CPU Burst Processes Prioritization Using Dynamic Quantum Time Algorithm Compared with Varying Time Quantum and Round Robin Algorithms

Maysoon A. Mohammed[1, 2], Mazlina AbdulMajid[1], Balsam A. Mustafa[1], and Rana Fareed Ghani[3]

[1] Faculty of Computer Systems & Software Engineering, UMP, Kuantan
26300 Pahang, Malaysia
maysoon.ameir1977@gmail.com

[2] Department of mechanical Engineering, UOT.
10066 Baghdad, Iraq
maysoon.ameir1977@gmail.com

[3] Department of Computer Sciences, UOT,
10066 Baghdad, Iraq
ranafghany@yahoo.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. Round Robin scheduling algorithm is the most suitable choice for time shared system but not for soft real time systems due to a large turnaround time, large waiting time and high number of context switches. The choice of the quantum time in RR is the optimal solution for the problem of large turnaround and waiting time with RR. In this study, we propose a priority algorithm with dynamic quantum time (PDQT), to improve the work of RR by improving the concept of Improved Round Robin with varying time quantum (IRRVQ). The proposed algorithm gave results better than RR and IRRVQ in terms of minimizing the number of context switches, average waiting time, average turnaround time, design and analysis. The simple Round-Robin algorithm has been improved by about 40%. By controlling quantum time according to the priorities and burst times of the processes, we experience fewer context switches and shorter waiting and turnaround times, thereby obtaining higher throughput.**

Index Term-- Round Robin; dynamic quantum time; priority; burst time; Priority Dynamic Quantum Time.

1. 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 [1] 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 [2]. Several CPU scheduling algorithms are available [3], [4], such as First Come First Serve Scheduling (FCFS), Shortest Job First Scheduling (SJF), Round-Robin Scheduling (RR), and Priority Scheduling (PS). However, due to disadvantages, these algorithms are rarely used in shared time operating systems, except for RR Scheduling [5]. RR is considered the most widely used scheduling algorithm in CPU scheduling [3], [6] also used for flow passing scheduling through a network device [7]. 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 [3], [6]. The scheduler is concerned mainly with the following tasks [8]:

- 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 process.

2. Performance Factors

CPU is an essential part in the operating system which is scheduled by many of the scheduling algorithms to keep it busy as much as possible to achieve the perfect utilization of CPU. The processes that need to be processed submit to the system and wait in the ready queue to be selected by the scheduler for the processing. The scheduler is responsible of picking the processes from the ready queue and allocate the CPU if it is idle for that process [9].

The moment that the process joins to the ready queue is called the arrival time. Burst time is the time that the process needs to complete its job inside the CPU. The turnaround time is the time that the process spends in the system from the moment of submission to the moment of completion the processing. Waiting time is the time that the process waits in the ready queue waiting for its turn to be selected by the scheduler for the processing. Therefore, we can conclude that a good scheduling algorithm for real time and time sharing system must possess the following characteristics [10]:

- 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 as shown in Fig.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 [11].

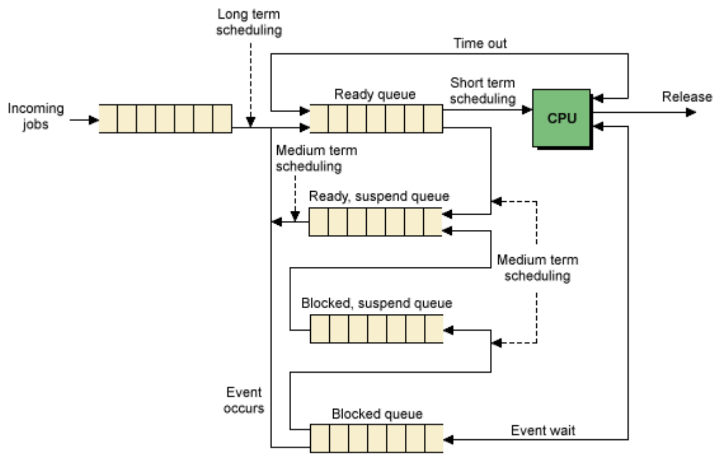

Fig. 1. Queuing diagram for scheduling

3. 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 [12] 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 [13]. Self-Adjustment Time Quantum in RR Algorithm is an algorithm in accordance to the burst time of the processes [14]. Reference [15] 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 [16]. 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. Reference [17] proposed algorithm similar to IRR [16]. The proposed algorithm uses two queues, which are ARRIVE and REQUEST. Compared with IRR, this algorithm indicated performance improvement. Reference [11] 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 is proposed by [18].

