I.J. Education and Management Engineering, **2015, 2, 10-19**
Published Online June 2015 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijeme.2015.02.02

Available online at http://www.mecs-press.net/ijeme

Efficient Round Robin Scheduling Algorithm with Dynamic Time Slice

Lipika Datta [a]

[a] *Computer Science and Engineering Department, College of Engineering and Management Kolaghat, K.T.P.P. Township, Purba Medinipur, W.B., PIN: 721171, India*

Abstract

Round Robin (RR) scheduling algorithm is the widely used scheduling algorithm in multitasking. It ensures fairness and starvation free execution of processes. Choosing the time quantum in RR algorithm is very crucial as small time slice results in large number of context switches and large time quantum increases the response time. To overcome these problems of RR scheduling, instead of static time slice dynamic time slice can be used to get optimal performance. The objective of this paper is to modify RR algorithm by adjusting time slices of different rounds depending on the remaining CPU bursts of currently running processes and considering their waiting times until that round in respect of the other processes' waiting times. Experimental analysis reveals that the proposed algorithm produces better average turnaround time, average waiting time and fewer number of context switches than existing algorithms.

Index Terms: Operating System, Scheduling, Round Robin Algorithm, Dynamic Time Slice, Context switch, Turnaround time, Average Waiting time.

© 2015 Published by MECS Publisher. Selection and/or peer review under responsibility of the Research Association of Modern Education and Computer Science.

1. Introduction

Modern operating systems support multitasking environment in which processes run in a concurrent manner. In a single-processor system, only one process can run in the CPU at a time. Others processes in the ready queue must wait until the CPU becomes free. The operating system must decide through the scheduler the order of execution of the processes in ready state. The objective of multiprogramming is to have some process running at all times to maximize CPU utilization. Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before using. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design. CPU scheduling determines which processes run when there are multiple run-able processes. CPU scheduling is important because it can have a

* Corresponding author.
E-mail address: lipika.datta@cemk.ac.in

big effect on resource utilization and the overall performance of the system. In general we want to optimize the behaviour of the system.

The goals of scheduling may be categorized as user based scheduling goals and system based scheduling goals [1]. User based goals are the criteria that benefit the user. Some User-based scheduling goals are:

- Turnaround Time: The time elapsed between the submission of a job and its termination is called the turnaround time.
  $t_r = wt + x$
  where t_r is turnaround time of a process
  wt is waiting time of the process in the ready queue.
  x is the execution time of the process.
  The scheduling algorithm should be designed such that turnaround time is minimized.
- Waiting Time: The time spent by the process in the ready queue is the waiting time. The scheduling algorithm should be designed such that waiting time is less.
- Response Time: It is the time interval between the time of submission of a process and the first response given by the process to the user. The scheduling algorithm should be designed such that the response time is within an acceptable range.
- Predictability: The algorithm should take care that a process does not take too long in processing as compared to the predictable behaviour of the process.
- Deadlines: The scheduling algorithm should be designed such that real-time processes will execute within their deadlines.

Some system-based scheduling goals are:

- Throughput: Throughput is the number of processes completed in a unit time. The scheduling algorithm should be designed in such a way that throughput in a system is maximized.
- CPU Utilization: It is the percentage of time that the CPU is busy in executing a process. The fundamental goal of scheduling is to keep the processor busy all the time.
- Fairness: All processes in the system should be treated in the same way unless there is some preference or priority for a specific process. In that case also processes with lower priority should not be ignored to avoid starvation.
- Context Switch: Context switching is the procedure of storing the state of an active process and restoring the state of another process for the CPU when it has to start executing the later process. Context switch is total overhead to the system and leads to wastage of CPU time. The scheduling algorithm should be designed such that the context switch is minimized.

So we can conclude that a good scheduling algorithm for real time and time sharing system must possess following characteristics:

- Minimum context switches.
- Maximum CPU utilization.
- Maximum throughput.
- Minimum turnaround time.
- Minimum waiting time.
- Minimum response time.

