IMPROVED ROUND ROBIN POLICY
A MATHEMATICAL APPROACH

D. Pandey
Dept of Mathematics
C C S University
Meerut, India,
pandey_diwakar2k1@rediffmail.com

Vandana
Dept of Computer Applications
Bharat Institute of Technology
Meerut, India,
ranavd@rediffmail.com

Abstract : **This work attempts to mathematically formulize the computation of waiting time of any process in a static n-process, CPU-bound round robin scheme. That in effect, can calculate other performance measures also. An improvement in the existing round robin algorithm has also been worked out that provides priority to processes nearing completion. The suggested approach uses two ready queues, wherein a process is returned to the second ready queue after the completion of its penultimate round. This policy reduces the average waiting time and increases the throughput, in comparison to the conventional round robin scheme, while maintaining the same level of CPU utilization and no substantial increase in the overheads. The mathematical formulation of this policy is transparent enough; it provides, for each process, the actual order of shifting to the second queue and also the order of termination.**

Keywords: CPU scheduling, quantum, dispatcher, context switch, throughput.

I. INTRODUCTION

In CPU scheduling, processes are allocated to CPU on the basis of some specified criteria. There are several methods in existence for this purpose. A preemptive scheduling algorithm possesses the right to take away the control of CPU from a process before its completion. During each such scheduling decision, a context switch occurs, meaning that the current process will stop its execution and put itself back to the ready queue and another process will be dispatched. An improvement in CPU scheduling algorithm is basically an attempt of optimizing some or all of the following criteria:

1. CPU Utilization: Keep CPU utilization as high as possible.
2. Throughput: Complete as many processes as possible in every unit of time.
3. Turnaround Time: Reduce the total time span of the process.
4. Waiting Time: Reduce the total time that a process spends in ready queue.
5. Response Time: Reduce the time between the submission of request and the first response. This measure is the amount of time it takes to start responding [1].

The Round Robin (RR) service discipline is a popular and widely used discipline in many real-world, time-sharing systems because of its fairness. In this discipline, a process is served by the CPU for a single quantum q at a time. If the burst time of the process is less than a quantum, this process is terminated in its first round and the process next in the ready queue is taken by the CPU, assigning a fresh quantum q to it. If the burst time of the process exceeds the quantum size q, the job's processing is interrupted at the end of its quantum and it is returned to the rear end of the ready queue, awaiting service quantum in the next round. This is done to provide fair service to all processes.

The RR scheduling policy gives better responsiveness but worse turnaround time and waiting time. However, it maintains the weighted turn around of processes approximately equal to the number of active processes in the system [2]**.** The RR policy does not perform well in terms of throughput since it treats all processes alike, without giving any favoured treatment to shorter processes.

In CPU-bound processes, reduction of turnaround time, waiting time and response time are all equivalent to a single.criterion of reducing the waiting time only. In waiting line phenomena, usually simulation models are the powerful tools to quantify the waiting time and other performance criteria because an exact probability distribution cannot be assigned to its behavior. In the present work, we have first attempted to mathematically formulize the computation of waiting time of any process in an n-process CPU-bound Round Robin service. That, in effect, can calculate other performance measures also.

Among several variants of Round Robin, the one called Selfish Round Robin [3] uses a structure of multilevel feedback queues in which only the last queue is active and the rest serve as holding queues with increasing priorities attached to them. In the present work, we have also suggested an improvement in the existing Round Robin algorithm that provides priority to the processes nearing completion. Unlike selfish round robin, the suggested approach uses two active ready queues, wherein a process is returned to the second ready queue after the completion of its penultimate round. The Improved Round Robin Policy (IRRP) reduces the waiting time of every process and hence improves all performance measures, including the throughput while maintaining the same level of CPU utilization and no substantial increase in the overheads. The mathematical formulation is transparent to the extent that for each process it provides actual order of it being transferred to the second queue and also its order of termination.

The rest of the paper is organized as follows. In Section 2, we analytically examine the components of the existing Round Robin service algorithm and obtain mathematical expressions for various performance criteria under this service scheme for CPU-bound processes. In Section 3, we develop a new Round Robin service scheme and in Section 4, several mathematical formulae have been obtained for the new scheme, which can significantly reduce simulation efforts. The fifth and last section presents the comparison of results obtained by working on different data-

sets with conventional Round Robin scheme and the new approach.

