VOL. 11, NO. 22, NOVEMBER 2016 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2016 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

CPU THREAD PRIORITIZATION USING A DYNAMIC QUANTUM TIME ROUND-ROBIN ALGORITHM

Maysoon A. Mohammed[1, 2], Mazlina Abdul Majid[1], Balsam A. Mustafa[1] and Rana Fareed Ghani[3]
[1]Faculty of Computer System & Software Engineering, University Malaysia Pahang, Pahang, Malaysia
[2]Department of Mechanical Engineering, University of Technology, Baghdad, Iraq
[3]Department of Computer Sciences, University of Technology, Baghdad, Iraq
E-Mail: maysoon.ameir@gmail.com

ABSTRACT

In Round-Robin Scheduling, the time quantum is fixed and processes are scheduled such that no process uses CPU time more than one time quantum in one go. If time quantum is too large, the response time of the processes will not be tolerated in an interactive environment. If the time quantum is too small, unnecessary frequent context switch may occur. Consequently, overheads result in fewer throughputs. In this study, we propose a priority Round-Robin algorithm with dynamic quantum time (PDQT). The algorithm used the old fixed quantum time to generate new one for each process depending on its priority. The simple Round-Robin algorithm has been improved by about 20%. By controlling quantum time, we experience fewer context switches and shorter waiting and turnaround times, thereby obtaining higher throughput.

Keywords: round robin, dynamic quantum time, priority, and PDQT.

INTRODUCTION

Modern operating systems are moving towards multitasking environments in which fast computer systems perform multitasking (executing more than one process at a time) and multiplexing (transmitting multiple flows simultaneously). This mainly depends on the CPU scheduling algorithm as the CPU is the essential part of the computer. In computer science, scheduling is the act by which processes are given access to system resources (e.g., processor cycles, communications bandwidth). CPU scheduling is an essential operating system task which permits allocating the CPU to a specific process for a time slice. In other words it determines which process runs when there are multiple runnable processes. As researchers (Kopetz 2011) previously pointed out that the need for a scheduling algorithm arises from the requirement for fast computer systems to perform multitasking and multiplexing. CPU scheduling is important because it affects resource utilization and other performance parameters(Hasan). Several CPU scheduling algorithms are available (Silberschatz, Galvin *et al.* 2013), (Oyetunji and Oluleye 2009), such as First Come First Serve (FCFS), Shortest Job First Scheduling (SJF), Round-Robin (RR) Scheduling, and Priority Scheduling (PS). However, due to disadvantages, these algorithms are rarely used in shared time operating systems, except for RR Scheduling (Cerqueira and Brandenburg 2013).

RR is considered the most widely used scheduling algorithm in CPU scheduling (Silberschatz, Galvin *et al.* 2013), (Yang, Schopf *et al.* 2003) also used for flow passing scheduling through a network device (Tong and Zhao 2007). An essential task in operating systems in CPU Scheduling is the process of allocating a specific process for a time slice. Scheduling requires careful attention to ensure fairness and avoid process starvation in the CPU. This allocation is carried out by software known as a scheduler(Silberschatz, Galvin *et al.* 2013), (Yang, Schopf *et al.* 2003).

The scheduler is concerned mainly with the following tasks (Chen and Liu 2013):

- CPU utilization - to keep the CPU as busy as possible
- Throughput - number of processes that complete their execution per time unit
- Turnaround - total time between submission of a process and its completion
- Waiting time - amount of time a process has been waiting in the ready queue
- Response time - amount of time taken from the time a request was submitted until the production of the first response
- Fairness - equal CPU time allocated to each thread

Therefore, we can conclude that a good scheduling algorithm for real time and time sharing system must possess the following characteristics (Singh, Goyal *et al.* 2010):

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

Operating systems may feature up to three distinct types of schedulers, which are long term, mid-term or medium term, and short-term (Figure-1). The long-term scheduler or job scheduler selects processes from the job pool and loads them into the memory for execution. The short-term scheduler or CPU scheduler selects from among the processes that are ready for execution and allocates a CPU to one of them. The medium term scheduler removes processes from the memory and reduces the degree of multiprogramming results in the scheme of swapping. Swapping is performed by the scheduler, which is the module that allows the CPU to control the process selected by the short-term scheduler(Noon, Kalakech *et al.* 2011).