In this paper a new version of Round Robin scheduling algorithm is introduced which calculates the time slice of each pass of the RR scheduling algorithm considering the remaining CPU burst times of the currently

executing processes. In each pass, the precedence factor for each process is also calculated. Precedence factor depends on the ratio of the remaining CPU burst of a process and the relative waiting time of the process until that round and it is used to decide the order of execution of the processes in that round. This modification makes the RR scheduling algorithm effective for real time task scheduling by reducing average turnaround time, average waiting time and number of context switches.

2. Related work done

In the last few years different approaches are used to enhance the performance of Round Robin scheduling. Research has been conducted to achieve good fairness in a dynamic environment while having a low scheduling overhead [2]. In [3] the authors have arranged the processes in ascending order of burst time. Time slice is chosen as the CPU burst of the mid process in case of odd number of processes, otherwise time slice is equal to the average CPU burst of all running processes. In [4] the authors have proposed an algorithm (ORR) executed in three phases. In first phase the processes execute according to RR scheduling with an initial time quantum. Then in the next round the time quantum doubles and the remaining processes are scheduled according to RR scheduling. In next phase they select the shortest process to execute. In [5] the authors have selected the median process and the time slice is calculated as the time slice of the median process plus the number of processes. In [6] authors have introduced weighted dynamic RR scheduling for scheduling cells in ATM switches to reduce waiting time by considering delay in each queue. In [7] the authors have assigned weights to processes. They have assigned more weight to the process with small CPU burst and they have also considered the waiting time of the processes for I/O and accordingly modified their weights. For heterogeneous processes i.e. CPU bursts of some processes are very smaller or larger than other processes, time quantum for RR scheduling is calculated using Arithmetic mean and Harmonic mean respectively in [8]. Time slice for different rounds of RR algorithm is dynamically calculated depending on the remaining CPU bursts of currently running processes in [9]. In [10] in DQRRR algorithm time Quantum is dynamically calculated for each round as the remaining CPU burst of the median process. In first round the processes are sorted according to the ascending order of their CPU burst time and in the following rounds they are arranged in the lowest followed by highest CPU burst processes among the currently running processes. Most of the aforesaid algorithms do not consider the waiting time of a process while calculating time slice for the next round of the RR scheduling.

2.1. Organization of paper

Section 3 presents the illustration of my proposed algorithm. In section 4, Experimental results and its comparison with existing algorithms is presented. Section 5 contains the conclusion.

3. Proposed approach

The proposed algorithm eliminates the drawbacks of implementing simple round robin architecture in real time system by introducing a concept of assigning different time quantum to different rounds of RR scheduling algorithm. At the beginning of each round of the RR algorithm the following matrices are calculated for each process:

$$\text{RRB} = \frac{\text{Remaining CPU burst}}{\text{Original CPU burst}} \tag{1}$$

$$WR = \frac{\text{Waiting time of the process till now}}{\text{Total waiting time of all currently running processes till now}} \quad (2)$$

$$\text{Precedence Factor (PF)} = \frac{WR}{RRB} \quad (3)$$

The proposed algorithm eliminates the drawbacks of implementing simple round robin architecture in real time system by introducing a concept of assigning different time quantum to different rounds of RR scheduling algorithm. At the beginning of each round of the RR algorithm the following matrices are calculated for each process:

3.1. Proposed algorithm

Input: Process id, Burst time.
Output: Average turnaround time, Average waiting time, Number of Context Switches.

