Homework Assignment #6 [50 pts] – Scheduling

You are expected to do your own work on all homework assignments.(See the statement of Academic Dishonesty on the Syllabus.)

Check the Syllabus for the late assignment policy for this course.

How to turn in?

Assignments need to be turned in via Laulima. Check the Syllabus for the late assignment policy for the course.

What to turn in?

You should turn in a tarred archive named `ics332_hw6_USERNAME.tar` that contains a single top-level directory called `ics332_hw6_USERNAME`, where `USERNAME` is your UH username. In that directory you should have all the files **named exactly** as specified in the questions below.

Expected contents of the `ics332_hw6_USERNAME` directory (note that in Linux text files do not need to have a .txt extension):

- `README.txt`: your report (for both Exercises)
- `workload_q4.txt`: a workload file (Exercise #2, Question #4)

Your report should be written in plain text (in which case it must be named `README.txt`. Your report must be readable "as is". points will be removed if the report is not readable.

Exercise #1 [12 pts]: Warm-up

Consider a system that uses Round-Robin scheduling with a context-switching/scheduling overhead of duration **0.1ms**. We represent the execution timeline of a job mix on this system by indicating 1ms of computation for a process via an uppercase letter. For example, the following string:

AAAoBoCoAAoDDD

means that process A executes for 3ms, then process B for 1ms, then process C for 1ms, then process A again for 2ms, then process D for 3ms, with appropriate context-switches in between. This example execution lasts 10ms + 4 * 0.1ms = 10.4ms (10ms of actual process execution, .4ms of context-switch overhead).

each one does one infinitely long CPU burst). The system begins with A on the CPU at the beginning its time quantum while B and C are in the Ready Queue, in that order.

a) Show the execution pattern (as a string of A's, B's, C', and o's) assuming that the scheduler time quantum is equal to 4 ms. Show the execution for more than 20ms (but less than 30ms).

b) In the long run (i.e, assuming jobs don't ever terminate), what percentage of the CPU time is wasted doing context-switching/scheduling?

Question #2 [4 pts]

Consider the following execution on our system:

AAAAAoBBBBoCoDDDDDoAAAAAoCoDDDDDoAAAoCoDDDDDoAAAAAoBBBBB

a) Out of the four jobs, which one do you think is a more interactive process (e.g., a text editor) and why?

b) In general (not specific to the execution above), is it possible to have a job run for longer than the time quantum with a Round Robin scheduling algorithm? Explain your answer.

Question #3 [4 pts]:

Consider a job mix in which there is one CPU-intensive job, A, that has a single infinitely long CPU burst, and 3 I/O-intensive jobs, B, C, and D. The scheduler uses Round Robin scheduling with a time quantum of 3 ms. Here is an observed execution, where initially A is running and B, C, and D are in the ready queue in this order:

AAAoBBoCoDoAAAoBBoAAAoDoAAAoBBoCoDoAAAoBBo

Answer the following questions based on the above. Explain and give a specify a part of the execution.

a) Could any of B's I/O-burst time be 6 ms? Explain.

b) Could any of B's I/O-burst time be 4 ms? Explain.

Exercise #2 [38 pts]: Reverse-Engineering a Scheduler

Overview

In this assignment you are provided a simple simulator of a **single core** computer: DiscreteTimeSchedulingSimulator.jar. This is an executable jar file, and if you run it from the command-line as is you get:

The simulator takes two command-line arguments:

1. A **workload file** that describes a set of jobs the execution of which is to be simulated
2. A **scheduling algorithm name**, which can be ALG1, ALG2, ALG3, ..., and ALG8. The goal of the assignment is to reverse-engineer the simulation output to determine what some of these algorithms are (hence their cryptic names).

The **workload file** describes one job per line (lines beginning with "#" are comments). Here is an example file:

```
# two jobs
job1    0    1   7   3   2
job2    4    2   1   2   4
```

This describes a system with two jobs, `job1` and `job2`. Each job is described by 6 features:

1. a name
2. an arrival time (integer)
3. a priority (integer, lower number means higher priority)
4. a CPU burst duration (integer)
5. an IO burst duration (integer)
6. a number of iterations

So, for instance, `job1` above arrives in the system at time 0, has priority 1, has a CPU burst duration of 7, an IO burst duration of 3, and goes through 2 iterations.

The **number of iterations** is the number of times the job does a CPU burst + IO burst execution. So, for instance, if `job1` executes by itself on the core, it will run from time 0 (its arrival time) until time 0

- (7 + 3) * 2 = 20. Similarly, `job2`, if by itself, would run from time 4 until time 4 + (1 + 2) * 4 = 16. However, when both these jobs run together, they compete for the CPU according to the OS's scheduling algorithm.

