CS 405 -- PROJECT 1

Write a simple CPU scheduler.

- When first invoked, the user can set the scheduler to use either:
    - a. non-preemptive SJF (up to next I/O burst
    - RR scheduling with a user-defined (i.e., user-entered) time quantum. (Of course, if a process finishes its CPU burst before its quantum is ended, the dispatcher dispatches the next process immediately.)
- After setting the scheduling algorithm (and time quantum), the user loads a series of jobs into a job pool. This can be done either interactively or from a file. The job characteristics are given below.
- In your implementation, you will need objects similar to the following:

    System clock/timer (NOTE: this is a counter -- it is NOT the actual system clock and it is NOT a Java timer object)
    JobQueue (all processes start out here -- FCFS scheduling)
    ReadyQueue (contains up to three processes)
    CPU (go here for processing)
    DiskQueue (go here when need to perform I/O)
    I/OWaitingQueue (go here when done with I/O -- no limit to length -- no re-ordering)

- You will probably find it helpful to keep the following kinds of information about each process. You may want more or less, depending on your implementation. Since a set of this data is associated with each process, it might also be helpful (but is not required) to keep it in the process's PCB. This is a little bit of a stretch, when compared to the real world -- since in the real world the system usually doesn't have ***a priori*** knowledge of the upcoming CPU/IO bursts -- although PCBs actually DO sometimes contain accounting information. In any case, you will probably need data similar to the following, and it is fine for this project to put it in a PCB object.

    Process ID
    List of CPU bursts for the process (list, vec, array, etc.)
    List of I/O bursts for the process (list, vec, array, etc.)
    Index/iterator for CPU burst list
    Index/iterator for IO burst list
    Counter indicating remaining time in current CPU burst
    Counter indicating remaining time in current I/O burst
    Pointer/reference to next PCB in queue

- All queues are queues of PCBs (i.e., the PCB really IS the process)
    - The PCBs must be built and enqueued on the job queue when the job pool is initially loaded into the system.
- The ready queue will hold up to **three** processes. It should be filled **before** the scheduler makes the

decision on what PCB to schedule next. After the scheduling decision is made, then the chosen process can move to the CPU. At that time the queue will have (at least) one empty space in it. The queue contents remain unchanged **until the next context switch**; it is not refilled until just before the scheduler has to make its next decision on which PCB to dispatch.

- When refilling the ready queue, the schedulers **first** take the process (if any) that has just been preempted from the CPU and place it in the ready queue. **Next** they try to get a process or processes from the IOWaitQueue, and **last**, if there is still any space in the ready queue, they take a new process from the job queue.

- Your schedulers must keep the PCBs updated **after each timer click**, and as the PCBs move between queues.

- When entering an IOWait, processes move to the disk for IO first, then to the IOWaitQueue **AFTER** completing IO. Processes **ALWAYS** go to the IOWaitQueue after completing I/O -- never directly to the ready queue. They wait there until space becomes available on the ready queue.

    - Assume processes can begin their I/O immediately, and do not have to wait for the disk(s) to be available.

- After the all the queues have been filled, scheduler has run, and the selected job has been moved to the CPU, but before it has run in the CPU for this timeslice, you must print out the system time in clock ticks, and a list of the processes in all the queues and the CPU. The process listed as being in the CPU should be the process that just got moved into the CPU and is ready to run. The output should be in the following format. If it is not, you will lose points:

```
Time:  12
    CPU:          1
    Job queue:    5
    Ready queue:  2, 3
    Disk queue:
    IOW queue:    4

Time:  16
    CPU:          2
    Job queue:    5
    Ready queue:  3, 4
    Disk queue:   1
    IOW queue:

Time:  18
    . . .
    . . .
    . . .
    . . .
```

- When an job enters the job pool, we know the length of all of its CPU and I/O bursts (by counting the instructions and number of pages in the I/O). This is loaded into the job's PCB and is used by the schedulers. Here is the data to use when you submit your project:

Job 1 -- CPU 5 I/O 2 CPU 6 I/O 3 CPU 3
Job 2 -- CPU 3 I/O 1 CPU 2 I/O 2 CPU 4 I/O 1 CPU 3

Job 3 -- CPU 2 I/O 6 CPU 2 I/O 6 CPU 2
Job 4 -- CPU 3 I/O 4 CPU 2
Job 5 -- CPU 6 I/O 5 CPU 6

/***USE THIS DATA TO GENERATE OUTPUT TO TURN IN WITH YOUR PROJECT***/

- For SJF, run the module with the jobs entering both in the above order and the jobs starting in reverse order (i.e., job 5 starts first, then job 4, etc. -- order and values of bursts within a job remain the same). For RR, use only the above order (no reverse order), but run four times, using time quanta of 2, 3, 4, and 5.

- Submit a listing and sample runs using the test data and instructions above. Soft copy is fine.

- The class website contains a file called *CS405Project1TestData.zip*. This is to help you understand and test your project, and verify that it is working correctly

  HOWEVER

  /***DO NOT USE THE EXAMPLE DATA TO GENERATE OUTPUT TO SUBMIT WITH YOUR PROJECT ***/

  /***USE THE DATA LISTED SEVERAL BULLETS ABOVE IN THE ASSIGNMENT***/