OS2 Assignment Report

KTHYUS001 – Yusuf Kathrada

CSC3002F

Aim

The aim is to compare and analyse two CPU scheduling algorithms namely First-Come-First-Serve (FCFS) and Round Robin (RR). To do this, a Java software framework was provided which simulated each algorithm as a kernel. A workload of 18 processes was given to each kernel. Thus, the algorithms could then be assessed based on metrics such as CPU utilisation, average turnaround time and average waiting time.

Method

The simulator outputted a trace file for each algorithm based on the context switch time (CST) and system call time (SCT) which could be varied for both algorithms as well as variable time quanta which could be adjusted for the RR policy. In order to extract the data from the trace files, a python script was made which scraped the data from the trace files to calculate the average turnaround times and average waiting times. The CPU utilisation was retrieved from executing the provided data files. Thus, the python scripts automated the process of data collection which could then be further analysed.

Results

Table A

	CPU Utilisation (%)			
Context Switch Time	FCFS	RR (Quanta = 50)	RR (Quanta = 150)	RR (Quanta = 250)
10	91.38	73.21	86.08	88.95
15	89.45	67.58	82.83	86.4
20	87.6	62.76	79.81	83.97
25	85.82	58.56	76.99	81.68

Note: System call time was kept constant at 5.

Table B

	Average Turnaround Time (virtual time units)					
Quanta	RR (CST = 5)	RR (CST = 10)	RR (CST = 15)	FCFS (CST = 5)	FCFS (CST = 10)	FCFS (CST = 15)
25	3.63	4.24	4.85	3.99	4.08	4.18
50	3.32	3.64	3.97	3.99	4.08	4.18
75	3.21	3.44	3.67	3.99	4.08	4.18
100	3.21	3.39	3.57	3.99	4.08	4.18
200	3.33	3.45	3.58	3.99	4.08	4.18
300	3.47	3.58	3.69	3.99	4.08	4.18
400	3.64	3.74	3.84	3.99	4.08	4.18
500	3.75	3.85	3.94	3.99	4.08	4.18
600	3.81	3.91	4.00	3.99	4.08	4.18
700	3.93	4.02	4.11	3.99	4.08	4.18
800	3.99	4.08	4.18	3.99	4.08	4.18
900	3.99	4.08	4.18	3.99	4.08	4.18
1000	3.99	4.08	4.18	3.99	4.08	4.18

Note:

1. System call time was kept constant at 1.
2. Values were divided by 10^5.

Table C

	Average Waiting Time (virtual time units)					
Quanta	RR (CST = 5)	RR (CST = 10)	RR (CST = 15)	FCFS (CST = 5)	FCFS (CST = 10)	FCFS (CST = 15)
25	3.34	3.95	4.56	3.72	3.81	3.90
50	3.03	3.36	3.69	3.72	3.81	3.90
75	2.93	3.16	3.39	3.72	3.81	3.90
100	2.92	3.11	3.29	3.72	3.81	3.90
200	3.05	3.17	3.3	3.72	3.81	3.90
300	3.2	3.3	3.41	3.72	3.81	3.90
400	3.36	3.46	3.56	3.72	3.81	3.90
500	3.47	3.57	3.67	3.72	3.81	3.90
600	3.53	3.63	3.72	3.72	3.81	3.90
700	3.65	3.74	3.83	3.72	3.81	3.90
800	3.72	3.81	3.90	3.72	3.81	3.90
900	3.72	3.81	3.90	3.72	3.81	3.90
1000	3.72	3.81	3.90	3.72	3.81	3.90

Note:

