Seth Cattanach
CSE-34341 Operating Systems
Assignment 3 due 27-Mar-19
Project Report: Scheduling Algorithms

I. Overview

This project implements and analyzes scheduling algorithms commonly used in modern operating systems. Simulated jobs from a text file are read by the program, *schedule.txt*, and the specified algorithm is used to process each job until all jobs have been completed. Metrics such as average turnaround time and total execution time are tracked using simulated time units, which means the metrics are accurate and independent across all platforms and processors (that is, "real time" – the amount of time the processor takes to run the program, which can vary for a given set of jobs and a scheduling algorithm – doesn't affect how metrics are measured. All metrics will be the same for a given set of jobs and a given scheduling algorithm).

To test these results, execute the program **schedule.py** (requires Python 3.0 or greater). In the *main()* function near the bottom of the script, there are six algorithms listed (lines 251-256). Uncomment one of these algorithms at a time to test. Specify the text file from which to read jobs in line 242 (default is "jobs.txt"). Two sample files are included in the zip file: "jobs.txt" (50-job sample) and "jobs2.txt" (30-job sample) – these are the two files upon which the algorithms were benchmarked.

Jobs are labelled with the following format:

[Job Name, Arrival Time, {+CPU-IO+CPU-IO…}, Priority, Owner]

…where "CPU" and "IO" are numbers that represent how many time units are in that CPU or IO burst. Two additional boolean fields ("arrived" and "completed") are appended to the end of each job when the job is read from its file; these fields are used to track job progress during several of the algorithms.

II. Algorithms Implemented

a. First Come, First Served *(non-preemptive)*
    i. Jobs are executed in the order they arrive. Once a job has started execution, it will execute until completion

b. Shortest Time First *(non-preemptive)*
    i. The shortest job that has arrived at a given time will be executed until completion (including its CPU and I/O bursts). If a new shortest job arrives during the execution of another job, it will be added to the front of the queue but will not begin execution until the previous job has finished

c. Shortest Time to Completion *(preemptive)*
    i. The algorithm will always select the job with the shortest next CPU burst. If a job is waiting for an I/O burst, this I/O wait will occur in the background while another process is executing its CPU burst. This I/O wait will also occur in parallel; if multiple jobs are waiting on an I/O burst, they will all execute in the background while the CPU is occupied. If a job with a shorter CPU burst arrives at any point, the algorithm will immediately switch and begin executing that job in preemptive fashion

d. Round Robin *(preemptive)*
    i. Similar to the shortest-time-to-completion algorithm, but jobs will be allowed to execute for a maximum of one *quantum* before the algorithm will switch to the next process in the queue (default quantum is 10 time units, but this can be set as an optional parameter in __*main*__). I/O bursts are again handled in the background and in parallel

e. Priority *(preemptive)*
    i. Jobs will be ordered in the queue based on their priority (1-5, where 1 is the highest priority). If a higher-priority job arrives during the execution of another job, the CPU will immediately begin executing the higher-priority job. Within a given priority level, jobs are executed in round-robin fashion. I/O bursts are again handled in the background and in parallel

III. Performance Metrics and Evaluation

These five algorithms were benchmarked according to the following metrics: total time to completion, average turnaround time, balance, and policy enforcement. The benchmarks were completed for two sets of jobs; one set contains 30 jobs and the other contains 50 jobs (these files are included as *jobs2*.txt and *jobs.*txt, respectively). All units are generic time units representing CPU clock cycles, as described in the **Overview**; these cycles are independent of real time and actual processing power.

The metrics used in benchmarking are described below:

Total time to completion (TTtC) measures how many time units the algorithm takes to schedule and complete all jobs.