4. RR Architecture

RR architecture is a pre-emptive 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 pre-empt 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 pre-empted 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 [19]. The time slice is proportional to the period of clock ticks [8]. 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.

4.1. 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 [20].

- Context switch: When the time slice of the task ends and the task is still executing in the processor, the scheduler forcibly pre-empts 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 turnaround 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 turnaround 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.

5. Improved Round Robin with Varying Time Quantum (IRRVQ)

The idea of improved Round Robin CPU scheduling algorithm with varying quantum time (IRRVQ) is depending on the combination between Shortest Job First (SJF) and RR with using dynamic quantum time in each round. First, the processes in the ready queue are ordered from lowest to highest burst times. The scheduler allocates the CPU to the first process using RR and assigns its burst time as quantum time for this round. The same procedure will be repeated in each round until all processes finish their execution and ready queue assigns to NULL.

6. The Proposed Algorithm, Priority Dynamic Quantum Time Scheduling Algorithm PDQT

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 process with fixed quantum time will be starved and preempted by the highest priority process. Hence, we propose an algorithm that depends on the existing RR.

The proposed algorithm is the Priority Dynamic Quantum Time Scheduling Algorithm (PDQT). The proposed algorithm focuses on the pitfalls of existing Round-Robin scheduling algorithm which have no priority for all the processes, where the processes arranged in first in first out architecture in the ready queue. This disadvantage of Round-Robin architecture affects negatively for processes with lower CPU burst. This leads to high turnaround and waiting times which leads to reduce throughput of the system. The proposed algorithm excludes the defects of implementing existing Round-Robin scheduling algorithm. Also the proposed algorithm focuses on IRRVQ scheduling algorithm and try to improve it by prioritizing the processes in the ready queue and changing the quantum time of each round besides changing the quantum time for each process in each round depending on its priority.
One of the optimal CPU scheduling algorithms in time sharing systems is Round Robin (RR). In time sharing systems the selection of the quantum time is an important factor that the performance of the CPU depends on [21]. The quantum time that taken by RR is fixed that reduces the CPU performance [22]. In this paper selection of quantum time and priority has been discussed by a new CPU scheduling algorithm for time shared systems, which is PDQT and compared with an existing one which is IRRVQ. It includes the advantage of RR CPU scheduling algorithm of less chance of starvation. Round robin CPU scheduling algorithm has high context switch rates, large response time, large waiting time, large turnaround time and less throughput [23], these disadvantages can be improved with new proposed CPU scheduling algorithm.

6.1. The Proposed Algorithm Design

The basic idea of our algorithm considers different priorities and different quantum times [24].

The steps of PDQT:

- Processes are arranged in increasing order in the ready queue. New priorities and quantum times are assigned according to the CPU bursts of processes; the process with lowest burst time is set with highest priority.
- Choose the first process in Round-Robin fashion and assign its burst time as quantum time for this round,

and allocate CPU to this process only for one quantum time.

- Calculate quantum times of all processes in this round depending on their priorities and the quantum time of this cycle by using a simple formula, which is q=k+p-1, where q is the new quantum time, k is the old quantum time, and p is the priority of the processes in the ready queue.
- Set different quantum times for the processes according to their 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.
- Each process gets the control of the CPU until they finished their execution.
- Repeat the steps above until all processes complete and ready queue assign to NULL.
- Apply the original RR, IRRVQ and our PDQT with the priorities and new different quantum times.
- Calculate context switches, average turnaround time and average waiting time.

By changing the quantum time of each cycle and all processes and assigning priority to the lowest burst time process, we could improve the existing IRRVQ algorithm by reducing context switches and lessening average turnaround and average waiting times. Hence, throughput will increase. The next section presents a case study to show the simulation between PDQT and IRRVQ algorithms.

6.2. Assumptions

The assumptions that we followed in the case studies are: The Quantum time has been taken in milliseconds, the CPU bound is active that mean all processes are in CPU bound not in I/O bound. For IRRVQ all processes with the same priorities while in our algorithm different priorities used for all processes. For experimental purposes, the burst times and arrival times of all processes are known and chosen by the researchers. The context switches in IRRVQ are considered zero while in PDQT are computed. The overhead of arranging the ready queue processes in ascending order has been considered zero in IRRVQ [25] as well in PDQT.

6.3. Experimental Simulation (Case Study)

In this case study, priorities are assigned to the processes according to their burst time. The highest priority sets to the lowest burst time process [26]. Thus, the lowest burst time process with the highest priority gets the highest quantum time. The proposed algorithm helps to minimize a number of performance factors like context switches, average turnaround time and average waiting time.