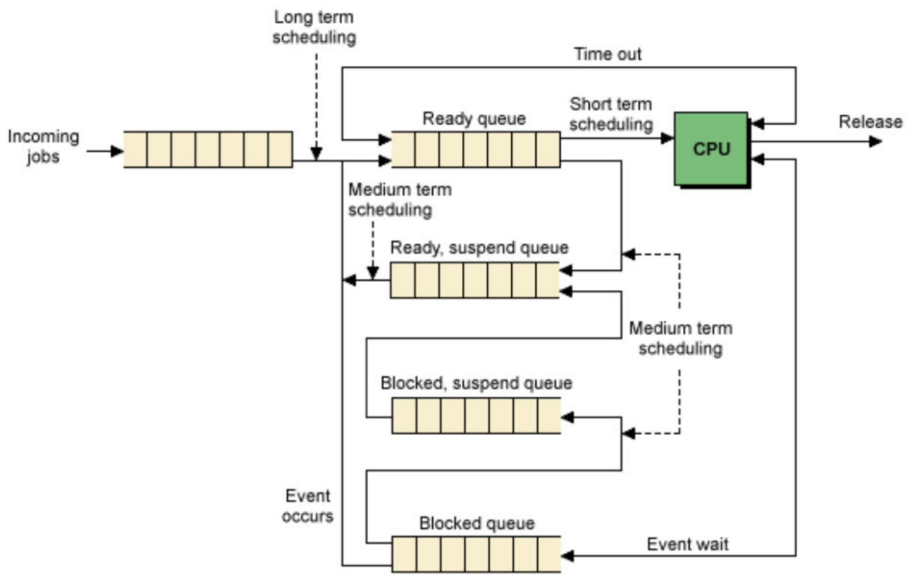


Figure-1. Queuing diagram for scheduling.

RELATED WORK

Many CPU scheduling algorithms have been introduced in the past years to improve the performance of the CPU. An algorithm for robust quantum time value (Lavanya[1] and Saravanan 2013) orders processes according to the smallest to the highest burst time. Then, quantum time would be calculated by taking the average of minimum and maximum burst times of the processes in the ready queue. An Improved Round Robin Scheduling using the feature of SJF in which the process in the ready queue would be allocated with static quantum time in the first cycle, and then the process would be selected by SJF (Yadav, Mishra *et al.* 2010). Self-Adjustment Time Quantum in RR Algorithm is an algorithm in accordance to the burst time of the processes (Nayak, Malla *et al.* 2012). (Behera 2011) assigned a fare-share weight to each process, such that the process with the minimum burst time would have the maximum weight. Quantum time would be calculated dynamically by using the weighted time slice method. Thus, the processes would be executed. An Improved RR (IRR) CPU Scheduling Algorithm was presented by (Mishra 2012). In this algorithm, the CPU time is allocated to the first process from the ready queue for a time interval of up to one quantum time. After the quantum time of the process is completed, the remaining burst time of this process would be compared with quantum time. If its burst time was less than one quantum time, the CPU would again allocate the same process until execution is completed and the task is removed from the queue. This algorithm reduces waiting time in the ready queue, and hence improves performance.

(Abdulrahim, E Abdullahi *et al.* 2014) proposed algorithm similar to IRR (Mishra 2012). The proposed algorithm uses two queues, which are ARRIVE and REQUEST. Compared with IRR, this algorithm indicated performance improvement. (Noon, Kalakech *et al.* 2011) presented a mechanism of dynamic quantum time, which overcame the problem of fixed quantum time. Meanwhile, an algorithm of feedback scheduling focused on lower priority queue process (Bhunia 2011).

RRARCHITECTURE

RR architecture is a preemptive version of First Come, First Serve scheduling algorithm. The tasks are arranged in the ready queue in first come, first serve manner and the processor executes the task from the ready queue based on time slice. If the time slice ends and the tasks are still executing on the processor, the scheduler will forcibly preempt the executing task and keep it at the end of ready queue. Then, the scheduler will allocate the processor to the next task in the ready queue. The preempted task will make its way to the beginning of the ready list and will be executed by the processor from the point of interruption.