1. System call time was kept constant at 1.
2. Values were divided by 10^5.

Analysis

Figure 1

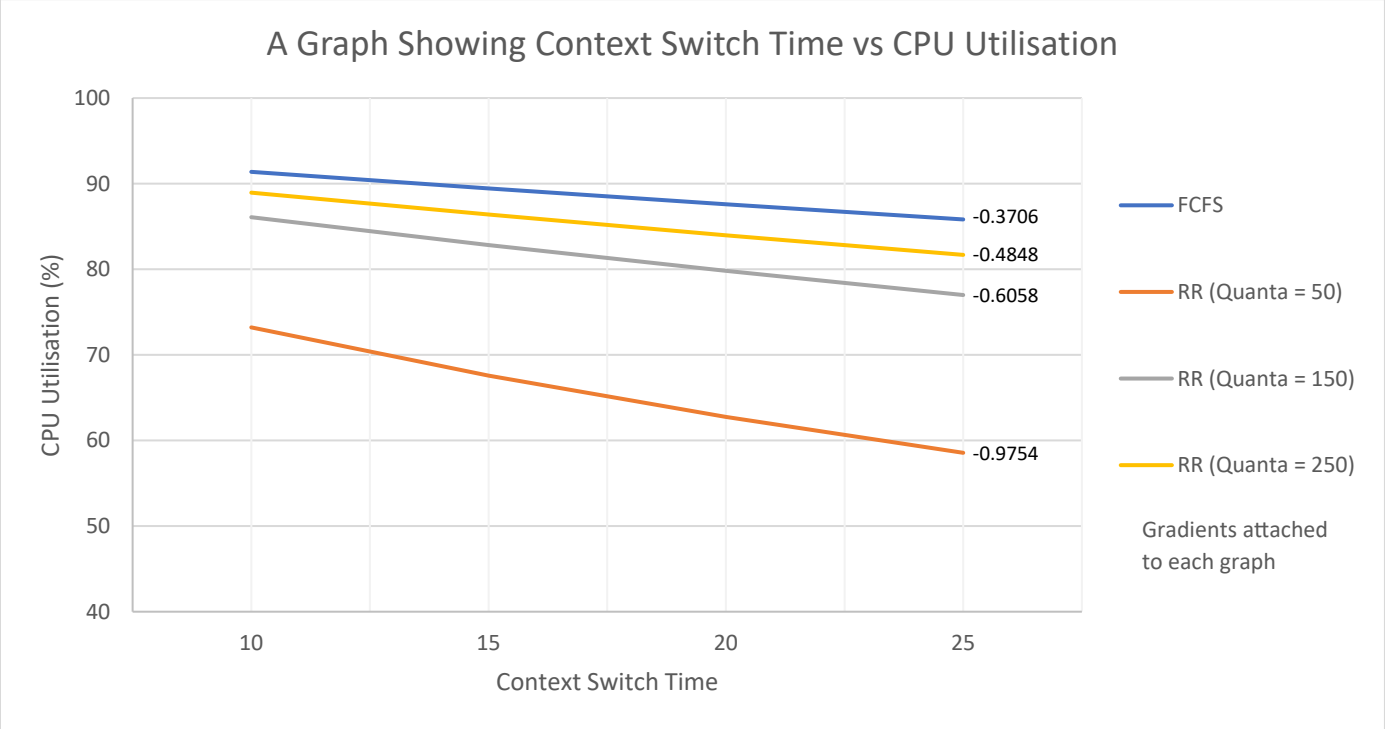


Figure 1 shows a graph which displays the effects of varying the context switch time on CPU utilization. It is important to note that the system call time was kept constant at 5. There are three critical trends which can be seen in Figure 1. The first trend is that there is an inverse relationship between context switch time and CPU utilisation. Therefore, the higher the context switch time the lower the recorded CPU utilisation was. This was seen in all four graphs in Figure 1, the three RR simulations where each had a different time quantum which was kept constant, as well as the for the FCFS simulation. The second trend to note is that the lower the time quantum for the RR algorithm, resulted in a steeper gradient of the curve. This is reinforced by the gradients which are attached to each graph. It can be seen that the lower the time quantum was, the stronger the negative correlation was observed between context switch time and CPU utilisation. The FCFS policy demonstrated the weakest negative correlation to which the RR algorithms tended towards as the quantum increased. This brings to light the final important trend which is that as the time quantum for the RR policy increases, the RR policy tends to degenerate into FCFS. This could be due to the fact that the time quantum is approaching a size which allows most processes to finish running before they are kicked off the CPU. Unlike when processes get removed from the CPU to be later executed, as they were not completed within the time quantum, and the next process in the queue is given a chance to run in a typical RR fashion. Finally, we can observe the best CPU utilisation at low context switch times for all algorithms and that FCFS tends to produce the highest CPU utilisation due to the lower cost of overheads from context switching between processes.

