

1. Introduction

This lab is done in the scope of the Operating System class at the Université of Neuchâtel, for the master in informatics. This is the second lab and the third assignment that proposes experiments focused on scheduling algorithms.

Extract of the lab:

During this practical, your goal is to experiment with different scheduling algorithms under different workloads. You should submit a short report containing your results. For each of the tasks, the report shall contain the following items:

- **A table with the performance summary of each algorithm** (CPU usage, throughput, turn around time, waiting time); a template using Excel is provided on the Moodle course page.
- **A concise description and analysis of the results** you obtained, e.g., performance of the various algorithms in relation to the task.

*The **Gantt charts of the experiments may be used** inside your report to detail your explanation. Apart from the report, you should also submit the modified run and exp files for each task. Those files shall be named according to the task number N, e.g., N.exp.*

This document should be joined with the excel table that summarize the data for all the runs. Nevertheless, the full data table is always provided with the explanation as a basis of hypothesis. Furthermore, this document should be accompanied by the run and experiment files used for the experiments.

1.1. Table of illustrations

Figure 1 – Run files for the experiments	4
Figure 2 – Experiment files	4
Figure 3 – Example of experiment file.....	4
Figure 4 – interactive experiment execution	5
Figure 5 – Table data for the interactive algorithm	5
Figure 6 – Gantt diagrams for the 4 algorithms during the interactive experiment.....	5
Figure 7 – CPU intensive experiment execution	6
Figure 8 – Table data for the CPU intensive algorithm	6
Figure 9 – Gantt diagrams for the 4 algorithms during the CPU intensive experiment.....	7
Figure 10 – mixed experiment execution.....	8
Figure 11 – Table data for the mixed algorithm.....	8
Figure 12 – Gantt diagrams for the 4 algorithms during the mixed experiment	9
Figure 13 – experiments for the mixed workloads	10
Figure 14 – run files for the mixed workloads.....	10
Figure 15 – (interactive // CPU) experiment execution	10
Figure 16 - Table data for the (interactive // CPU) experiment	11
Figure 17 - Gantt diagrams for the 4 algorithms during the (interactive // CPU) experiment	11
Figure 18 – (CPU // Mixed) experiment execution	12
Figure 19 - Table data for the (CPU // Mixed) experiment	12
Figure 20 - Gantt diagrams for the 4 algorithms during the (interactive // CPU) experiment	13

Figure 21 – (Mixed // interactive) experiment execution	13
Figure 22 - Table data for the (Mixed // interactive) experiment.....	14
Figure 23 - Gantt diagrams for the 4 algorithms during the (CPU // Mixed) experiment.....	14
Figure 24 – experiments for the varying RR	16
Figure 25 – RR varying time slice analysis - interactive experiment execution	16
Figure 26 - RR varying time slice analysis - Table data for the interactive algorithm	16
Figure 27 - RR varying time slice analysis - Gantt diagrams for the algorithms during the interactive experiment	17
Figure 28 – RR varying time slice analysis - CPU intensive experiment execution	17
Figure 29 – RR varying time slice analysis - Table data for the CPU intensive algorithm	18
Figure 30 – RR varying time slice analysis - Gantt diagrams for the 4 algorithm during the CPU intensive experiment.....	18
Figure 31 – RR varying time slice analysis - mixed experiment execution	19
Figure 32 – RR varying time slice analysis - Table data for the mixed algorithm	19
Figure 33 – RR varying time slice analysis - Table data for the mixed algorithm	20
Figure 34 – run files for the context switch overhead test	21
Figure 35 – experiments for the context switch overhead test	21
Figure 36 – Difference between the execution time without and with overhead during context switching.....	21
Figure 37 – Difference between the data tables without and with overhead during context switching	22
Figure 38 – Gantt diagrams with overhead during context switching	23
Figure 39 – Zoomed-in Gantt diagram with overhead during context switching with time slice = 0.1	23
Figure 40 – Zoomed-in Gantt diagram with overhead during context switching with time slice = 10	24
Figure 41 – runs for the high load performance	25
Figure 42 – experiments for the high load performance	25
Figure 43 – High load interactive experiment execution	25
Figure 44 – Table data for the High load interactive algorithm	26
Figure 45 –Gantt diagrams for the 4 algorithms during the High load interactive experiment	26
Figure 46 – High load CPU intensive experiment execution	27
Figure 47 –Table data for the High load CPU intensive algorithm	27
Figure 48 – Gantt diagrams for the 4 algorithms during the High load CPU intensive experiment	28
Figure 49 – High load mixed experiment execution.....	28
Figure 50 – Table data for the high load mixed algorithm	29
Figure 51 – Gantt diagrams for the 4 algorithms during the high load mixed experiment	29

Table des matières

1. Introduction.....	1
1.1. Table of illustrations.....	1
2. Individual workloads	4
2.1. Interactive	4
2.2. CPU intensive.....	6
2.3. Mixed workload.....	7
3. Mixing workloads	10
3.1. Interactive – CPU	10
3.2. CPU – Mixed	12
3.3. Mixed – interactive.....	13
4. Varying round-robin time slice	16
4.1. Interactive	16
4.2. CPU intensive.....	17
4.3. Mixed workload.....	18
4.4. Context switch overhead.....	21
5. High load performance.....	25
5.1. Interactive	25
5.2. CPU intensive.....	27
5.3. Mixed workload.....	28
6. Conclusion	31

2. Individual workloads

Three run files are created: one for each experiment.




 cputensive.run	07.04.2022 13:43	Fichier RUN	1 Ko
 interactive.run	07.04.2022 13:40	Fichier RUN	1 Ko
 mixedload.run	07.04.2022 13:43	Fichier RUN	1 Ko

Figure 1 – Run files for the experiments

Each run file corresponds to one of the experiment's setups. These run files are used with different algorithm throughout the lab. The parameters are set up according to the part 2.1 of the lab subject, and the run files are joined in annex of this document.

For this experiment, the three runs are executed (interactive, CPU intensive, mixed workload) with the four algorithms (FCFS, RR-1, SJF, SJFP-0.5). Each experiment runs one setup under all the scheduling algorithm - this provides a good basis of comparison between the algorithms on the same context.




 workloads_cputensive.exp	07.04.2022 13:48	Fichier EXP	1 Ko
 workloads_interactive.exp	07.04.2022 13:47	Fichier EXP	1 Ko
 workloads_mixed.exp	07.04.2022 13:48	Fichier EXP	1 Ko

Figure 2 – Experiment files

The runs are separated in three experiment files, one for each experiment setup.

```
name workloads_cputensive
comment This experiment contains 4 runs of the cpu intensive experiment with the algorithms

run cputensive algorithm FCFS key "FCFS"
run cputensive algorithm RR 1 key "RR"
run cputensive algorithm SJF key "SJF"
run cputensive algorithm SJFA 0.5 key "SJFA"
```

Figure 3 – Example of experiment file

Each experiment file runs the same setup under each of the algorithms. This method is used for all the tests of this lab.

2.1. Interactive

The first experiment setup is the interactive one – it simulates a GUI-like software that mostly waits and received occasional burst of interactions.

```

Starting experiment of 4 runs from workloads_interactive
Experimental Run for interactive
  151 events done in 42 milliseconds
Experimental Run for interactive
  151 events done in 27 milliseconds
Experimental Run for interactive
  151 events done in 31 milliseconds
Experimental Run for interactive
  151 events done in 30 milliseconds

```

Figure 4 – interactive experiment execution

The interactive experiment is executed on the same run for the four algorithms. The execution time (respectively FCFS, RR, SJF, SJFA) is not very variable between the four, with a difference of ~10 milliseconds between the slowest (FCFS) and the faster (RR).

Table Data												
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	Entries		Average Time	
									CPU	I/O	CPU	I/O
interactive_3	FCFS	38.83	5	5	.643870	.128774	0.00	.26	50	45	.50	3.23
interactive_4	RR	38.83	5	5	.643870	.128774	0.00	.26	50	45	.50	3.23
interactive_5	SJF	38.83	5	5	.643870	.128774	0.00	.26	50	45	.50	3.23
interactive_6	SJFA	38.83	5	5	.643870	.128774	0.00	.26	50	45	.50	3.23

Name	Key	Turnaround Time				Waiting Time			
		Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
interactive_3	FCFS	36.07	31.60	38.83	2.58	1.98	.89	3.21	.17
interactive_4	RR	36.07	31.60	38.83	2.58	1.98	.89	3.21	.17
interactive_5	SJF	36.07	31.60	38.83	2.58	1.98	.89	3.21	.17
interactive_6	SJFA	36.07	31.60	38.83	2.58	1.98	.89	3.21	.17

Done

Figure 5 – Table data for the interactive algorithm

A more thorough analysis of the table shows that the results are the same for all the algorithms (including – but not only - CPU utilization, throughput, response times and waiting times).

