COSC 4740
Program 2

Due date: Oct 2
Worth 100 points

The goal of this assignment is to implement a subset of the process management component of an operating system, in particular, the process state transitions, process scheduling, and context switching. In addition, we will also learn the usage of *fork*, *exec*, *wait*, *pipe*, and *sleep* system calls in Unix. Read the man pages of these system calls for details.

Your program should consist of three Unix processes: *commander* and *process manager*. The commander process spawns the process manager, and the process manager spawns a reporter process when needed. The commander process reads commands from the standard input and passes them to the process manager. The process manager maintains a PCB table and implements three process management functions: process state transitions, process scheduling, and context switching. In addition, it will fork when calling a reporter method whenever it needs to print out the state of the system. The reporter method simply prints the current state of the system and terminates.

The state of each process managed by the process management component includes an integer variable whose value is changed during execution. The commander process issues one of the following commands to the process manager every two second:

1. **S** < *pid* > < *value* > < *run_time* >: Start a new process whose process id is *pid* and the value of its integer item is *value*. The total running time of this process is *run_time* time units.
2. **B** < *rid* >: Block the currently running process for a resource whose resource id is *rid*.
3. **U** < *rid* >: Unblock the process that is currently using the resource with resource id *rid*.
4. **Q**: End of one unit of time.
5. **C** < *cmd* > < *num* >: Change the value of the integer variable of the currently running process as follows: (1) if *cmd* == 'A', then *value* = *value* + *num*, (2) if *cmd* == 'S', then *value* = *value* - *num*, (3) if *cmd* == 'M', then *value* = *value* x *num*, and (4) if *cmd* == 'D', then *value* = *value*/*num*.
6. **P**: Print the current state of the system.
7. **T**: Print the average turnaround time, and terminate the system.

The process manager implements the scheduling policy of multiple queues with priority classes that was discussed in class. A new process starts with priority 0 (highest priority) and there are a maximum of 4 priority classes. Quantum size for priority class 0 is 1 unit of time. Assume that there are 3 resources (*rid* = 0, 1, and 2) on which processes may block. Resources are allocated to processes based on priorities. The process manager also manages the current time (an integer initialized to 0). The value of the current time is incremented after commands **Q** and **C**. It spawns a reporter process to print the system state on receiving a **P** command from the commander process.

Use Unix pipes for communication between the commander and the process manager processes. Write C++ programs for process manager and commander processes in separate files.

Implement the process manager as a set of five data structures: *Time*, *PcbTable*, *ReadyState*, *BlockedState*, and *RunningState*. Use the **QueueArray** class you implemented in homework 1 in

ReadyState and *BlockedState*; the queue items should be PcbTable indices. The output from the reporter process should be as follows:

****
The current system state is as follows:
****

CURRENT TIME: <time>

RUNNING PROCESS: <pid, priority, value, start time, CPU time used so far>

BLOCKED PROCESSES:
Queue of processes blocked for resource 0:
<pid, priority, value, start time, CPU time used so far>
<pid, priority, value, start time, CPU time used so far>
..
..
Queue of processes blocked for resource 3:
<pid, priority, value, start time, CPU time used so far>
<pid, priority, value, start time, CPU time used so far>
..

PROCESSES READY TO EXECUTE:
Queue of processes with priority 0:
<pid, value, start time, CPU time used so far>
<pid, value, start time, CPU time used so far>
..
..
Queue of processes with priority 4:
<pid, value, start time, CPU time used so far>
<pid, value, start time, CPU time used so far>
..

****

Turning in the Assignment:
Hard copy:

1. A cover page with Name, program #2, cosc 4740 a repo name (see github and below for your repo name), Section # and statement of help.
2. Output of the driver for ONLY the following times, 0, 6, 11 (4 of them), 15, 118, and 125

Soft copy:

1. Use this link to create your repo https://classroom.github.com/a/GRmLtiHW
2. Following good git commit styles. They are completely up to you now.
3. Create/Edit the readme.md file, add the following:
    - Name

    - How to compile the code, if there is no makefile.
    - List anything that doesn't work (that you know of)
4. Lastly ensure everything has uploaded to the github website and not just the local repo.

Code will be graded on correctness, comments, and coding style.