Prioritized Fair Round Robin Algorithm with Variable Time Quantum

Arfa Yasin[1], Ahmed Faraz[1], Saad Rehman[2]

[1] *Center for Advanced Studies in Engineering (CASE), Islamabad, Pakistan*
[2]*National University of Sciences and Technology (NUST), Islamabad, Pakistan*
engr.arfa.yasin@gmail.com, a.faraz009@gmail.com, saadrehman@ceme.nust.edu.pk

***Abstract*—The performance of the time sharing systems and multiprocessor systems is greatly dependent on CPU scheduling algorithms. Some of the famous algorithms are Shortest Job First (SJF), First Come First Served (FCFS), Priority Scheduling and Round Robin (RR). Round Robin is the preferable choice for time shared systems. The main aim of this paper is to formulate a new methodology for RR algorithm that enhances the performance of the time sharing systems by reducing the waiting time, turnaround time and number of context switches. The proposed algorithm Prioritized Fair Round Robin (PFRR) with variable time quantum is a combination of SJF, Priority Scheduling, and RR with variable time quantum. The idea is to set the time quantum value according to the priority and burst times of the processes in the ready queue. The algorithm is designed for both equal priority processes and different priority processes.**

***Index Terms*—CPU Scheduling Algorithm for time sharing systems, Improved Round Robin Algorithm, Waiting Time, Turnaround Time, Number of context switches in Round Robin algorithm**

I. Introduction

SCHEDULING is a fundamental function for an operating system. All resources must be scheduled before their use and so CPU, being a most essential resource must be scheduled [6]. Nowadays, operating systems focus on multitasking environments in which round robin algorithm plays its part. CPU scheduling needs that all processes allocate the CPU fairly and processes do not get into starvation.

Scheduler is part of the kernel and it selects the process. Three distinct schedulers are short term scheduler, long term scheduler, and medium term scheduler [6]. Long term scheduler selects any process from the job pool and put it in main memory in the ready queue. Short term scheduler allocates the CPU to a process that it selects from the ready queue. Medium term scheduler performs swapping to reduce the degree of multiprogramming.

The selection criteria of a CPU scheduling algorithm depend upon the following [6]:

1) *Fairness*: All processes must fairly get the CPU and no one gets into starvation.
2) *CPU Utilization*: CPU should remain busy for 100% time.
3) *Throughput*: Increase the number of processes that have finished their execution within a certain time interval.
4) *Response time*: It must be kept minimum. It is the time when request is submitted for the process till the first response of the process is produced.
5) *Waiting Time*: It is the time a process spends in the ready queue. It must be kept minimum.
6) *Turnaround Time*: It is the time from submission of the request till the time it is completed. It must be kept minimum.
7) *Context Switch*: When a process is preempted, its context is stored so that it can resume later from the same point. It is totally an overhead because CPU does no useful work during context switch. Also it adds overhead for the scheduler. Hence, context switches should be made minimum.

A. CPU Scheduling Algorithms

In First Come First Served algorithm, the process which comes in the ready queue first, will allocate the CPU first. Here average waiting time is long. In Shortest Job First algorithm, shortest CPU burst time process executes first. Here average waiting time is optimal [6]. FCFS policy is implemented when two processes arrive at the same time and have same burst time. In Priority Scheduling algorithm, the highest priority process is executed first. Here the major issue is starvation. A lower priority process can wait

978-1-4673-9666-0/15 $31.00 © 2015 IEEE
DOI 10.1109/FIT.2015.62

indefinitely if too many higher priority processes arrive which can prohibit the lower priority process to get the CPU.

In RR a small time slice value is assigned to all processes in the ready queue. RR undergoes a severe problem with its time quantum value. If the time quantum is extremely large, then RR is just similar to FCFS, and the response time increases. Very small time slice leads to slow processors as it adds overhead to the CPU.

Disadvantages of RR include the following:

1) Large Waiting time
2) Large Response time
3) Increased Context Switches
4) Large Turnaround time
5) Less Throughput

A major problem with RR is fixed time slice value. A scheduler sets a fixed time slice with every process mostly between 10-100 ms [8].