1. Let BT_i be the CPU burst of process P_i.
   TQ be the time quantum
   RBT_i be the remaining CPU burst of P_i
   WT_i be the waiting time of P_i till that time
   TWT_i be the total waiting time of all the currently running processes till that time.
   PF_i be the precedence factor.
2. If number of processes in ready queue is more than one (suppose n) sort them in ascending order of their CPU burst time.
3. TQ = $\sum_{i=1}^{n} \frac{BTi}{n}$
4. Execute the processes as per RR scheduling with TQ time quantum.
5. If $BT_i < TQ$ delete process P_i from ready queue.
   Else $RBT_i = BT_i - TQ$ and calculate WT_i
6. If ready queue! = NULL execute following steps. Suppose k be the number of processes present in ready queue.
7. Calculate $RRB_i = \frac{RBTi}{BTi}$
8. If $\forall i, j, 1 \leq i \leq k, 1 \leq j \leq k$, if $RRB_i = RRB_j$ sort the processes in ascending order of their RBT. Go to step 13.
9. Calculate $TWT_i = \sum_{i=1}^{k} WTi$
10. Calculate $PF_i = \frac{WTi}{RRBi}$
11. For $\forall i, 1 \leq i \leq k$ sort the processes in descending order of PF_i
12. TQ = $\left\lceil \left(\sum_{i=1}^{k} \frac{RBTi}{k}\right) \right\rceil$
13. Schedule the processes in the ready queue according to RR algorithm with time quantum TQ.
14. Calculate $RBT_i = RBT_i - TQ$ and calculate WT_i, for $\forall i, 1 \leq i \leq k$
15. Repeat steps 7 to 15
16. If new process arrives, after expiry of current time quantum goto step 7.
17. Calculate average waiting time, average turnaround time and number of context switches for each process.

End

4. Experimental results

4.1. Assumption

Experiments are performed in single processor environment and on independent processes. All the parameters like number of processes, and burst time of all the processes are known before submitting the processes to the processor. All processes are CPU bound and none I/O bound. Context switching overhead and time taken for calculating the time slices are ignored while calculating average turnaround time and average waiting time.

4.2. Data set

To compare the performance of the algorithm with the algorithms described in [10] (DQRRR) and six different data sets are taken. For first three cases arrival times of processes are considered with zero and next three cases are with non-zero arrival times. Again comparison is done between algorithms introduced in [4] (ORR) with two sets of data, one with zero arrival time and another with non-zero arrival time.

4.2.1. Same data set applied to DQRRR and proposed algorithm

4.2.1.1. Data set with zero arrival time

Case a: Processes with increasing Burst Time:

Table 1. Inputs for case 4.2.1.1.a

Process id	Arrival time	Burst time
P1	0	30
P2	0	42
P3	0	50
P4	0	85
P5	0	97

P1	P2	P3	P4	P5	P5	P4	P5	
0	30	72	122	183	244	247	298	304

Fig.1. Gantt chart for case 4.2.1.1.a

Table 2.Comparison between algorithms for case 4.2.1.1.a

Algorithm	Average Turnaround Time	Average Waiting Time	No. of Context Switch
DQRRR	195.2	134.4	7
Proposed Algorithm	165.2	104.4	6

Case b: Processes with decreasing burst time:

Table 3. Inputs for case 4.2.1.1.b

Process id	Arrival time	Burst time
P1	0	105
P2	0	90
P3	0	60
P4	0	45
P5	0	35

P5	P4	P3	P2	P1	P1	P2	P1	
0	35	80	140	207	274	305	328	335

Fig.2. Gantt chart for case 4.2.1.1.b

Table 4.Comparison between algorithms for case 4.2.1.1.b

Algorithm	Average Turnaround Time	Average Waiting Time	No. of Context Switch
DQRRR	219.4	152.4	7
Proposed Algorithm	183.6	116.6	7