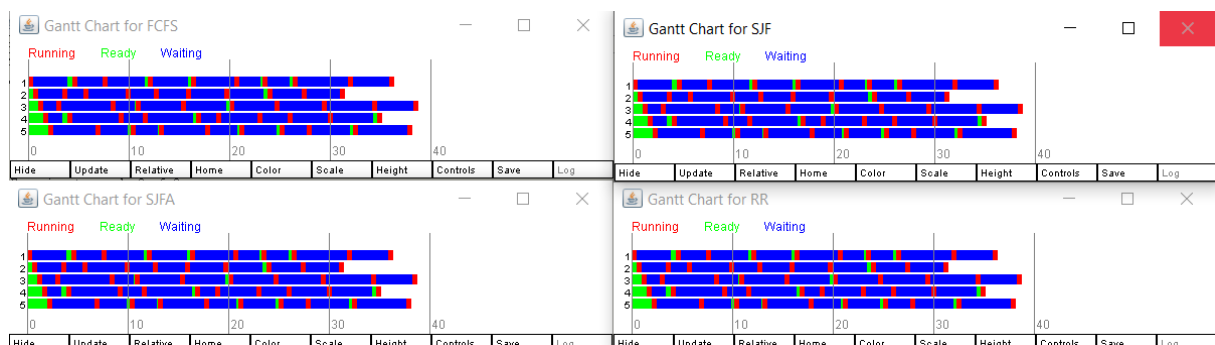


Figure 6 – Gantt diagrams for the 4 algorithms during the interactive experiment

The Gantt diagrams show that a fully interactive-focused software is noticeably light on the CPU. As the processes request very little CPU time, they are mostly in the “waiting” state during their lives. When they switch to ready, they are immediately processed by the idle CPU. This seed (42) did not

bring any conflict in the processes, meaning they did not in fact compete for CPU time. This first experiment shows that there is no need for a scheduling algorithm if there is no conflict of competition for resource (CPU time).

2.2. CPU intensive

The second part of this section is focused on a CPU-intensive task. The first experiment proved that competition was at the core of the need for a scheduling algorithm – it is expected that intensive CPU tasks will bring competition for CPU time.

```
Starting experiment of 4 runs from workloads_cpuintensive
Experimental Run for cpuintensive
61 events done in 14 milliseconds
Experimental Run for cpuintensive
61 events done in 25 milliseconds
Experimental Run for cpuintensive
421 events done in 88 milliseconds
Experimental Run for cpuintensive
61 events done in 24 milliseconds
```

Figure 7 – CPU intensive experiment execution

The execution times are more spread-out between the processes (Warning: the algorithms were badly ordered, they are respectively FCFS, SJF, RR, SJFA). The fastest scheduling algorithm, first come first serve, executed the tasks in 14ms meanwhile the slowest, Round Robin, completed the task in 88milliseconds – about less than eight times the value. Multiple tests have been re-produced (as another execution on the machine could skew the test in one of the algorithm favor: the multiple runs have produced similar results, with ~<20ms for the FCFS and ~80ms for the SJF.

Table Data										Entries				Average Time			
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA		CPU	I/O	CPU	I/O				
cpuintensive_5	FCFS	200.00	5	5	1.000000	.025000	0.00	3.40		20	15	10.00	1.35				
cpuintensive_6	SJF	200.00	5	5	1.000000	.025000	0.00	3.40		20	15	10.00	1.35				
cpuintensive_7	RR	200.00	5	5	1.000000	.025000	0.00	3.82		200	15	1.00	1.35				
cpuintensive_8	SJFA	200.00	5	5	1.000000	.025000	0.00	3.40		20	15	10.00	1.35				

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
cpuintensive_5	FCFS	180.00	160.00	200.00	14.14	135.94	116.18	156.08	2.79
cpuintensive_6	SJF	180.00	160.00	200.00	14.14	135.94	116.18	156.08	2.79
cpuintensive_7	RR	197.00	193.00	200.00	2.61	152.94	149.18	155.08	.42
cpuintensive_8	SJFA	180.00	160.00	200.00	14.14	135.94	116.18	156.08	2.79

Figure 8 – Table data for the CPU intensive algorithm

The data table shows a CPU utilization of one and a throughput of 0.025 in all cases. The algorithms use the CPU in the same manner and no capacity seems to be lost. The number of entries increases very quickly for the RR algorithm (10x more than the other algorithms), with an average CPU time logically divided by the same coefficient.

However, the turnaround and waiting time are not the same:

- For the turnaround, the maximums are the same at 200, but the Round Robin has a greater minimum and average turnaround time.
- For the waiting time, the maximums are also the same at 15, but the average and minimum is greater for the round robin

A higher turnaround time means the processes have taken longer on minimum and on average to be executed. A higher waiting time means the processes have waited longer in the queue on average. This seems to indicate that the processes might have waited more in the RR.

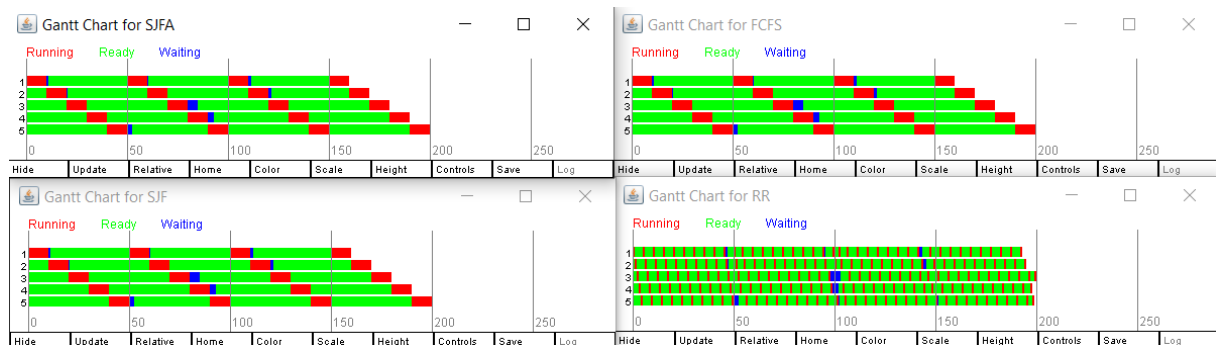


Figure 9 – Gantt diagrams for the 4 algorithms during the CPU intensive experiment

The Gantt diagram shows that the first come first serve algorithm and both SJF and SJA are almost identical. Indeed, the SJFA tries to improve the SJF by providing a better estimation of the next CPU burst. As the CPU-intensive tasks are exactly predictable, there is no distinction between them. Similarly, as there is no distinction between the CPU burst estimation, the SJF(a) have the same behavior as the FCFS with one difference: they will allocate the same CPU time to each process one by one, but there is no guarantee of order (this will depend on the resolution of equality inside the SJF algorithms).

The round robin algorithm tries to allocate the same amount of time by switching between all five processes. As is clearly visible in the Gantt diagram, this means that:

- all algorithms will finish at the same time. The allocation is more equal between the tasks, but this means some indicators may go down (as the first process takes almost the same time as the last process to execute). This explains the turnaround time.
- As the scheduler assign time for Process 1-2-3-4-5-1-2-3-4-5-....., there can be more waiting time: when the time is for process 5 to be executed and this particular process is in the waiting state and not the ready one, then this CPU time is wasted, and no other process take the place.

This waiting happened a few times in this experiment and explains the difference in execution time (a process is ready but has to wait its turn in the queue). This is also bound to the very little Q value of the RR algorithm: on the other schedulers, the execution time allocated is significant and the waiting times did not block other processes. If the Q value was larger, then the round robin would have probably avoided most of those waiting times. Nonetheless, this shows a weakness of the RR (What happens when an allocated process is not in the ready state ?).

2.3. Mixed workload

Lastly, the third part of this section is a test on a mixed workload of CPU tasks and interactions.

```

Starting experiment of 4 runs from workloads_mixed
Experimental Run for mixedload
  112 events done in 40 milliseconds
Experimental Run for mixedload
  276 events done in 54 milliseconds
Experimental Run for mixedload
  112 events done in 41 milliseconds
Experimental Run for mixedload
  112 events done in 38 milliseconds

```

Figure 10 – mixed experiment execution

The algorithms (Respectively FCFS / RR / SJF / SJFA) are close but not identical in execution time. The FCFS is generally the fastest, SJF-type algorithms are very close behind by a few milliseconds, and the RR trails behind with +~10ms.

Table Data

Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	Entries		Average Time	
									CPU	I/O	CPU	I/O
mixedload_9	FCFS	101.32	5	5	.986927	.049346	0.00	2.65	37	32	2.70	3.08
mixedload_10	RR	101.88	5	5	.981578	.049079	0.00	2.88	119	32	.84	3.08
mixedload_11	SJF	104.06	5	5	.960961	.048048	0.00	2.06	37	32	2.70	3.08
mixedload_12	SJFA	100.49	5	5	.995121	.049756	0.00	2.51	37	32	2.70	3.08

Name	Key	Turnaround Time				Waiting Time			
		Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
mixedload_9	FCFS	93.49	75.70	101.32	9.20	53.80	38.75	59.46	1.54
mixedload_10	RR	98.41	95.63	101.88	2.61	58.72	55.92	62.03	.41
mixedload_11	SJF	82.63	60.37	104.06	17.14	42.94	20.27	64.04	3.56
mixedload_12	SJFA	90.14	73.36	100.49	11.26	50.45	33.26	63.54	2.50

Done

Figure 11 – Table data for the mixed algorithm

The data table shows a difference in CPU utilization with the SJA being the better CPU allocator, and the RR being the worst one. The throughput is also won by the SJA followed by the FCFS before the SJF. The RR generates a lot of CPU entries (119 vs 37) of little time (0.84 vs 2.70 average).

