REPORT:

SIMULATION ANALYSIS OF IMPLEMENTED SCHEDULER SIMULATOR

PART I - DESIGN AND IMPLEMENTATION OF A SCHEDULER SIMULATOR

Akshavi Baskaran	101302106
Liam Addie	101315124

Repository Link for Part 1: *https://github.com/akshavib/SYSC4001_A3_P1.git*

Lab Section: L4

Group #: 5

Due: December 1st, 2025

OBJECTIVE

This report presents an analysis of the small simulator scheduler for an operating system, with a fixed CPU and fixed memory partitions. The scheduler displays the lifecycle of a process, from ready, to running, to waiting, then terminated. This report includes the analysis of a simulator to study how different scheduling algorithms affect process performance.

IMPLEMENTATION

This simulation was developed by including the interrupts.hpp header file and significantly modifying the interrupts_EP.cpp, interrupts_RR.cpp and interrupts_EP_RR.cpp files to implement an external priorities scheduler, a round robin scheduler, and a scheduler that implements both. Multiple test files (such as test1.txt) were formulated to model diverse process workloads, including varying arrival times, CPU burst durations, and I/O frequency patterns.

For each simulation run, the complete execution timeline, detailing state transitions (New → Ready → Running → Waiting → Terminated), was recorded in output files (such as execution.txt) in an organized chart. This displayed the Time of Transition, the PID, the Old State, and the New State. The simulation logic incorporated a ready_queue to manage process scheduling, a wait_queue to handle I/O delays, and a master job_list to track the overall progress of all processes within the system.

ANALYSIS OF SIMULATION EXECUTION RESULTS

Below is the spreadsheet created from the values produced by calculations.py (can be found in our GitHub repository). These results come from the calculations.py script, which was used to automatically process the raw execution logs from the scheduler programs. The script reads each test's trace file and its matching execution log, picks out all the important events, and calculates the required metrics like turnaround time, wait time, response time, and throughput. It does this for every scenario across all three schedulers (EP, RR, EP_RR), then averages the results and saves everything into one CSV file.

Scheduler	Trace File	Throughput	Avg Turnaround	Avg Wait	Avg Response
EP	test1	0.1	10	0	0
RR	test1	0.1	10	0	0
EP_RR	test1	0.1	10	0	0
EP	test2	0.1	10	0	0
RR	test2	0.1	10	0	0
EP_RR	test2	0.1	10	0	0
EP	test3	0.133333	11	3.5	3.5
RR	test3	0.133333	11	3.5	3.5
EP_RR	test3	0.133333	10	2.5	0
EP	test4	0.142857	9.5	1.5	0
RR	test4	0.153846	9	1	0
EP_RR	test4	0.153846	9	1	0
EP	test5	0.003333	600	300	300
RR	test5	0.003333	800	500	100
EP_RR	test5	0.003333	600	300	300

EP	test6	0.0025	1100	700	700
RR	test6	0.0025	566.67	166.67	100
EP_RR	test6	0.0025	1100	700	700
EP	test7	0.015789	93.33	30	30
RR	test7	0.015789	96.67	33.33	33.33
EP_RR	test7	0.015789	116.67	53.33	0
EP	test8	0.003636	525	25	25
RR	test8	0.003656	522	23.5	25
EP_RR	test8	0.003656	522	23.5	25
EP	test9	0.001724	1055	475	475
RR	test9	0.001887	685	105	25
EP_RR	test9	0.001887	609	30	0
EP	test10	0.00303	585	245	5
RR	test10	0.003333	545	205	5
EP_RR	test10	0.003333	383	45	5
EP	test11	0.001667	1000	400	400
RR	test11	0.001667	800	200	50
EP_RR	test11	0.001667	700	100	0
EP	test12	0.002143	1050	583.33	583.33
RR	test12	0.002143	750	283.33	83.33
EP_RR	test12	0.002143	666.67	200	0
EP	test13	0.002	750	250	250
RR	test13	0.002	950	450	50
EP_RR	test13	0.002	750	250	250
EP	test14	0.001667	1105	480	5
RR	test14	0.001905	800	175	50
EP_RR	test14	0.001905	648	25	5
EP	test15	0.000588	1950	250	250
RR	test15	0.000694	1740.5	50	10
EP_RR	test15	0.000592	1938	238	250
EP	test16	0.002	1750	1250	1250
RR	test16	0.002	2750	2250	250
EP_RR	test16	0.002	1750	1250	1250
EP	test17	0.000579	7728.57	6000	6000
RR	test17	0.000579	11800	10071.43	1928.57
EP_RR	test17	0.000579	7728.57	6000	6000
EP	test18	0.001481	2011.67	1336.67	1336.67
RR	test18	0.001481	753.33	78.33	70
EP_RR	test18	0.001481	2011.67	1336.67	1336.67