Average Turnaround Time (AvgTT) measures the average time between a job's arrival and completion

Balance (BAL) measures the percentage of time units in which the CPU was occupied (as opposed to waiting on I/O)

Policy Enforcement (POL) notes the algorithm's required policy (or policies) and whether it is always enforced

Benchmarking results are reported below in **Figure 1** (for the round robin algorithm, benchmarks were completed for three different quantum sizes):

	First Come First Served		Shortest Job First		Shortest Time to Completion		Round Robin quantum size=(X)		Priority	
	Jobs=30	*Jobs=50*	*Jobs=30*	*Jobs=50*	*Jobs=30*	*Jobs=50*	*Jobs=30*	*Jobs=50*	*Jobs=30*	*Jobs=50*
TTtC	7205	13164	7205	13164	1286	7197	3384 **(4)** 3362 **(10)** 3366 **(30)**	6791 **(4)** 6791 **(10)** 6791 **(30)**	3361	6791
AvgTT	3019	5519	1681	3305	822	1598	1176 **(4)** 1215 **(10)** 1482 **(30)**	2574 **(4)** 2686 **(10)** 3101 **(30)**	1506	2860
BAL	45.5%	51.6%	45.5%	51.6%	89.0%	94.4%	96.8% **(4)** 97.4% **(10)** 97.4% **(30)**	100% **(4)** 100% **(10)** 100% **(30)**	97.6%	100%
POL	Jobs are executed in the order they arrive (always enforced)		Shortest jobs always executed first (if arrived)		Shortest CPU burst always executed next (enforced as long as all jobs are not currently IO-bound)		Always enforces policy that no job should execute longer than one quantum at a time, and jobs should be executed in round-robin fashion		Always enforces policy that higher-priority jobs should be executed first (but if multiple jobs have equally high priority, executed in round-robin fashion)	

<u>Figure 1:</u> Benchmarking results for 5 algorithms, 2 sets of jobs, and 4 metrics (and quantum size when applicable). Units for "Total Time to Completion" and "Average Turnaround Time" are simulated CPU clock cycles

Discussion of results

As expected, the two non-preemptive algorithms (first come-first served and shortest job first) had the worst performance across the metrics measured. In particular, their balance is quite low since they must wait on each I/O burst individually and sequentially. However, they do have several strengths: since they are non-preemptive, the order of execution can easily be predicted. These algorithms would be especially suitable for CPU-bound processes, since their performance suffers with IO-bound processes (no ability to handle I/O wait in the background/in parallel). Additionally, these non-preemptive algorithms are much easier to implement and thus suitable for situations where performance isn't hugely important (smaller groups of jobs, for instance).

The shortest time to completion algorithm consistently yielded the best (lowest) average turnaround time, which was expected due to its preemptive nature and its emphasis toward always completing the shortest job remaining. It also provides a significant improvement in total execution time over the non-preemptive algorithms, although other preemptive algorithms were able to improve slightly more on total execution time. One disadvantage of this algorithm is that its balance isn't always maximized: when only a few jobs have arrived, finishing the shortest remaining job can create situations where all jobs are I/O bound and thus the CPU isn't fully maximized.

The round robin algorithm with a small quantum [smallest quantum benchmarked was q=4] acts very much like the shortest time to completion algorithm; the results indicate that average turnaround time decreases and the total execution time slightly increases as the quantum size decreases (this makes logical sense because the shortest time to completion algorithm acts similarly to the round robin algorithm with quantum size q=1). However, the round robin algorithm can never yield a lower average turnaround time than the shortest time to completion algorithm, because round robin ensures an equitable allocation of CPU time to all available processes (whereas shortest time to completion will always prioritize a job that is close to completion, regardless if it has been executing for a long time). The round robin algorithm also

yielded excellent balance metrics due to its preemptive nature and its ability to handle multiple I/O bursts in the background and in parallel (in the benchmark with 30 jobs, an increase in quantum size led to a slight increase in balance/CPU utilization. In the benchmark with 50 jobs, quantum size was irrelevant because there were enough jobs to always ensure a CPU burst was available)

Lastly, the priority algorithm, which implements round robin scheduling within a given priority level, yielded similar total execution times and slightly worse average turnaround times when compared to the pure round robin algorithm. This slight decrease in performance was also expected, because the priority algorithm includes an additional policy on top of round robin: it requires that higher-priority tasks are always completed first. So, a very short but low-priority task won't be completed for potentially a very long time, which significantly hurts average turnaround time.