Ten processes have been defined with CPU arrival time, burst time, and their priorities. These ten processes are scheduled in RR technique as well as according to the IRRVQ and PDQT algorithms. The context switch, average waiting time, and average turnaround time are calculated; the results are compared and analyzed using ten equations of queueing theory. To accomplish this task, we implemented the algorithm in JAVA programming language and conducted several experiments. However, only one experiment is discussed and analyzed here for dynamic quantum time process, and we assure that the analysis remain the same for the other experiments.

The processes with their CPU arrival times and burst times are given in Table I. Quantum time is 10 milliseconds.

Table I
Inputs of the processes

Processes	Arrival	Burst Time	Priority
A	0	10	10
B	5	18	8
C	10	25	3
D	15	15	9
E	15	20	6
F	20	19	7
K	25	22	5
L	30	42	2
M	35	25	4
N	35	45	1

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

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

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

, where n = number of processes, T = completion time – arrival time and B = turnaround time – burst time.
Table 2 represents the results that obtained from the three algorithms.

Table II
Inputs of the processes

Algorithm	Average TAT	Average WT	CS
RR	138.3	114.2	28
IRRVQ	142.4	118.3	28
PDQT	127.9	103.8	25

The results above proved that the existing algorithm IRRVQ is not efficient to improve RR for the large number of processes where RR gives average turnaround and average waiting time better that IRRVQ. On the contrast our proposed PDQT conducts better results with the large number of processes over that RR and IRRVQ, so IRRVQ has been improved in effective way by using PDQT.

Figs. 2, 3, and 4 represent the Gantt chart of the three algorithms RR, IRRVQ and PDQT respectively, and figure

8 shows the comparison of performance of the three algorithms.

