COMP 346 Operating Systems

Assignment 2

Due Date: 11:55 PM, Sunday, July 26

Note: All submissions are made via Moodle (see "deliverables" section). Late penalties (< 1 hour = 10%, < 24 hours = 25%) will be applied automatically.

After you upload the assignment, please verify that your submission is recorded as submitted.

Description: This second assignment will allow you to become more familiar with the notion of CPU scheduling. As we have discussed in class, there are many different scheduling algorithms that can be used to optimize various performance metrics. In this assignment, we will focus on the comparison of two fairly basic mechanisms: **First Come First Served** and **Shortest Job First**. Moreover, we will look specifically at how each performs relative to *average wait time*.

Details: You will be creating a simulation that evaluates the two algorithms on a common "scheduling trace." The trace consists of a set of CPU burst/wait cycles that are associated with each process. Each set of cycles is unique to a given process. For example, two processes might have the following cycle sets:

Process 0: <24, 45>, <45, 128>, <67, 345>

Process 1: <123, 65>, <45, 918>

Note that the number of cycles does not have to be the same for each process, since some processes will simply run longer than others. In this case, the first cycle for Process 0 would consist of a 24 millisecond CPU burst, followed by a 45 ms wait in the IO queue. Process 1 would start with a 123 ms CPU burst, and then a 65 ms wait. When Process 0 eventually gets back to the CPU, it would spend 45 ms running, before moving to the IO queue for 128 ms. The simulation would continue until both processes eventually completed all of their cycles.

Of course, the order of execution depends entirely upon the scheduling algorithm. With FCFS, processes go to the CPU in the order that they arrive in the readyQ. This order may change during each iteration, as they will return from the IO queue only when their wait cycle has been completed.

For SJF, the scheduling is a little more sophisticated. In this case, the idea is to use exponential averaging (as described in the slides), to try to schedule the shortest/fastest process first during each iteration. The idea is that this would reduce the average wait time for each process.

Parameters:

In order to perform the evaluation, constraints must be specified on the command line and/or within the code.

On the command line itself, two parameters will be specified. The first is the **process_count**. This will be a number between 2 and 20. The second is the **cycle_max**. This is a number between 2 and 1000 that specifies the maximum length of the cycle trace for each process. In practice, for each process you will generate a random number between 1 and cycle max that will represent the number of burst/wait cycles for that process. Again, each process will be of a different length.

In terms of the burst and wait cycles, they will be defined as follows. The wait cycles are simple. They will be represented by a randomly generated number between 1 and 1000 (these numbers represent milliseconds).

The CPU burst times are slightly more involved. In order to allow the SJF estimate to converge fairly quickly (since we don't want to have to run the simulation for hours), we will ensure that individual processes produce cycles that are fairly similar. To do this, you can use the following approach. Randomly generate a number between 1 and 10, then multiply this value by 100. This will be the *root*. Now, for each cycle, generate a second number between 1 and 10 and add it to the root. This will be the actual burst time. For example, if for a given process, we have a root of 300, then the burst times might be {301, 307, 304, 301}. For another process, they might be {705, 710, 709}.

Scheduling:

Your job is to create a group of processes, each with a pre-defined set of burst/wait cycles. You will then feed this stream first into the FCFS scheduler and then the SJF scheduler. To ensure that your algorithm is doing what it is supposed to be doing, you will display information continuously about the scheduling process. Once both scheduling streams have been run, you will compute the average wait times and print them to the screen.

NOTE: In order for the results to be meaningful, both schedulers must work on exactly the same trace sets.

In order to put this in context, let's look at the output that was generated from a very small run (at the end of the assignment description). The command line was

Scheduler 3 3

So we generate just three processes and allow each one to have between 1 and 3 burst/wait cycles.

You can see that the output shows results for each scheduling iteration. We begin by listing each process in the readyQ and the waitQ (initially, this is empty). For the FCFS algorithm, we simply select the first process in the list, then show (i) the time it has been waiting, (ii) its current burst time (iii) the cycles remaining for this process (iv) the new status of the process – either it is now finished or it will be moved to the waitQ for some period of time.