IV. Implementation of New Algorithm (and its performance against the previous five)

The algorithm I developed was a variation of the two non-preemptive algorithms; it executes jobs in a non-preemptive fashion sorted by the fewest number of total bursts in the job. The benchmark results for this algorithm, "fewest bursts," are shown against the same metrics as the previous algorithms in **Figure 2**, below:

	Fewest Bursts First	
	Jobs=30	*Jobs=50*
Total Time to Completion	7205	13164
AvgTT	2496	4248
BAL	45.5%	51.6%
POL	Jobs are executed non-preemptively in order of fewest number of bursts to complete (always enforced)	

<u>Figure 2:</u> Benchmarking the new "fewest bursts first" algorithm using the same metrics and same jobs as in Figure 1

Since this is a non-preemptive algorithm, it makes sense that its total time to completion and balance metrics are the same as the other two non-preemptive algorithms (in both the 30 job and 50 job benchmarks). The key difference for this algorithm is its average turnaround time: it marks an improvement over the first come-first served algorithm (by about 23% in the 50-job benchmark and about 17% in the 30-job benchmark), though it's slightly worse than the shortest job first metric.

Though not simulated in this project, the concept of overhead could potentially cause this "fewest bursts first" algorithm to yield an average turnaround time that is *better* than shortest job first, in practice. In this project, there is no easy way to track overhead associated with switching from CPU to I/O bursts very often (since there are not actual jobs being executed, but rather simulated). With real jobs, however, a very short job with many separate bursts would incur lots of overhead, and if this job were executed first (as would happen in a shortest-job-first algorithm), it may actually take longer than a slightly longer job with only one or two bursts. **With that in mind, this algorithm is most efficient in terms of average turnaround time when a) there is overhead associated with switching between bursts, and b) there are processes with many bursts**. A hypothetical example is given below:

Three simplified jobs are given to be {J1, 0, +1-1+1-1+1-1+1}, {J2, 0, +3-4+1}, and {J3, 0, +9}. Assume there is an additional 1 CPU clock cycle of overhead to switch from a CPU burst to an I/O burst or an I/O burst to a CPU burst:

- The shortest job first algorithm would execute the jobs in the order **J1, J2, J3**. J1 would terminate at t=13 (7 cycles of execution and 6 cycles of overhead) for a turnaround time of 13, J2 would terminate at t=23 (13 plus 8 cycles of execution and 2 cycles of overhead for a turnaround time of 23), and J3 would terminate at t=32 (23 plus 9 cycles of execution). The average turnaround time would thus be (13+23+32)/3 = 22.67
- The fewest bursts first algorithm would execute the jobs in the order **J3, J2, J1**. J3 would terminate at t=9 (9 cycles of execution and 0 cycles of overhead), J2 would terminate at t=19 (9 plus 8 cycles of execution and 2 cycles of overhead) and J3 would terminate at t=32 (19

plus7 cycles of execution and 6 cycles of overhead). The average turnaround time would thus be (9+19+32)/3 = 20.

In this example, choosing the shortest job first incurred additional overhead and would yield a worse average turnaround time than choosing the job with the fewest total number of bursts.

The weaknesses of this “fewest bursts first” algorithm are obvious in a similar manner: when jobs have an equal or roughly-equal number of bursts, this algorithm will not always choose the shortest job and further, this algorithm will not provide significant benefit if overhead is negligible.

As previously mentioned, this is a non-preemptive algorithm and therefore it will likely never outperform some of the preemptive versions (shortest time to completion, round robin, etc.) in terms of turnaround time and total time to completion (once a job is started, it must execute to completion and cannot handle I/O bursts in the background). The advantage of this non-preemptive characteristic is stability and minimal overhead: in general, constantly switching between processes can incur additional overhead costs and cause additional problems should an error occur (since there can be many “half completed” processes at a given time), whereas a non-preemptive algorithm only switches when the job is complete.