II. MATHEMATICAL FORMULATION OF ROUND ROBIN SCHEME

Let $P_1, P_2, ...P_n$ be *n*-processes arranged in FCFS manner in a ready queue (to be referred as *Q1*), having x_i and r_i; $i = 1, 2, ...n$, as their expected CPU bursts and number of rounds required for completion, respectively. Further, let *q* denote a fixed quantum in time-units. Following expression can be used to calculate r_i .

$$r_i = \begin{cases} x_i / q, if\ x_i \% q = 0 \\ (x_i / q) + 1, otherwise. \end{cases} \quad .....(1)$$

where x_i *and* q are assumed to have integral values and the symbols '%' and '/' denote modulus and division operators, respectively. For example, if P_1 *and* P_2 are two processes having their CPU bursts as $x_1 = 250$ *and* $x_2 = 275$, then for a quantum size *q* =50, r_1 *and* r_2 will be 5 and 6 respectively. Every process will be executed for one complete quantum in each round, except for the last round where it may be less than or equal to one quantum. The time requirement for a process P_i in its last round (denoted by q_i) is the remainder of (x_i / q).

Remaining CPU burst-time is updated after every round. We use $x_{i,j}$ and $q_{i,j}$ to denote the remaining CPU burst and the executed portion of CPU burst, respectively, for the i^{th} process in j^{th} round. Obviously, the remaining CPU time requirement of a process P_i in r^{th} round can be given by:

$$x_{i,r} = x_{i,r-1} - q_{i,r-1} \quad .....(2)$$

where $q_{i,r-1}$ can be obtained from the following

$$q_{i,r-1} = \begin{cases} q_i, if\ x_{i,r-1} < q; \\ q, if\ x_{i,r-1} \geq q. \end{cases} \quad .....(3)$$

Total waiting time for a process P_i that terminates in r_i^{th} round (P_i (TWT)) can be calculated by summing up its Waiting Time (WT) in all rounds from first to r_i^{th} round.

WT of P_i in round 1 = $(q_{1,1} + q_{2,1} + ... + q_{i-1,1})$,

WT of P_i in round 2

= $(q_{i+1,1} + q_{i+2,1} + ... + q_{n,1}) + (q_{1,2} + q_{2,2} + ... + q_{i-1,2})$,

WT of P_i in round 3

= $(q_{i+1,2} + q_{i+2,2} + ... + q_{n,2}) + (q_{1,3} + q_{2,3} + ... + q_{i-1,3})$,

……..

WT of P_i in round r_i

= $(q_{i+1,r_i-1} + q_{i+2,r_i-1} + ... + q_{n,r_i-1}) + (q_{1,r_i} + q_{2,r_i} + ... + q_{i-1,r_i})$.

Total waiting time of the process P_i can be obtained by summing up all above expressions.

$$P_i(TWT) = (\sum_{k=1}^{i-1} q_{k,1}) + [\sum_{k=i+1}^{n} q_{k,1} + \sum_{k=1}^{i-1} q_{k,2}] + [\sum_{k=i+1}^{n} q_{k,2} + \sum_{k=1}^{i-1} q_{k,3}] + ... + [\sum_{k=i+1}^{n} q_{k,r_i-1} + \sum_{k=1}^{i-1} q_{k,r_i}] \quad .....(4)$$

=

$$[\sum_{k=1}^{n} q_{k,1} - q_{i,1}] + [\sum_{k=1}^{n} q_{k,2} - q_{i,2}] + ... + [\sum_{k=1}^{n} q_{k,r_i-1} - q_{i,r_i-1}] + [\sum_{k=1}^{i} q_{k,r_i} - q_{i,r_i}]$$

$$= \sum_{k=1}^{n} [q_{k,1} + q_{k,2} + ... + q_{k,r_i-1}] + \sum_{k=1}^{i} q_{k,r_i} - \sum_{k=1}^{r_i} q_{i,k}$$

Using $\sum_{k=1}^{r_i} q_{i,k} = x_i$, following expression is obtained for $P_i(TWT)$

$$P_i(TWT) = \begin{cases} \sum_{k=1}^{n} \sum_{j=1}^{r_i-1} q_{k,j} + \sum_{k=1}^{i} q_{k,r_i} - x_i, \ for\ r_i > 1; \\ \sum_{k=1}^{i} q_{k,r_i} - x_i, \qquad for\ r_i = 1. \end{cases}$$

.....(5)

where $q_{j,r} = 0$ for, $r > r_j$, since the processes that have already been completed in earlier rounds will not contribute to the right hand side of (5).

$$Average\ Waiting\ Time = \sum_{i=1}^{n} P_i(TWT) / n \quad .....(6)$$

Turnaround time of a process P_i can be calculated by the following expression:

$$Average\ Turnaround\ Time = \sum_{i=1}^{n} P_i(TT) / n \quad .....(7)$$

III. IMPROVED ROUND ROBIN POLICY (IRRP)

In Round Robin principle, processes are executed in the order of their arrival. However unlike FCFS, the processes get only a fixed quantum of CPU time in each round. RR therefore avoids a long wait for first CPU response. A process may thus need several rounds for completion. A major drawback in Round Robin policy is that even if a process is near completion, it is still placed at the rear end of *Q1*, which not only increases the total waiting time but also lowers the throughput.

