Operating Systems Laboratory (CS39002)

Spring Semester 2021-2022

Assignment 4: Usage of threading library and locks

Assignment given on: February 28, 2022

Assignment deadline: March 8, 2022, 1:00 PM

In this assignment you will learn hands-on how to use pthreads and locks. The details of the assignment are as follows.

a) Initialize a master process. This master process (let's call it A) will create a job dependency tree. In the tree a child job must finish before a parent job can be started. A job has a job id (which is just a random number between 1 and 10^8) and a time of completion. This tree is base tree T. This tree resides in a shared memory segment.

b) Now, process A will spawn P producer threads (P is an input). Each producer thread runs for a random time between 10 to 20 seconds. The job of producer threads is to add new dependency jobs to randomly chosen jobs that have not been scheduled yet (so essentially add new children to a job node). Between each addition the threads will sleep a random time between 0 to 500 milliseconds.

c) The following is the minimum structure of a node in the tree describing a job. Process A must create an array of these nodes in a *shared memory* with room to spare so that producers can add more jobs. Your tree T should have a random number of jobs between 300 to 500 nodes, and each job time has a maximum completion time of 250ms.

```
struct Node {
        job id
        time for completion
        dependent jobs []
        mutex lock
        status (completed, on going, done)
        :
        :
        <other data>
}
```

d) The process A will also fork a consumer process named B. Process B spawns *y* consumer threads (y is an input parameter). Each consumer thread will traverse the tree and find the first job that has no dependency and run it. While *running the job*, the process simply sleeps for the specified amount of time (as defined in the job) and then the job is completed.

e) Every time any event occurs, the details of the event should be outputted to the console with details like job id for more information where applicable (events can be addition of new jobs, start of new jobs, completion of jobs). The producers will stop after their time of completion. The consumers will stop after all jobs are executed.

f) Use mutex locks (the ones in the node, as well as other locks if and when needed) to prevent race conditions between different producers and consumers. Make sure you use your mutex locks as efficiently as possible (too many locks and it will be just a sequential program, too few and you will have race coditions) and not hold it more than necessary by a single thread. Other data structures can be used locally as well to simplify your implementation.

Hint: *Use the POSIX Pthread library for creating/managing the threads.*

Submission Guideline:

- Create the program as a single file as **`Assignment_4_<groupno>.c`** or **`.cpp`**, and upload it.