CS342 Operating Systems - Fall 2020
Project #2 – Synchronization / Scheduling

Assigned: Oct 25, 2020, Sunday.
Due date: Nov 18, 2020, Wednesday, 23:59. Document version: 1.3

- *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).*
- *The project can be done in groups of 2 or individually.*

Assignment

In this project you will develop a multi-threaded scheduling simulator. There will be N threads simulating N processes. Each such thread is called a PS thread (process simulator thread). During its lifetime, a process goes through cpu-burst/io-burst cycle many times. A PS thread will simulate this behavior. There will be also a scheduling and executor thread, called SE thread, that will simulate the scheduling and execution of the cpu bursts of these processes. Hence your program will run with N+1 threads. If you wish, you can have the main thread of your program to take the role of the SE thread.

A PS thread will generate a random cpu burst length (in the order of 100ms) and will put that information into the ready queue. With this we are simulating arrival (generation) of a cpu burst and putting that into the ready queue (cpu burst is added to the tail of the ready queue). Ready queue is protected by a lock. Then the PS thread will wait on a condition variable until the burst is executed. When burst is executed, the PS thread will be waken up from waiting on the condition variable. Then it will generate a random I/O wait time (i.e., an I/O burst) and will sleep that much time (you can use usleep for that purpose). Then it will generate another cpu burst and submit that burst to the ready queue again. It will continue behaving like this.

The SE thread will select one cpu burst from the ready queue. It will look to the length L of the cpu burst and will simulate the execution of the burst by sleeping L time units (you can use usleep function). Then it will inform the related thread about the fact that burst execution has finished. It will do this by signaling on the respective condition variable. Then the SE thread will select another cpu burst and simulate the execution of that cpu burst (by sleeping again). It will continue acting like this.

In SE thread, which burst will be selected to run next depends on the scheduling algorithm simulated. The SE thread should be able to simulate the following scheduling algorithms: FCFS, SJF (non-preemptive), and RR(q). For RR, q should be in order of 100 ms.

The program will have the following parameters:

schedule N minCPU maxCPU minIO maxIO outfile duration algorithm quantum infileprefix

- N is the number of PS threads.
- minCPU is the minimum CPU burst length and maxCPU is the maximum CPU burst length, that a process can have. Similarly, minIO is the minimum I/O wait time, and maxIO is the maximum I/O wait time, that a process can have.
- If infileprefix is equal to "no-infile" string, cpu burst times are randomly generated (uniformly distributed between minCPU and maxCPU) and used inside your program. Similarly, io-burst (i.e., io-wait) times are randomly generated (uniformly distributed between minIO and maxIO).
- If infileprefix is something different than "no-infile", cpu and io burst lengths will be obtained from an input file (the name of the input file will have the specified prefix) for each PS thread. The input file for PS thread X (1<=X<=N) will have a name with the format "infileprefixX.txt". For example: if infileprex is "infile", then the input filename for simulated process 4 will be "infile4.txt".
- Duration determines how long the simulation will last. If set to a value M, then each PS thread will generate M cpu bursts. For example, M can be "100".
- <alg> is the scheduling algoritm to simulate. It can be "FCFS", "SJF", or "RR". If FCS or SJF is specified, then q will be set to 0; if RR is specified, then q is a value greater than 100 and less than 300. The q value can be, for example, 100.
- The outfile parameter is the name of the output file generated.

The SE thread will generate an output file. For each burst executed, it will generate one line of output to the output file. The output information will include the time the burst started (in microseconds), and the length of the burst (in ms) and the process to which the burst belongs. It will use gettimeofday function to get the current time. The unit will be microseconds. The time should be expressed relative to the start time of the first burst. That means the first burst will start at time 0 (microseconds). You will format the time using 10 digits. Then, time 100 for example, will look like: 0000000100. Output can be like the following:

0000000000 200 1
0000200000 300 2
0000550000 150 1
0000650000 200 3
...

If input is to be taken from files, the format of each input file will be like the following. First burst type is specified, then the length of burst (in ms).
CPU 200
IO 600
CPU 300
IO 800
CPU 200

An example invocation of the program can be like the following:

schedule 5 100 300 500 8000 out.txt 50 RR 100 no-infile

That means cpu and io burst times will be generated randomly inside the program (not to be taken from input files); there will be 5 processes simulated; scheduling algorithm to simulate is RR with time quantum being 100 ms. Minimum cpu burst time is 100 ms, and maximum is 300 ms. That means all cpu burst times will be between 100 and 300 (uniformly distributed). Each process will generate 50 cpu bursts and then terminate.

When terminated, your program will also write to the screen the following information: the total waiting time in the ready queue for each process, the average response time for each process.

Submission

You will submit your schedule.c file and your Makefile. Put all these files into a directory named with your Student ID, and 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/she 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.

References

[1] There are a lot resources in Internet explaining POSIX mutex and condition variables.

Tips and Clarifications

- Start early.
- N can be at most 5.
- You will use POSIX threads (pthreads) mutex and conditions variables.
- There is no specific order about which process will add its burst to the queue, if there are several processes that have bursts. The process that wants to add the burst to the ready queue can do that when it gets the lock for the ready queue.
- If input is read from files, then "duration" is not effective. You will read each input file once. When all input files are processed, the simulation will terminate.
- In the output, the start time of a burst does not have to be equal to the sum of all bursts up to that time, since we are getting real-time (using gettimeofday function) at the start of a burst. For example, after the very first burst of length

200 ms is executed, the start time of the next burst can be, for example, 200030 microseconds (bigger than 200000 microseconds).

- For RR, if burst length is more than the time quantum (q), the execution will be in rounds of q ms. For each such execution of length q at most, information should be written to the output file. The output file should reflect the true execution order of the processes. In RR, if the time quantum of a burst has finished, but the burst did not finish yet, the burst needs to be added back to the ready queue.
- In RR, a new burst is added to the tail of the ready queue.
- In RR, SE will signal a PS after the whole burst is executed (not after every time quantum).