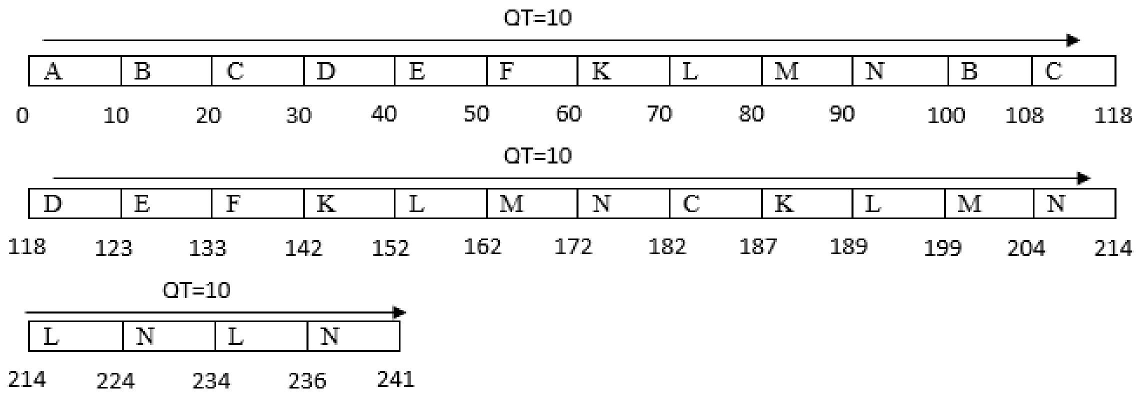

Fig. 2. Gantt chart of RR

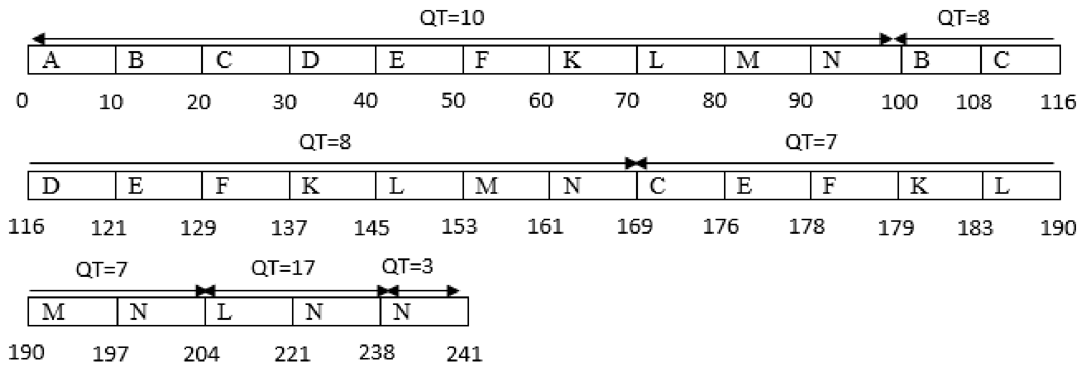

Fig. 3. Gantt chart of IRRVQ

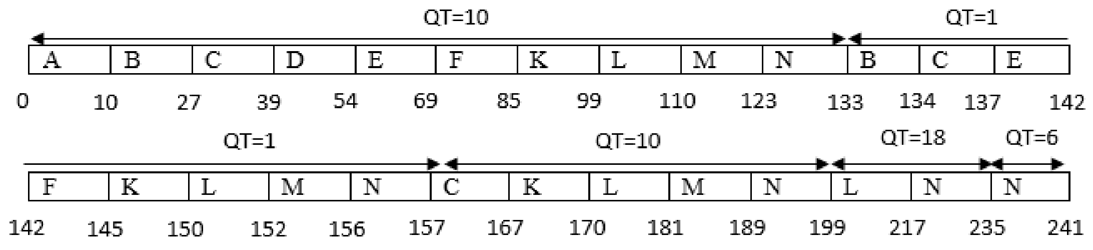

Fig. 4. Gantt chart of PDQT

6.4. Analysis of the case study

In this section the three algorithms are applied with the given arrival time, burst times, and priorities, where the processes still enter to the ready queue until the processing is finished and go out the CPU. Analyzing for the times of these processes in the ready queue and the system by using equations of queueing theory which is the mathematical approach of the waiting queues has been obtained. Tables III and IV and V show the analysis of the RR, IRRVQ and PDQT processing times, respectively.

Table III
Analysis Times of RR

Task	Arrival times	Burst times	Quantum time	Start service time	End service time	Time in queue	Waiting time	Finishing of service
A	0	10	10	0	10	0	0	completed

I J E N S

B	5	18	10	10	20	1	5	
C	10	25	10	20	30	1	10	
D	15	15	10	30	40	1	15	
E	15	20	10	40	50	1	25	
F	20	19	10	50	60	1	30	
K	25	22	10	60	70	1	35	
L	30	42	10	70	80	1	40	
M	35	25	10	80	90	1	45	
N	35	45	10	90	100	1	55	
B	-	8	10	100	108	0	80	completed
C	-	15	10	108	118	1	78	
D	-	5	10	118	123	0	78	completed
E	-	10	10	123	133	0	73	completed
F	-	9	10	133	142	0	73	completed
K	-	12	10	142	152	1	72	
L	-	32	10	152	162	1	72	
M	-	15	10	162	172	1	72	
N	-	35	10	172	182	1	72	
C	-	5	10	182	187	0	64	completed
K	-	2	10	187	189	0	35	completed
L	-	22	10	189	199	1	27	
M	-	5	10	199	204	0	27	completed
N	-	25	10	204	214	1	22	
L	-	12	10	214	224	1	15	
N	-	15	10	224	234	1	10	
L	-	2	10	234	236	0	10	completed
N	-	5	10	236	241	0	2	completed
Total	190					18	1142	

Table IV
Analysis Times of IRRVQ

Task	**Arrival times**	**Burst times**	**Quantum time**	**Start service time**	**End service time**	**Time in queue**	**Waiting time**	**Finishing of service**
A	0	10	10	0	10	0	0	completed
B	5	18	10	10	20	1	5	
C	10	25	10	20	30	1	10	
D	15	15	10	30	40	1	15	
E	15	20	10	40	50	1	25	
F	20	19	10	50	60	1	30	
K	25	22	10	60	70	1	35	
L	30	42	10	70	80	1	40	
M	35	25	10	80	90	1	45	
N	35	45	10	90	100	1	55	
B	-	8	8	100	108	0	80	completed
C	-	15	8	108	116	1	78	
D	-	5	8	116	121	0	76	completed
E	-	10	8	121	129	1	71	
F	-	9	8	129	137	1	69	
K	-	12	8	137	145	1	67	

L	-	32	8	145	153	1	65	
M	-	15	8	153	161	1	63	
N	-	35	8	161	169	1	61	
C	-	7	7	169	176	0	53	completed
E	-	2	7	176	178	0	47	completed
F	-	1	7	178	179	0	41	completed
K	-	4	7	179	183	0	34	completed
L	-	24	7	183	190	1	30	
M	-	7	7	190	197	0	29	
N	-	27	7	197	204	1	28	
L	-	17	17	204	221	0	14	completed
N	-	20	17	221	238	1	17	
N	-	3	3	238	241	0	0	completed
Total	190					19	1183	