II. Literature Review

In the past many improved RR algorithms have been introduced because of the fixed time slice problem observed in RR. The burst time and priority of each process is used to compute time quantum [1]. The time quantum [2] is computed for two processors system which showed better results than DQRRR [3]. First process in ready queue allocates the CPU for one time slice. Processor is allocated by the same process if its remaining burst time is less than time quantum; otherwise next process in ready queue gets the CPU. Waiting time is reduced and it gave better results than traditional RR [4]. Processes with short burst times are given more weight so that they leave ready queue earlier [5]. Ready queue is sorted according to the processes' burst times in ascending order. Time slice is equal to the median of all processes' burst times present in the ready queue [7]. Compute time slice by finding the average of all processes in ready queue [8]. IRRVQ combines the SJF with RR to produce better performance [9]. Allocate the CPU according to priorities of the processes for one time quantum [10]. Sort the ready queue according to the processes' burst times and assign priorities. Highest priority is assigned to the shortest burst time process. This approach gave good results than traditional RR.

III. Proposed Algorithm

Our proposed algorithm PFRR presents a solution to the fixed time slice problem in RR by adjusting the time slice value for each process according to its priority and burst time, hence, using variable time quantum. Our proposed approach gives better results than IRRVQ [9] for equal priority processes, Priority Based Round Robin [10] for different priority processes, and RR scheduling. IRRVQ is developed for equal priority processes. Priority Based Round Robin considers different priority processes and uses fixed time slice for round one. Moreover, in second round it does not pay more attention to priorities of the processes, rather it considers only the burst times of the processes. PFRR considers both burst times and priorities of processes in each round for different priority processes. It also assigns time quantum fairly for an equal priority process according to its burst time.

A. Abbreviations and Acronyms

Table I shows the acronyms and abbreviations which are used in our paper.

TABLE I
ABBREVIATIONS AND ACRONYMS

Abbreviation	Acronym
N	Total number of processes present in the ready queue
Tavg	Average turnaround time
Wavg	Average waiting time
NCS	Total number of context switches
Rtime	Remaining burst time of the Process
Btime	Burst time of Process
TQ	Time Quantum
ms	Millisecond
RQ	Ready Queue
Algo	Algorithm
Bavg	Average of burst time of all processes present in ready queue
HBT	Highest burst time of a process in ready queue
MDR	Midrange of all processes in ready queue
Priority(i)	Priority of i^{th} process in ready queue
Wb(i)	Weight assigned to i^{th} process according to its burst time
Wp(i)	Weight assigned to i^{th} process according to its priority

B. Flow Diagram of PFRR:

Fig. 1 shows the detailed flow diagram of proposed PFRR algorithm.

