ASSIGNMENT 2 CS330

Group 29

EXPERIMENTS

1. Comparison between non-preemptive FCFS and preemptive round-robin:
   a. batch1.txt:

	FCFS	Round-Robin
Batch Execution Time	8962	8994
Avg turnaround time	8959	8972
Avg waiting Time	8057	8067
Completion Time (Difference between max and min)	7	49

Explanation:

- Both the policy give more or less the same result as the process have large cpu bursts and then a small I/O bursts. Basically, the FCFS runs the entire inner loop once, then due to the I/O call, it goes to sleep, and so another process is scheduled, and this continues in a round robin fashion. The only difference is that in RR scheduling, the process may be context switched out in between executing the inner for loop. So the processes will end more or less together. And due to the similar execution time of all the processes, they also will have similar waiting times and turnaround times.
-

b. batch2.txt:

	FCFS	Round-Robin
Batch Execution Time	8686	9452
Avg turnaround time	8684	9405
Avg waiting Time	7814	8460
Completion Time (Difference between max and min)	7	143

Explanation:

Similar to above, the only difference is that we call yield instead of sleep, and scheduling wise, for these two schedulers, it won't make much of a difference. Basically, the yield/sleep call is making them look alike, if we had removed it, there would be a marked difference in the stats.

c. batch7.txt:

	FCFS	Round-Robin
Batch Execution Time	9072	8182
Avg turnaround time	4889	8158
Avg waiting Time	3981	7340
Completion Time (Difference between max and min)	8254	45

Explanation:
The process runs to completion in FCFS, and therefore the processes scheduled first will end first, hence the average turnaround time is less. And so is the average waiting time. But the scheduling is not fair as there is a large difference in the completion time difference. In round-robin scheduling, the processes are frequently context switched out, and so they will all get small amounts of CPU time, and they will end more or less together as the process codes are basically same. So the average turnaround time is worse, but this scheduling is fair as the completion time difference is less.

2.

SJF	batch2.txt	batch3.txt
Batch Execution Time	9091	40605
Avg turnaround time	8816	31278
Avg waiting Time	7907	27217
Completion Time (Difference between max and min)	371	22455
CPU bursts	count: 60, avg: 151, max: 235, min: 1	count: 212, avg: 191, max: 308, min: 1
CPU burst estimates	count: 50, avg: 146, max: 198, min: 89	count: 202, avg: 191, max: 260, min: 1

CPU burst estimation error:	count: 50, avg: 70	error: count: 202, avg: 31

the average CPU burst estimation error per estimation instance)/(the average CPU burst length) for batch2.txt = 70 / 151 = 0.46

the average CPU burst estimation error per estimation instance)/(the average CPU burst length) for batch3.txt = 31 / 191 = 0.16

We observe that the ratio in batch3.txt is significantly lesser than the ratio in batch2.txt. The reason for this is the more no of outer loops in batch3.txt. So there are almost equal length CPU bursts for a greater no of consecutive CPU bursts in batch3 compared to batch2, so the CPU burst estimator is able to make better guesses in the case of batch3 as the estimate converges to the actual value (s(n) converges to t(n))
3.

	FCFS	SJF
Batch Execution Time	7575	6726
Avg turnaround time	7573	4933
Avg waiting Time	6816	4261
Completion Time (Difference between max and min)	6	3589
CPU bursts		count: 59, avg: 114, max: 183, min: 1
CPU burst estimates		count: 49, avg: 108, max: 175, min: 45
CPU burst estimation error:		count: 49, avg: 50

We can clearly observe that the average turn-around time for SJF is much less compared to FCFS. This is on expected lines, as we know that SJF always optimizes the average turnaround time. The batch execution time is also lesser in SJF. Also the completion time difference is huge in SJF. So it is not fair. And this is also expected as it always schedules the shortest available job, and in this process, some processes may have to wait very long in the ready queue in order to get scheduled.

One can also explain why the diff in completion time of FCFS is low, since the process yields the CPU frequently, the next process in line is scheduled, and this continues on till all the processes end, so almost all the processes end together.

4. Comparison between preemptive round-robin and preemptive UNIX:
    a. batch5.txt:

	Round-Robin	Preemptive UNIX
Batch Execution Time	10637	10642
Avg turnaround time	10623	7012
Avg waiting Time	9555	5943
Completion Time (Difference between max and min)	25	7369

Clearly the average turnaround time is much lesser in UNIX scheduler, since the processes with lower priorities are scheduled first, and they have a better chance of finishing first, thus the overall average turnaround time comes down.

    b. batch6.txt

	Round-Robin	Preemptive UNIX
Batch Execution Time	10691	10758
Avg turnaround time	10680	7130
Avg waiting Time	9610	6055
Completion Time (Difference between max and min)	36	7548

The turnaround time for UNIX is less as the process with higher priority will get finished first. And also we see a huge difference in completion time as we had processes with different priorities.

IMPLEMENTATION DETAILS:

SCHED_NPREMPT_SJF (Shortest job scheduled first)

We use the given formula to compute the expected next CPU burst time inside the yield function (RUNNING -> RUNNABLE transition), sleep function (RUNNING->SLEEPING) transition and the exit function (RUNNING->ZOMBIE) transition (though not needed here as process has ended). We measure the ticks at the beginning of the current CPU burst when the process is scheduled by the scheduler (using p->sticks) and inside the yield and sleep functions, we again obtain the ticks and store it in p->eticks, and the difference of the two gives us the length of the just ended CPU burst. We calculate the expected length of the next cpu burst s(n+1) using s(n) which is p->next_burst in the code and the above calculated CPU burst. In the scheduler, we run over all the processes and schedule the non batch processes first, then we obtain the process with the minimum expected next burst, and schedule it. And the process repeats over again.

SCHED_PREMPT_UNIX:
Similar to the above implementation, we update the CPU usage in the yield, and sleep functions but using the values as given in the instructions. In the scheduler function, we update the priority and the CPU usage as given in the instructions. And then we loop over again to pick the process with the minimum priority and schedule it.

ASSUMPTIONS:

1. For all the scheduling policies no matter what happens we first finish submitjobs then only we start the execution of other batch process so we switch off the timer interrupt for it.
2. The reading in the tables may differ on different cpu as it depends highly on the processor.