The smallest max turnaround time is provided by the SJA while retaining a good average. The SJF is surprisingly the worst one on maximum turnaround time, but also has the smallest average and minimum turnaround (showing a sign of imbalance). We can note that although the RR is the slowest algorithm, its maximum turnaround time is not the worst (but its average is the highest by far).

The average waiting time is higher for the RR than for all the other algorithms, with the SJF has the smallest one. Once again, the SJF retains the extremes with the slowest minimum and highest maximum waiting times.

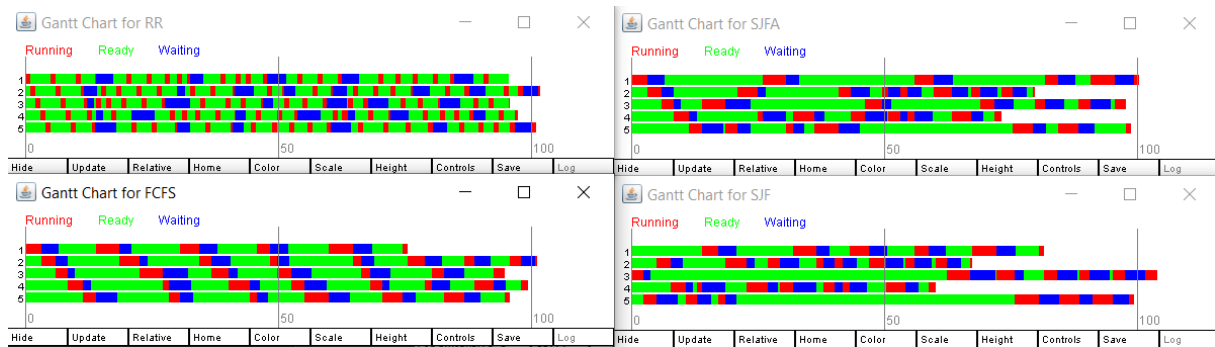


Figure 12 – Gantt diagrams for the 4 algorithms during the mixed experiment

The FCFS unsurprisingly allocates time for process 1,2,3,4, 5,... In order. As the processes are taken as soon as they are ready, this avoids some waiting time, although we could see more convoy problems if, typically, interactive tasks were more mixed and numerous between long CPU intensive tasks.

The SJFA is similar to the FCFS, but we see that the interactive jobs are prioritized at the end of the experiment with short jobs executed before long tasks. This means some processes are executed very soon meanwhile the CPU tasks are delayed. Although this does not bring a lot of improvement over the numbers themselves (see data table comments), this could prove especially important for usability of the system where interactions must often be processed in real time meanwhile background computations can be delayed. The Gantt diagram shows the great variation between the end of execution time between the processes.

The SJFA still favors interactive tasks but does so more efficiently: we see the tasks end-of-execution time is closer. The SJFA was not able to bring improvement during the CPU intensive workload because they were equal, this time, it helps estimate better when the processes change between interactive and CPU intensive tasks.

Lastly, the RR suffers the same problems as before: some processes are waiting while called by the scheduler and lose CPU time. Instead of allocating resources to the CPU processes when the interactive ones are waiting, it still give them their fair share: all the processes finish almost at the same time, but the wasted CPU time has a cost on the total execution time.

3. Mixing workloads

This section reports the results of mixing the workloads to observe this effect on the different algorithms.




 mixing_cpu_mixed.run	07.04.2022 16:34	Fichier RUN	1 Ko
 mixing_inter_cpu.run	07.04.2022 16:50	Fichier RUN	1 Ko
 mixing_mixed_inter.run	07.04.2022 16:31	Fichier RUN	1 Ko

Figure 13 – experiments for the mixed workloads

Three new runs start two set of processes in parallel, the two sets being taken from the first runs.




 mixing_cpu_mixed.exp	07.04.2022 16:52	Fichier EXP	1 Ko
 mixing_inter_cpu.exp	07.04.2022 16:52	Fichier EXP	1 Ko
 mixing_mixed_inter.exp	07.04.2022 16:52	Fichier EXP	1 Ko

Figure 14 – run files for the mixed workloads

A set of three new experiment files are used to start the three runs with the four scheduling algorithm each time.

3.1. Interactive – CPU

The first part of this section mixes the two opposite setups: Interactive (mostly waiting) with CPU intensive (mostly computing with bursts of IO).

```
Starting experiment of 4 runs from mixing_inter_cpu
Experimental Run for mixing_inter_cpu
211 events done in 43 milliseconds
Experimental Run for mixing_inter_cpu
211 events done in 74 milliseconds
Experimental Run for mixing_inter_cpu
571 events done in 91 milliseconds
Experimental Run for mixing_inter_cpu
211 events done in 44 milliseconds
```

Figure 15 – (interactive // CPU) experiment execution

The algorithms correspond respectively to FCFS, SJF, RR, SJFA. The FCFS and the SJFA are very close in performance (~40ms) while the SJF stands at ~70ms, and the RR is yet again the last one at ~90ms.

Table Data

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
mixing_inter_cpu_17	FCFS	234.40	10	10	.959912	.042663	0.00	7.36	70	60	3.21	2.61
mixing_inter_cpu_18	SJF	225.00	10	10	1.000000	.044444	0.00	5.39	70	60	3.21	2.61
mixing_inter_cpu_19	RR	225.00	10	10	1.000000	.044444	0.00	5.21	250	60	.90	2.61
mixing_inter_cpu_20	SJFA	225.00	10	10	1.000000	.044444	0.00	6.28	70	60	3.21	2.61

Name	Key	Turnaround Time				Waiting Time			
		Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
mixing_inter_cpu_17	FCFS	210.72	170.00	234.40	23.04	172.58	128.05	198.26	2.67
mixing_inter_cpu_18	SJF	159.50	113.00	225.00	46.59	121.36	77.71	182.86	4.26
mixing_inter_cpu_19	RR	155.30	83.00	225.00	67.36	117.16	50.71	181.86	6.33
mixing_inter_cpu_20	SJFA	179.50	153.00	225.00	27.40	141.36	117.71	182.86	2.37

Done

Figure 16 - Table data for the (interactive // CPU) experiment

The data shows a good CPU utilization of all algorithms, with the FCFS having some flaws visible in the throughput. The RR has once again the most entries and smallest average time.

The FCFS algorithm is very bad on turnaround time: it has the worst average, the biggest minimum and the biggest maximum. The RR is the best in the turnaround by all metrics (smallest max, smallest min, smallest average).

The FCFS is also bad on waiting time meanwhile the RR is the best one.



Figure 17 - Gantt diagrams for the 4 algorithms during the (interactive // CPU) experiment

The Gantt diagram show that the FCFS algorithm, although not bad in execution time, tries to allocate CPU time to each process that request it in order. This cause the interactive processes to be delayed by the intense CPU processes. This would be terrible if those processes were 5 GUI processes running with five background computation processes.

The RR algorithm once again gives the same amount of resources to all processes – thus the interactive processes are finished somewhat rapidly and at the same time. It seems to lessen the waiting time for the interactions, at the cost of waiting time for the processes that are only able to execute fraction of their work at a time. This waiting time would be even more terrible if overhead were present (see 4.4). The round robin of one seems to lessen the waiting on the interactive process

switches whereas a higher time slice would be better for the intensive computations (at the cost of less responsivity for the interactions).

The SJF and SKF-A both prioritize correctly the interactions while according to a sufficient amount of time to the computations. The SJF executes all the processes of interaction between each process of intensive computation. The SJFA does the same after the first execution – the first batch of computation is done before the second batch of interactions. This is due to the different estimation that is less abrupt and takes more time to calibrate.


3.2. CPU – Mixed

The second part of this section consist in mixing the CPU intensive tasks with the mixed workloads. This makes tasks with some interactions and lot of intensive CPU usage.

```
Starting experiment of 4 runs from mixing_cpu_mixed
Experimental Run for mixing_cpu_mixed
163 events done in 41 milliseconds
Experimental Run for mixing_cpu_mixed
163 events done in 40 milliseconds
Experimental Run for mixing_cpu_mixed
689 events done in 117 milliseconds
Experimental Run for mixing_cpu_mixed
163 events done in 40 milliseconds
```

Figure 18 – (CPU // Mixed) experiment execution

The algorithms (respectively FCFS, SJF, RR, SJFA) are similar in execution time except for the slowest round robin (40 to 120ms).

 Table Data — □ ×

Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	Entries		Average Time	
									CPU	I/O	CPU	I/O
mixing_cpu_mixed_13	FCFS	300.00	10	10	1.000000	.033333	0.00	7.44	54	44	5.56	2.03
mixing_cpu_mixed_14	SJF	300.00	10	10	1.000000	.033333	0.00	4.81	54	44	5.56	2.03
mixing_cpu_mixed_15	RR	300.00	10	10	1.000000	.033333	0.00	7.30	317	44	.95	2.03
mixing_cpu_mixed_16	SJFA	300.00	10	10	1.000000	.033333	0.00	5.80	54	44	5.56	2.03

Name	Key	Turnaround Time				Waiting Time			
		Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
mixing_cpu_mixed_13	FCFS	262.19	209.86	300.00	33.94	223.25	166.04	263.94	3.88
mixing_cpu_mixed_14	SJF	183.37	60.65	300.00	99.16	144.44	30.39	256.08	9.40
mixing_cpu_mixed_15	RR	258.07	204.83	300.00	40.44	219.13	174.58	256.02	3.52
mixing_cpu_mixed_16	SJFA	213.02	124.84	300.00	68.62	174.08	88.79	256.08	6.37

Done

Figure 19 - Table data for the (CPU // Mixed) experiment

There is no difference in CPU utilization or throughput between the algorithms. The turnaround time are the same maximums, with the SJF once again having the least average / minimum and the FCFS having the worst.