Case c: Processes with Random Burst Time

Table 5. Inputs for case 4.2.1.1.c

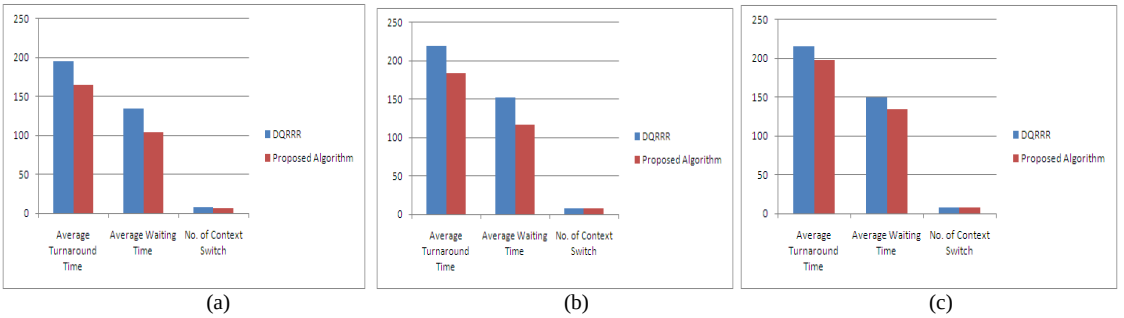
Process id	Arrival time	Burst time
P1	0	92
P2	0	70
P3	0	35
P4	0	40
P5	0	80

P3	P4	P2	P5	P1	P2	P5	P1	P1	
0	35	75	139	203	267	273	289	314	317

Fig.3. Gantt chart for case 4.2.1.1.c

Table 6.Comparison between algorithms for case 4.2.1.1.c

Algorithm	Average Turnaround Time	Average Waiting Time	No. of Context Switch
DQRRR	215.6	150.2	7
Proposed Algorithm	197.8	134.4	8



(a) (b) (c)

Fig.4. Analysis of performance among algorithms (a) (case 4.2.1.1.a), (b) (case 4.2.1.1.b), (c) (case 4.2.1.1.c)

4.2.1.2. Data set with non-zero arrival time

Case a: Processes with increasing Burst Time:

Table 7. Inputs for case 4.2.1.2.a

Process id	Arrival time	Burst time
P1	0	28
P2	2	35
P3	6	50
P4	6	82
P5	8	110

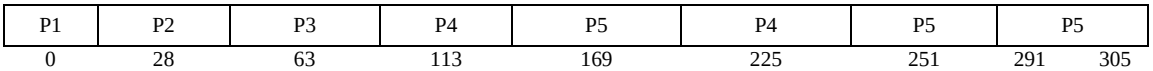


Fig.5. Gantt chart for case 4.2.1.2.a

Table 8.Comparison between algorithms for case 4.2.1.2.a

Algorithm	Average Turnaround Time	Average Waiting Time	No. of Context Switch
DQRRR	173.2	112.2	7
Proposed Algorithm	152	94.6	7

Case b: Processes with decreasing Burst Time:

Table 9. Inputs for case 4.2.1.2.b

Process id	Arrival time	Burst time
P1	0	80
P2	2	72
P3	3	65
P4	4	50
P5	5	43

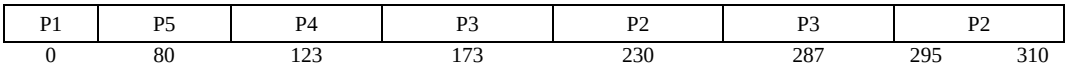


Fig.6. Gantt chart for case 4.2.1.2.b

Table 10.Comparison between algorithms for case 4.2.1.2.b

Algorithm	Average Turnaround Time	Average Waiting Time	No. of Context Switch
DQRRR	209.8	147.8	7
Proposed Algorithm	196.2	131.4	6

Case c: Processes with Random Burst Time:

