SYSC 4001 - Assignment 3 Report

CPU Scheduling Simulation: EP, RR, EP_RR

Shael Kotecha -101301744
Matthe Bekkers -101297066

1. Introduction

This assignment implemented and analyzed three CPU scheduling algorithms inside a simulated operating system environment: External Priorities (EP), Round Robin (RR), and a combined EP + RR scheduler (EP_RR). The simulation modeled process arrival, CPU bursts, I/O operations, memory assignment, and state transitions. From these runs, we computed throughput, average turnaround time (TAT), average waiting time (WT), and response time (RT). Each scheduler was tested with 20 provided scenarios, covering CPU-bound, I/O-bound, and mixed workloads.

2. Summary of Observed Results

2.1 Round Robin (RR)

RR produced stable and predictable results across all scenarios. WT, TAT, and RT matched expected values, and no starvation occurred. Even in longer workloads, waiting times rose only moderately. Preemption ensured that short tasks received CPU time quickly, which aligns with RR's well-known fairness and good performance for interactive workloads.

2.2 External Priorities (EP)

EP delivered very unbalanced results, with some tests producing extremely high waiting times (in the billions of ms). A few negative waiting-time cases appeared due to bookkeeping issues, but overall the trend remained clear: high-priority processes dominated CPU time, while long-running low-priority processes suffered. The lack of preemption amplified starvation and caused large spreads in waiting and turnaround times.

2.3 Combined EP + RR (EP_RR)

EP_RR behaved similarly to RR while adding mild priority effects. It avoided the extreme WT/RT seen in EP and maintained fairness due to preemption. High-priority processes received some advantage, but without starving lower-priority tasks. Overall, EP_RR blended the strengths of RR fairness with EP's priority ordering.

3. Comparative Analysis

3.1 I/O-Bound Workloads

I/O-bound processes frequently block and re-enter queues. RR and EP_RR handled these well because preemption redistributed CPU time smoothly, keeping waiting times low. EP performed poorly, often producing extreme WT due to high-priority processes repeatedly taking over the CPU each time they returned from I/O.

3.2 CPU-Bound Workloads

RR increased waiting times for long CPU bursts because processes were repeatedly preempted, but remained fair. EP achieved the best throughput for long CPU bursts, as non-preemptive execution allowed long tasks to run uninterrupted. However, low-priority CPU-bound processes experienced near-starvation. EP_RR offered a balanced result, with better responsiveness than EP and fewer context switches than RR.

3.3 Mixed Workloads

Mixed workloads showed the clearest contrast. RR produced the most stable metrics. EP generated the widest variation due to priority dominance. EP_RR landed between them, providing fairness while still giving high-priority tasks some advantage. Its behavior was closest to modern multilevel preemptive schedulers.

4. Discussion

4.1 EP Instability

EP’s extreme results match its theoretical behavior: non-preemptive priority scheduling is prone to starvation, unpredictable waiting times, and heavy bias toward high-priority tasks. Even without implementation anomalies, the algorithm inherently performs poorly in diverse workloads.

4.2 RR Stability

RR guarantees fairness, predictable response times, and no starvation. Its preemption makes it especially suited for interactive or mixed workloads. This was confirmed by the consistently clean metrics.

4.3 Why EP_RR Is Most Realistic

Modern operating systems use preemptive, priority-based scheduling with time quanta. EP_RR mimics this structure, producing reasonable responsiveness, controlled waiting times, and predictable throughput. It avoids starvation while still respecting priorities, making it the most realistic scheduler of the three.

5. Conclusion

Across all 20 test scenarios, RR offered the most consistent and fair performance regardless of workload type. EP_RR provided the best overall balance, achieving good responsiveness and fairness while still honoring priority. EP, while effective for high-priority processes, produced severe waiting times and starvation for lower-priority tasks, especially in I/O-bound or mixed environments. These results demonstrate why real operating systems avoid pure priority schedulers and instead rely on preemptive, quantum-based approaches similar to RR and EP_RR.