A scheduler requires a time management function to implement the RR architecture and requires a tick timer (Goel and Garg 2013). The time slice is proportional to the period of clock ticks (Chen and Liu 2013). The time slice length is a critical issue in real time operating systems. The time slice must not be too small, as it would result in frequent context switches. Moreover, the time slice should be slightly greater than the average task computation time.

RR pitfalls in real time systems

RR when implemented in real time operating systems faces two drawbacks, which are high rate of context switch and low throughput. These two problems of RR architecture are interrelated (Lampard 2011).

- Context switch: When the time slice of the task ends and the task is still executing in the processor, the scheduler forcibly preempts the tasks on the processor. The interrupted task is then stored in stacks or registers, and the processor is allocated the next task in the ready queue. This action performed by the scheduler is called “context switch.” Context switch leads to wastage of time, memory, and scheduler overhead.

- Larger waiting and response times: In RR architecture, the time the process spends in the ready queue waiting for the processor for task execution is known as “waiting time.” The time the process completes its job and exits from the task-set is called “turnaround time.” Larger waiting and response times are clearly a drawback in RR architecture, as it leads to degradation of system performance.
- Low throughput: Throughput refers to the number of processes completed per time unit. If RR is implemented in real time operating systems, throughput will be low and results in severe degradation of system performance. If the number of context switches is low, then the throughput will be high. Context switch and throughput are inversely proportional to each other.

PROPOSED ALGORITHM

RR scheduling algorithm has no priority and fixed quantum time. However, this scheduling algorithm is not suitable for real time operating system (RTOS) because of drawbacks. In other words, the high context switch, high waiting and response times, and low throughput are pitfalls of RR. These disadvantages do not make the optimal choice for RTOS. Priority RR scheduling still has the problem of starvation, where the lowest priority thread with fixed quantum time will be starved and preempted by the highest priority thread. Hence, we propose an algorithm that depends on the existing RR. The proposed algorithm is the Priority Dynamic Quantum Time RR Scheduling Algorithm (PDQT).

The basic idea of this algorithm considers different priorities and different quantum times(Mohanty, Behera *et al.* 2011).

The steps of PDQT:

a) Set priorities for the processes that enter the ready queue.
b) Calculate new quantum time depending on the old one by using a simple formula, which is q=k+n-1, where q is the new quantum time, k is the old quantum time, and n is the priority of the processes in the ready queue.
c) Set different quantum times for the processes depending on the priorities. The highest priority process will get the largest quantum time, which is q, and the lowest priority process will get the smallest quantum time, which is k.
d) Assign the process in between to get quantum time less than the time of the process before it by 1.
e) Apply the original RR with the priorities and new different quantum times.
f) Calculate context switches average turnaround, and average waiting times.

By changing the quantum time, we could improve the existing RR algorithm by reducing context switches and lessening waiting and response times. Hence, throughput will increase. The next sections present two case studies to show the differences between PDQT and RR Algorithm.

Case study 1:
PDQT vs. existing RR Algorithm

Five processes have been defined with CPU burst time, arrival times, and their priorities. These five processes are scheduled in RR technique as well as according to the PDQT algorithm. The context switches, average waiting timeand average turnaround time are calculated, and the results are compared. To accomplish this task, we implemented the algorithm in JAVA programming language and conducted several experiments. However, only two experiments are discussed here for dynamic quantum time process, and we assure that the analysis remain the same for the other experiments.

We consider five processes (A, B, C, D, and E) with different arrival times, burst time, and priorities as shown in Table-1, where quantum time is 5 millisecond. Figure-2 illustrates a diagram of the case study.

Table-1. The inputs for the processes of case study 1.