The waiting time is in favor of both SJF and SJFA, with the RR having the worst minimum waiting time and the FCFS having the worst average.



Figure 20 - Gantt diagrams for the 4 algorithms during the (interactive // CPU) experiment

The Gantt diagrams show that the round robin has less problems with “mostly CPU intensive tasks” as they have little interaction, and thus an equal allocation of resources is not problematic. These processes also have less time waiting for input, which is a real problem with RR as this time will be a direct loss on the CPU.

The FCFS once again finished the CPU intensive task first as they take very long batches of CPU time. This also means that the mixed processes suffer for more waiting time.

The SJF and SJFA prioritize once again the interactions over the intense processes if given the choice. The SJFA seems to stabilize the allocation between the mixed processes, meanwhile the mixed processes end of execution varies more widely in the shortest job first.

3.3. Mixed – interactive

The third and last part of this section is an experiment mixing the mixed workload with the interaction – this time, some CPU tasks will occur, but the focus will be on short interactions.

```
Starting experiment of 4 runs from mixing_mixed_inter
Experimental Run for mixing_mixed_inter
  262 events done in 49 milliseconds
Experimental Run for mixing_mixed_inter
  262 events done in 55 milliseconds
Experimental Run for mixing_mixed_inter
  426 events done in 87 milliseconds
Experimental Run for mixing_mixed_inter
  262 events done in 56 milliseconds
```

Figure 21 – (Mixed // interactive) experiment execution

The algorithm (respectively FCFS, SJF, RR, SJFA) are close with the RR once again lagging being in execution time.

Table Data

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
mixing_mixed_inter_21	FCFS	126.88	10	10	.985158	.078813	0.00	6.59	87	77	1.44	2.82
mixing_mixed_inter_22	SJF	127.26	10	10	.982270	.078582	0.00	3.51	87	77	1.44	2.82
mixing_mixed_inter_23	RR	127.32	10	10	.981766	.078541	0.00	5.15	169	77	.74	2.82
mixing_mixed_inter_24	SJFA	125.49	10	10	.996093	.079687	0.00	4.52	87	77	1.44	2.82

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
mixing_mixed_inter_21	FCFS	117.75	89.20	126.88	10.93	83.57	52.25	97.76	1.45
mixing_mixed_inter_22	SJF	78.85	45.37	127.26	32.75	44.67	14.30	88.18	2.78
mixing_mixed_inter_23	RR	99.80	72.94	127.32	23.71	65.62	45.37	85.00	1.81
mixing_mixed_inter_24	SJFA	90.90	64.20	125.49	25.58	56.73	37.14	88.54	2.07

Done

Figure 22 - Table data for the (Mixed // interactive) experiment

The CPU utilization is not perfect as there will inevitably be waiting time as the experiment is focused on interactions. The SJFA is the best user of CPU time and has the best throughput.

The turnaround time, on average, is best on SJF / SJFA, followed by RR and FCFS. The maximum turnaround is globally similar for all the algorithms, meaning all processes take about the same amount of time to be executed.

The waiting times are the worst on the FCFS, while being the smallest ones on the SJF / SJFA.



Figure 23 - Gantt diagrams for the 4 algorithms during the (CPU // Mixed) experiment

The Gantt diagrams show that the FCFS once again does not prioritize interactions and the mixed task / interactions tasks have a similar end-of-execution time. The Round robin tries to allocate regularly the CPU to processes that go / are into waiting status.

The SJF and SJFA prioritize the interactions once more. The SJFA takes some initial calibration before being fully efficient and starts by allocating the same resources to mixed and interactions. Once the calibration is over, all the interaction processes receive a good and balanced amount of CPU time: the SJFA is again more stable than the SJF.

These experiments show us that FCFS / RR are more adapted to intensive CPU tasks, meanwhile shortest job first offer a level of prioritization that avoids the convoy effect. The SJFA is way more balanced than the SJF but takes time to calibrate at first. The RR is consistently the slowest scheduling algorithm, although this could be caused / tweaked by the time slice and worsened by the context switch time. A challenge of these algorithms seems to be the mixing of CPU-intensive tasks wanting big chunks of CPU time and fully using them and interaction-focused tasks wanting little chunks of CPU time very quickly and going back into waiting mode.

4. Varying round-robin time slice

The fourth section of this lab is focused on looking at the differences that tweaking the round robin algorithm makes. This algorithm has two meta-parameters that are important: the time slice allocated for each process, which can be controlled, and the context switch time, which has to be endured by the scheduler.




 varying_rr_cpu.exp	07.04.2022 17:59	Fichier EXP	1 Ko
 varying_rr_interactive.exp	07.04.2022 17:58	Fichier EXP	1 Ko
 varying_rr_mixed.exp	07.04.2022 17:58	Fichier EXP	1 Ko

Figure 24 – experiments for the varying RR

The previous run files are used in three new experiment files with the RR algorithm with a time slice of 0.1, 0.5, 1 and 2. No overhead is set for the following subsections.


4.1. Interactive

The first difference is observed on the interactive context.

```
Starting experiment of 4 runs from varying_rr_interactive
Experimental Run for interactive
585 events done in 91 milliseconds
Experimental Run for interactive
151 events done in 41 milliseconds
Experimental Run for interactive
151 events done in 42 milliseconds
Experimental Run for interactive
151 events done in 26 milliseconds
```

Figure 25 – RR varying time slice analysis - interactive experiment execution

The execution time is very long for the RR 0.1. It seems to stabilize for 0.5 and 5, and to be even more efficient with a time slice of ten.

 Table Data												
									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
interactive_5	RR 0.1	42.58	5	5	.587098	.117420	0.00	.64	267	45	.09	3.23
interactive_6	RR 0.5	38.83	5	5	.643870	.128774	0.00	.26	50	45	.50	3.23
interactive_7	RR 5	38.83	5	5	.643870	.128774	0.00	.26	50	45	.50	3.23
interactive_8	RR 10	38.83	5	5	.643870	.128774	0.00	.26	50	45	.50	3.23

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
interactive_5	RR 0.1	39.54	35.97	42.58	2.42	5.46	5.05	6.28	.08
interactive_6	RR 0.5	36.07	31.60	38.83	2.58	1.98	.89	3.21	.17
interactive_7	RR 5	36.07	31.60	38.83	2.58	1.98	.89	3.21	.17
interactive_8	RR 10	36.07	31.60	38.83	2.58	1.98	.89	3.21	.17

Done

Figure 26 - RR varying time slice analysis - Table data for the interactive algorithm

The RR with time slice 0.5, 5 and 10 algorithms are very similar in term of numbers. The round robin with a time slice of 0.1 has way more entries, with a bigger average turnaround time and a lot more waiting time.

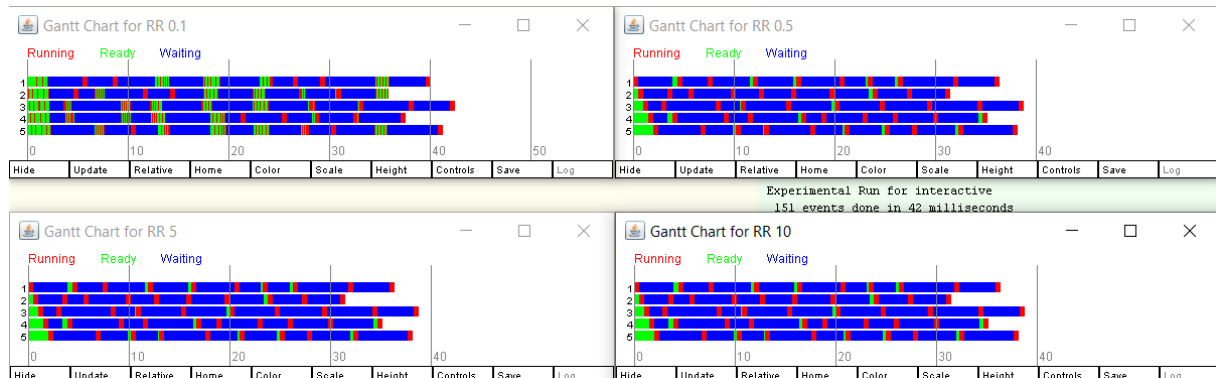


Figure 27 - RR varying time slice analysis - Gantt diagrams for the algorithms during the interactive experiment

The RR 0.1 is different from the others because the threads are mostly in waiting status themselves and rarely in ready mode. When an interaction process is in ready mode, it is allocated a very little amount of CPU time. When a bunch of processes are in ready mode (as shown in the Gantt diagram), they are themselves allocated a lot of tiny CPU time and are competing hard for resources. With the other and bigger time slices, the processes have time to finish before giving the hand and less time is wasted waiting in ready mode for tiny bursts of CPU. This explains the bigger execution time and bigger waiting time, and the execution time would be even worse with overhead during context switches.

