CS330 Group 28 Assignment 2

Aryan Vora(200204) **S Pradeep(200826)** **Tejas Ramakrishnan(201050)**

Yash Gupta(201144)

1 Part 1: SCHED_NPREEMPT_SJF

We find the shortest job, which will be scheduled next using the $minp$ process. We iterate the process table and execute any processes which are non batch, otherwise, the shortest job is stored in $minp$. We maintain $flag_1$, which is 0 when there are only batch processes, else it is 1. We also maintain another flag, $flag_2$, which is used to ensure that the scheduling algorithm has not changed.
We now perform a context switch and execute $minp$. We maintain t_i as the actual running time for the i^{th} iteration and estimate s_{i+1} using a as 0.5, as instructed.

2 Part 2: SCHED_PREEMPT_UNIX

We introduce bp in the process structure which maintains the priority of the process. Initialize cpu_usage as 0. We go through all the processes in the process table and check for main processes, else we divide the cpu_usage of all processes by 2. We then update the priority and find the minimum priority process and schedule it. We update cpu_usage of the executed process using the mentioned values and conditions.

3 Part 3: Observations

3.1 Comparison between non-preemptive FCFS and preemptive round-robin:

batch1.txt	FCFS	RR
Batch execution time:	15522	15721
Average turn-around time:	15516	15635
Average waiting time:	13962	14061
Completion time: avg:	16184	16391
Completion time: max:	16189	16473
Completion time: min:	16180	16265

Batch1.txt has ten processes, each one is testloop1.c. Testloop1.c has long CPU bursts, three for loops, the outermost for loop executes 5 times. The two inner for loops will take a lot of time(1 million and 100 iterations each) and those two inner for loops will be executed before sleep() is called. So in non-preemptive FCFS, the process will be scheduled, will run the two inner for loops and will go to sleep(Willingly giving up the CPU). It will continue the remaining iterations of the outer for loops when it is scheduled again. (5 CPU Bursts). However, in Preemptive round robin, OS will give timer interrupt before process can willingly call sleep(), so it will need to be scheduled multiple more times, not just 5 like in FCFS. FCFS has unbounded waiting time in worst case.

batch2.txt	FCFS	RR
Batch execution time:	15694	15898
Average turn-around time:	15687	15834
Average waiting time:	14120	14248
Completion time: avg:	18490	16515
Completion time: max:	18496	16577
Completion time: min:	18486	16453

Preprint. Under review.

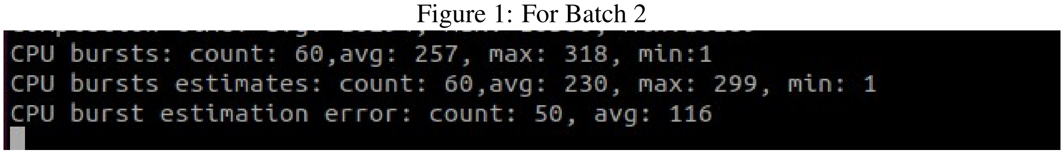
Here, instead of sleep(1), yield() is called, and the only difference I can see is larger completion times in FCFS compared to RR now.

batch7.txt	FCFS	RR
Batch execution time:	15556	15726
Average turn-around time:	8535	15686
Average waiting time:	6981	14116
Completion time: avg:	9388	16403
Completion time: max:	16408	16453
Completion time: min:	2397	16307

Batch7.txt has ten processes, each one is testloop4.c. Testloop4.c has long CPU bursts, three for loops, the outermost for loop executes 5 times. The difference here is that neither sleep() nor yield() is called. So in non-preemptive FCFS, the process will be scheduled , and will complete exectution, since it does not willingly give up the CPU and has only CPU Burst. However, in Preemptive round robin, OS will give timer interrupt before process completes, so it will need to be scheduled multiple more times, not just once like in FCFS. We can see that the minimum completion time in FCFS is really low, and maximum is really high, meaning that FCFS clearly does poorly on fairness. This is because first process that is scehduled will be allowed to complete in one long CPU burst, and that process will have very low completion time. There is also much lower average waiting time in FCFS.

