# A3 Part1: Report

https://github.com/auriza-jpg/SYSC4001_A3P1

https://github.com/ibuomi/SYSC4001_


## Brodie Macleod 101302779 (student 2)

## Tamunoibuomi Okari 101287549 (student 1)


This assignment involved implementing a CPU scheduling simulator. This scheduler parses input files containing the arrival times, CPU burst times, PID, IO frequency and duration of each respective process. The algorithms that were implemented were the Round Robin, Efective Priority without preemption and a combo RR and EP algorithm, which used preemption.

When generating test cases for this simulator, the main objective was to observe how each of the algorithms responded to various conditions and scenarios, including scheduling IO-bound, Balanced, and CPU-bound processes. Some other factors that were deemed of interest were how the algorithms responded to varying numbers of processes across these conditions, as well as the order that processes arrived in.

Each of the simulation scenarios (CPU Bound, Balanced and IO bound) were tested with different numbers of processes, as well as different levels of intensity. For example:

**# IO_LIGHT_S**
4, 5, 0, 20, 5, 2
1, 8, 3, 18, 6, 1

Involves “Light” IO-bounded process, and “S” (small) number of processes, compared to

**#IO_HEAVY_L**
9, 1, 0, 40, 6, 2
12, 2, 1, 38, 5, 2
43, 1, 3, 30, 6, 2
23, 4, 5, 36, 5, 2

This involved more process and more demanding IO. An additional two tests per scenario involved “clumped” cases, where all processes arrived at t = 0 and “spread” cases, where processes arrived with a significant delay in between each other. This gives us a far better understanding of how each algorithm performs across a wide range of conditions.

**IO AVERAGES**
interrupts_EP  Throughput=0.0447  Wait=16.73  Turnaround=53.54  Response=6.64

interrupts_RR  Throughput=0.0434  Wait=13.44  Turnaround=50.19  Response=5.76
interrupts_EP_RR  Throughput=0.0435  Wait=14.25  Turnaround=51.00  Response=5.99

**CPU AVERAGES**

interrupts_EP  Throughput=0.0188  Wait=80.28  Turnaround=177.16  Response=25.89
interrupts_RR  Throughput=0.0181  Wait=77.66  Turnaround=174.53  Response=23.98
interrupts_EP_RR  Throughput=0.0183  Wait=81.00  Turnaround=177.88  Response=24.89

**Balanced AVERAGES**
interrupts_EP  Throughput=0.0213  Wait=99.66  Turnaround=195.19  Response=19.07
interrupts_RR  Throughput=0.0212  Wait=98.94  Turnaround=194.47  Response=15.64
interrupts_EP_RR  Throughput=0.0213  Wait=109.28  Turnaround=204.62  Response=18.04

**IO-bound**
Throughput is nearly identical across all three algorithms, with EP showing slightly better results. Round Robin performed the best in the IO-bound test, as it can utilize the CPU efficiently when the ready queue is small due to the frequent IO operations. Most processes complete their IO cycles quickly and return to the CPU in short bursts, or spend more time In IO, efficiently processing those ready to do CPU. EP_RR, while better than EP, is limited by the priority scheduling as processes with lower effective priority can sit in the ready queue longer when higher-priority tasks repeatedly return from IO.

**CPU-Bound**

Throughput is nearly identical, with a slight preference for EP, and overall higher since most of the processes spend their time in the running state. EP's response time suffers more here than in the IO tests, as processes that are approaching their IO time are forced to re-enter the ready queue, causing response delays, especially for lower-priority processes. EP_RR has an advantage over EP, as the RR aspects guarantees 100 ms of CPU time when multiple processes share the same priority, preventing this behavior. But there are also disadvantages. Unlike in the IO tests, EP_RR shows longer wait times, since lower-priority processes may get stuck behind both the EP ordering and long RR cycles among higher-priority. These same effects also causes the slower response and turnaround times.

**Balanced:**
In the balanced case, round robin again performs the best, close to EP in most categories except response time, where it performs noticeably better. Its fixed time quantum prevents delays during longer CPU cycles and ensures that a process can reach its IO time without being preempted. These benefits do not appear in EP_RR, which will preempt processes that are trying to complete longer CPU bursts. This happens more frequently when IO-bound processes keep entering and leaving the ready queue, causing repeated interruptions. As a result, EP_RR shows longer wait times and slower turnaround compared to RR, as in CPU bound tasks. Again,

as in IO, EP_RR still shows slightly better response time due to the RR behaviour with the same priorities.