Figure 2

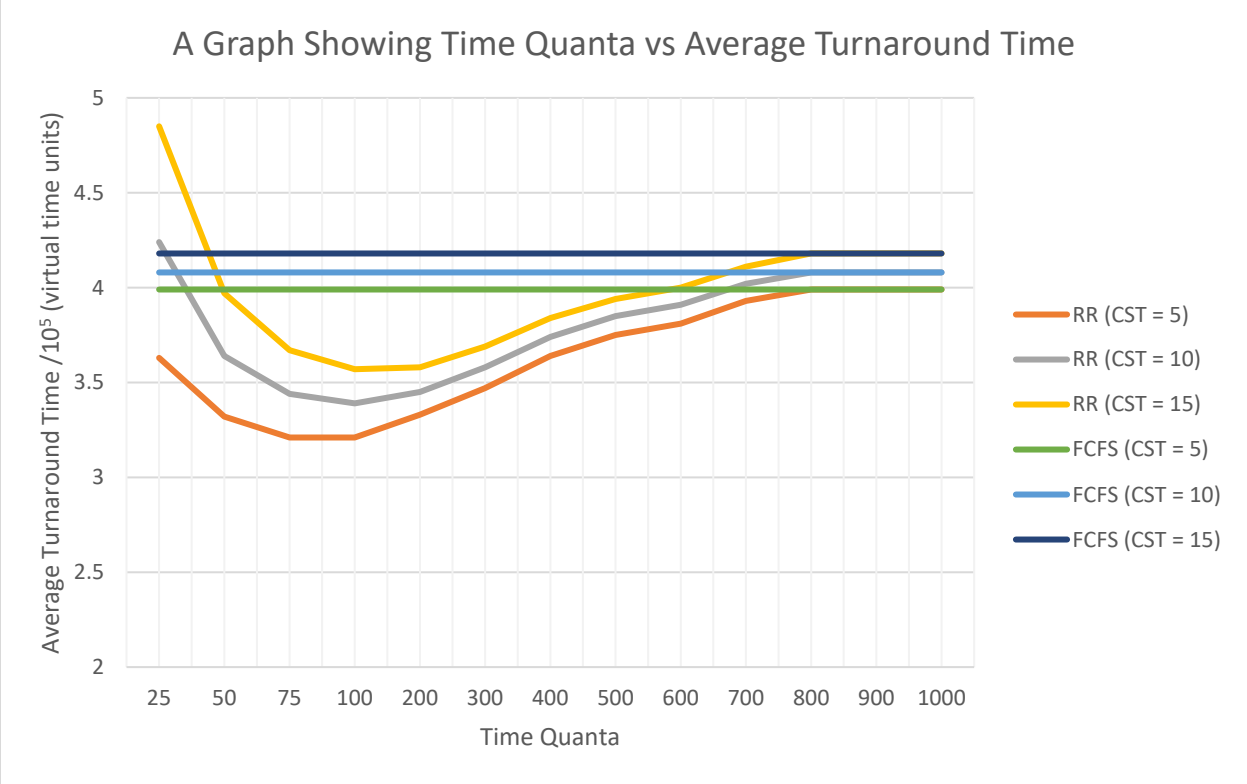
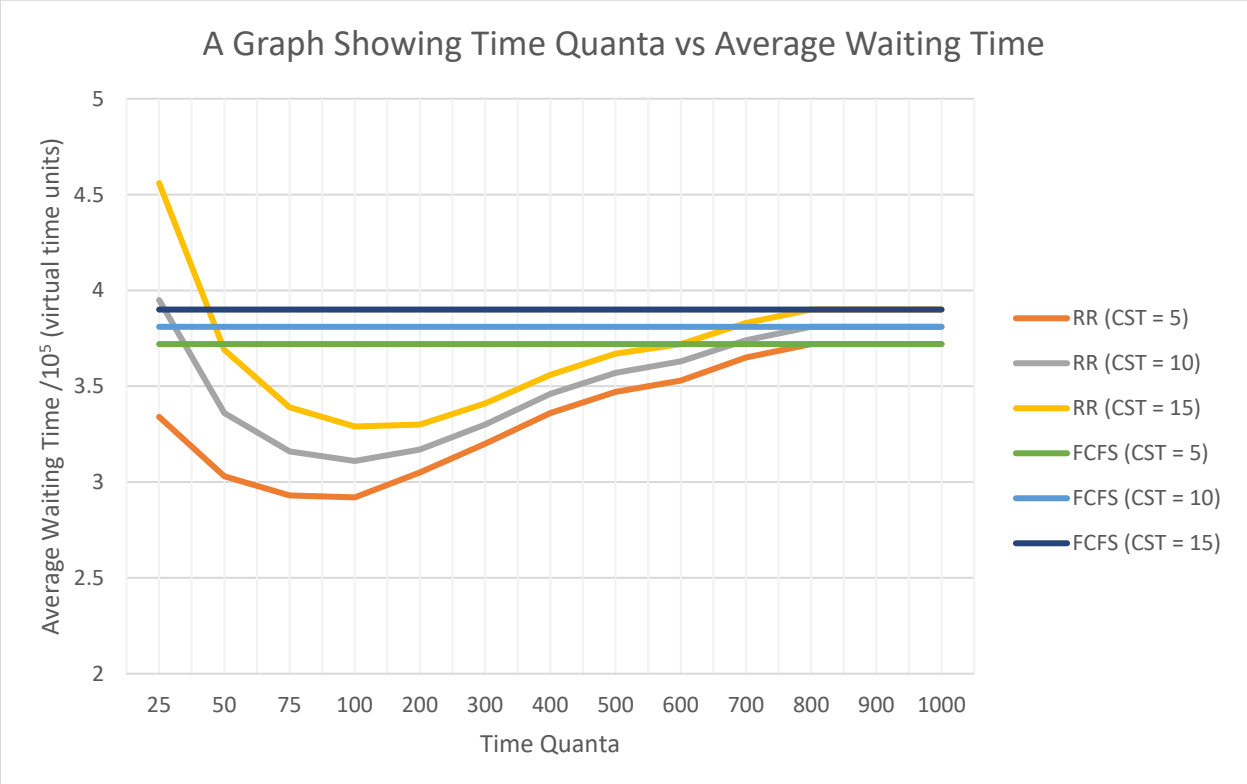