Task	Arrival time	Burst time	Priority
A	0	12	2
B	10	8	4
C	15	4	5
D	20	10	3
E	25	6	1

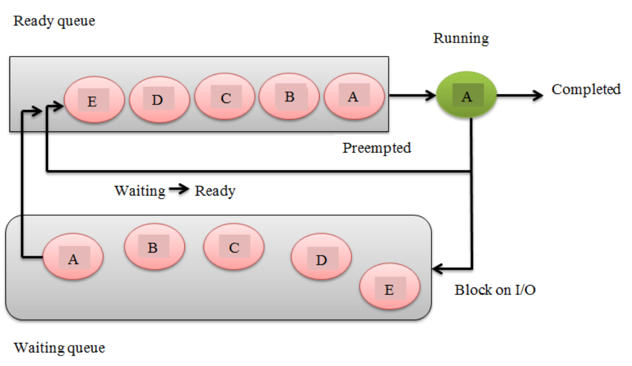
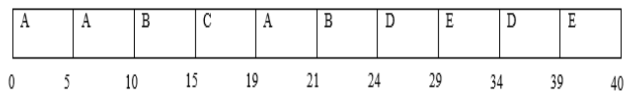


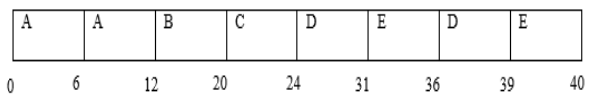
Figure-2. Diagram of case study 1.

According to the original RR, Simple RR does not use priority. Hence, five processes have been scheduled by using simple RR architecture. The time slice of 5 millisecond was used. In RR algorithm, no process is allocated in the CPU for more than one time slice in a row. If the CPU process exceeds one time slice, the concerned process will be preempted and placed into the ready queue. The process is preempted after the first time quantum, and the CPU is given the next process, which is in the ready queue (process B). Similar process is conducted for the schedule until the first cycle is completed. In the second cycle, the same method is used to schedule the processes. The process time slicing in simple RR architecture is shown in Gantt chart 1.



Gantt chart-1. Process time slicing in simple RR.

According to PDQT, the algorithm used priorities of the processes and different quantum times depending on these priorities, where each process gets quantum time different from the other. The highest priority process gets the largest quantum time, the lowest priority process gets the smallest quantum time, and the processes in between get quantum time less than the one before by 1. The process time slicing in PDQT architecture is shown in Gantt chart 2.



Gant chart-2. Process time slicing in PDQT.

The equations used to calculate average turnaround and average waiting time are:

$$\text{Average turnaround time} = \sum_{k=1}^{n} T/n \quad (1)$$

$$\text{Average waiting time} = \sum_{k=1}^{n} B/n \quad (2)$$

where n = number of processes, T = completion time - arrival time; B = turnaround time - burst time

By using Equations 1 and 2 and initially applying RR, we obtained 14.6 millisecond for average turnaround time and 6.6 millisecond for average waiting time. The context switch is 9. By applying PDQT, we got 13.0 and 5.0 millisecond for average turnaround and average waiting times, respectively, and 7 for context switch. These results indicated that PDQT behaves more efficiently than simple RR.

Case study 2:

In this example, we applied two algorithms with different arrival times, but the two processes (B and C) will arrive at the same time. This situation means the priority of the process will play an important role in executing the processes. The process with the largest priority should go first whenever it arrives at the same time with another process with smallest priority. Table-2 shows the inputs of the processes and Figure-3 shows the diagram of the case study. Gant charts 3 and 4 illustrate the process time slicing in RR and PDQT, respectively. The quantum time is 3.

Table-2. Inputs of the processes of case study 2.

Task	Arrival time	Burst time	Priority
A	0	12	2
B	5	8	4
C	5	4	5
D	10	10	3
E	15	6	1

