Assignment 3 Group Report
SYSC 4001 A, L4

Justin Kim
101298662

Dorian Bansoodeb
101309988

Monday, December 1st, 2025

Github Repository Link Part 1: https://github.com/Dorianbansoodeb/SYSC4001_A3_P1
Github Repository Link Part 2: https://github.com/kimmerr9/SYSC4001_A3_P2

Objective:
The goal of this assignment is to create a CPU scheduling simulator. The template code given for Assignment 3 was used, and the previous code from Assignment 1 and Assignment 2. All 3 schedulers that were implemented are: EP (External Priorities, non preemptime), RR (Round Robin with 100ms quantum), EP_RR (External Priorities with preemption and 100 ms quantum). The simulator has the same behaviour as it would be in real life such as Process arrival times, CPU burst execution, I/O requests, memory allocations using the fixed partitions, and the process state transitions from New to Running for example. The test cases below are used to compare the scheduling algorithms by using comparable data to observe the average wait time, average response, average turnaround time, throughput, the use of memory (BONUS) , and the I/O responsiveness.

Methodology:
20 test cases were performed for each of the scheduling algorithms. Our team varied each test by changing the arrival times, the cpu burst lengths, the i/o frequency and its duration and the memory requirements of them. Some of the key tests were those that were CPU bound, I/O bound, Mixed Workloads, and tests where memory was not able to be allocated at arrival. The results were tabulated by using a python script.

Test Cases:

EP

Test Case	Number of Processes	Completion Time	Avg Wait Time	Avg Response Time	Avg Turnaround Time	Throughput	Notes
1	3	25	7.000	2.333	17.333	0.12000	
2	4	38	13.500	4.500	24.000	0.10526	
3	3	32	7.000	1.667	18.667	0.09375	High wait time, due to the high I/O
4	4	42	12.250	5.750	25.500	0.09524	
5	3	26	4.667	0.667	15.000	0.11538	Short jobs finished quickly, increasing throughput

6	4	43	14.250	4.250	26.750	0.09302	
7	3	31	10.000	2.000	22.667	0.09677	
8	4	40	11.750	6.000	23.250	0.10000	
9	3	25	5.000	3.333	14.000	0.12000	Frequent I/O interrupts leads to move context switches.
10	3	29	4.333	2.333	13.000	0.10345	Frequent context switches increased the wait time
11	3	32	8.000	1.667	20.667	0.09375	
12	4	36	9.000	3.250	20.500	0.11111	
13	3	33	6.667	1.667	21.000	0.09091	
14	4	40	13.250	2.750	24.750	0.10000	Increased wait time as there is one long cpu bound process
15	3	34	7.667	3.667	21.000	0.08824	
16	4	52	12.750	4.500	29.000	0.07692	All have very long burst times
17	4	41	14.500	6.750	25.750	0.09756	Balanced mix of CPU and I/O
18	3	33	9.000	2.000	22.667	0.09091	
19	4	43	14.750	2.750	27.500	0.09302	A CPU heavy process kept the CPU busy, while frequent I/O from another process lead to many ready state

							transitions
20	4	53	15.750	2.500	30.750	0.07547	Lower I/O leads to less context switches, more CPU utilization

RR

Test Case	Number of Processes	Completion Time	Avg Wait Time	Avg Response Time	Avg Turnaround Time	Throughput	Notes
1	3	25	7.000	2.333	17.333	0.12000	Frequent I/O interrupts cause the processes to go from Running to waiting many times. Leads to more completion time.
2	4	38	13.500	4.500	24.000	0.10526	
3	3	32	7.000	1.667	18.667	0.09375	
4	4	42	12.250	5.750	25.500	0.09524	
5	3	26	4.667	0.667	15.000	0.11538	Rare I/O block. Minimal context switching keeps waiting time low.
6	4	43	14.250	4.250	26.750	0.09302	
7	3	31	10.000	2.000	22.667	0.09677	
8	4	40	11.750	6.000	23.250	0.10000	
9	3	25	5.000	3.333	14.000	0.12000	
10	4	32	6.750	3.000	16.000	0.12500	
11	3	32	8.000	1.667	20.667	0.09375	
12	4	36	9.000	3.250	20.500	0.11111	Mixed

							workloads.
13	3	33	6.667	1.667	21.000	0.09091	
14	4	40	13.250	2.750	24.750	0.10000	
15	3	34	7.667	3.667	21.000	0.08824	Short jobs and staggered arrival times lets the rr to keep the cpu busy with little to no idle times.
16	4	52	12.750	4.500	29.000	0.07692	
17	4	41	14.500	6.750	25.750	0.09756	
18	3	33	9.000	2.000	22.667	0.09091	
19	4	43	14.750	2.750	27.500	0.09302	
20	4	53	15.750	2.500	30.750	0.07547	Heavy I/O process (PID 10), which increases context switches and waiting time.

