#CS330:Operating Systems
Assignment-2 report : Group 22
Report

Batch 1

Scheduling Policies	CPU utilization	Avg. Wait Time
1.Non-preemptive(default)	56.2147	24409
2.Non-preemptive(sjf)	56.2147	24409
3.Round-robin(Q1 = ¼ * 118.16)	68.0560	108278
4.Round-robin(Q2 = ½ * 118.16)	63.600	84207
5.Round-robin(Q3 = ¾ * 118.16)	60.3785	75724
6.Round-robin(Q4 = 20)	73.277	143739
7.Unix scheduler(Q1)(Q1 = ¼ * 118.16)	68.062	109194
8.Unix scheduler(Q2)(Q2 = ½ * 118.16)	63.5135	84168
9.Unix scheduler(Q3)(Q3 = ¾ * 118.16)	60.94	76287
10.Unix scheduler(Q4)(Q4 = 20)	73.733	143882

As we can quite clearly see, CPU utilization decreases on increasing the time quanta. This can be explained by examining the mechanism we are using to calculate the CPU utilization. We just calculate the ratio of total CPU bursts and total timer ticks. If quantum is not short enough, we can see that the algorithm becomes similar to default non-preemptive NachOS. If we decrease the time quanta

then total time during context switches increases while total time during i/o remains constant. Hence CPU utilization goes up.

But this much short quanta won't work well in real life because the processes given to system may not be exactly same nature as in this case. Also high CPU utilization need not be a good indicator of performance. Low time quanta will increase context switch overhead and will drastically reduce system throughput. Even though the CPU will be busy at all times, it will not get any work done.

Batch 2

Scheduling Policies	**CPU utilization**	**Avg. Wait Time**
1.Non-preemptive(default)	**82.866**	**24355**
2.Non-preemptive(sjf)	**82.866**	**24355**
3.Round-robin(Q1 = ¼ * 118.16)	**91.172**	**109146**
4.Round-robin(Q2 = ½ * 118.16)	**89.11**	**85111**
5.Round-robin(Q3 = ¾ * 118.16)	**88.19**	**74847**
6.Round-robin(Q4 = 20)	**92.98**	**143307**
7.Unix scheduler(Q1)(Q1 = ¼ * 118.16)	**91.17**	**108434**
8.Unix scheduler(Q2)(Q2 = ½ * 118.16)	**88.465**	**85309**
9.Unix scheduler(Q3)(Q3 = ¾ * 118.16)	**88.00**	**75991**
10.Unix scheduler(Q4)(Q4 = 20)	**93.26**	**143101**

Batch 3

	CPU utilization	Avg. Wait Time
1.Non-preemptive(default)	**94.724**	**24355**
2.Non-preemptive(sjf)	**94.724**	**24355**
3.Round-robin(Q1 = ¼ * 118.16)	**99.341**	**108798**
4.Round-robin(Q2 = ½ * 118.16)	**98.77**	**84552**
5.Round-robin(Q3 = ¾ * 118.16)	**98.90**	**73886**
6.Round-robin(Q4 = 20)	**99.36**	**142924**
7.Unix scheduler(Q1)(Q1 = ¼ * 118.16)	**99.35**	**107993**
8.Unix scheduler(Q2)(Q2 = ½ * 118.16)	**99.46**	**85282**
9.Unix scheduler(Q3)(Q3 = ¾ * 118.16)	**98.63**	**75315**
10.Unix scheduler(Q4)(Q4 = 20)	**99.54**	**142812**

Batch 4

	CPU utilization	Avg. Wait Time
1.Non-preemptive(default)	**99.86**	**36577**
2.Non-preemptive(sjf)	**99.86**	**36577**
3.Round-robin(Q1 = ¼ * 118.16)	**99.90**	**110763**
4.Round-robin(Q2 = ½ * 118.16)	**99.87**	**87573**
5.Round-robin(Q3 = ¾ * 118.16)	**99.86**	**81957**
6.Round-robin(Q4 = 20)	**99.92**	**144916**
7.Unix scheduler(Q1)(Q1 = ¼ * 118.16)	**99.90**	**111270**
8.Unix scheduler(Q2)(Q2 = ½ * 118.16)	**99.88**	**87858**
9.Unix scheduler(Q3)(Q3 = ¾ * 118.16)	**99.87**	**82100**

10.Unix scheduler(Q4)(Q4 = 20)	**99.93**	**145801**

PART II

Scheduling Policies used in Batch 5	**Average waiting time in the ready queue**
1	55450
2	40195

Explanation:

In default NachOS non preemptive scheduling algorithm, the processes are scheduled on First Come First Serve basis. Each process runs an iteration of outerloop and then yields the CPU. Now, the difference between testloop4 and testloop5 is the length of the inner loop. The order in which the first algorithm schedules the processes is fixed. But as we have seen in class, that shortest job first algorithm is provably optimal for minimizing average waiting time. testloop5 has shorter bursts hence it should be optimal to schedule it first if we want to reduce the avg. waiting time. Algorithm 2 does exactly that. It tries to predict the shortest job first and and thus reduces the average waiting time in ready queue.

PART III

Quanta used : 100

	Overall estimation error in CPU burst OUTER_BOUND = 4	Overall estimation error in CPU burst OUTER_BOUND = 8
Batch - 1	**81.59**	**45.09**
Batch - 2	**86.23**	**46.18**
Batch - 3	**73.69**	**40.21**
Batch - 4	**97.03**	**98.5**
Batch - 5	**62.08**	**32.95**

As we see, prediction error reduces by huge margin if we double the OUTER_BOUND. This is because batch-1, batch-2, batch-3, batch-5 periodically yield cpu after some time (either by calling YieldCPU or by an IO command). Thus there more number of context switches and SJF gets more chance to predict. Thus prediction error decreases as we make more predictions. Batch-4 has testloop3 ten times, which doesn't yield the cpu at any point. Thus we get only one chance to predict irrespective of OUTER_BOUND. Hence, prediction error remains same and will most probably be quite large.

PART IV

***(To run Part 4, uncomment line 125 in userprog/progtest.cc)**

(for job completion times)	Round - robin	Unix scheduling
Maximum	256234	202873
Minimum	250184	61824
Average	254826	136149
Variance	2932556.0000	2241989918.5000

As we can clearly see unix-scheduling outperforms round robin in every case. Unix honours priorities hence it tries to finish the process with highest priority as early as possible leading to very small minimum job completion time. Round robin on the other hand gives equal amount of quantum to each process. Thus each process has more or less same job completion time which is very high. This also means that average job completion time of unix-scheduling will be smaller than round-robin.

However variance of round robin will be much less than unix as all the processes take almost same time.