Dec 1 2025
SYSC 4001

SIMULATION REPORT

Group 35
Esteban Heidrich - 101267959
Riley Foxton - 101304022

Github links
P1 https://github.com/rileyfoxton/SYSC4001_A3_P1
P2 https://github.com/Hey1121/SYSC4001_A3P2

Analysis of Testing:

The tests were designed both to ensure the functionality of the program and to see which algorithm performed best under various conditions. The simple tests help see that the program works as designed (ie the slices and IO are correct).

Points to discuss:

Which algorithm performed best overall and why

Overall the EP RR has the most potential since it has the most preemption with a timeout and priority, one can schedule programs so that each program waits very little time, by giving short processes or heavy IO processes higher priority, shortening average wait time and turnaround time . However if one was not able to control when jobs appeared the RR algorithm is beneficial when processes that are shorter than the slice time can appear anywhere in the job list (as the fairness ensures a cap as to when they will be reached). However if shorter jobs routinely appear at the start of a sequence then the EP will perform better.

Which algorithm performs best with IO bound
With IO bound programs EP combined with RR works best because most of the programs have low CPU times. Therefore the fairness of the access to the CPU doesn't matter as much. Also since a process has access to the CPU again after an IO (since it is a preemptive algorithm) the average turnaround time goes down.

Which algorithm performs best with CPU bound
With CPU bound programs external priorities works best, since all the programs have large CPU time fair access to the CPU only increases the average turnaround time, and if there is a program that is vital it would be able to move out of the busy CPU quickly by raising the priority, however because the algorithm does not time share the time spent in the ready queue also increases

Which algorithm performs best with with similar best with similar IO and CPU

With similar IO and CPU round robin works best because it prevents unfair access to the CPU and allows processes that have execution times that are smaller than the slice time to get out of the ready queue fast.

Statistics From Test Cases (comparing the calculated averages)
All three algorithms have similar throughput which is expected as they were all given the same processes that have the same execution times. However on average wait time, turnaround time and response time the RR and EP algorithm performed best. However there are certain situations (detailed in the overall best algorithm section) where the other algorithms will perform better. EP and RR likely performed better because the preemptive but ordered aspect allows for processes to resume their CPU usage as soon as possible after waiting for IO, reducing wait time and turnaround time.

Appendix:

Tests:

Test Case #	Purpose	Test
1	To see a single process works	10, 1, 0, 50, 0, 0
2	To see a single process will be kicked out at end of slice and reinserted	10, 1, 0, 150, 0, 0
3	To ensure IO only run when necessary	10, 1, 5, 50, 100, 10
4	To see a single process works with IO	10, 1, 5, 50, 10, 5
5	To see a single process works with IO and will get kicked out at slice	10, 1, 0, 500, 100, 10
6	To see that multiple processes will run one after another	1, 1, 0, 50, 0, 0 10, 1, 20, 50, 0, 0
7	To see that the EP and RR EP will prioritise the higher priority	10, 1, 0, 50, 0, 0 1, 1, 0, 50, 0, 0
8	To see that the EP and RR will reprioritize the higher priority	10, 1, 0, 150, 0, 0 1, 1, 0, 150, 0, 0
9	To see the EP programs will continue to the higher process	10, 1, 0, 50, 30, 10 1, 1, 0, 50, 30, 10
10	To see the EP programs will continue to the higher process	10, 1, 0, 150, 75, 10

		1, 1, 0, 150, 75, 10
11	To see if the priorities work	5, 1, 0, 150, 0, 0 1, 1, 0, 150, 0, 0 10, 1, 10, 200, 0, 0
12	To see if the priorities work	10, 1, 0, 50, 0, 0 5, 1, 0, 150, 0, 0 1, 1, 0, 200, 0, 0
13	To see if the priorities work with IO	5, 1, 0, 50, 0, 0 10, 1, 0, 150, 50, 50 1, 1, 0, 150, 50, 50
14	To see if there is any strange behaviour if cpu time matches slice time	3,10, 0, 100, 0, 0 10, 15, 0, 100, 0, 0
15	Same as above	3,10, 0, 200, 0, 0
16	Testing memory space freeing	3, 25, 0, 20, 0, 0 10, 40, 0, 125, 0, 0 15, 25, 25, 75, 0, 0
17	Testing memory space assignment behaviour	10, 25, 0, 80, 0, 0 15, 25, 0, 90, 0, 0
18	Testing	15, 40, 0, 90, 0, 0 20, 15, 1, 90, 0, 0
19	Testing for differences between scheduling algorithms	3, 10, 0, 60, 0, 0 7, 12, 0, 90, 0, 0 9, 35, 0, 120, 0, 0 12, 2, 0, 210, 0, 0
20	Same as above	3, 10, 0, 60, 15, 20 7, 12, 0, 90, 15, 20 9, 35, 0, 120, 15, 20 12, 2, 0, 210, 15, 20

Test Results:
Thru Put = Throughput (# of processes able to be done per 100ms)
Avg wait time = Avg wait time (ms)
Avg turnrnd time = Avg turnaround time (ms)
Avg response time = Avg response time (ms)

Test #	RR				EP				RR and EP			
	Thru put	Avg wait time	Avg turnrnd time	Avg respse time	Thru put	Avg wait time	Avg turnrnd time	Avg respse time	Thru put	Avg wait time	Avg turnrnd time	Avg respse time
1	2.00	0	50	0	2.00	0	50	0	2.00	0	50	0
2	0.67	0	150	0	0.67	0	150	0	0.67	0	150	0
3	1.82	0	50	0	1.82	0	50	0	1.82	0	50	0
4	1.33	0	70	5	1.33	0	70	5	1.33	0	70	5
5	0.19	0	540	10	0.19	0	540	10	0.19	0	540	10
6	2.00	15	65	0	2.00	15	65	0	2.00	15	65	0
7	2.00	25	75	0	2.00	25	75	0	2.00	25	75	0
8	0.67	125	275	0	0.67	75	225	0	0.67	75	225	0
9	2.00	30	90	25	2.00	30	90	25	1.81	25	85	10
10	0.67	102.5	262.5	75	0.67	102.5	262.5	75	0.64	75	235	10
11	0.60	247	414	0	0.60	147	317	0	0.60	147	317	0
12	0.75	116	250	0	0.75	183	317	0	0.75	183	317	0
13	0.83	50	233	50	0.83	66	250	50	0.86	66	250	50
14	1.00	50	150	50	1.00	50	150	50	1.00	50	150	50
15	0.50	0	200	0	0.50	0	200	0	0.50	0	200	0
16	1.36	38.3	136.7	63.3	1.36	46.7	120	46.7	1.36	46.7	120	46.7
17	1.18	40	125	40	1.18	40	125	40	1.18	40	125	40
18	1.11	44.5	134.5	44.5	1.11	44.5	134.5	44.5	1.11	44.5	134.5	44.5
19	0.83	115	265	145	0.83	120	240	120	0.83	120	240	120
20	0.80	22.5	375	125	0.69	65	350	100	.69	53.8	336.3	86.3

AVG	1.1	51.0	195.6	31.6	1.1	50.5	189.1	28.3	1.1	48.3	186.7	23.6