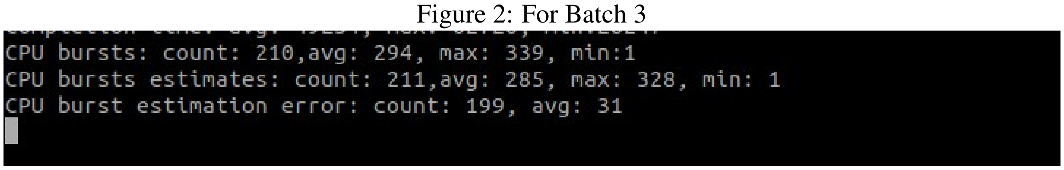
3.2 CPU burst estimation error using exponential averaging:

Figure 1: For Batch 2



(the average CPU burst estimation error per estimation instance)/(the average CPU burst length)=$\frac{116}{257}$
The error is quite high. This might be due to the unevenness of the cpu bursts due to the short amount of I/O bursts. This can lead to the estimate deviating from the optimal value.

Figure 2: For Batch 3



(the average CPU burst estimation error per estimation instance)/(the average CPU burst length)=$\frac{31}{294}$
The error is quite low. This might be due to the evenness of the cpu bursts due to the continuos, intersped I/O bursts which results in the extimate being quite close to the optimal value.

3.3 Comparison between non-preemptive FCFS and non-preemptive SJF:

batch4.txt	FCFS	SJF
Batch execution time:	11581	11536
Average turn-around time:	11575	8300
Average waiting time:	10419	7166
Completion time: avg:	12406	9159
Completion time: max:	12411	12242
Completion time: min:	12402	6078
CPU bursts: count:	-	61
CPU bursts: avg:	-	189
CPU bursts: max:	-	316
CPU bursts: min:	-	1
CPU burst estimates: count:	-	61
CPU burst estimates: avg:	-	168
CPU burst estimates: max:	-	301
CPU burst estimates: min:	-	1
CPU burst estimation error: count:	-	51
CPU burst estimation error: avg:	-	86

We observe FCFS is fair while SJF has a wide range of completion times. This is expected from SJF as it will strive towards finishing the shortest job first, this will lead to the minimum completion time being quite small. However, the maximum completion time remains comparable to FCFS. Now, due to the minimum being quite small, the average is pulled down on all statistics, and that is what we observe.

3.4 Comparison between preemptive round-robin and preemptive UNIX:

batch5.txt	RR	UNIX	Difference (RR-UNIX)
Batch execution time:	15443	15575	-132
Average turn-around time:	15340	10257	5083
Average waiting time:	13793	9021	4772
Completion time: avg:	16247	11206	5041
Completion time: max:	16346	16525	-179
Completion time: min:	16146	5643	10503

(a) We understand that the UNIX scheduler is quite unfair compared to the Round Robin scheduler based on the major difference between the maximum and minimum completion times. This batch has processes with long CPU bursts and short I/O bursts of sleeping. I expect that the RR scheduler in general takes more time as the small I/O bursts are wasteful of the time quantum for the scheduler. This leads to a higher completion time in general for the RR scheduler.

batch6.txt	RR	UNIX	Difference (RR-UNIX)
Batch execution time:	15419	15487	-68
Average turn-around time:	15345	10175	5170
Average waiting time:	13808	8748	5060
Completion time: avg:	16379	11373	5006
Completion time: max:	16446	16686	-240
Completion time: min:	16307	5817	10490

(b) We understand that the UNIX scheduler is quite unfair compared to the Round Robin scheduler based on the major difference between the maximum and minimum completion times. This batch has processes with long CPU bursts and short I/O bursts of yielding. I expect that the RR scheduler in general takes more time as the small I/O bursts are wasteful of the time quantum for the scheduler. This leads to a higher completion time in general for the RR scheduler.

4 Limitations

Due to lack of floating arithmetic while computing priorities and CPU usage, multiple processes begin to have the same value for priorities (due to rounding). This leads to the first few processes being executed, while there is a delay for execution of the last few processed (namely pid 10, 11, 12, 13).
This can be seen in our code, and on uncommenting our print statements, one can observe the above phenomenon.