We then check the waitQ to see if anything is ready to be moved back to the readyQ. This would happen, for example, if the current burst cycle was longer than the remaining wait time for one or more processes in the waitQ.

This process completes a single scheduling iteration and we may now turn our attention back to the readyQ. Eventually, the readyQ and waitQ are both empty. At this point the scheduler has completed its job.

The process is repeated for the SJF scheduler (again, using the same trace information). The only difference that you will see in the running output is that when the readyQ is displayed, the estimated burst time for each process is displayed. This is the time -- derived by exponential averaging – that the scheduler "thinks" will be the next burst for the process. It should be clear that it will not be exactly the same as the burst recorded in the trace. This value is not known by the scheduler until after a process has been selected and actually runs. Note that the estimates are initialized with the value 500 ms since we don't have any previous burst data at that point.

Finally, once both schedulers have done their job, you can print out the results. Here, we will print the average wait time per process (rather than a single overall value). Essentially, this is the total wait time for each process (i.e., how long they sat in the readyQ), divided by the number of cycles.

What you will see in the sample output is that SJF is really no better than FCFS. This might seem discouraging at first glance but it is really not surprising. With just 3 cycles (or less), there isn't enough time for the SJF estimates to converge on the real values. As a result, the SJF estimates are complete guesses and are no better than FCFS.

However, I have also included the average wait time output for a longer run of 10 processes and 500 cycles. When you look at these results, you can a dramatic improvement. While a few of the processes show similar averages, the majority of the times are significantly better for SJF.

Deliverables: All code is to be written in Java. Multiple source files should be combined into a single zip file and submitted to Moodle (not the Faculty's EAS system). The zip file should be named

LastName_FirstName_A2.zip (e.g., Smith_John_A1.zip)

No other files should be included (no class files or extra files generated by your development environment). The source code will be extracted and compiled by the marker, who will then execute a series of test cases against the code to ensure correctness.

Finally, you should include a README text file to indicate what parts of the assignment work or don't work. This may make it easier for the grader to give you points if there are problems/limitations with your solution.

Good luck...