4.2. CPU intensive

The second part of this section has the same experiments done on CPU-intensive threads.

```
Starting experiment of 4 runs from varying_rr_cpu
Experimental Run for cpuintensive
4051 events done in 637 milliseconds
Experimental Run for cpuintensive
821 events done in 148 milliseconds
Experimental Run for cpuintensive
101 events done in 40 milliseconds
Experimental Run for cpuintensive
61 events done in 25 milliseconds
```

Figure 28 – RR varying time slice analysis - CPU intensive experiment execution

The problem of tiny time slice is exacerbated on the CPU intensive tasks. The execution time are in descending order, with an order of magnitude of difference between a slice of 0.1 and a slice of 10.

Table Data

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
cpuintensive_1	RR 0.1	200.00	5	5	1.000000	.025000	0.00	3.87	2015	15	.10	1.35
cpuintensive_2	RR 0.5	200.00	5	5	1.000000	.025000	0.00	3.86	400	15	.50	1.35
cpuintensive_3	RR 5	200.00	5	5	1.000000	.025000	0.00	3.65	40	15	5.00	1.35
cpuintensive_4	RR 10	200.00	5	5	1.000000	.025000	0.00	3.40	20	15	10.00	1.35

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
cpuintensive_1	RR 0.1	198.90	196.70	200.00	1.16	154.84	154.02	155.72	.13
cpuintensive_2	RR 0.5	198.30	196.00	200.00	1.54	154.24	153.18	155.52	.19
cpuintensive_3	RR 5	190.00	180.00	200.00	7.07	145.94	136.18	156.08	1.39
cpuintensive_4	RR 10	180.00	160.00	200.00	14.14	135.94	116.18	156.08	2.79

Done

Figure 29 – RR varying time slice analysis - Table data for the CPU intensive algorithm

The turnaround times and waiting times are also in descending order. The minimum waiting time and turnaround time are notably smaller for the RR 5 and 10, hinting that the processes wait a smaller amount of time and are generally executed in a smaller period.

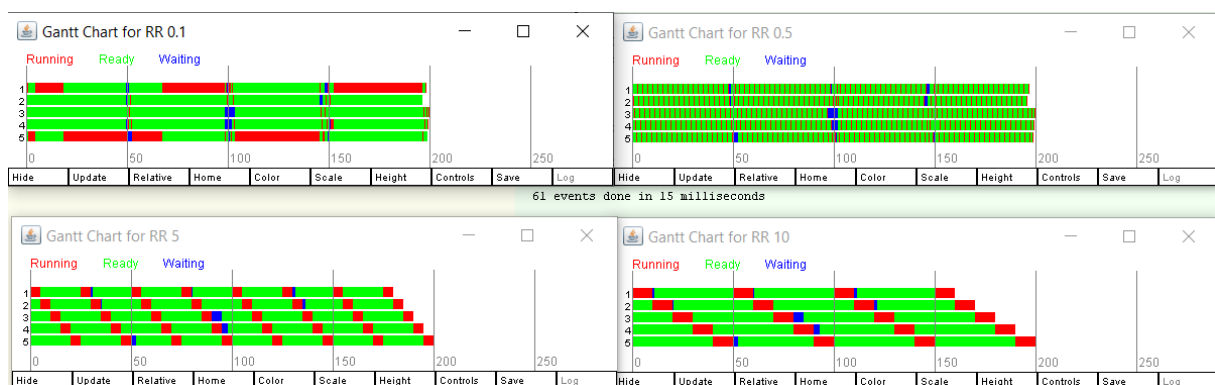


Figure 30 – RR varying time slice analysis - Gantt diagrams for the 4 algorithm during the CPU intensive experiment

The RR of 0.1 does not seem to work correctly and the first diagram tend to indicate a failure of the algorithm, as process 2,3 and 4 are ready and not given their allocation of resource.

The RR 0.5, 5 and 10 are working correctly. The RR 5 and 10 show that in case of high CPU intensive tasks, the round robin benefits from having a higher slice time (meanwhile that would be the opposite for interactions, as seen before).

4.3. Mixed workload

The round robin is now evaluated with the time slices on the mixed workloads.


```

Starting experiment of 4 runs from varying_rr_mixed
Experimental Run for mixedload
2076 events done in 328 milliseconds
Experimental Run for mixedload
476 events done in 84 milliseconds
Experimental Run for mixedload
112 events done in 39 milliseconds
Experimental Run for mixedload
112 events done in 28 milliseconds

```

Figure 31 – RR varying time slice analysis - mixed experiment execution

The difference of execution time between the time slices is less significant than for the CPU intensive tasks, but still suffers in comparison to the full interaction processes. This could hint that longer time slices are heavily preferred by CPU intensive task but induce lost CPU for interactions which prefer very small slices.


Table Data

☐
☐
☒
☐

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
mixedload_9	RR 0.1	101.42	5	5	.985984	.049299	0.00	2.91	1019	32	.10	3.08
mixedload_10	RR 0.5	102.89	5	5	.971879	.048594	0.00	2.92	219	32	.46	3.08
mixedload_11	RR 5	101.32	5	5	.986927	.049346	0.00	2.65	37	32	2.70	3.08
mixedload_12	RR 10	101.32	5	5	.986927	.049346	0.00	2.65	37	32	2.70	3.08

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
mixedload_9	RR 0.1	98.66	96.43	101.42	1.79	58.97	56.40	60.48	.29
mixedload_10	RR 0.5	99.83	97.00	102.89	1.91	60.14	56.97	62.05	.34
mixedload_11	RR 5	93.49	75.70	101.32	9.20	53.80	38.75	59.46	1.54
mixedload_12	RR 10	93.49	75.70	101.32	9.20	53.80	38.75	59.46	1.54

Done

Figure 32 – RR varying time slice analysis - Table data for the mixed algorithm

The CPU utilization of the table seems to corroborate this hypothesis at first: The time slices 5 and 10 have the best CPU utilization, the slowest turnaround time, and the slowest waiting time. The RR0.5 seems to be the worst time slice all-around for this scenario for all indicators (turnaround, waiting time, CPU utilization). This could mean that the RR algorithm struggles to satisfies both short-burst (with small time slices) and highly intensive (which prefer long time slices) processes at the same time.



Figure 33 – RR varying time slice analysis - Table data for the mixed algorithm

The Gantt diagram show that the RR 0.1 is constantly giving resources to processes that are ready, even though it does a lot of contexts switching. The RR 5 and 10 give long chunks of CPU resources that is large enough to be rarely wasted, even with multiple processes going on waiting mode simultaneously. The RR with a time slice of 0.5 seems to be losing CPU time by having more processes waiting while being ready.

It is important to note that the RR of 0.1 is usually the worst in term of execution time despite not always being the worst in term of data table and not having an overhead when switching context. This could be due to the software itself adding some overhead when running the experiments. This context switch overhead is studied in the next and final subsection.

4.4. Context switch overhead

The last part of this section studies the change in dispatch latency when an overhead cost is associated to context switching.

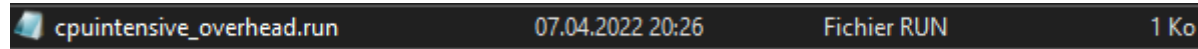


Figure 34 – run files for the context switch overhead test

A new run file is introduced – they are similar to the CPU intensive one but with context switching cost.

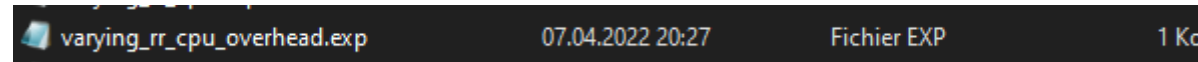


Figure 35 – experiments for the context switch overhead test

An experiment file is associated and runs the experiment once with each RR time slice (0.1 to 10). The goal is to analyze the difference an overhead makes between a little time slice and a big time slice. One of the previous observations suggests that the overhead would be terrible with little time slices for CPU intensive tasks that can fully utilize the big time slices of CPU time. As the goal is to observe if there is a real distinction, the experiment is based on the CPU intensive task.

Starting experiment of 4 runs from varying_rr_mixed	Starting experiment of 4 runs from varying_rr_cpu_overhead
Experimental Run for mixedload	Experimental Run for cpuintensive_overhead
2076 events done in 328 milliseconds	6036 events done in 939 milliseconds
Experimental Run for mixedload	Experimental Run for cpuintensive_overhead
476 events done in 84 milliseconds	1224 events done in 205 milliseconds
Experimental Run for mixedload	Experimental Run for cpuintensive_overhead
112 events done in 39 milliseconds	144 events done in 41 milliseconds
Experimental Run for mixedload	Experimental Run for cpuintensive_overhead
112 events done in 28 milliseconds	81 events done in 26 milliseconds

Figure 36 – Difference between the execution time without and with overhead during context switching

The round robins of 0.1 and 0.5 are highly impacted by the overhead, going from 300ms to 900ms and from 84ms to 200ms for a time slice of 0.1 and 0.5. The overhead has little to no impact for time slices of 5 and 10.

