Operating Systems - Summer 2019

Steven Li <lis18>

Justin Chen <chenj37>

Ishita Padhiar <padhii>

July 16, 2019

1) Truthfully, there should not be a “best” algorithm. Each of the four simulated algorithms have strengths and weaknesses that are appropriate for different situations, depending on the type of processes that you are running. As we will elaborate below for example, for CPU-bound processes Round Robin is best suited and for I/O bound processes SRT is usually best. Similarly, there are also scenarios when non-preemptive algorithms perform better than preemptive ones.

```
Algorithm FCFS
-- average CPU burst time: 103.027 ms
-- average wait time: 57.459 ms
-- average turnaround time: 164.486 ms
-- total number of context switches: 37
-- total number of preemptions: 0
Algorithm SJF
-- average CPU burst time: 103.027 ms
-- average wait time: 57.459 ms
-- average turnaround time: 164.486 ms
-- total number of context switches: 37
-- total number of preemptions: 0
Algorithm SRT
-- average CPU burst time: 103.027 ms
-- average wait time: 43.162 ms
-- average turnaround time: 151.054 ms
-- total number of context switches: 45
-- total number of preemptions: 8
Algorithm RR
-- average CPU burst time: 103.027 ms
-- average wait time: 46.486 ms
-- average turnaround time: 154.378 ms
-- total number of context switches: 45
-- total number of preemptions: 8
```

```
Algorithm FCFS
-- average CPU burst time: 84.304 ms
-- average wait time: 622.507 ms
-- average turnaround time: 710.810 ms
-- total number of context switches: 537
-- total number of preemptions: 0
Algorithm SJF
-- average CPU burst time: 84.304 ms
-- average wait time: 718.043 ms
-- average turnaround time: 806.346 ms
-- total number of context switches: 537
-- total number of preemptions: 0
Algorithm SRT
-- average CPU burst time: 84.304 ms
-- average wait time: 742.488 ms
-- average turnaround time: 832.035 ms
-- total number of context switches: 704
-- total number of preemptions: 167
Algorithm RR
-- average CPU burst time: 84.304 ms
-- average wait time: 646.331 ms
-- average turnaround time: 735.626 ms
-- total number of context switches: 670
-- total number of preemptions: 133
```

```
Algorithm FCFS
-- average CPU burst time: 903.431 ms
-- average wait time: 3185.499 ms
-- average turnaround time: 4092.930 ms
-- total number of context switches: 473
-- total number of preemptions: 0
Algorithm SJF
-- average CPU burst time: 903.431 ms
-- average wait time: 2889.805 ms
-- average turnaround time: 3797.237 ms
-- total number of context switches: 473
-- total number of preemptions: 0
Algorithm SRT
-- average CPU burst time: 903.431 ms
-- average wait time: 2811.782 ms
-- average turnaround time: 3720.338 ms
-- total number of context switches: 606
-- total number of preemptions: 133
Algorithm RR
-- average CPU burst time: 903.431 ms
-- average wait time: 3253.886 ms
-- average turnaround time: 4161.782 ms
-- total number of context switches: 528
-- total number of preemptions: 55
```

Figure 1: Simout03, Simout04, Simout05 Statistics

Specifically when a process burst times vary dramatically preemptive algorithms such as SRT or RR work better, but if process burst times are of similar length non preemptive algorithms such as SJF or FCFS will perform just as well, if not better. An example of this can be seen in Figure 1; Simout4 contains statistics for a test case that has which has lower variation in burst lengths, as opposed to Simout5 which

shows the statistics of a test case that has greater differences in burst lengths. In Simout4 we can see that FCFS and SJF each out-perform both RR and SRT, while in Simout5 SRT does better than FCFS and SJF.

The algorithm that is best-suited for CPU-bound processes is the Shortest Job First (SJF) algorithm. CPU-bound processes are processes that go faster if the CPU is faster. So, the algorithms best suited for these types of processes are algorithms that spend a lot of time running processes as opposed to using context switches. The two algorithms that have the fewest number of context switches are FCFS and SJF, since they don't use preemptions. However, SJF is a bit better suited for CPU-bound processes, since it benefits more from a faster CPU. The process of adding something to the queue requires more computational power with the SJF algorithm as compared to the FCFS algorithm, since it must calculate the location in the queue where the new process should go. This is reflected in the statistics calculated in

```
Algorithm FCFS
-- average CPU burst time: 1000.038 ms
-- average wait time: 1838.596 ms
-- average turnaround time: 2842.635 ms
-- total number of context switches: 52
-- total number of preemptions: 0
Algorithm SJF
-- average CPU burst time: 1000.038 ms
-- average wait time: 1838.673 ms
-- average turnaround time: 2842.712 ms
-- total number of context switches: 52
-- total number of preemptions: 0
Algorithm SRT
-- average CPU burst time: 1000.038 ms
-- average wait time: 1838.673 ms
-- average turnaround time: 2842.712 ms
-- total number of context switches: 52
-- total number of preemptions: 0
Algorithm RR
-- average CPU burst time: 1000.038 ms
-- average wait time: 1933.596 ms
-- average turnaround time: 2962.942 ms
-- total number of context switches: 381
-- total number of preemptions: 329
```

Figure 2: CPU-Bound Statistics Example

Figure 2, where we tested the 4 algorithms on processes that had large CPU burst times and small I/O times. When this was the case, the FCFS algorithm had the lowest wait times and turnaround times, but only marginally better than the SJF algorithm. We hypothesize that in a more accurate system, the SJF algorithm would perform better. Thus, SJF is best-suited for CPU-bound processes.