Table 11. Inputs for case 4.2.1.2.c

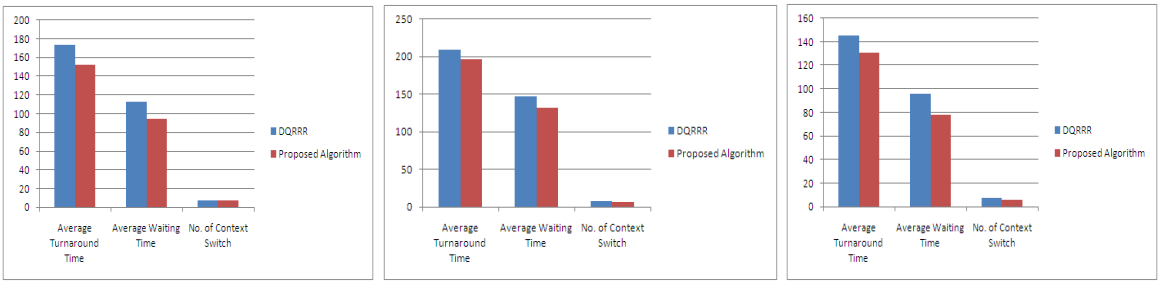
Process id	Arrival time	Burst time
P1	0	26
P2	1	82
P3	2	70
P4	5	31
P5	7	40

P1	P4	P5	P3	P2	P3	P2	
0	26	57	97	153	209	223	249

Fig.7. Gantt chart for case 4.2.1.2.c

Table 12.Comparison between algorithms for case 4.2.1.2.c

Algorithm	Average Turnaround Time	Average Waiting Time	No. of Context Switch
DQRRR	145.4	95.6	7
Proposed Algorithm	130.4	77.6	6



(a) (b) (c)

Fig.8. Analysis of performance among algorithms (a) (case 4.2.1.2.a), (b) (case 4.2.1.2.b), (c) (case 4.2.1.2.c)

4.2.2. Same data set applied to ORR and proposed algorithm

4.2.2.1. Data set with zero arrival time

Table 13. Inputs for case 4.2.2.1

Process id	Arrival time	Burst time
P1	0	22
P2	0	18
P3	0	9
P4	0	10
P5	0	5

P5	P3	P4	P2	P1	P2	P1	
0	5	14	24	37	50	55	64

Fig.9. Gantt chart for case 4.2.2.1

Table 14.Comparison between algorithms for case 4.2.2.1

Algorithm	Average Turnaround Time	Average Waiting Time	No. of Context Switch
ORR	42.6	29.8	9
Proposed Algorithm	32.4	19.6	6

4.2.2.2. Data set with non-zero arrival time

Table 15. Inputs for case 4.2.2.2

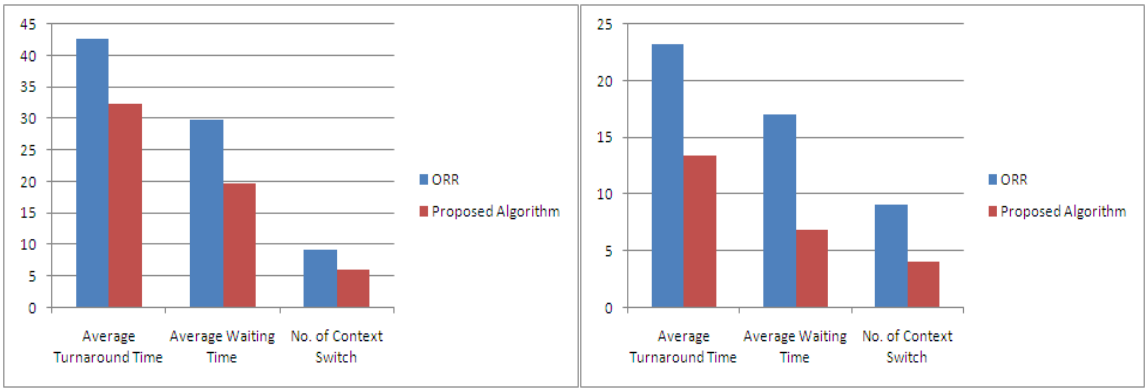
Process id	Arrival time	Burst time
P1	0	4
P2	2.004	7
P3	5.010	5
P4	6.02	8
P5	8.019	9

P1	P2	P3	P4	P5	
0	4	11	16	24	33

Fig.10. Gantt chart for case 4.2.2.2

Table 16.Comparison between algorithms for case 4.2.2.2

Algorithm	Average Turnaround Time	Average Waiting Time	No. of Context Switch
ORR	23.2	17	9
Proposed Algorithm	13.39	6.79	4



(a) (b)

Fig.11. Analysis of performance among algorithms (a) (case 4.2.2.1), (b) (case 4.2.2.2)

Figure 4 and Figure 6 graphically represent the performances of the DQRR and proposed algorithm in terms of Average Turnaround Time, Average Waiting Time and number of context switching considering zero arrival parameters considering zero and non-zero arrival time respectively. Figure 11 graphically represents the performances of the ORR and the proposed algorithm in terms of the same parameters.