It is to note that for a time slice of 0.1 and 0.5, the overhead of one is actually more costly than the time slice itself. A significant impact on execution time is therefore expected. For the time slices of 5 and 10, the overhead has a cost that stays below the time slice.

In term of efficiency, an in-and-out time slice of 0.1 adds 2 overheads, and that an overhead has a tenth of the cost of a path. By considering the efficiency as CPU used for computation divided by total CPU used for a pass, we have $1 / (1+2) = 0.33$, or 33% of CPU time efficiency, which is very bad. For a time slice of 10, an in-and-out pass has an efficiency of $100 / (100+2) = 0.98$, so 98% of efficiency in CPU time. This is assuming the context has switched twice for a pass.

Again, a bigger time slice seems to be even better in term of context switching but could be costly in case of interactions as observed previously. This is a trade-off that has to be defined when setting up the algorithm depending on the cost of switching context and the type of processes to schedule.

Table Data

Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	Entries		Average Time	
									CPU	I/O	CPU	I/O
cpuintensive_1	RR 0.1	200.00	5	5	1.000000	.025000	0.00	3.87	2015	15	.10	1.35
cpuintensive_2	RR 0.5	200.00	5	5	1.000000	.025000	0.00	3.86	400	15	.50	1.35
cpuintensive_3	RR 5	200.00	5	5	1.000000	.025000	0.00	3.65	40	15	5.00	1.35
cpuintensive_4	RR 10	200.00	5	5	1.000000	.025000	0.00	3.40	20	15	10.00	1.35

Name	Key	Turnaround Time				Waiting Time			
		Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
cpuintensive_1	RR 0.1	198.90	196.70	200.00	1.16	154.84	154.02	155.72	.13
cpuintensive_2	RR 0.5	198.30	196.00	200.00	1.54	154.24	153.18	155.52	.19
cpuintensive_3	RR 5	190.00	180.00	200.00	7.07	146.94	136.18	156.08	1.39
cpuintensive_4	RR 10	180.00	160.00	200.00	14.14	135.94	116.18	156.08	2.79

Done

Table Data

Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	Entries		Average Time	
									CPU	I/O	CPU	I/O
cpuintensive_overhead_1	RR 0.1	601.00	5	5	.332779	.008319	401.00	4.62	2005	15	.10	1.45
cpuintensive_overhead_2	RR 0.5	280.20	5	5	.713776	.017844	80.20	4.17	401	15	.50	1.45
cpuintensive_overhead_3	RR 5	208.20	5	5	.960615	.024015	8.20	3.63	41	15	4.88	1.45
cpuintensive_overhead_4	RR 10	204.00	5	5	.980392	.024510	4.00	3.41	20	15	10.00	1.45

Name	Key	Turnaround Time				Waiting Time			
		Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
cpuintensive_overhead_1	RR 0.1	599.58	597.60	600.90	1.12	515.52	514.92	516.32	.10
cpuintensive_overhead_2	RR 0.5	278.14	274.50	280.10	1.95	226.16	223.92	227.58	.25
cpuintensive_overhead_3	RR 5	195.62	182.10	208.10	9.64	150.84	137.58	163.32	1.87
cpuintensive_overhead_4	RR 10	183.50	163.10	203.90	14.42	139.14	118.98	159.68	2.85

Figure 37 – Difference between the data tables without and with overhead during context switching

The previous observation is clearly shown in the data table that shows the CPU utilization of 0.33 for the time slice of 0.1 going up to 0.98 for the time slice of ten. The numbers are actually very consistent with the previous efficiency calculations. The RR 0.1 has a terrible waiting time, whether on average or on minimum / maximum due to the context switching.

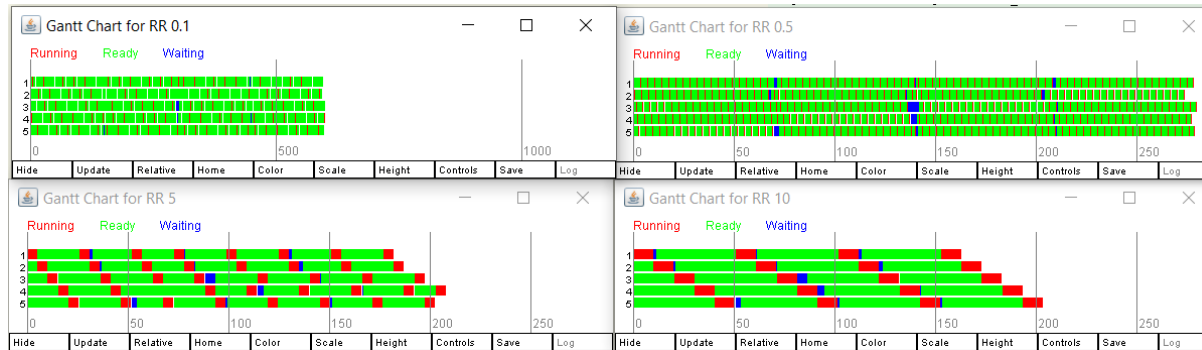


Figure 38 – Gantt diagrams with overhead during context switching

The Gantt diagrams do not seem to yield information at first unless we zoom in on the two extremes – the time slices of 0.1 and 10.

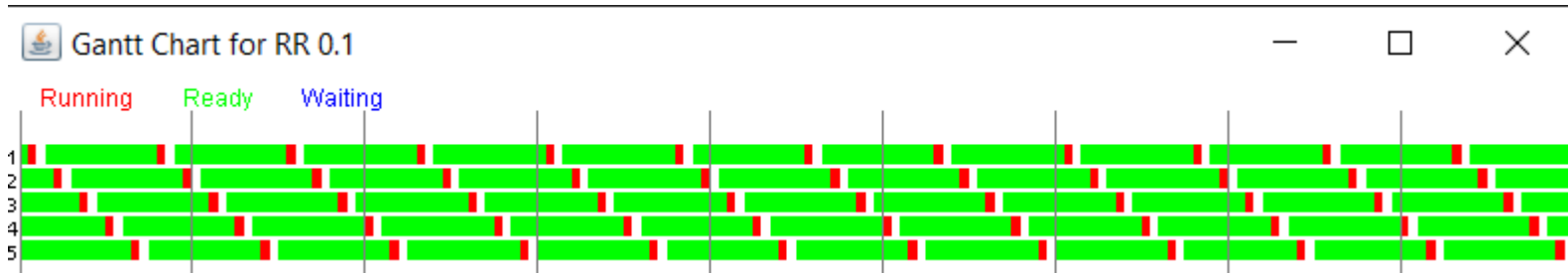


Figure 39 – Zoomed-in Gantt diagram with overhead during context switching with time slice = 0.1

The time slice of 0.1 with an overhead cost of one clearly shows the gaps created during the allocated CPU time at each pass of the round robin. Although the CPU intensive is the perfect use case for round robin, this poor efficiency is terrible for the execution time and waiting time of the algorithm.

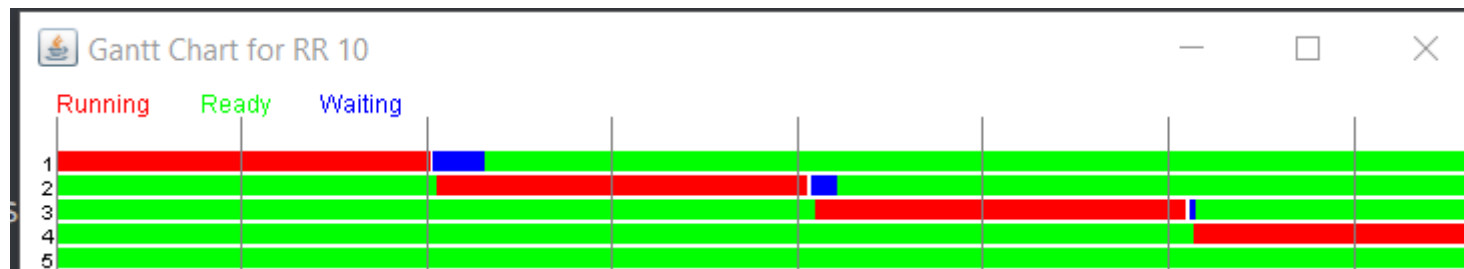


Figure 40 – Zoomed-in Gantt diagram with overhead during context switching with time slice = 10

The time slice of ten clearly shows that the overhead become nearly negligible with a big time slice. Once again, this time slice could become problematic in case of interaction (and more context switches would occur).

5. High load performance

This last section tests the same 4 algorithms, First Come First Serve, Round robin, Shortest-job First and SJF with alpha averaging. The processes are more numerous and coming at various times to simulate a higher load of tasks.




 highload_cpuintensive.run	07.04.2022 13:43	Fichier RUN	1 Ko
 highload_interactive.run	07.04.2022 18:16	Fichier RUN	1 Ko
 highload_mixedload.run	07.04.2022 13:43	Fichier RUN	1 Ko

Figure 41 – runs for the high load performance

Three high load runs are created. They are mostly the same as the precedent runs, but have some parameters tweaked: a higher number of processes coupled to an interarrival with a uniform distribution.




 highload_cpuintensive.exp	07.04.2022 18:18	Fichier EXP	1 Ko
 highload_interactive .exp	07.04.2022 14:58	Fichier EXP	1 Ko
 highload_mixed.exp	07.04.2022 14:58	Fichier EXP	1 Ko

Figure 42 – experiments for the high load performance

Three experiments file are created for this experiment, one for each setup.