The algorithm that is best-suited for the I/O-bound processes is the Shortest Remaining Time (SRT) algorithm. I/O-bound processes are processes that go faster if the I/O subsystem is faster. So, algorithms that utilize more preemptions and context switches would use the I/O subsystem more, thus the SRT and SJF algorithms would be more ideal than the FCFS and RR ones. However, the SRT algorithm has the advantage over the SJF algorithm since SRT will finish processes more quickly. Because of this, the I/O subsystem will almost always be running, which is more ideal for I/O-bound processes. The statistics in Figure 3 also reflect this sentiment, as we tested the 4 algorithms on processes that had large I/O times and small CPU burst times. When this was the case, the SRT algorithm had the lowest wait times and turnaround times.

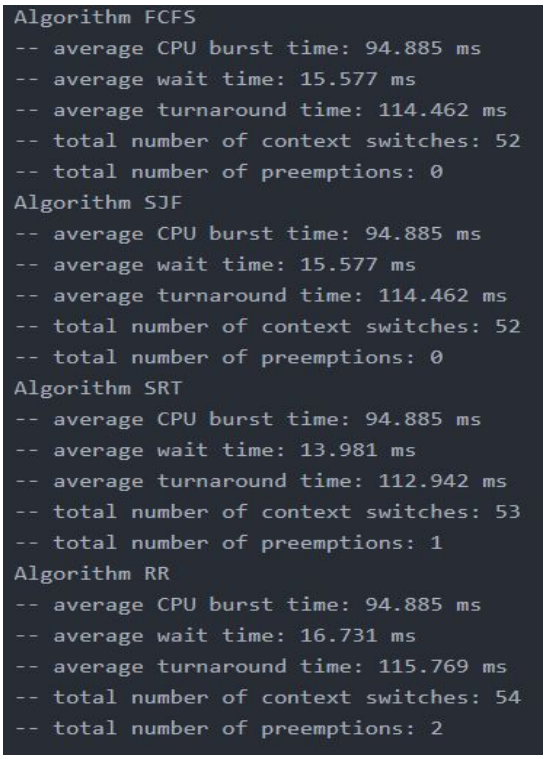

Figure 3: I/O-Bound Statistics Example

2) For the RR algorithm, changing rr_add from END to BEGINNING would cause the preempted process due to an expired time slice to be moved to the beginning instead of the end of the queue, which would make it the next process again to use the CPU. The "better" approach here would be to set rr_add to END, which would have it add preempted processes to the end of the queue. Adding processes to the END of the queue would allow the CPU to be used by other processes that have also been waiting in the queue. These processes could be shorter than the previously preempted process, allowing them to have time with the CPU and carrying out their CPU burst. If they finish earlier or within the allotted time slice, that means they can move on to their I/O bursts and future CPU bursts, causing the system to be steadily going through all of the processes. Adding processes to the BEGINNING of the queue wouldn't be as beneficial because if the previously preempted process is extremely long, then it would indefinitely block the queue and starve all other processes from using the CPU. At that point, it would be extremely similar

to the First Come First Serve (FCFS) algorithm, in that the faster arriving process would have priority in the queue.

3) Generally, an alpha value of 0.5 is accepted as standard, since it evenly balances out the approximation of the next CPU burst time between the previous approximation and the actual burst time. However, there's no singular "best" alpha value. If a process has burst times that vary greatly, it would be better to have a relatively low alpha value, since you wouldn't want to fully commit to changing to a value that would be wildly different from the next burst time. As demonstrated by Figure 4 (left), for processes with a big range of burst times, running SJF with a smaller alpha value yielded lower wait times and turnaround times. However, if a process has burst times that are all pretty similar, it would be better to have a higher alpha value, as each burst time can predict the next one pretty well. This is demonstrated in Figure 4 (right), as those statistics are for a process with very similar burst times, and running SJF with a larger alpha value yielded lower wait times and turnaround times. So, certain alpha values are better for certain cases, and an alpha value of 0.5 is a good neutral balance of the two extremes.

```
alpha = 0.1
Algorithm SJF
-- average CPU burst time: 5000.000 ms
-- average wait time: 8807.019 ms
-- average turnaround time: 13811.019 ms
-- total number of context switches: 52
-- total number of preemptions: 0
alpha = 0.9
Algorithm SJF
-- average CPU burst time: 5000.000 ms
-- average wait time: 9191.827 ms
-- average turnaround time: 14195.827 ms
-- total number of context switches: 52
-- total number of preemptions: 0
```

```
alpha = 0.1
Algorithm SJF
-- average CPU burst time: 5000.000 ms
-- average wait time: 11872.212 ms
-- average turnaround time: 16876.212 ms
-- total number of context switches: 52
-- total number of preemptions: 0
alpha = 0.9
Algorithm SJF
-- average CPU burst time: 5000.000 ms
-- average wait time: 11108.635 ms
-- average turnaround time: 16112.635 ms
-- total number of context switches: 52
-- total number of preemptions: 0
```