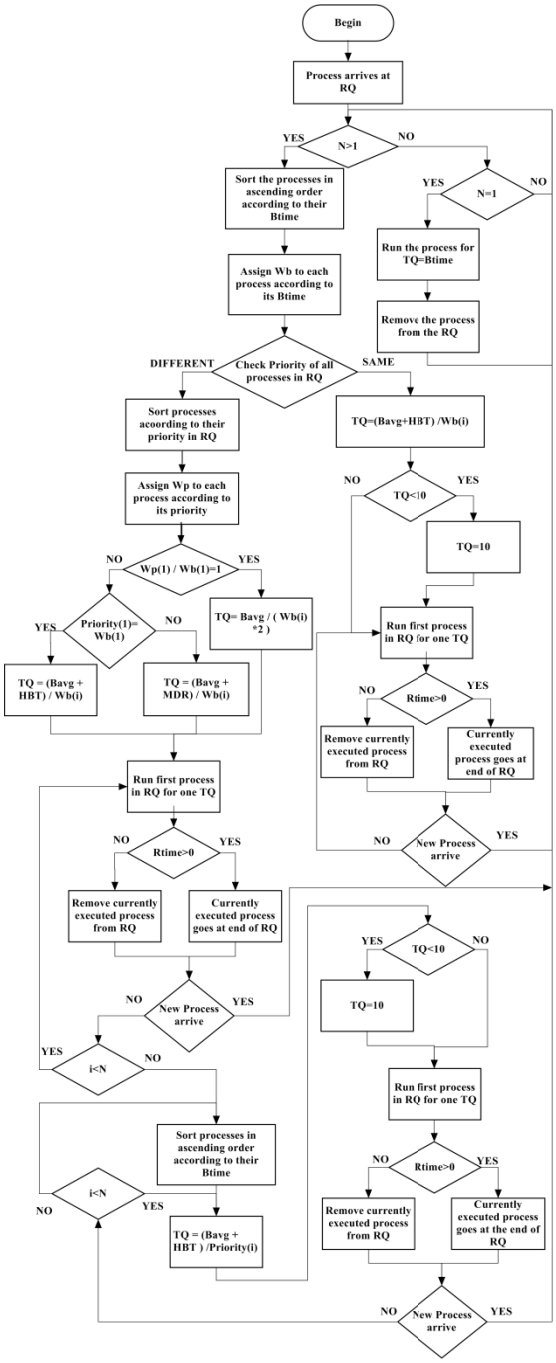

Fig. 1. Flow Diagram of PFRR algorithm

C. *Methodology*

Following is the proposed PFRR algorithm:

1) When a new process arrives in RQ, check N. If it is the only process in RQ, allocate the CPU to it for TQ set to the Btime of the process.

2) If processes in the RQ are more than one, sort the processes according to their burst times.

3) Assign weight (Wb) to each process according to its burst time. Least weight is given to the lowest burst time process, as Wb would be inversely proportional to the TQ.

4) Check the priorities of all processes in the RQ.

Case 1: Priority of all processes is same in the RQ.

a) TQ must be computed by the following formula:

$$TQ = \frac{Bavg+HBT}{Wb(i)} \quad (1)$$

b) Check if TQ is less than 10 ms, then set TQ to 10 ms and go to step c. Otherwise, skip this step and go to step c.

c) Allocate CPU to the first process in RQ for one TQ.

d) If Rtime of currently executed process is greater than zero, then, put the process at the tail of RQ. Otherwise, remove the current process from RQ, as it has finished its execution i.e., its remaining burst time is zero.

e) Check if a new process arrives, then go to step 1. Otherwise go to step f.

f) Repeat steps a to e till Rtime of all processes in RQ gets zero.

Case 2: Priority of all processes in RQ is different.

a) Sort the processes according to the priority of processes in RQ.

b) Assign weight to each process. Higher priority process is given more weight (Wp).

c) Compute TQ as follows for round 1:

If highest priority process has maximum burst time, i.e., if Wp(1) / Wb(1) is one, then, set TQ as follows:

$$TQ = \frac{Bavg}{(Wb(i)*2)} \quad (2)$$

Wp(1) is the weight assigned to first process in RQ according to its priority. Wb(1) is the weight assigned to first process in RQ according to its burst time. Here, first process in RQ is the highest priority process.

Else if highest priority process has least burst time, i.e., if Priority(1)=Wb(1), then, set TQ as follows:

$$TQ = \frac{(Bavg+HBT)}{Wb(i)} \quad (3)$$

Priority(1) is the priority of first process in RQ.

Otherwise, set TQ as below:

$$TQ = \frac{Bavg+MDR}{Wb(i)} \quad (4)$$

d) Allocate CPU for first process in RQ for one TQ.

e) If the currently running process has finished its execution, remove it from RQ. Otherwise, put it at the tail of RQ.

f) Check if a new process arrives, then go to step 1. Otherwise go to step g.

g) If i < N, go to step c. Otherwise, go to step h.

h) In round 2, sort the processes according to their burst times in RQ.

i) Compute TQ as follows:

$$TQ = \frac{Bavg+HBT}{Priority(i)} \quad (5)$$

j) If TQ is less than 10 ms, set TQ to 10 ms and go to step k, otherwise skip this step and go to step k.

k) Run the process in RQ for 1 TQ.

l) If the currently running process has finished its execution, remove it from RQ. Otherwise, put it at the tail of RQ.

m) Check if a new process arrives, then go to step 1. Otherwise go to step n.

n) If i < N, go to step h. Otherwise, go to step o.

o) Repeat steps h to n till all processes in RQ have finished their execution i.e., their Rtime becomes zero and are removed from RQ.