5.1. Interactive

Firstly, the algorithms are evaluated with interactive processes with a higher load requirement.

```
Starting experiment of 4 runs from highload_interactive
Experimental Run for highload_interactive
15001 events done in 2350 milliseconds
Experimental Run for highload_interactive
15001 events done in 2346 milliseconds
Experimental Run for highload_interactive
15001 events done in 2354 milliseconds
Experimental Run for highload_interactive
15001 events done in 2339 milliseconds
```

Figure 43 – High load interactive experiment execution

Similarly as for the previously tested interactive processes, all four algorithms perform the same in term of execution time. This can be the same issue of zero-conflict between the processes.

Table Data

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
highload_interactive_1	FCFS	4143.89	500	500	.603327	.120665	0.00	.24	5000	4500	.50	2.97
highload_interactive_2	RR	4143.89	500	500	.603327	.120665	0.00	.24	5000	4500	.50	2.97
highload_interactive_3	SJF	4143.89	500	500	.603327	.120665	0.00	.24	5000	4500	.50	2.97
highload_interactive_4	SJFA	4143.80	500	500	.603339	.120668	0.00	.23	5000	4500	.50	2.97

Name	Key	Turnaround Time				Waiting Time			
		Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
highload_interactive_1	FCFS	33.74	22.80	43.94	3.59	2.00	.02	6.33	.00
highload_interactive_2	RR	33.74	22.80	43.94	3.59	2.00	.02	6.33	.00
highload_interactive_3	SJF	33.74	22.80	43.94	3.59	2.00	.02	6.33	.00
highload_interactive_4	SJFA	33.64	23.11	43.36	3.71	1.89	0.00	7.88	.00

Done

Figure 44 – Table data for the High load interactive algorithm

The data table is also almost identical between all four algorithms. The CPU utilization is bad because there are times where there are no processes to run, wasting CPU time. The waiting time are still very low (average of two with a very low minimum) and the turnaround times are very good, which is a good indication that the waiting processes are executed quickly. The sparse environment is responsible of this poor efficiency.

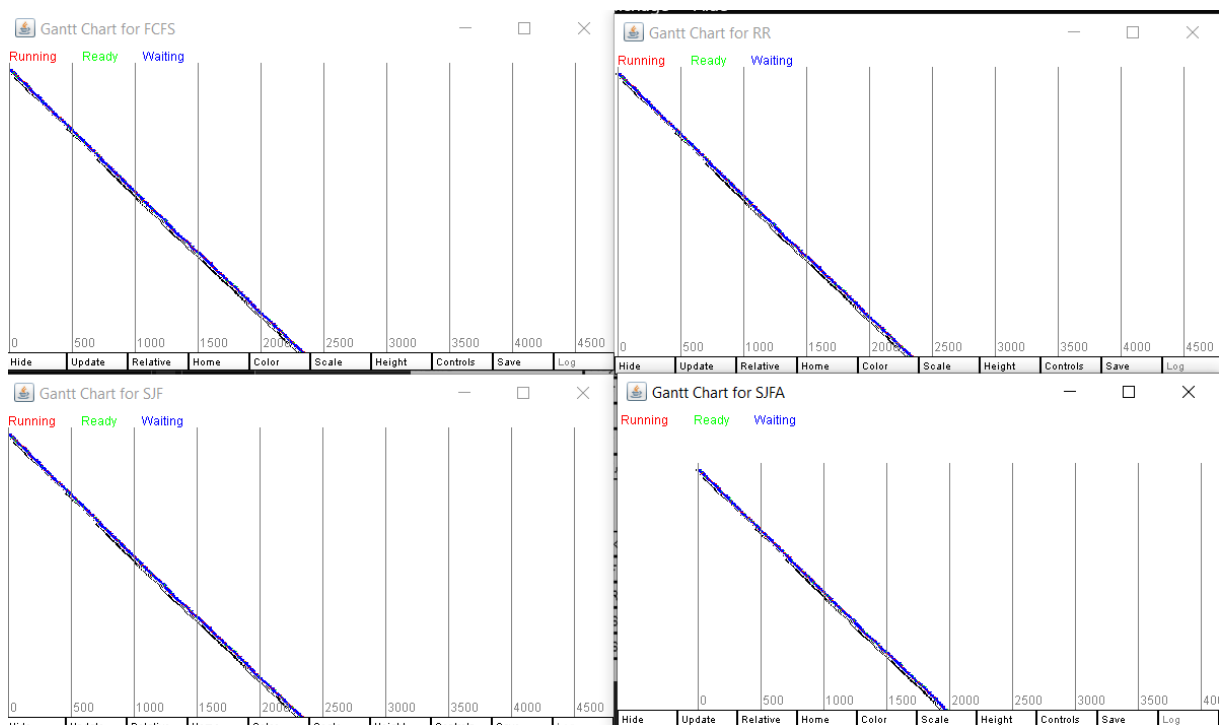


Figure 45 –Gantt diagrams for the 4 algorithms during the High load interactive experiment

The Gantt diagram show that the processes have no conflict for resources, and therefore the scheduling algorithms are not useful.

5.2. CPU intensive

Secondly, the CPU intensive task is evaluated with more processes competing for resources.

```
Starting experiment of 4 runs from highload_cpuintensive
Experimental Run for highload_cpuintensive
6001 events done in 941 milliseconds
Experimental Run for highload_cpuintensive
6001 events done in 945 milliseconds
Experimental Run for highload_cpuintensive
42009 events done in 6544 milliseconds
Experimental Run for highload_cpuintensive
6001 events done in 957 milliseconds
```

Figure 46 – High load CPU intensive experiment execution

The FCFS, SJF and SJFA are similarly at 1000ms, and the round robin lags behind at 6500ms.

Table Data

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
highload_cpuintensive_5	FCFS	21253.51	500	500	.941021	.023526	0.00	.36	2000	1500	10.00	1.49
highload_cpuintensive_6	SJF	21253.51	500	500	.941021	.023526	0.00	.36	2000	1500	10.00	1.49
highload_cpuintensive_7	RR	21255.57	500	500	.940930	.023523	0.00	.47	20004	1500	1.00	1.49
highload_cpuintensive_8	SJFA	21253.51	500	500	.941021	.023526	0.00	.36	2000	1500	10.00	1.49

Name	Key	Turnaround Time				Waiting Time			
		Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
highload_cpuintensive_5	FCFS	59.77	43.82	84.99	6.49	15.31	0.00	34.52	.01
highload_cpuintensive_6	SJF	59.77	43.82	84.99	6.49	15.31	0.00	34.52	.01
highload_cpuintensive_7	RR	64.47	46.82	86.39	5.51	20.01	3.00	36.51	.01
highload_cpuintensive_8	SJFA	59.77	43.82	84.99	6.49	15.31	0.00	34.52	.01

Done

Done

Figure 47 –Table data for the High load CPU intensive algorithm

The RR does not loose that much CPU utilization compared to the other three algorithms. Nevertheless, it has a worse turnaround and waiting time in average: although the maximum waiting time and turnaround time are similar, the minimum ones are higher for the round robin, indicating that the processes might be waiting longer than with the other algorithms.

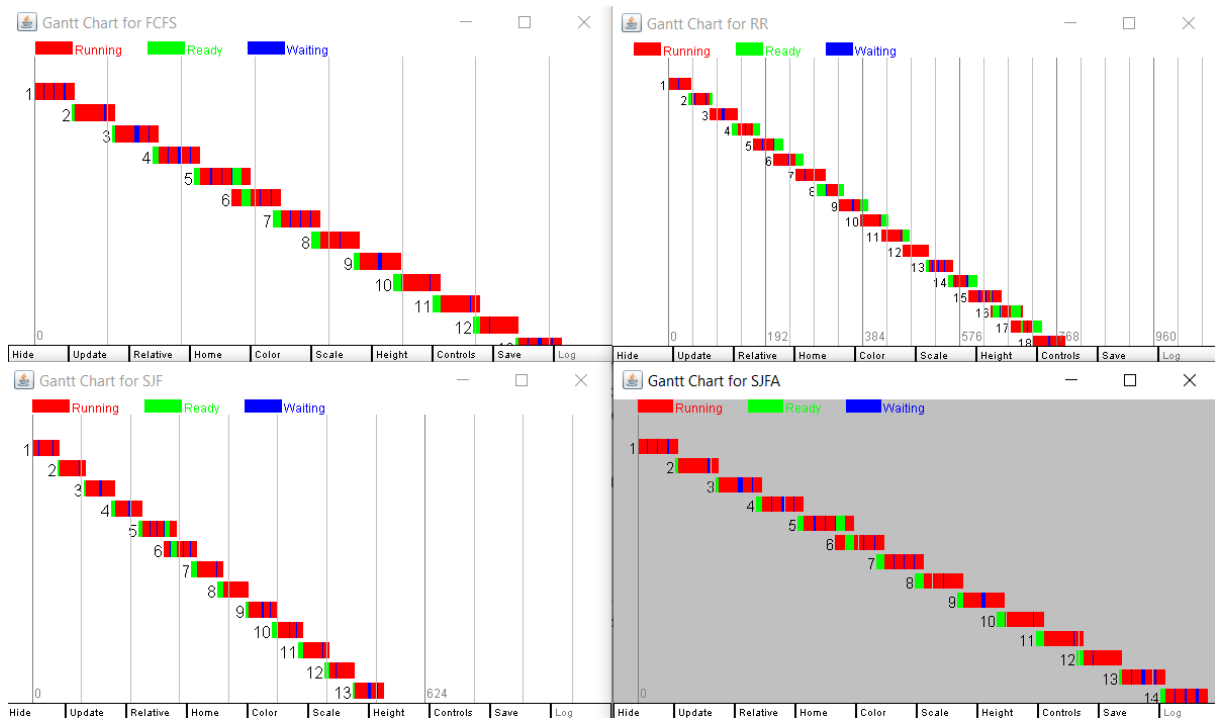


Figure 48 – Gantt diagrams for the 4 algorithms during the High load CPU intensive experiment