EP_RR

Test Case	Number of Processes	Completion Time	Avg Wait Time	Avg Response Time	Avg Turnaround Time	Throughput	Notes
1	3	26	6.667	3.667	17.000	0.11538	Short CPU burst with some I/O, low wait time.
2	4	37	11.500	3.250	22.000	0.10811	
3	3	31	7.333	2.000	19.000	0.09677	
4	4	43	12.250	7.500	25.500	0.09302	
5	3	26	4.667	0.667	15.000	0.11538	

6	4	44	14.250	11.750	26.750	0.09091	Mix of both I/O bound and CPu bound processes, high wait time due to repeated I/O.
7	3	32	9.000	1.333	21.667	0.09375	
8	4	44	9.750	9.250	21.250	0.09091	Frequent I/O interrupts, however has short response time as there is priority.
9	3	24	5.333	1.667	14.333	0.12500	
10	3	29	4.333	3.333	13.000	0.10345	Many long jobs, early I/O leads to lower priority processes to regain cpu.
11	3	33	6.667	2.667	19.333	0.09091	
12	4	39	6.750	4.500	18.250	0.10256	
13	3	35	5.667	4.000	20.000	0.08571	
14	4	42	13.750	3.500	25.250	0.09524	CPU heavy jobs. Lower priority jobs are the I/O ones , leading to high wait times as the CPU is taken by higher priority processes.
15	3	32	7.000	4.000	20.333	0.09375	
16	4	52	12.500	3.750	28.750	0.07692	
17	4	42	10.750	6.750	22.000	0.09524	
18	3	34	7.333	5.333	21.000	0.08824	

19	4	41	10.750	5.500	23.500	0.09756	
20	4	54	18.000	6.750	33.000	0.07407	

Scheduler Comparison:
EP vs RR: EP has less context switches (less timeouts), giving more consistent turnaround times for the CPU bound workloads. The RR algorithm allows all processes to run, but there are more timeouts causing more Ready to Running transitions.

In the tables, EP completes around 25-33 time units, while RR completes higher at 31-38, as there are more timeouts. The wait times for RR are also higher.

EP_RR vs EP: The EP_RR helps with responsiveness for higher priority processes as it takes priority over lower ones on quantum expiration. This reduces response time for higher priority jobs but increases the wait time for lower priority processes. The EP has less context switches, which reduces overhead. The EP leads to shorter turnaround times for long cpu jobs.

EP_RR's wait times in the table is greater than the EP's wait times, (12-14 vs 8-11), as lower priority tasks have to wait.

EP_RR vs RR: RR leads to processes having fairness, by giving each an equal timeslice to run. EP_RR takes this a step further by prioritizing jobs with higher priorities. Giving priority to higher priority processes may lead to starvation of the lower priority processes.

In the tables, the completion time of the processes in EP_RR (41-53) are higher than the processes in RR (38-43) as the EP_RR makes the lower priority tasks not run for long as a higher priority task will want to run instead.

Bonus:
Across all schedulers, the memory usage allocation was following a similar pattern as the partitions available were of fixed size. The logs showed that the largest partition which is 40MB was often fragmented, and many processes stayed in the NEW state, as their needed memory size did not fit in the other available partitions. This caused delays in the Ready, Running State transitions as processes are waiting to join the ready queue when memory becomes available. The use of fixed partitions will lead to internal fragmentation, and becomes an issue when small processes are placed inside large partitions which leads to less processes being able to be loaded in memory. The logs showed that the scheduling algorithms do not control memory usage, however the memory limitations from fragmentation from the fixed partitions lead to delays for when processes start, which will extend the total completion.

Conclusion:
This assignment allowed us to simulate three scheduling algorithms (EP,RR, and EP_RR), and we were able to compare their performances by comparing certain fields such as waiting time or

Average Turnaround time. These comparisons allowed us to observe how each algorithm differs from the others and have their own pros and cons. For example, EP has less context switches and benefits CPU bound processes, RR is a fairer algorithm, and EP_RR improves on responsiveness by introducing priority-based preemption. Having fixed memory allocation had an impact on completion time, as processes were stuck in the NEW state as they couldn't be loaded into memory. These simulations showed how these scheduling algorithms behaved with real OS, and we could observe their behaviours and their tradeoffs.