EP	test19	0.001765	1350	783.33	783.33
RR	test19	0.001765	1083.33	516.67	416.67
EP_RR	test19	0.001765	1350	783.33	783.33
EP	test20	0.00125	1714	854	834
RR	test20	0.001477	1239	379	122
EP_RR	test20	0.00129	1713.2	853.2	842

Test Case 1: Single Process, No I/O
In this first test case, all the schedulers behaved identically. Their turnaround and throughput times are identical, and perhaps this case was too simple for the schedulers to differ from each other.

Test Case 2: Single Process with I/O
Once again, the schedulers behave identically. Although there was I/O, it seemed that all the scheduling algorithms responded in the same way.

Test Case 3: 2 Processes, No I/O
EP and RR separately have identical results, giving the same throughput and turnaround. However, EP_RR performs better with a lower turnaround time of 10 and lower wait time of 2.5. Much lower response time, even 0, meaning the first process got CPU immediately with little to no delay. Same throughput, meaning it didn’t process more tasks overall, but it handled them more efficiently.

Test Case 4: 2 Processes with I/O
RR and EP_RR performed slightly better than EP in this scenario, in throughput and turnaround time. EP works slowed for this test, most probably because it is a priority-based scheduler. Some processes had to wait longer in EP than they had to for RR and EP_RR before they completed.

Test Case 5: 3 Processes, No I/O, Reversed Priority
The throughput is identical for all schedulers (0.003), so they all completed the same number of processes per unit of time. However the turnaround and wait times were larger for RR (800 and 500) compared to EP and RR (600 and 300). The response time for RR is much lower than EP/EP_RR though, which could mean that RR gives a faster initial response, but higher overall turnaround and wait. Whereas the other two have a reasonable balance.

Test Case 6: 1 Long Process, Short I/O Processes
The throughput is the same for all the schedulers, but EP and EP_RR have a higher turnaround and wait times compared to RR. Even more, the response time for EP and EP_RR (700) is a lot higher than RR (100) showing RR starts processes sooner. In this case, RR gives faster response and completion, while EP and EP_RR perform poorly.

Test Case 7: Random Mix of Arrival Times and Types
EP completes the tasks fastest, and RR is slightly slower, and EP_RR has immediate responses, but increases overall process completion time. This is seen as the response time for EP_RR is 0, but it has the highest turnaround and wait (116.67 and 53.33).

Test Case 8 - 12: Different Memory Sizes
For throughput, RR and EP_RR often match or slightly outperform EP, especially on workloads with many processes, showing they can complete more tasks per unit of time. EP has the highest turnaround overall, RR is in between, and EP_RR is lowest but not by too much. EP_RR also gives the lowest response times, which could mean that EP_RR is the most efficient and responsive when it comes to varying memory sizes. And EP would be the worst.

Test Case 13: Long Processes, No I/O
EP and EP_RR handle the workload efficiently, with low turnaround and wait times (750 and 250). RR's turnaround and wait times were greater (950 and 450). Their response times were also lower than RR, which shows overall that EP and EP_RR was more efficient this round.