For same priority processes, least burst time process is given more TQ. For different priority processes, TQ is computed according to the burst time when processes are sorted according to their priority. When processes are sorted according to their burst times in ascending order in round 2, TQ is computed according to the priority of processes.

PFRR uses an efficient way to compute TQ in first round for different priority processes. In the worst case when a highest priority process has maximum burst time, PFRR uses a small TQ, so that the waiting time of other processes do not suffer much. If a highest priority process has least burst time, PFRR uses large TQ, as it will not cause a very large increase in the waiting time and turnaround time of other processes in RQ. For all other cases, PFRR uses a moderate TQ value. Initially in first round, processes are sort according to their priority, and processes with little burst times are given more TQ by dividing TQ by Wb. The idea is to allow them to leave the RQ quickly after they finish their execution. In round 2 processes are sort in ascending order according to their burst times. Process with high priority is given more TQ by dividing the TQ by priority of the process. Highest priority is represented by least integer.

IV. Simulations

We used MATLAB R2014a to present our results. To evaluate the performance of our proposed algorithm, we compare the results of RR with PFRR and proved that PFRR increase the performance of the system by reducing Wavg, Tavg, and NCS. We then compare our results of PFRR with IRRVQ for same priority processes and with PBRR for different priority processes and gain interesting results.

A. Assumptions

It has been assumed that it is a single processor system. The overhead to sort the processes on the basis of priority and burst times and to compute the TQ is zero. TQ is in milliseconds (ms). Processes are CPU bound and not I/O bound.

B. Experiments

In PFRR we have considered both cases of different priority processes in RQ and equal priority processes in RQ.

Case I: Same Priority Processes

Consider five processes in RQ along with their burst times as shown in Table II. This data is collected from [9]. It is assumed that the priority of all processes is same and the number of processes in RQ is known before sending these processes to the CPU. All processes arrive at the same time in RQ.

The comparison results of RR, IRRVQ, and PFRR are shown in Table III.

Gantt charts for RR (TQ is 10 ms), IRRVQ, and PFRR are shown in Fig. 2, Fig. 3, and Fig. 4 respectively. The least number of context switches occur in PFRR as are evident from the Gantt charts.

Fig. 5 shows the graph obtained from MATLAB for Wavg, Tavg, and NCS of all processes in Table II for RR (TQ = 10 ms), IRRVQ, and PFRR. The proposed algorithm PFRR has shown significant improvement by reducing Wavg, Tavg, and NCS compared to RR and IRRVQ.

TABLE II
BURST TIMES OF PROCESSES IN RQ [9]

Process	Burst Time (ms)
P1	15
P2	32
P3	10
P4	26
P5	20

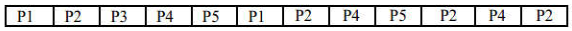

Fig. 2. Gantt Chart for RR when TQ = 10 ms (Case 1)

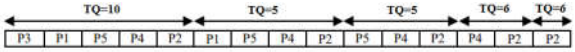

Fig. 3. Gantt Chart for IRRVQ (Case 1)

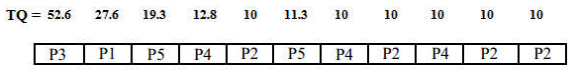

Fig. 4. Gantt Chart for PFRR (Case 1)

TABLE III
COMPARISON OF RR, IRRVQ, AND PFRR

Algo	TQ (ms)	Wavg (ms)	Tavg (ms)	NCS
RR	5, 10	56.2, 54.2	76.8, 74.8	21, 11
IRRVQ	10, 5, 5, 6, 6	46.2	66.8	14
PFRR	52.6, 27.6, 19.33, 12.88, 10, 11.3, 10, 10, 10, 10	38.77	59.37	10

