c and topk_process_synchron.c. When we type make, all your programs should be compiled and the related executables should be generated. Then tar and gzip the directory. For example a student with ID 21404312 will create a directory named "21404312" and will put the files there. Then he will tar the directory (package the directory) as follows:

```
tar cvf 21404312.tar 21404312
```

Then he will gzip the tar file as follows:

```
gzip 21404312.tar}
```

In this way he will obtain a file called 21404312.tar.gz. Then he will upload this file in Moodle.

Late submission will not be accepted (no exception). A late submission will get 0 automatically (you will not be able to argue it). Make sure you make a submission one day before the deadline. You can then overwrite it.

3 Tips and Clarifications

- Work incrementally. Be persistent.
- The "man shm_overview" command will give you help information about POSIX shared memory in Linux shell.
- The "man sem_overview" command will give you help information about POSIX semaphores.
- The "man pthreads" command will give you information about POSIX threads and also mutex and condition variables.
- Max value of K is 10000. Min value is 100.
- Max value of N is 10. Min value is 1.
- An input file can contain a very large number of integers.