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

EP Results (waiting times fixed)

Test	Proc	Throughput	Avg Turnaround	Avg Waiting (fixed)	Avg Response
1	1	0.1	10	0	0
2	1	0.0909091	11	0	0
3	2	0.133333	11	5	3.5
4	2	0.142857	9.5	2	0
5	3	0.0166667	106.667	56.0	46.6667
6	2	0.0173913	101.5	30	0
7	3	0.0193548	98.3333	40	21.6667
8	1	0.00714286	140	0	0
9	3	0.0193548	105	50	11.6667
10	4	0.0222222	95	55	50

11	2	0.0181818	105	28	6
12	1	0.0047619	210	0	0
13	3	0.04	43.3333	18.3333	18.3333
14	2	0.0148148	120	40	0
15	5	0.025	99	70	59
16	2	0.00952381	155	60	0
17	2	0.0173913	102.5	25	1
18	1	0.025	40	0	0
19	3	0.00847458	258	110	6
20	4	0.0145455	173.75	85	23.75

RR Results

Test	Proc	Throughput	Avg Turnaround	Avg Waiting	Avg Response
1	1	0.1	10	0	0
2	1	0.0909091	11	0	0
3	2	0.133333	11	3.5	3.5
4	2	0.142857	9.5	1.5	0
5	3	0.0166667	106.667	46.6667	46.6667
6	2	0.0165289	98.5	21.5	5
7	3	0.0193548	98.3333	33.6667	21.6667
8	1	0.00714286	140	0	0
9	3	0.0175439	103.667	40.6667	21.6667
10	4	0.0222222	95	50	50
11	2	0.0181818	105	28	6

12	1	0.0047619	210	0	0
13	3	0.04	43.3333	18.3333	18.3333
14	2	0.013986	114	29.5	7.5
15	5	0.025	99	59	59
16	2	0.01	160	55	10
17	2	0.0173913	102.5	25	1
18	1	0.025	40	0	0
19	3	0.00847458	258	107	6
20	4	0.013468	165.5	80	53.75

EP_RR Results

Test	Proc	Throughput	Avg Turnaround	Avg Waiting	Avg Response
1	1	0.1	10	0	0
2	1	0.0909091	11	0	0
3	2	0.133333	11	3.5	3.5
4	2	0.142857	9.5	1.5	0
5	3	0.0166667	106.667	46.6667	46.6667
6	2	0.0173913	101.5	24.5	0
7	3	0.0193548	98.3333	33.6667	21.6667
8	1	0.00714286	140	0	0
9	3	0.0193548	105	42	11.6667
10	4	0.0222222	95	50	50
11	2	0.0181818	105	28	6
12	1	0.0047619	210	0	0

13	3	0.04	43.3333	18.3333	18.3333
14	2	0.0148148	120	35.5	0
15	5	0.025	99	59	59
16	2	0.00952381	155	55	10
17	2	0.0173913	102.5	25	1
18	1	0.025	40	0	0
19	3	0.00847458	258	107	6
20	4	0.0145455	173.75	80	23.75

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. Conclusion

Across all 20 test scenarios, RR consistently offered the fairest and reliable performance, regardless of workload type. EP_RR provided the best overall balance, achieving good responsiveness and fairness while still prioritizing key elements. 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.