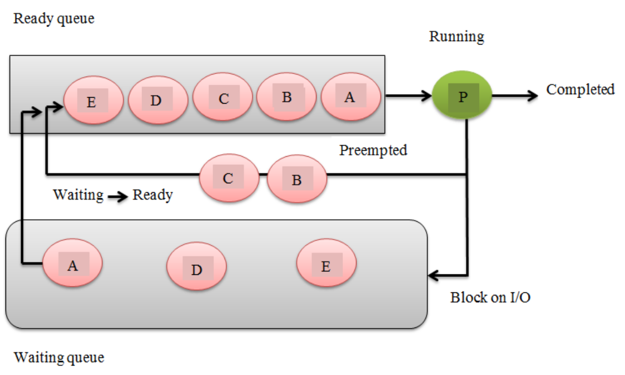
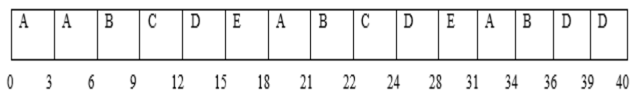
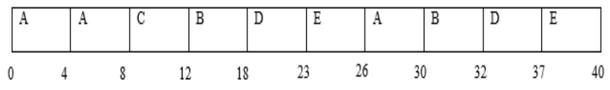


Figure-3. Diagram of case study 2.



Gantt chart-3. Process time slicing in RR.



Gantt chart-4. Process time slicing in PDQT.

By using Equations 1 and 2 and applying RR, we obtained 26.2 milliseconds for average turnaround time and 18.2 milliseconds for average waiting time, and the context switch is 14. By applying PDQT, we got 23.2 and 15.2 milliseconds for average turnaround and average waiting time, respectively, and the context switch is 9. The same efficient results of PDQT have been obtained in this case study. The advantage of our algorithm as we experience from the different case studies is high performance with the large number of processes which will be the next improvement of the algorithm to compare with other techniques, however, there is a limitation faces us is the low performance with the large burst times with high quantum time.

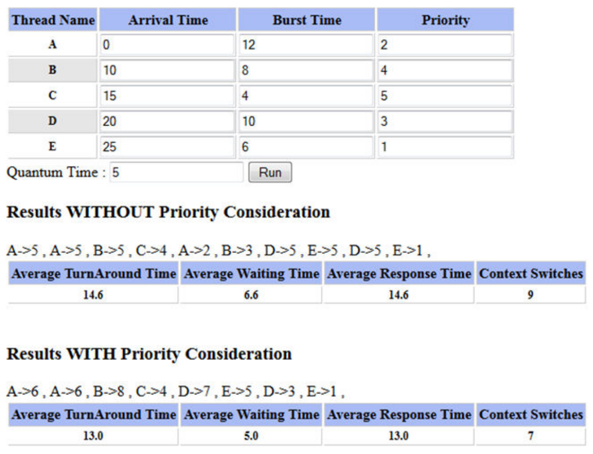
Tables 3 and 4 illustrate the results that obtained in case study 1 and 2, figures 4 and 5 show the simulation of the results for the two case studies.

Table-3. Results of case study 1.

Algorithm / Factors	RR	PDQT
Context switches	9	7
Average turnaround time	14.6	13.0
Average waiting time	6.6	5.0

Table-4. Results of case study 2.

Algorithm / Factors	RR	PDQT
Context switches	14	9
Average turnaround time	26.2	23.2
Average waiting time	18.2	15.2



Thread Name	Arrival Time	Burst Time	Priority
A	0	12	2
B	10	8	4
C	15	4	5
D	20	10	3
E	25	6	1

Quantum Time : 5 Run

Results WITHOUT Priority Consideration

A->5 , A->5 , B->5 , C->4 , A->2 , B->3 , D->5 , E->5 , D->5 , E->1 ,

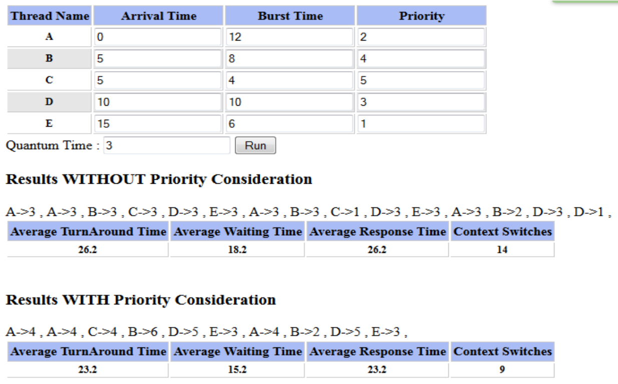
Average TurnAround Time	Average Waiting Time	Average Response Time	Context Switches
14.6	6.6	14.6	9

Results WITH Priority Consideration

A->6 , A->6 , B->8 , C->4 , D->7 , E->5 , D->3 , E->1 ,

Average TurnAround Time	Average Waiting Time	Average Response Time	Context Switches
13.0	5.0	13.0	7

Figure-4. Simulation for case study 1.