The Gantt diagram show that the FCFS executes one process after the other – but a process coming in the middle of an execution can steal the CPU time of the first process that arrived when it goes into waiting status. The SJF and SJFA have a strong tendency of allocating the resources to the processes that are the closest to finishing their tasks. Their estimations seem good as the processes finish one by one in the order of arrival (and not in the order of request as in FCFS). This means that the first processes are less impacted by processes coming during execution, proving useful for CPU-intensive tasks efficiency. The round robin indices some strong waiting time when multiple processes are waiting on the CPU as it does not prioritize any of them. This makes the processes wait longer during the execution of the experiment.

5.3. Mixed workload

The last part of this experiment is to evaluate the algorithm under a high load of mixed workload processes.

```
Starting experiment of 4 runs from highload_mixedload
Experimental Run for highload_mixedload
10795 events done in 1678 milliseconds
Experimental Run for highload_mixedload
27251 events done in 4244 milliseconds
Experimental Run for highload_mixedload
10795 events done in 1681 milliseconds
Experimental Run for highload_mixedload
10795 events done in 1675 milliseconds
```

Figure 49 – High load mixed experiment execution

The execution times are almost identical, excepted for the round robin which is once again lagging behind.



Table Data

										Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA		CPU	I/O	CPU	I/O
highload_mixedload_9	FCFS	29907.51	500	500	.334364	.016718	0.00	.02		3598	3098	2.78	2.97
highload_mixedload_10	RR	29907.51	500	500	.334364	.016718	0.00	.02		11826	3098	.85	2.97
highload_mixedload_11	SJF	29907.51	500	500	.334364	.016718	0.00	.02		3598	3098	2.78	2.97
highload_mixedload_12	SJFA	29907.51	500	500	.334364	.016718	0.00	.02		3598	3098	2.78	2.97

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
highload_mixedload_9	FCFS	39.32	27.79	61.09	4.52	.94	0.00	11.58	.00
highload_mixedload_10	RR	39.72	27.79	59.59	4.69	1.35	0.00	12.93	.00
highload_mixedload_11	SJF	39.31	27.79	57.57	4.49	.93	0.00	11.46	.00
highload_mixedload_12	SJFA	39.32	27.79	61.09	4.53	.94	0.00	11.58	.00

Done

Figure 50 – Table data for the high load mixed algorithm

The CPU utilization is the same for all algorithms: very sparse at 33% efficiency. The RR has a lot more entries, and a waiting time that is worse than the other algorithms although the turnaround times are still in-line and even better than FCFS.

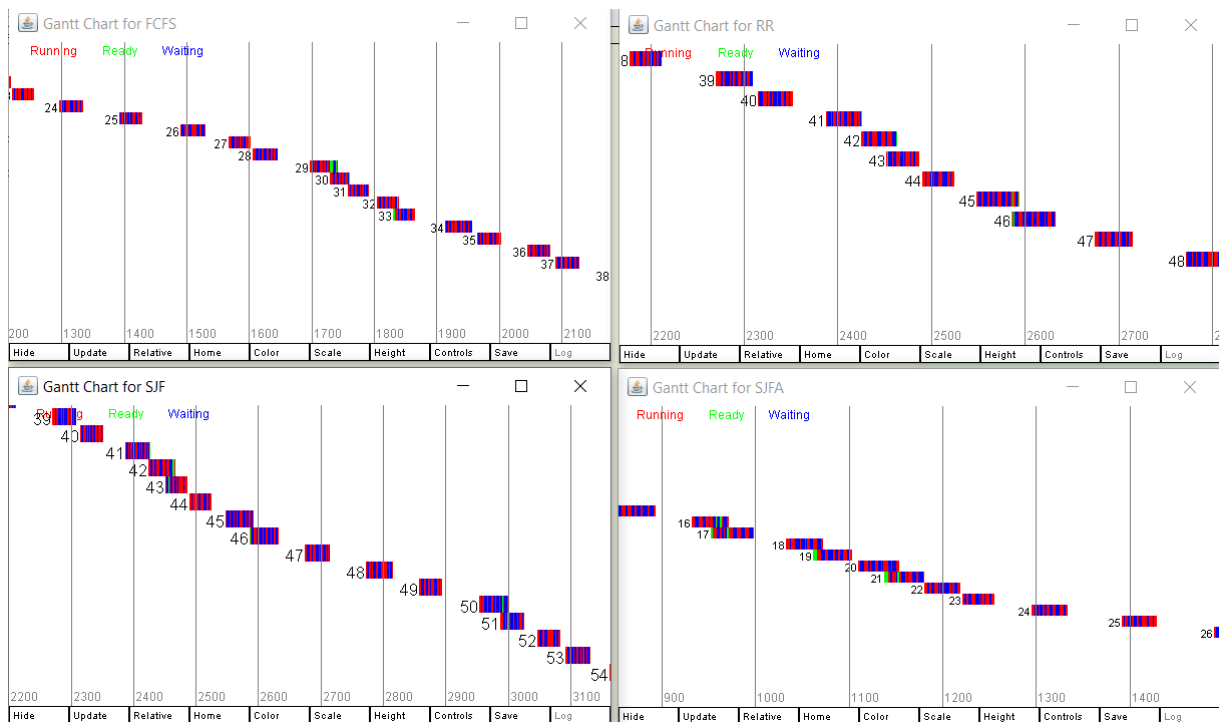


Figure 51 – Gantt diagrams for the 4 algorithms during the high load mixed experiment

The Gant diagram show that once again the SJF are generally the best one when it comes to balanced tasks of short and long CPU/IO bursts with waiting times: the CPU resources are correctly allocated by prioritizing little chunks of CPU time to interactions followed by large chunks for computations.

The FCFS does not prioritize one over the other – as a result, the waiting times are the same but the turnaround times are generally higher because some interaction processes are waiting on computations (convoy effect) when they could be quickly dispatched.

The round robin is the clear loser on mixed tasks workloads as it does not prioritize any of the tasks and tries to allocate the same amount to every process asking. This generates a lot of waiting times that are visible on the data table. The overhead would worsen the problem, although the time slice could also be adjusted. It is at 1 for this experiment, and as seen before, 0.5 was the worst one for balanced workloads. A slice of 0.1 works better for interaction but would suffer from overhead, meanwhile a slice of five would give a chance to the moderate CPU tasks to finish before switching context.

6. Conclusion

Firstly, these experiments have identified the main reason to use scheduling algorithm: to efficiently resolve conflict between resource allocation (here CPU time). The different algorithms resolve this problem differently, and these resolutions are more adapted to some type of problem. For example, the round robin is fair, which is a good thing for CPU intensive tasks, but terrible for mixed tasks. The question of what a good solution is also an important one and is answered by looking at different indicators on the data table. For example, an interaction-focused system will want low waiting time meanwhile a CPU intensive system would be focused on CPU utilization, throughput and a mixed system could focus on turnaround time to finish the numerous tasks that it has to execute.

The mixed experiments also highlighted that the context of the conflicts is important: are the tasks asking for small or big chunks of CPU times ? Or both at the same type ? What is the nature of the conflict and the requests for resources ? For example, the shortest-come-first type of algorithms did not shine when asked to schedule a singular type of tasks, although they are fairly good for mixed tasks. When mixing two types of tasks, they outshine the others by prioritizing low-latency interactions over long-time computations. The SJF-A is also proving to be a very good improvement over the SJF at determining the next CPU burst of the processes when they are rapidly varying in this context.

The varying experiment show the importance of adapting hyper-parameters of the algorithms to the process's nature. The round robin with a very low time slice is slightly less terrible with interactions meanwhile big time slices make it shine where it is the most efficient: Big CPU-intensive tasks. The overhead experiment also show that the parameters also have to be adapted to the system's properties, such as context-switching overhead.

Lastly, the high load performance seemed to validate the hypothesis made on the individual short-load workloads.

This lab was very insightful when it comes to firsthand practice of the scheduling algorithms. It highlights that the choice of a scheduling algorithms must be a very thought-out choice based on multiple criteria which can be:

- The nature of the processes competing for resources
- The different indicators that are important for the system (CPU utilization, turnaround time, waiting time, latency, etc....)
- The intrinsic properties of the system the scheduler must work on

And even when the choice is made, the algorithms themselves can also be tweaked to be adapted to the problematic. Scheduling algorithms are precise mechanism that must be hand-picked to respond to specific problems, and this lab shows that they all provide different results that can be interesting during different setup.