Table V
Analysis Times of PDQT

Task	Arrival times	Burst times	Quantum time	Priority	Start service time	End service time	Time in queue	Waiting time	Finishing of service
A	0	10	19	10	0	10	0	0	completed
B	5	18	17	8	10	27	1	5	
C	10	25	12	3	27	39	1	17	
D	15	15	18	9	39	54	0	24	completed
E	15	20	15	6	54	69	1	39	
F	20	19	16	7	69	85	1	49	
K	25	22	14	5	85	99	1	60	
L	30	42	11	2	99	110	1	69	
M	35	25	13	4	110	123	1	75	
N	35	45	10	1	123	133	1	88	
B	-	1	8	8	133	134	0	106	completed
C	-	13	3	3	134	137	1	95	
E	-	5	6	6	137	142	0	68	completed
F	-	3	7	7	142	145	0	57	completed
K	-	8	5	5	145	150	1	46	
L	-	31	2	2	150	152	1	40	
M	-	12	4	4	152	156	1	29	
N	-	35	1	1	156	157	1	23	
C	-	10	12	3	157	167	0	20	completed
K	-	3	14	5	167	170	0	17	completed
L	-	29	11	2	170	181	1	18	
M	-	8	13	4	181	189	0	25	completed
N	-	34	10	1	189	199	1	32	
L	-	18	19	2	199	217	0	18	completed
N	-	24	18	1	217	235	1	18	
N	-	6	6	1	235	241	0	0	completed
Total	190						16	1038	

The basic model of the queue in our case study is shown in fig. 5, which contains three general elements.

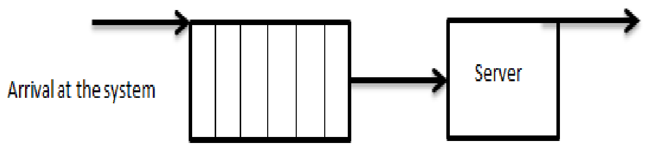

Fig. 5. Basic model of a queue

- The arrival process in the system, which needs to be executed within a specific period of time.
- The order by which the process obtains access to the CPU service, once it joins the queue, where the queue may be finite or infinite, is also known as the queue size.
- The server is the CPU and the departure from the system.

Queue discipline refers to the priority system by which the next process to be serviced is selected from a set of waiting processes. One common queue discipline is the First-In-First-Out process. Service rate (or service capacity) refers to the overall average number of processes a system can handle in a given time. Utilization refers to the proportion of time that a server is busy handling processes.

Several types of queues, such as M/M/1, M/G/1, and G/G/1, are handled using the queueing theory. Our model is focused on M/M/1.

The suitable model with our case study is M/M/1 queue, in which a single server serves jobs that arrive according to a Poisson process and has exponentially distributed service requirements [27]. Equations 1 to 10 illustrate the equations of queueing theory that have been used in the mathematical analysis of the above case study:

$$\text{Avg. T}= \sum_{k=1}^{n} T/n \quad (1)$$

$$\text{Avg. W} = \sum_{k=1}^{n} W/n \quad (2)$$

$$P= nw/n \quad (3)$$

$$C_{active} = 1- (I/To) \quad (4)$$

$$C_{capacity}= \sum_{k=1}^{n} \lambda/\mu \quad (5)$$

$$\text{Avg. S}= \sum_{k=1}^{n} S/n \quad (6)$$

$$\text{Avg. A}= \sum_{k=1}^{n} A/n \quad (7)$$

$$\text{Avg.}_{stay}= To/n \quad (8)$$

$$P1= (1- P) Pn \quad (9)$$

$$M= P/ (1- P) \quad (10)$$

, where Avg. T is the average turnaround time, n is the number of processes, T is the turnaround time (which is equal to the difference between the completion time and arrival time), Avg. W is the average waiting time, W is the difference between the turnaround time and burst time, P is the probability to wait in queue, nw is the number of waiting processes, Cactive is the CPU activity, I is the idle time of the processor, To is the total time in the system, Ccapacity is the CPU capacity, λ is the arrival rate (process arrival per millisecond), µ is the service rate (which is equal to 1/quantum time), Avg. S is the average service time, S is the service time, Avg. A is the average arrival time, A is the arrival time, Avg.stay is the average stay time in the system, P1 is the probability of n processes in CPU, and M is the mean number of processes in CPU.

Table 6 shows the results obtained after applying the ten equations with the input processes of the three algorithms, the traditional algorithm (RR), the improved algorithm (IRRVQ), and the proposed algorithm (PDQT).