Thread Name	Arrival Time	Burst Time	Priority
A	0	12	2
B	5	8	4
C	5	4	5
D	10	10	3
E	15	6	1

Quantum Time : 3 Run

Results WITHOUT Priority Consideration

A->3 , A->3 , B->3 , C->3 , D->3 , E->3 , A->3 , B->3 , C->1 , D->3 , E->3 , A->3 , B->2 , D->3 , D->1 ,

Average TurnAround Time	Average Waiting Time	Average Response Time	Context Switches
26.2	18.2	26.2	14

Results WITH Priority Consideration

A->4 , A->4 , C->4 , B->6 , D->5 , E->3 , A->4 , B->2 , D->5 , E->3 ,

Average TurnAround Time	Average Waiting Time	Average Response Time	Context Switches
23.2	15.2	23.2	9

Figure-5. Simulation for case study 2.

COMPARRISION WITH ORIGINAL RR

The performance of two algorithms can be compared by considering the number of context switches, average waiting time, and average turnaround time as shown in Figures 6, 7, and 8, respectively.

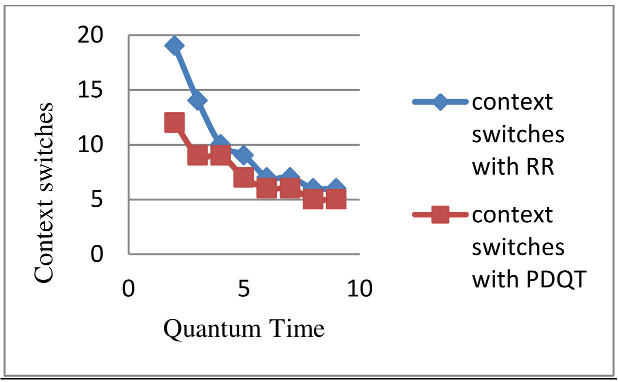


Figure-6. Performance of RR and PDQT for quantum time and context switches.

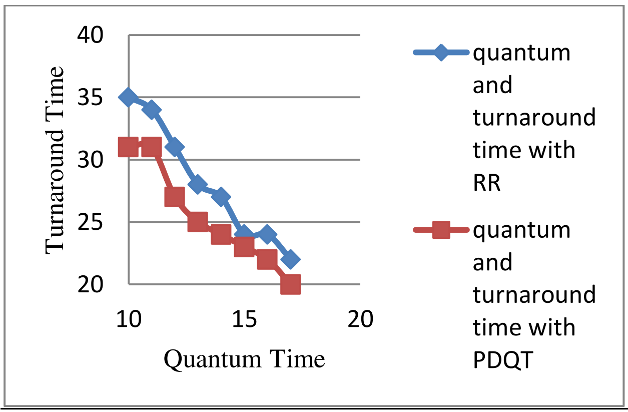


Figure-7. Performance of RR and PDQT for quantum time and turnaround time.

VOL. 11, NO. 22, NOVEMBER 2016 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2016 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

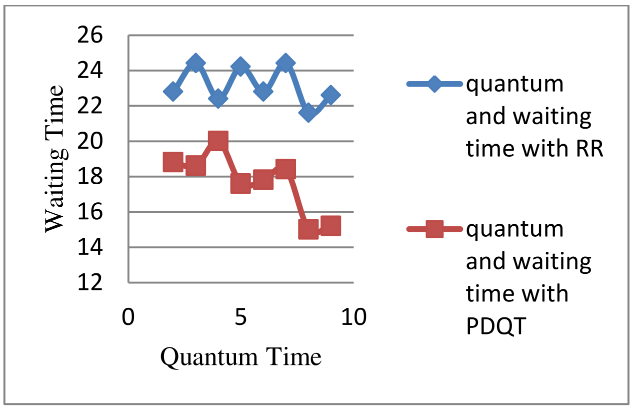


Figure-8. Performance of RR and PDQT for quantum time and waiting time.

