

FCFS	seed	average wait time	average turnaround time	average CPU burst time
1 processes	2	0.000 ms	84.312 ms	80.312 ms
2 processes	2	57.459 ms	164.486 ms	103.027 ms
16 processes	2	622.507 ms	710.810 ms	84.304 ms
8 processes	64	3185.499 ms	4092.930 ms	903.431 ms

SJF	seed	average wait time	average turnaround time	average CPU burst time
1 processes	2	0.000 ms	84.312 ms	80.312 ms
2 processes	2	57.459 ms	164.486 ms	103.027 ms
16 processes	2	669.520 ms	757.823 ms	84.304 ms
8 processes	64	2889.805 ms	3797.237 ms	903.431 ms

SRT	seed	average wait time	average turnaround time	average CPU burst time
1 processes	2	0.000 ms	84.312 ms	80.312 ms
2 processes	2	43.162 ms	151.054 ms	103.027 ms
16 processes	2	695.521 ms	785.248 ms	84.304 ms
8 processes	64	2811.782 ms	3720.338 ms	903.431 ms

RR	seed	average wait time	average turnaround time	average CPU burst time
1 processes	2	0.000 ms	84.312 ms	80.312 ms
2 processes	2	46.486 ms	154.378 ms	103.027 ms
16 processes	2	669.814 ms	761.492 ms	84.304 ms
8 processes	64	3253.886 ms	4161.782 ms	903.431 ms

1.

By using data collection and predictions, our team decided to choose the shortest Remaining time(SRT) algorithm as the “best” algorithm out of four. Even with a different seed number, the SRT algorithm performs the best out of the four by having the least average run time. SRT is most efficient for processes to complete its CPU burst and I/O burst by allowing processes to preempt and “sort” based on the situation.

SJF algorithm is the best-suited for CPU-bound processes, it always allows the process with the shortest CPU burst from the ready state to enter running state. Which allows the CPU bound to complete first compare to other algorithms.

FCFS is best-suited for I/O-bound, for the reason of choosing FCFS is that a process always has the chance to enter its I/O burst after finishing its CPU burst. The process under the FCFS scheduling algorithm will never get preempted by other processes, which will lose the chance to enter I/O burst immediately. FCFS ensures the I/O burst for each process completes consistently.

2.

We compared the results of the RR algorithm from END to BEGINNING. We found that most of the average turnaround times for the BEGINNING algorithm are a bit smaller than that for END algorithm. Hence we think the RR algorithm will be better if the rr\_add is BEGINNING.

3.

Since SJF and SRT both rely on the estimated burst time for CPU burst time, the key initial value  $\alpha$  determines the estimate calculation. SRT has a much lower average time around time compared to the SJF in  $\alpha$  range from 0 to 1. The lowest turnaround time appears to happen with the SRT algorithm.

4.

By changing from a non-preemptive algorithm (SJF) to a preemptive algorithm (SRT), the simulation result changes by changing the performance time and starvation problem. The starvation issue occurs when changing from SJF to SRT since there is preemption, a process with huge CPU burst time might result in starvation. Also, the SRT would likely to perform a long time due to the preemption time waste. Relatively, SJF might have a slightly better running performance compare to SRT.

5.

Limitation 1: There is a big difference between the proportion of CPU burst time and IO burst time in our simulation and the proportion of CPU burst time and IO burst time in the real condition. The running speed is much faster than reading and writing speed in the real world. So the IO burst time will be much greater than CPU burst time. Our simulation does not have a good prediction of this proportion. We can ask the user for the proportion of CPU burst time and IO burst time. So that we can make a more accurate prediction.

Limitation 2: We cannot know if there is a process that is idle for too long times. For example, if we use the SJF algorithm and there is a process that needs much more CPU burst time than other processes, this process may be idle for a very long time or it will start running after all other processes have finished running. We can create a list to store the idle time for each processes so we can make the model better.

Zhaoxi Sun<sunz5>, Xinyan Sun<sunx15>, Tongyu Wang<wangt15>, Yueting Liao<liaoy3>

Limitation 3: Our simulation can just simulate a one-core CPU. But in the real world, most CPUs we use are multiple-core CPUs, so the simulation cannot simulate the real condition computer well now. We can change our code to let it simulate multiple-core CPUs.

6. Describe a priority scheduling algorithm of your own design (i.e., how do you calculate priority?). What are its advantages and disadvantages? Support your answer by citing specific simulation results.

Our new scheduling algorithm would be “Longest Burst time algorithm”

Advantages:

-Solve the starvation and ensure the large process complete

-Steady processing the processes to the burst

Disadvantage: process with short burst time would not finish calling first