Table VI
Analysis Times of PDQT

Factors	RR	IRVVQ	PDQT
n	10	10	10
T	1383	1424	1279
Avg. T	138.3	142.4	127.9
W	1142	1183	1038
Avg. W	114.2	118.3	103.8
CS	28	28	25
nw	18	19	16
P	1.8	1.9	1.6
I	0	0	0
To	241	241	241
C_{active}	100%	100%	100%
λ	0.6	0.6	0.6
µ	2.8	3.5	4
$C_{capacity}$	0.2	0.2	0.2
S	1573	1613	1469
Avg. S	157.3	161.3	146.9
A	190	190	190
Avg. A	19	19	19
Avg. $_{stay}$	24.1	24.1	24.1
P1	1.44n	1.71n	0.96n
M	2.25	2.11	2.66

In the table above, nw for RR is 18 and 19 in IRRVQ while in PDQT is 16. That mean the number of processes in RR and IRRVQ that had to repeat their cycle to enter again to the ready queue and waiting for processing is more than in PDQT by 2 and 3 respectively. Thus, the probability of the processes to wait in the ready queue with PDQT less than in RR and IRRVQ. Because the total time for service in the three algorithms is the same and there is no idle time in CPU (i.e. CPU always busy), so the activity of CPU is 100% for the algorithms. Moreover, the capacity of CPU with RR, IRRVQ and PDQT is the same for this case study. However, other experiments and case studies conducted higher CPU capacity with PDQT over the other algorithms. Processes in RR and IRRVQ take more time for service and average time for service than in PDQT where they are 1573, 157.3, 1613, 161.3 and 1469, 146.9 for RR, IRRVQ and PDQT respectively. The probability of n number of processes to stay in CPU in PDQT less than in RR and IRRVQ, this lead to high mean number of processes in CPU with PDQT over RR and IRRVQ. From all these results we conclude that by setting priorities and changing the quantum time from fixed to dynamic will give RR more flexibility to execute more processes with less time and high throughput.

6.5. Simulation and comparison RR, IRRVQ and PDQT

The performance of the three algorithms can be compared by considering the number of context switches, average waiting time, and average turnaround time with different quantum times to ensure that the same effective results with PDQT are obtained over RR and IRRVQ. We got results of context switches, average turnaround time and average waiting time after applying five different quantum times. Table 7 and Figs 6, 7 and 8 show the obtained results from these different times of RR, IRRVQ and PDQT, respectively.

Table VII
Results of Simulation between RR, IRRVQ and PDQT for five different quantum times

Algorithms	RR			IRRVQ			PDQT		
Factors / QT	CS	Avg. T.T.	Avg. W.T.	CS	Avg. T.T.	Avg. W.T.	CS	Avg. T.T.	Avg. W.T.
10	28	138.3	114.2	28	142.4	118.3	25	127.9	103.8
11	25	139.8	115.6	28	144.8	120.6	19	119.8	95.6
12	24	145.6	100.7	28	143.3	119	19	122.9	98.6
13	22	152.1	117.7	28	149.6	125.2	18	118.7	94.3
14	21	145.6	121.1	28	152	127.5	18	121.3	96.8

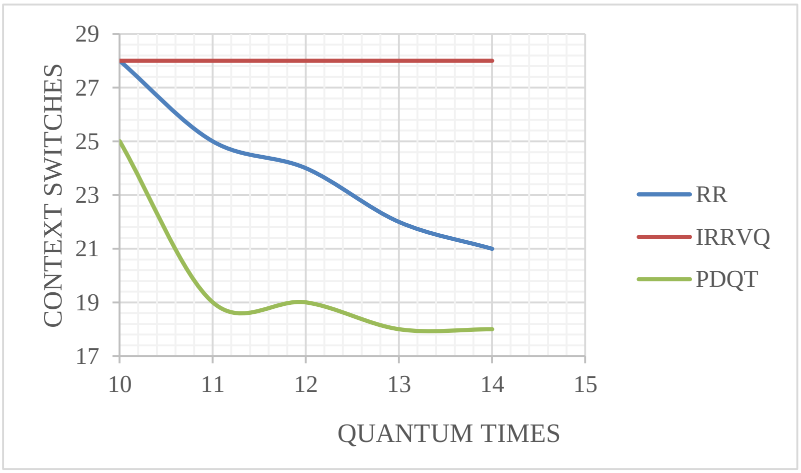

Fig. 6. Performance of RR, IRRVQ and PDQT with different quantum times and context switches