Figure 4: Statistics for large range of bursts (left) and small range of bursts (right)

4) The first major difference in switching from a non-preemptive algorithm to a preemptive one is that there will be more context switches in the preemptive algorithm. This is because each preemption requires a context switch. This also means that preemptive algorithms spend more time in the I/O subsystem. Another change from switching from SJF to SRT is that it introduces the possibility of starving. This is when a burst doesn't activate for a long time. The reason why SRT is more susceptible to this is because preemption allows for long bursts to be stalled after they begin, whereas SJF finishes a burst once it's been started. This means that a long process may not be finished for a very long time, but it also means that shorter processes get done even quicker.

5) There are a number of ways that the project specifications could be expanded to better model a real-world operating system. The first is the number of processes. For the project, we only had to deal with a maximum of 26 processes, but in a real operating system, there could be hundreds, thousands, or an even greater amount of processes.

The second is the individualization of each algorithm. For the project, we run the produced pseudo-random values on each scheduling algorithm individually to analyze the statistics of each algorithm, such as average wait times, burst times, and turnaround times, but a CPU ultimately is looking for process efficiency. It is looking to appropriately run all processes in the best way possible. For this reason, a CPU implementation should not solely rely on a single algorithm to prioritize and run its processes. It should use a combination of algorithms or a combination of characteristics of each algorithm in order to create the most efficient system.

The third is the way the average wait time, average CPU burst time, and average turnaround time is calculated. For the project, all of the averages were calculated over all wait times, executed bursts, and turnaround times. There was no differentiation if the wait times, burst times, or turnaround times were

from Process A or Process B. However, we believe that there should be a differentiation. We believe that the correct way of calculating the average wait time, burst times, and turnaround times should be differentiated between each process. We take all of the wait times, burst times, and turnaround times for each process, add them together, divide by the number of recorded wait times, burst times, or turnaround times respectively, and record them for each process. Then, we take all of these recorded values for each process, add each of them together, and divide by the number of processes to calculate each respective average wait time, average burst time, and average turnaround time.

6) If we were to create our own priority scheduling algorithm, we would use a combination of the four scheduling algorithms used in the project to meet certain conditions that arise in the CPU implementation. For the main algorithm, we would use Shortest Job First (SJF). However, when we have a tie for multiple processes, we would implement First Come First Serve (FCFS) as the tie-breaker algorithm to prioritize the processes. Finally, we would also add an AGING mechanism to the system, allowing processes that have been sitting in the queue for a large amount of time to have increased priority and not have long processes indefinitely block the CPU and starve out all other processes.

The advantages of the algorithm are that the average wait time and turnaround times will be relatively consistent at a medium range, not being too unreasonable large, but also not unrealistically low. According to Figure 1, the statistics output files demonstrate that SJF has a relatively medium average wait time and turnaround time compared to FCFS, SRT, and RR for varying CPU simulations. In addition, the algorithm will have a fair distribution of work between the CPU and the I/O subsystem, allowing for a more efficient CPU workflow. Although not perfectly, this algorithm will ultimately be able to somewhat depict a realistic CPU system.

The disadvantages of the algorithm are that it is difficult to predict burst time for SJF, and that it is difficult to implement the AGING mechanism. For the project, we were indirectly trying to determine

`argv[6]` : For the SJF and SRT algorithms, since we cannot know the actual CPU burst times beforehand, we will rely on estimates determined via exponential averaging (as discussed in class on 6/24). As such, this command-line argument is the constant α. And note that the initial guess for each process is $\tau_0 = \frac{1}{\lambda}$. When calculating τ values, use the "ceiling" function for all calculations.

Figure 5: Alpha value argument in the project PDF

future CPU burst times via exponential averaging by finding *tau* values, as depicted in Figure 5. The *tau* values is calculated based on previous CPU burst times, which has no correlation to future CPU burst times, and it might even be an inaccurate representation of the future CPU burst times. This may cause problems when we inaccurately display *tau* values for long processes and short processes.

For AGING, it will be difficult to increase the priority of a process that has been waiting a long time with respect to the priority placed by FCFS during tiebreakers and SJF for the main algorithm. For example, if FCFS and SJF placed priority on certain processes, should AGING increase a priority equal to, less than, or greater than the placed priority? Ideally, it would be different depending on how long the process has been inside the queue, but it would just be difficult to say what *x* amount of time waited results in a *y* amount of priority added.