The figures above shows that the proposed algorithm performs better over existing RR for dynamic time quantum. We see that the PDQT RR has less number of context switches, turnaround time, and waiting time in comparison to simple RR for same value of time quantum.

CONCLUSIONS

We have successfully compared both algorithms, namely, simple RR and the proposed algorithm (PDQT). Results indicated that PDQT is more efficient because this proposed algorithm has fewer context switches and shorter average turnaround and waiting times compared to simple RR. Moreover, the results reduced operating system overhead and increased throughput. PDQT lessened the problem of starvation as the processes with highest priorities are assigned with largest quantum time and are executed before lower priority processes. Performance of time-sharing systems can be improved with the proposed algorithm, and can be modified to enhance the performance of real time system.

REFERENCES

Abdulrahim, A., *et al.* 2014. "A New Improved Round Robin (NIRR) CPU Scheduling Algorithm." International Journal of Computer Applications. 90(4): 27-33.

Behera, H. 2011. "Experimental Analysis of New Fair-Share Scheduling Algorithm with Weighted Time Slice for Real Time Systems." Journal of Global Research in Computer Science2 (2).

Bhunia, A. 2011. "Enhancing the Performance of Feedback Scheduling." International Journal of Computer Applications 18(4): 11-16.

Cerqueira, F. and B. Brandenburg. 2013. A comparison of scheduling latency in linux, PREEMPT-RT, and LITMUSRT. Proceedings of the 9th Annual Workshop on Operating Systems Platforms for Embedded Real-Time applications.

Chen, M.-X. and S.-H. Liu. 2013. Hierarchical Deficit Round-Robin Packet Scheduling Algorithm. Advances in Intelligent Systems and Applications-Volume 1, Springer: 419-427.

Goel, N. and R. Garg. 2013. "An Optimum Multilevel Dynamic Round Robin Scheduling Algorithm." arXiv preprint arXiv: 1307.4167.

Hasan, T. F. "CPU scheduling visualization."

Kopetz, H. 2011. Real-time systems: design principles for distributed embedded applications, Springer.

Lampard, B. 2011. "Program scheduling and simulation in an operating system environment."

Lavanya¹, M. and S. Saravanan. 2013. "Robust Quantum Based Low-power Switching Technique to improve System Performance."

Mishra, M. K. 2012. "An Improved Round Robin CPU scheduling algorithm." Journal of Global Research in Computer Science 3(6): 64-69.

Mohanty, R., *et al.* 2011. "Priority based dynamic round robin (PBDRR) algorithm with intelligent time slice for soft real time systems." arXiv preprint arXiv: 1105.1736.

Nayak, D., *et al.* 2012. "Improved round robin scheduling using dynamic time quantum." International Journal of Computer Applications (0975–8887) Volume.

Noon, A., *et al.* 2011. "A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average." arXiv preprint arXiv: 1111.5348.

Oyetunji, E. and A. Oluleye. 2009. "Performance Assessment of Some CPU Scheduling Algorithms." Research Journal of Information and Technology 1(1): 22-26.

Silberschatz, A., *et al.* 2013. Operating system concepts, Wiley.

Singh, A., *et al.* 2010. "An Optimized Round Robin Scheduling Algorithm for CPU Scheduling." IJCSE) International Journal on Computer Science and Engineering 2(07): 2383-2385.

Tong, W. and J. Zhao. 2007. Quantum varying deficit round robin scheduling over priority queues. Computational Intelligence and Security, 2007 International Conference on, IEEE.

Yadav, R. K., *et al.* 2010. "An improved round robin scheduling algorithm for CPU scheduling." International Journal on Computer Science and Engineering 2(04): 1064-1066.

Yang, L., *et al.* 2003. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. Proceedings of the 2003 ACM/IEEE conference on Supercomputing, ACM.