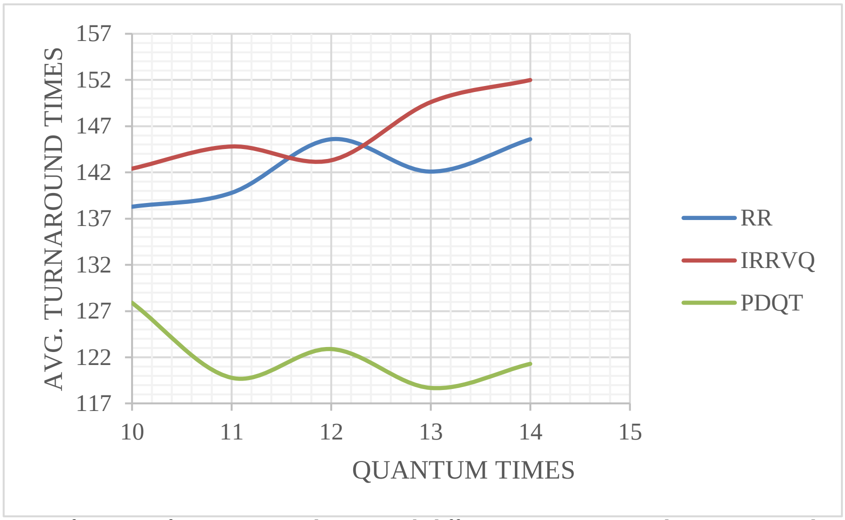

Fig. 7. Performance of RR, IRRVQ and PDQT with different quantum times and Avg. turnaround times

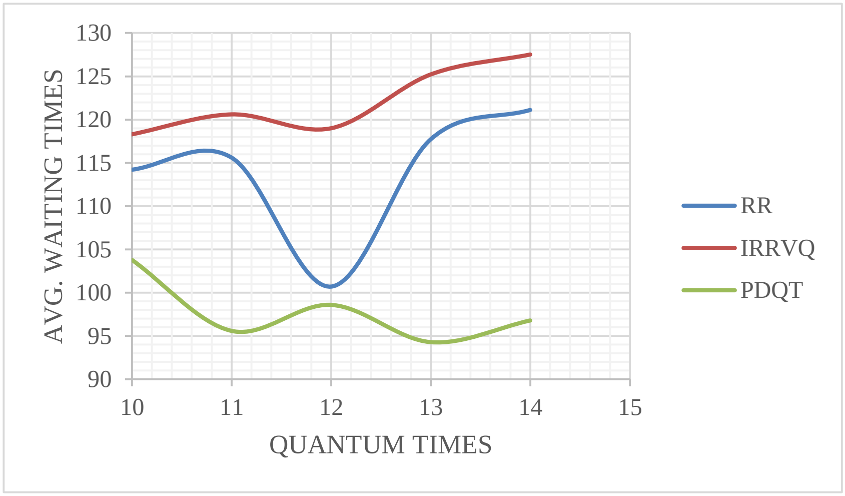

Fig. 8. Performance of RR, IRRVQ and PDQT with different quantum times and Avg. waiting times

The figures above show that the proposed algorithm performs better over the existing algorithm RR and the improved one IRRVQ for dynamic time quantum. It is obvious that the proposed algorithm conduct better results over all factors of performance (context switches, average turnaround time and average waiting time). On the other hand, the improved algorithm called IRRVQ did not actually improve RR for the large number of processes. However, IRRVQ gives a good result over RR with small values of burst times. So, with change the quantum time from fixed to dynamic for processes and cycles will increase the performance of Round Robin scheduling algorithm and forward it to the level of soft real systems.

7. Conclusions and Future Works

Our focusing in this study is the priority and dynamic quantum time for Round-Robin scheduling algorithm and varying time quantum algorithm.

The idea of our algorithm considers different priorities and different quantum times [24]. The two factors that we studied and used in the algorithm are:

- Priority: Fixed Priorities have been assigned to every process, on contrast to RR which has no priorities, and the scheduler arranges the processes in the ready queue in order to their priority.
- Dynamic quantum time: Varying quantum time has been taken instead of fixed quantum time; where the quantum time changes depending on the priorities of the processes. Our experimental results show that our algorithm performs better than RR and IRRVQ algorithms in terms of reducing the number of context switches, average turnaround time and average waiting time.

We have successfully compared three algorithms, namely, simple RR, improved IRRVQ and the proposed algorithm (PDQT). Results indicated that PDQT is more efficient because the fewer context switches and shorter average turnaround and average waiting times over the other both algorithms. 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.