Figure 3



Figures 2 and 3 display the effects of varying the time quanta on the average turnaround times and average waiting times respectively. It should be noted that in both figures, the times were divided by 10^5 for clarity and that the system call costs were constant at 1. The FCFS simulations were added for comparison however it is known that varying the time quanta does not affect FCFS, hence the constant graphs. Both figures are near identical which suggests a correlation between the average turnaround times and average waiting times. From the figures it can be observed that the lower the context switch time was, the lower the average turnaround times and average waiting times were, suggesting a direct relationship. This is seen in both RR and the FCFS algorithms.

A clear trend can be seen in the RR graphs in both figures. At a small time quantum, the RR simulations performed poorly, worse than the FCFS policy, and the average turnaround times and waiting times recorded were high. However, as the time quantum entered the range of 75-100, this provided the best average turnaround times and average waiting times. The best average turnaround time recorded was 3.21×10^5 virtual time units, recorded in the RR graph with a context switch cost of 5, and the best average waiting time recorded was 2.92×10^5 virtual time units, recorded in the RR graph with a context switch time of 5 as well. This suggests that the optimal quanta for the RR schedule for this simulation is in that range of 75-100, specifically it was found to be 87. This is further reinforced by the results observed in Figure 1, as the figure suggests that at a time quantum of between 75-100 will result in good CPU utilisation as well leading to a well-balanced performing CPU scheduling algorithm. Moving forward, as the time quanta in figures 2 and 3 increase after 100, the average turnaround times and average waiting times steadily increase up until a time quantum of 800. At this point, the RR algorithms degenerate into FCFS, with RR performance matching that of its FCFS counterpart with the same context switch time and thus will remain constant from 800 onwards.

These results therefore show that if the time quantum selected is too small, the cost of context switching will lead to poor performance and that if the time quantum is too large, the RR policy will degenerate into FCFS policy.

Conclusion

This report demonstrated the effects of context switch time and system call time on CPU utilisation, average turnaround time and average waiting time for a First-Come-First-Serve and Round Robin algorithm. It was found that there is an inverse relationship between context switch time and CPU utilisation and that the best CPU utilisation was found in the FCFS algorithm at low context switch time. A direct relationship was found for context switch time on average turnaround time and average waiting time as the higher the context switch time was, the higher the average turnaround time and average waiting times were (higher in this case means worse performance). The optimal time quantum was found to be in the range of 75-100 as this produced the best average turnaround time and average waiting time whilst also maintaining good CPU utilisation. Further, it was found that if the time quantum is too small the Round Robin algorithm produced average waiting times and average turnaround times worse than that of First-Come-First-Serve. Conversely, if the time quantum was too large, the Round Robin policy would degenerate into First-Come-First-Serve.