Sample output stream: 3 processes and up to three cycles

```
** FCFS Scheduler **


Current ReadyQ:
 Process 0, cycle count: 3
 Process 2, cycle count: 1
 Process 1, cycle count: 2

Current WaitQ:

Running process 0 ...
 wait time in readyQ (ms): 0
 burst time (ms): 208
 remaining cycles: 2
 process moved to wait Q for 411 ms

Checking WaitQ...
 No processes ready to be moved to readyQ

 ** Scheduling iteration complete


Current ReadyQ:
 Process 2, cycle count: 1
 Process 1, cycle count: 2

Current WaitQ:
 Process 0, wait time: 411

Running process 2 ...
 wait time in readyQ (ms): 208
 burst time (ms): 204
 remaining cycles: 0
 process has terminated

Checking WaitQ...
 No processes ready to be moved to readyQ

 ** Scheduling iteration complete


Current ReadyQ:
 Process 1, cycle count: 2

Current WaitQ:
 Process 0, wait time: 207

Running process 1 ...
 wait time in readyQ (ms): 412
 burst time (ms): 609
 remaining cycles: 1
 process moved to wait Q for 499 ms

Checking WaitQ...
 Process 0 wait time: -402
 moving Process 0 to ready Q


 ** Scheduling iteration complete


Current ReadyQ:
 Process 0, cycle count: 2

Current WaitQ:
 Process 1, wait time: 499

Running process 0 ...
```

```
 wait time in readyQ (ms): 0
 burst time (ms): 202
 remaining cycles: 1
 process moved to wait Q for 595 ms

Checking WaitQ...
 No processes ready to be moved to readyQ

 ** Scheduling iteration complete


Current ReadyQ:

Current WaitQ:
 Process 1, wait time: 297
 Process 0, wait time: 595

Running process 1 ...
 wait time in readyQ (ms): 0
 burst time (ms): 604
 remaining cycles: 0
 process has terminated

Checking WaitQ...
 Process 0 wait time: -9
 moving Process 0 to ready Q


 ** Scheduling iteration complete


Current ReadyQ:
 Process 0, cycle count: 1

Current WaitQ:

Running process 0 ...
 wait time in readyQ (ms): 0
 burst time (ms): 203
 remaining cycles: 0
 process has terminated

Checking WaitQ...
 Queue is Empty
 No processes ready to be moved to readyQ

 ** Scheduling iteration complete


Current ReadyQ:

Current WaitQ:


We are done, all processes have terminated



** SJF Scheduler **


Current ReadyQ:
 Process 2, cycle count: 1 , estimated burst: 500
 Process 1, cycle count: 2 , estimated burst: 500
 Process 0, cycle count: 3 , estimated burst: 500

Current WaitQ:

Running process 2 ...
 wait time in readyQ (ms): 0
 burst time (ms): 204
 remaining cycles: 0
 process has terminated

Checking WaitQ...
 Empty
 No processes ready to be moved to readyQ

 ** Scheduling iteration complete


Current ReadyQ:
 Process 0, cycle count: 3 , estimated burst: 500
 Process 1, cycle count: 2 , estimated burst: 500

Current WaitQ:

Running process 0 ...
 wait time in readyQ (ms): 204
 burst time (ms): 208
 remaining cycles: 2
 process moved to wait Q for 411 ms

Checking WaitQ...
 No processes ready to be moved to readyQ

 ** Scheduling iteration complete


Current ReadyQ:
 Process 1, cycle count: 2 , estimated burst: 500

Current WaitQ:
 Process 0, wait time: 411

Running process 1 ...
 wait time in readyQ (ms): 412
 burst time (ms): 609
 remaining cycles: 1
 process moved to wait Q for 499 ms

Checking WaitQ...
 process 0 wait time: -198
 moving 0 to ready Q


 ** Scheduling iteration complete


Current ReadyQ:
 Process 0, cycle count: 2 , estimated burst: 354

Current WaitQ:
 Process 1, wait time: 499

Running process 0 ...
 wait time in readyQ (ms): 0
 burst time (ms): 202
 remaining cycles: 1
 process moved to wait Q for 595 ms

Checking WaitQ...
 No processes ready to be moved to readyQ

 ** Scheduling iteration complete


Current ReadyQ:

Current WaitQ:
 Process 1, wait time: 297
 Process 0, wait time: 595
```

```
Running process 1 ...
 wait time in readyQ (ms): 0
 burst time (ms): 604
 remaining cycles: 0
 process has terminated

Checking WaitQ...
 process 0 wait time: -9
 moving 0 to ready Q


 ** Scheduling iteration complete


Current ReadyQ:
 Process 0, cycle count: 1 , estimated burst: 351

Current WaitQ:

Running process 0 ...
 wait time in readyQ (ms): 0
 burst time (ms): 203
 remaining cycles: 0
 process has terminated

Checking WaitQ...
 Empty
 No processes ready to be moved to readyQ

 ** Scheduling iteration complete


Current ReadyQ:

Current WaitQ:


We are done, all processes have terminated



 ** Results: Average Wait Time per process

Process 0
 FCFS: 0
 SJF: 68
Process 1
 FCFS: 206
 SJF: 206
Process 2
 FCFS: 208
 SJF: 0
```

Results for a larger run: 10 process and 500 cycles

```
 ** Results: Average Wait Time per process

Process 0
 FCFS: 467
 SJF: 578
Process 1
 FCFS: 637
 SJF: 671
Process 2
 FCFS: 1607
 SJF: 15
Process 3
 FCFS: 1400
 SJF: 5
Process 4
 FCFS: 2186
 SJF: 72
Process 5
 FCFS: 2050
 SJF: 404
Process 6
 FCFS: 637
 SJF: 14
Process 7
 FCFS: 1726
 SJF: 751
Process 8
 FCFS: 1152
 SJF: 231
Process 9
 FCFS: 906
 SJF: 13
```