Test Case 14: Very Long Processes
Once again, like test case 13, EP and EP_RR's response time was pretty low (5) and RR's was ver high (50). However, the turnaround for EP was higher than RR and EP_RR, and the throughput for EP was lower than RR and EP_RR. This means EP performed the worst, EP_RR handles it most efficiently, and RR improves throughput but has higher response.

Test Case 15: Short Processes, Frequent I/O
Response time is lowest for RR (10), moderate for EP (250), and highest for EP_RR (250), meaning RR starts processes quickly but EP_RR focuses more on maintaining overall efficiency across all processes. The turnaround is also lower for RR, and Throughput is slightly higher for RR (0.000694) than EP (0.000588) and EP_RR (0.000592), meaning RR completes slightly more processes per unit time. RR works best in this case, EP works worst.

Test Case 16: Processes with 80% of their time waiting
With a lot of wait time, EP and EP_RR handle these processes more efficiently. The throughput is the same for all schedulers, and the wait time is the highest for RR (2750) and lower for EP and EP_RR (1750). RR gives fast initial response, but higher total turnaround and wait.

Test Case 17: Many Processes Arriving at Once
In this test, all the processes were in need of different needs. The throughput ended up being the same for all schedulers (0.000579), so they completed the same number of processes per unit of time. Turnaround and wait is highest for RR, and lower for EP and EP_RR. And response time for RR is lower which proves that EP and EP_RR are far more efficient for this type of process.

Test Case 18: Long Process First, Blocking Short I/O
In this test, we expected EP to be the fastest, but it ended up being RR that performed the best. RR was able to run short I/O processes despite the long process first, while EP and EP_RR suffered from long waits and slow completion. RR's response time was lower (70 compared to 1336.67) and turnaround time was much lower as well.

Test Case 19: Preemption Test
In this case, the high priority arrives after the low priority starts. The throughput is same for all schedulers, turnaround and wait is highest for EP and EP_RR, and response time is lowest for RR. RR performed the best.

Test Case 20: Rapid Fire Arrivals
RR handles rapid-fire arrivals best, reducing wait and turnaround while maintaining good responsiveness, whereas EP and EP_RR are slower and more affected by context switch overhead.

Looking at the test cases and their results, RR appears to be best for fast response and I/O-heavy tasks, EP_RR is best for balanced efficiency across mixed workloads, and EP is generally the least efficient except in simple cases.

OVERALL RESULTS FOR EACH SCHEDULER ALGORITHM

The table below represents the overall performance for each scheduler by calculating the total average of the metrics for all 20 tests it simulated.

Scheduler	Average Throughput	Average Wait Time [ms]	Average Turnaround Time [ms]	Average Response Time [ms]
EP	0.02607	1220.40	698.37	661.54
RR	0.02667	1296.08	774.59	166.12
EP_RR	0.02665	1131.29	609.58	587.35

Ideally, the best performing scheduler would be optimized to have maximum throughput, and minimum waiting time, turnaround time, and response time. In this case, the highest (average) throughput between the three scheduler algorithms by a very small amount is RR, followed by EP_RR, then EP. The wait time from lowest to highest is EP_RR, EP, RR, the lowest turnaround time is EP_RR, EP, RR, and finally the lowest response time goes RR, EP_RR, EP. This shows that EP_RR was the best performing and optimized algorithm overall, which makes sense because it has the benefits of both (External) Priority scheduling and Round Robin, and it is also preemptive, unlike EP.

CONCLUSION

After looking at the results of this simulation, the simulation shows that the ideal, best performing algorithm really depends on the type of processes that it is dealing with. While a small difference in timing may be negligible for one workload, it can have a significant impact in another. Overall, EP_RR proves to be the most balanced and efficient scheduler, combining RR's responsiveness with EP's prioritization, making it the most effective choice across a wide range of workloads.