SYSC4001

Assignment 3 Part 1 Report

Group members:
Mithushan Ravichandramohan 101262467
Dennis Chen 101236818

Simulation Analysis Report

- This document summarises the results of an experimental evaluation of three scheduler implementations included in the repository:

  - EP (priority-based)
  - RR (round-robin)
  - EP+RR (priority with round-robin time-slicing)

We generated 20 distinct input scenarios programmatically (see `scripts/generate_tests.py`) across three workload categories: CPU-bound, I/O-bound and Mixed. For each scenario the three schedulers were executed and their execution logs saved as `executiongen_<id>_<cat>_<sched>.txt`.

The metrics computed from logs are:

- Throughput = terminated processes / simulation time (processes / ms)
- Average Wait Time = mean time spent in READY state per process (ms)
- Average Turnaround = mean(completion_time - arrival_time) (ms)
- Average Response (arrival->first RUNNING) = mean initial response time (ms)
- Average I/O Duration = mean WAITING duration observed (ms)

All raw parsed results are in `results_summary.csv`. Below we summarise aggregated numbers and provide an interpretation of the behaviour observed across schedulers and workload types.

High-level aggregated findings

Across the 60 executions (20 scenarios × 3 schedulers) the three schedulers produced broadly similar overall throughput on average, but they differ in wait time, turnaround and response depending on workload mix.

Key trends:

- CPU-bound workloads produce the worst throughput and highest average wait/turnaround for every scheduler (long CPU bursts dominate the processor).
- I/O-bound workloads produce higher throughput and lower turnaround because processes frequently block on I/O and release the CPU.
- RR and EP+RR improve fairness and initial response for short/interactive jobs, while EP helps meet priority requirements but can increase wait for lower-priority tasks unless fairness mechanisms are present.

Per-category behaviour (summary)

The CSV `results_summary.csv` contains per-run metrics; the scripts aggregated these internally to produce per-scheduler and per-category averages. Representative observations:

- CPU-bound (average across CPU-heavy tests): throughput ≈ 0.029 processes/ms; avg wait and turnaround are substantially larger than in I/O-bound tests (waits in the tens to low hundreds of ms depending on exact burst lengths).
- I/O-bound: throughput ≈ 0.04 processes/ms; avg wait is much lower (often < 25 ms) and average I/O durations observed match the inputs (verifying correct WAITING->READY timing logging).
- Mixed: intermediate values; EP+RR often achieves a better balance by avoiding extremes of starvation while still giving priority preference.

Concrete examples from the generated runs (refer to `results_summary.csv` for exact values):

- `executiongen_6_cpu_*` (CPU-bound): average waits ~90–95 ms and throughputs ≈ 0.025 processes/ms across schedulers.
- `executiongen_8_io_*` (I/O-bound): throughputs ≈ 0.04–0.045 processes/ms and avg waits ≈ 19–24 ms.
- `executiongen_16_mixed_*` (mixed): EP+RR shows a trade-off where time-slicing slightly increases average wait in some runs but reduces tail latencies for long tasks.

Interpretation and why the results match expectations

- Throughput: Strongly influenced by processing-time demand. CPU-bound jobs occupy the CPU for longer, reducing the number of completions per unit time.
- Wait and Turnaround: Queueing discipline matters. RR reduces head-of-line blocking for short jobs, improving their response; EP can reduce response for high-priority jobs at the expense of others.
- I/O Durations: The measured average WAITING durations in the logs match the I/O durations configured in the test inputs. This confirms the simulator correctly records the WAITING → READY transitions and timing.

Practical recommendations

- Use RR or EP+RR for interactive workloads where per-request latency matters. These provide better initial response and fairness for short tasks.
- Use EP (priority) for systems that must favour certain classes (ex: real-time). However, we need to add aging or time-slicing to avoid starvation of lower-priority work.