Team Members: John Foster <fostej5> Keerti Sundaram <sundak3> Anya Tralshawala <tralsa>

CSCI 4210 - Operating Systems Project Writeup

1. **(4 points) Of the four simulated algorithms, which algorithm is the "best" algorithm for CPU-bound processes? Which algorithm is best-suited for I/O-bound processes?**

SJF is the best algorithm for CPU-bound processes. A CPU process is defined as a process that for most of its execution time is in the CPU. Generally, non-preemptive algorithms favor CPU-bound processes since there are no preemptions and processes running on the CPU do not get interrupted. Looking at our simulation results, we see that between the two non-preemptive algorithms, SJF has the lowest average wait time (for 16 processes, SJF: 199.345ms vs FCFS: 215.423ms). Therefore, we see that SJF performs the best for CPU-bound processes.

SRT is the best algorithm for I/O-bound processes. An I/O-bound process is defined as a process that spends most of its execution time in the Waiting for IO state. Generally, preemptive algorithms favor I/O-bound processes more since processes can preempt others and are less likely to have a higher waiting time. Looking at our simulation results, SRT had the lowest wait time compared to the other preemptive algorithms (16 processes, SRT: 199.043ms vs RR: 229.361ms). Therefore, SRT performs the best for I/O-bound processes.

2. **(4 points) For the SJF and SRT algorithms, what value of α produced the "best" results?**

For SJF and SRT, $\alpha = 0.75$ produced the best results, compared to $\alpha = 0.5$, wait time, and turnaround time both saw performance uplifts. For $\alpha = 0.5$ the simulation

resulted in a wait time of 206.35ms and an average turnaround time of 294.66ms, compared to α= 0.75 where the simulation saw an average wait time of 199.345 ms and an average turnaround time of 287.648 ms.

3. **(4 points) For the SJF and SRT algorithms, how does changing from a non-preemptive algorithm to a preemptive algorithm impact your results?**

Switching from a non-preemptive algorithm to a preemptive algorithm requires more overhead. The preemptive (SRT) algorithm also resulted in a faster execution time because the currently running processes could be preempted by faster running processes.

In terms of wait time, this varies between the SJF and SRT algorithms. SJF will always have a minimized average wait time for each process but for SRT the average weight time may or may not minimize based on the number of processes and when they arrive. For example in test case 5 the average weight time for SJF was 507.518 ms and for SRT it was 544.928 ms but for test case 2 and 3 the average weight time for both SJF and SRT was 0.00ms and 12.973ms respectively.

The preemptive SRT algorithm usually has a higher number of context switches. In test case 4 the number of context switches for SJF was 537 and SRT 613. In general the SRT algorithm showed good response times for shorter processes and poorer response times for longer processes because they were preempted by the shorter processes.

4. **(6 points) Describe at least three limitations of your simulation, in particular how the project specifications could be expanded to better model a real-world operating system.**

1) Our simulation only considers that one CPU is in use. Real-world operating systems use multiple CPUs, allowing for parallel processing.

2) Much of the algorithm is based on randomization (determine number of cpu bursts, cpu burst time, io burst time), whereas in a real-world situation certain instructions have a specific number of bursts/burst time.

3) Due to the theoretical nature of the simulation, it is not very scalable to a real CPU, alot of our implementation relies on custom classes that would not be present on a real machine, thus breaking alot of our functionality.

5. **(6 points) Describe a priority scheduling algorithm of your own design (i.e., how could you calculate priority?). What are its advantages and disadvantages?**

Longest Job First: This algorithm does not include preemptions. As a process comes in depending on the tau (predicted CPU burst time) value the order of the ready queue is determined. The higher the predicted CPU burst time the earlier the process is placed in the ready queue. The queue is also reordered whenever a process finishes an IO burst, when it is ready to be placed on the ready queue. Ties are broken via FCFS.

An advantage to this algorithm is that the CPU utilization will be high. And a disadvantage to this algorithm would be that the algorithm would have to wait for all processes to complete to make sure you compare the predicted CPU burst times for each of the processes. Another disadvantage would be that processes with larger CPU burst times might end up facing indefinite blocking or starvation.