So, assuming the above 2-job file is named `workload.txt`, here is what an execution of the simulator would look like with, say, scheduling algorithm ALG2:

```
% java -jar DiscreteTimeSchedulingSimulator.jar workload.txt ALG2

------------------------------------------------------
|  Simulated workload                                |
------------------------------------------------------

                arriv.  prio.   CPUb    IOb     iter.
```

```
------------------------------------------------------
|  X: execute   |     R: ready    |     B: blocked   |
------------------------------------------------------

    job1        job2

0:      X       -
1:      X       -
2:      X       -
3:      X       -
4:      X       R
5:      X       R
6:      X       R
7:      B       X
8:      B       B
9:      B       B
10:     R       X
11:     X       B
12:     X       B
13:     X       R
14:     X       R
15:     X       R
16:     X       R
17:     X       R
18:     B       X
19:     B       B
20:     B       B
21:     -       X
22:     -       B
23:     -       B

------------------------------------------------------
|  Summary statistics                                |
------------------------------------------------------

        turnaround time             wait time
       ------------------------------------
job1          21                    1
job2          20                    8
```

The above output has three sections:

1. A summary of the workload in the workload file
2. The execution trace with what each job is doing at each timestep
3. Summary statistics about the execution

operation. As expected job2 arrives at time 4. Although job2 by itself would complete at time 15, due to competition with job1, it completed at time 23. And indeed, we can see that its wait time is 8, i.e., it was in the Ready Queue for 8 time units unable to run on the CPU because job1 was running.

Reverse engineering the scheduling algorithms

Each question below focuses on one scheduling algorithm. Based on the output of the simulator you must then answer some questions, and eventually identify the scheduling algorithm.

IMPORTANT: you must justify your answers

Below are example justifications to use as templates. Note that in your justifications, when referencing a timestep to prove a point, always refer to the **earliest** such timestep. You can assume that if you don't see some behavior in the simulator output, then it never happens. The lower the number, the higher the priority (e.g., priority 1 is higher than priority 2).

Here are the two example acceptable justication format:

- "At timestep 12, ALGx picks job2 over job4 out of the ready queue, showing that it does not prioritize jobs by arrival time"
- "ALGy always picks from the ready queue the job with the highest priority, so it does select jobs by their priority"

If you find an algorithm to be Round-Robin, you have to give a time quantum

For all questions below, use this 4-job workload.txt workload file:

```
# <name> <arrival> <priority> <CPU> <IO> <iterations>
job1        0          1        4     3       4
job2        1          4        5     2       4
job3        2          3        4     2       5
job4        1          1        3     5       3
```

Pick algorithms from this algorithm bank when guessing the scheduling the algorithms:

```
Preemptive Priority             STCF
Non-Premptive Priority          SJF
FCFS                            MLFQ
RR (specify time quantum)
```

Question #1: ALG5 [13 pts]

Based on the simulator output using algorithm ALG5 answer the following questions:

q1.2 [3 pts] Does ALG5 prioritize jobs based on earliest arrival times? ("arrival time" means "time of first arrival in the system", not "arrival time in the ready queue")

q1.3 [3 pts] Is ALG5 Round-Robin?

q1.4 [3 pts] Does ALG5 prioritize jobs based on their priorities?

q1.5 [1 pts] Which scheduling algorithm (out of the ones in the bank) is ALG5?

Question #2: ALG3 [13 pts]

Based on the simulator output using algorithm ALG3 answer the following questions:

q2.1 [3 pts] Is ALG3 preemptive?

q2.2 [3 pts] Does ALG3 prioritize jobs based on earliest arrival times? ("arrival time" means "time of first arrival in the system", not "arrival time in the ready queue")

q2.3 [3 pts] Does ALG3 prioritize jobs based on shortest remaining CPU burst duration?

q2.4 [3 pts] Does ALG3 prioritize jobs based on their priorities?

q2.5 [1 pts] Which scheduling algorithm (out of the ones in the bank) is ALG3?

Question #3: ALG6 [8 pts]

Based on the simulator output using algorithm ALG6 answer the following questions:

q3.1 [3 pts] Is ALG6 preemptive?

q3.2 [3 pts] Is there an upper bound on the number of consecutive timesteps a process can spend in the Running state? If so, what is that upper bound?

q3.3 [2 pts] Which scheduling algorithm (out of the ones in the bank) is ALG6?

Question #4 [4pts]

q4 [4 pts] Construct and turn in a **4-job** workload file, workload_q4.txt, so that with ALG6:

- All jobs arrive by time 9
- All jobs do 10 iterations
- All jobs have a CPU burst of at least 3
- All jobs have an IO burst of at most 10
- All jobs have wait time of 9