5. Conclusion

A comparative study of Dynamic Quantum with Re-Adjusted Round Robin Scheduling algorithm, Optimized RR algorithm and proposed one is made. It is concluded that the proposed algorithm is superior to the other two algorithms as it has less waiting time, less turnaround time and usually less context switching thereby reducing the overhead and saving of memory space. Future work can be based on this algorithm modified and implemented for hard real time system where deadlines of the processes are to be taken into consideration.

References

[1] Principles of Operating System, Naresh Chauhan, Oxford University Press, 2014.
[2] Kevin Jeffay, F. Donelson Smith, Arun Moorthy, James Anderson, "Proportional Share Scheduling of Operating System Services for Real-Time Applications", In Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.
[3] Saroj Hiranwal, Dr. K.C. Roy, "Adaptive Round Robin Scheduling using Shortest Burst Approach Based on Smart Time Slice", International Journal of Computer Science and Communication Vol. 2, No. 2, July-December 2011, pp. 319-323.
[4] Ajit Singh, Priyanka Goyal, Sahil Batra, "An Optimized Round Robin Scheduling Algorithm for CPU Scheduling", International Journal on Computer Science and Engineering Vol. 02, No. 07, 2010, 2383-2385.
[5] Chhayanath Padhy, Dillip Ranjan Nayak, "Revamped Round Robin Scheduling Algorithm", Journal of Engineering Computers & Applied Sciences (JECAS) ISSN No: 2319-5606 Volume 3, No.4, April 2014.
[6] Taeck-Geun Kwon, Sook-Hyang Lee, and June-Kyung Rho, "Scheduling Algorithm for Real-Time Burst Traffic using Dynamic Weighted Round Robin", R&D Lab., LG Information & Communications, LTD., 1998 IEEE.
[7] Tarek Helmy**,** Abdelkader Dekdouk, "Burst Round Robin as a Proportional - Share Scheduling Algorithm", IEEEGCC, http://eprints.kfupm.edu.sa/1462.
[8] Ashkan Emami Ale Agha, Somayyeh Jafarali Jassbi," A New Method to Improve Round Robin Scheduling Algorithm with Quantum Time Based on Harmonic-Arithmetic Mean (HARM)", I.J. Information Technology and Computer Science, 2013, 07, 56-62 Published Online June 2013 in MECS (http://www.mecs-press.org/).
[9] Rami J. Matarneh," Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes", American Journal of Applied Sciences 6 (10): 1831-1837, 2009 ISSN 1546-9239 © 2009 Science Publications.
[10] H.S.Behera, R. Mohanty, Debashree Nayak," A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis", International Journal of Computer Applications (0975 – 8887) Volume 5– No.5, August 2010.

Author(s) Profile

Lipika Datta: Lipika Datta is presently working as an Assistant Professor in Computer Science and Engineering department in College of Engineering and Management, Kolaghat, West Bengal, India. She received B.E. degree from Regional Engineering College Durgapur, W.B. and M.E. degree from West Bengal University of Technology. She has 9 years teaching experience. She has published several research papers in international journal and conferences. Her research interest includes CPU scheduling and load balancing.

How to cite this paper: Lipika Datta,"Efficient Round Robin Scheduling Algorithm with Dynamic Time Slice", IJEME, vol.5, no.2, pp.10-19, 2015.DOI: 10.5815/ijeme.2015.02.02