**Analysis**

1. **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?**

**CPU-bound processes**

FCFS SJF SRT RR

Simulation 1: 1436 1436 1436 1436

Simulation 2: 4014 4014 4117 4132

Simulation 3: 47488 47515 48804 49266

Simulation 4: 754869 756364 756400 757215

Simulation 1: bash$ ./a.out 1 2 0.01 256 4 0.50 64 > output04.txt

Simulation 2: bash$ ./a.out 2 2 0.01 256 4 0.50 64 > output04.txt

Simulation 3: bash$ ./a.out 16 2 0.01 256 4 0.50 64 > output04.txt

Simulation 4: bash$ ./a.out 26 2 0.01 256 4 0.50 64 > output05.txt

Figure 1

**I/O-bound processes**

FCFS SJF SRT RR

Simulation 1: 85658 85658 85658 85658

Simulation 2: 136480 136480 136480 136480

Simulation 3: 629781 629737 630079 630367

Simulation 4: 754869 756364 756400 757215

Simulation 1: bash$ ./a.out 1 2 0.01 256 4 0.50 64 > output04.txt

Simulation 2: bash$ ./a.out 2 2 0.01 256 4 0.50 64 > output04.txt

Simulation 3: bash$ ./a.out 16 2 0.01 256 4 0.50 64 > output04.txt

Simulation 4: bash$ ./a.out 26 2 0.01 256 4 0.50 64 > output05.txt

Figure 2

As shown in Figure 1 FCFS is the best algorithm for CPU-bound processes because it is the fastest time. As shown in Figure 2 FCFS is the best algorithm for I/O-bound processes because it is the fastest time.

1. **For the SJF and SRT algorithms, what value of α produced the “best” results?**

For SJF and SRT the value of α that produces the best results would be when α is approximately 0.5.

1. **For the SJF and SRT algorithms, how does changing from a non-preemptive**

**algorithm to a preemptive algorithm impact your results?**

**Text

Description automatically generated**

**Figure 3**

./a.out 25 2 0.01 256 4 0.5 64 > output.txt

As shown in Figure 3, changing to a preemptive algorithm increases the number of context switches and increases the average turnaround time. The number of context switches increases because every preemption causes a context switch. The average turnaround time increases for SRT because more context switches occur. Figure 3 also shows that changing to a preemptive algorithm increases the average wait time and increases the CPU utilization.

1. **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.**

One limitation of our simulation is that we don’t kill the processes or close any processes. In a real-world operating system, processes could get terminated before they finish all of their bursts.

Another limitation is that our simulation does age any processes. Therefore, our simulation does not have a way to prevent indefinite blocking of processes. A real-world operating system would have aging implemented to prevent indefinite blocking or starvation.

Finally, our implementation is limited by certain variables being constant, such as the time slice for round-robin. Tuning those variables based on the real-time data of the running processes would better model a real-world operating system.

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

CustomPriorityScheduling is a system of queues. It goes as follows:

- Systems Queue is where OS processes are pushed and operates on RR.

- A Queue is where incoming processes are pushed. Top Queue Operates on FCFS, but only for 1 CPU burst. This is to provide and initial prediction of possible CPU burst times for the following bursts, modifiable by following bursts. After a burst, processes can be pushed to B queue if it is a foreground process, or C queue if it is a background process

- B Queue operates on the metric (waiting time + burst time)/ burst time. Higher values mean higher priority, allowing longer jobs to eventually get processed over shorter jobs. Processes from here can be preempted by Systems Queue.

- C Queue also operates on the metric (waiting time + burst time)/ burst time, and apart from being a lower priority than B queue, behaves exactly the same.

CPU prioritizes queue based on the following:  
Systems preempts all other queues, and processes here are evaluated first.  
A queue takes priority over B, C queues.

B, C queues keep the metric, sigma\_((waiting time + burst time)^2)/   
sigma\_((burst time)^2), in order to schedule the next process. Processes are taken from the queue with the higher valued metric