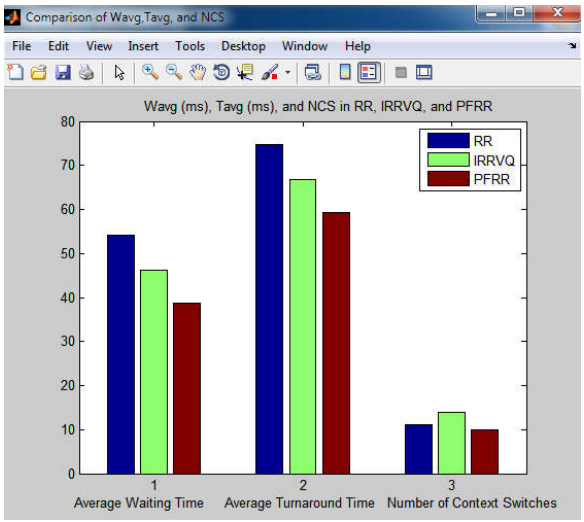

Fig. 5. Wavg, Tavg, and PFRR of all Processes in Table II for RR (TQ = 10 ms), IRRVQ, and PFRR

Case II: Different Priority Processes

Consider five processes in RQ with their burst times and priorities as shown in Table IV. We assume that the arrival time of all processes is same and N is known before sending the processes to CPU. Priorities of all processes are different.

The results obtained from RR, Priority Based Round Robin, and PFRR are tabulated in Table V. The success of PFRR over both RR and Priority Based Round Robin is visible from the fact that PFRR shows more reduced Wavg, Tavg than RR and Priority Based Round Robin. NCS is less in our proposed algorithm than RR but are same for Priority Based Round Robin.

The Gantt charts for RR, Priority Based Round Robin, and PFRR are shown in Fig. 6, Fig. 7, and Fig. 8 respectively.

Wavg, Tavg, and NCS are compared in Fig. 9 for RR, Priority Based Round Robin, and PFRR. The TQ is 10 ms for both RR and Priority Based Round Robin. Our result clearly shows significant improvement in Wavg and Tavg, with same number of context switches in our proposed algorithm PFRR and Priority Based Round Robin. NCS are less in PFRR than RR. The result clearly depicts the better performance of PFRR over both RR and Priority Based Round Robin in terms of Wavg and Tavg.

TABLE IV
PRIORITY AND BURST TIMES OF PROCESSES IN RQ

Process	Priority	Burst Time (ms)
P1	3	16
P2	2	28
P3	4	13
P4	1	38
P5	5	12

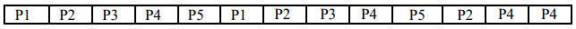

Fig. 6. Gantt Chart for RR when TQ = 10 ms (Case 2)

Fig. 7. Gantt Chart for Priority Based Round Robin, when TQ = 10 ms in first round (Case 2)

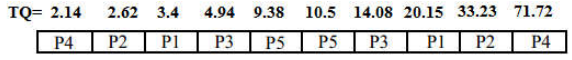

Fig. 8. Gantt Chart for proposed algorithm PFRR (Case 2)

TABLE V
COMPARISON OF RR, PRIORITY BASED ROUND ROBIN AND PFRR

Algo	TQ (ms)	Wavg (ms)	Tavg (ms)	NCS
RR	5, 10	59.6, 59	81, 80.4	23, 12
Priority Based Round Robin	5, 10	39.4, 49.4	60.8, 70.8	9, 9
PFRR	2.14, 2.62, 3.4, 4.94, 9.38, 10.55, 14.08, 20.15, 33.23, 71.72	35.03	56.43	9