For future work, 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

[1] H. Kopetz, Real-time systems: design principles for distributed embedded applications: Springer. (2011).
[2] H. J. Hassan, "Operating system Concepts," Memory, p. 8. (2013).
[3] A. Silberschatz, P. B. Galvin, and G. Gagne, Operating system concepts vol. 8: Wiley. (2013).
[4] E. Oyetunji and A. Oluleye, "Performance Assessment of Some CPU Scheduling Algorithms," Research Journal of Information and Technology, vol. 1, pp. 22-26. (2009).
[5] F. Cerqueira and B. Brandenburg, "A comparison of scheduling latency in linux, PREEMPT-RT, and LITMUSRT," in Proceedings of the 9th Annual Workshop on Operating Systems Platforms for Embedded Real-Time applications, pp. 19-29. (2013).
[6] L. Yang, J. M. Schopf, and I. Foster, "Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments," in Proceedings of the 2003 ACM/IEEE conference on Supercomputing, p. 31. (2003).
[7] W. Tong and J. Zhao, "Quantum varying deficit round robin scheduling over priority queues," in Computational Intelligence and Security, 2007 International Conference on, pp. 252-256. (2007).
[8] M.-X. Chen and S.-H. Liu, "Hierarchical Deficit Round-Robin Packet Scheduling Algorithm," in Advances in Intelligent Systems and Applications-Volume 1, ed: Springer, pp. 419-427. (2013).
[9] T. F. Hasan, "CPU SCHEDULING VISUALIZATION." Diyala Journal of Engineering Sciences, Vol. 07, No. 01, pp. 16-29. (2013).
[10] A. Singh, P. Goyal, and S. Batra, "An Optimized Round Robin Scheduling Algorithm for CPU Scheduling," IJCSE) International Journal on Computer Science and Engineering, vol. 2, pp. 2383-2385. (2010).
[11] A. Noon, A. Kalakech, and S. Kadry, "A new round robin based scheduling algorithm for operating systems: dynamic quantum using the mean average," arXiv preprint arXiv:1111.5348. (2011).
[12] M. Lavanya¹ and S. Saravanan, "Robust Quantum Based Low-power Switching Technique to improve System Performance," International Journal of Engineering and Technology, Vol 5 No 4. (2013).
[13] R. K. Yadav, A. K. Mishra, N. Prakash, and H. Sharma, "An improved round robin scheduling algorithm for CPU scheduling," International Journal on Computer Science and Engineering, vol. 2, pp. 1064-1066. (2010).
[14] D. Nayak, S. K. Malla, and D. Debadarshini, "Improved round robin scheduling using dynamic time quantum," International Journal of Computer Applications (0975–8887) Volume. (2012).
[15] H. Behera, R. Mohanty, and D. Nayak, "A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis," arXiv preprint arXiv:1103.3831. (2011).
[16] M. K. Mishra, "An Improved Round Robin CPU scheduling algorithm," Journal of Global Research in Computer Science, vol. 3, pp. 64-69. (2012).
[17] A. Abdulrahim, S. E Abdullahi, and J. B Sahalu, "A New Improved Round Robin (NIRR) CPU Scheduling Algorithm," International Journal of Computer Applications, vol. 90, pp. 27-33. (2014).
[18] A. Bhunia, "Enhancing the Performance of Feedback Scheduling," International Journal of Computer Applications, vol. 18, pp. 11-16. (2011).
[19] N. Goel and R. Garg, "An Optimum Multilevel Dynamic Round Robin Scheduling Algorithm," arXiv preprint arXiv:1307.4167. (2013).
[20] B. Lampard, "Program scheduling and simulation in an operating system environment,". (2011).
[21] P. S. Varma, "A FINEST TIME QUANTUM FOR IMPROVING SHORTEST REMAINING BURST ROUND ROBIN (SRBRR) ALGORITHM," Journal of Global Research in Computer Science, vol. 4, pp. 10-15. (2013).
[22] M. Mahesh Kumar, B. Renuka Rajendra, M. Sreenatha, and C. Niranjan, "AN IMPROVED APPROACH TO MINIMIZE CONTEXT SWITCHING IN ROUND ROBIN SCHEDULING ALGORITHM USING OPTIMIZATION TECHNIQUES." International journal of research in engineering and technology, vol. 3. (2014).
[23] M. U. Siregar, "A New Approach to CPU Scheduling Algorithm: Genetic Round Robin," International Journal of Computer Applications, vol. 47. (2012).
[24] R. Mohanty, H. Behera, K. Patwari, M. Dash, and M. L. Prasanna, "Priority based dynamic round robin (PBDRR) algorithm with intelligent time slice for soft real time systems," arXiv preprint arXiv:1105.1736. (2011).
[25] M. K. Mishra and F. Rashid, "AN IMPROVED ROUND ROBIN CPU SCHEDULING ALGORITHM WITH VARYING TIME QUANTUM," International Journal of Computer Science, Engineering & Applications, vol. 4. (2014).
[26] I. S. Rajput and D. Gupta, "A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems," International Journal of Innovations in Engineering and Technology. (2012).
[27] M. Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action: Cambridge University Press. (2013).