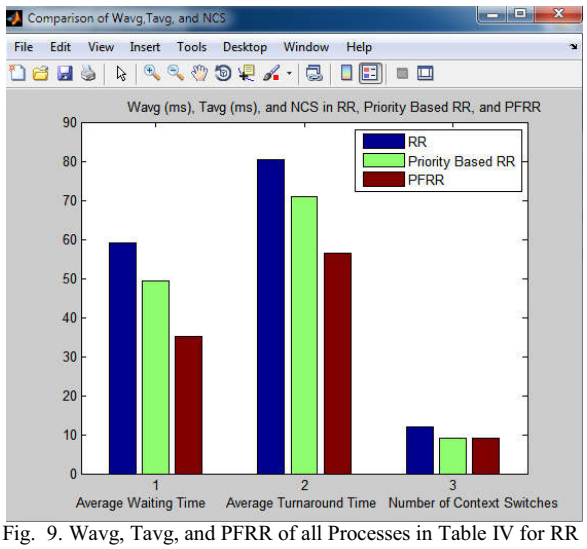

Fig. 9. Wavg, Tavg, and PFRR of all Processes in Table IV for RR (TQ = 10 ms), Priority Based Round Robin (TQ = 10 ms), and PFRR

V. CONCLUSION

In this paper, PFRR algorithm is proposed for time sharing systems. The performance of a CPU scheduling algorithm is dependent on the turnaround time, waiting time, and total number of context switches. Therefore, variable time quantum is used to improve the performance in PFRR by reducing Wavg, Tavg, and NCS. PFRR is compared with various CPU scheduling algorithms including IRRVQ, Priority Based Round Robin, and traditional Round Robin algorithm. From the results and observations we have proved that our proposed CPU scheduling algorithm PFRR is better than Round Robin, IRRVQ, and Priority Based Round Robin because of the reduced Wavg and Tavg. NCS in PFRR is reduced than RR but it remains the same for Priority Based Round Robin. PFRR also uses a variable time quantum that resolves the fixed time quantum problem observed in both RR and Priority Based Round Robin. Thus, PFRR covers CPU scheduling for both same priority processes as well as different priority processes by computing time quantum fairly for both cases.

REFERENCES

[1] Asst. Proff. M. K. Srivastav, Sanjay Pandey, Indresh Gahoi, Neelesh Kumar Namdev, "Fair Priority Round Robin with Dynamic Time Quantum", International Journal of Modern Engineering Research, vol. 2, issue3, pp. 876-881, 2012.

[2] H.S. Behera, Jajnaseni Panda, Dipanwita Thakur & Subasini Sahoo, "A New Proposed Two Processor Based CPU Scheduling Algorithm with Varying Time quantum for Real Time Systems", Journal of Global Research in Computer Science, vol. 2, no. 4, pp. 81-87, 2011.

[3] H.S. Behera, R. Mohanty, Debashree Nayak, "A New Proposed Dynamic Quantum with Readjusted Round Robin Scheduling Algorithm and its Performance Analysis", International Journal of Computer Applications(0975-8887) vol. 5, no.5, August 2010.

[4] Manish Kumar Mishra & Abdul Kadir Khan, "An Improved Round Robin CPU Scheduling Algorithm", Journal of Global Research in Computer Science, vol. 3, no. 6, pp. 64-69, 2012.

[5] Helmy, T. & A. Dekdouk. (2007). Burst Round Robin as a Proportional-share Scheduling Algorithm IEEEGCC. [Online]. Available: http://eprints.kfupm.edu.sa/1462/

[6] Silberschatz, Galvin, and Gagne, "Operating systems concepts", 8th ed., Wiley, 2009.

[7] Debashree Nayak, Sanjeev Kumar Malla & Debashree Debadarshini, "Improved Round Robin Scheduling using Dynamic Time Quantum", International Journal of Computer Applications, vol. 38, no. 5, pp. 34-38, 2012.

[8] Abbas Noon, Ali Kalakech & Seifedine Kadry, "A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average", IJCSI International Journal of Computer Science Issues, vol. 8, issue 3, no. 1, May 2011.

[9] Manish Kumar Mishra and Dr. Faizur Rashid "An Improved Round Robin CPU Scheduling Algorithm with varying Time Quantum", International Journal of Computer Science, Engineering and Applications (IJCSEA), vol. 4, no. 4, August 2014.

[10] Ishwari Singh Rajput & Deepa Gupta, "A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems", International Journal of Innovations in Engineering and Technology (IJIET), vol. 1, issue 3, Oct 2012.