IRRP is an attempt to combine the basic functionalities of RR with an improvement towards the priority assigned to the processes nearing completion. In view of (1), it is obvious that the time requirement for completion of a process P_i after

$(r_i - 1)^{th}$ round will be at the most one time quantum. We therefore, consider a priority queue (to be referred as *Q2*) in addition to the ready queue *Q1*. An additional queue has been used by Pandey et.al. [4] for dispatching priority in context of FCFS scheduling. All processes, after being served by the CPU in penultimate round, are sent to the rear end of *Q2* instead of *Q1*. Thus the processes which need only one quantum or less will be terminated in the first round itself from *Q1*, while all others will be terminated on being dispatched from *Q2*. Therefore processes going to CPU through *Q1*, if not terminated, may return back to the rear end of either *Q1* or *Q2*. As shown in Fig.1, this approach organizes the pending requests in two queues. The scheduling policy adopts cycle of *three* processes for the purpose of sequential allocation to CPU; it starts with *two* processes from *Q1* followed by *one* process from *Q2*.The policy can be violated only under following special situations:

- **(i)** At any stage, after dispatching two processes from *Q1*, if *Q2* is found to be empty, another pair of processes will be dispatched from *Q1*.
- **(ii)** If *Q1* is left with a single process, *Q2* will have its turn immediately after the dispatch of the single process from *Q1*.
- **(iii)** If *Q1* is left with no process, *Q2* will function as a single ready queue.

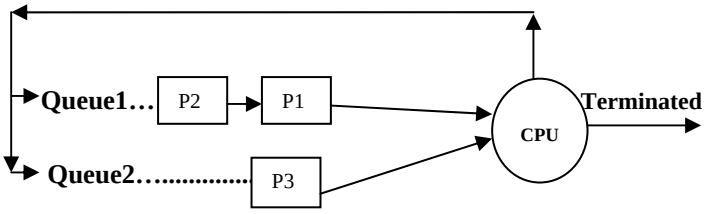

Figure 1

The scheduling policy can further be improved by adopting some different cycle. Precise idea is to appropriately choose a pair of numbers *p* and *q (p>q)* that determine the number of processes from *Q1* and *Q2* for allocation to CPU in the cycle. An optimal choice may however, depend on the number of processes and the size of their CPU bursts. In the present work, we shall confine our discussion to *p=2* and *q=1*.This policy provides better estimates than the conventional RR policy in respect of all performance measures, including the throughput, without any significant increase in the overheads.

IV. MATHEMATICAL COMPUTATIONS FOR IRRP

In an effort to obtain mathematical expressions for various performance measures of IRRP, we first need to know the order in the sequence (in terms of elapsed time slices of CPU use) in which a process is finally terminated under the new policy. Let $P_1, P_2, ...P_n$ be *n processes* under consideration, needing $r_1, r_2, ...r_n$ rounds of CPU execution respectively. We use following notations in addition to those given in Section 2.

$p_{i,r}$: Number of processes P_j (having index j < i) gone to *Q2* up to r^{th} round;

$p_{r,i}$: Number of processes P_j (having index j > i) gone to *Q2* up to r^{th} round;

$p_r := p_{i,r} + p_{r,i}$;

$Q(P_i)$: Order in the sequence in which a process P_i goes to *Q2;*

$T'(P_i)$: Tentative order in the sequence in which a process P_i terminates on completion;

$T(P_i)$: Actual order in the sequence in which a process P_i terminates on completion.

To determine $T(P_i)$ we first need to know $Q(P_i)$.

Determination of $Q(P_i)$

In IRRP, a process P_i, requiring r_i rounds of CPU executions, moves to *Q2* after $(r_i\text{-}1)^{th}$ execution. Thus, its order in the sequence for going to *Q2* can be obtained by first determining its order at $(r_i\text{-}1)^{th}$ execution under conventional RR policy (to be denoted by $Q_{RR}(P_i, r_{i-1})$) and subtracting from it the number of processes gone to *Q2* prior to P_i up to $(r-2)^{th}$ round, that may affect $Q_{RR}(P_i, r_{i-1})$. In fact all processes that have gone to *Q2* up to $(r-3)^{th}$ round and only those processes P_j having *j < i*, gone to *Q2* in $(r-2)^{th}$ round will affect the order of P_i in $(r-1)^{th}$ round. Therefore, this number is ($p_{r-3,i} + p_{i,r-2}$).

For $r_i = 1$; $Q(P_i) = 0$, since a process terminating in the first round would never go to *Q2*.

For $r_i = 2$; $Q(P_i) = Q_{RR}(P_i, 1) - 0 = i - 0 = i$, since the order of execution of P_i in first round under RR policy is i^{th} and no process can go to *Q2* up to $(r-2)^{th}$ round.

For $r_i \geq 3$; $$Q(P_i) = Q_{RR}(P_i, r_i - 1) - (p_{r_i-3,i} + p_{i,r_i-2}) \quad \text{....(8)}$$

To determine, $Q(P_i)$ for $r_i = 3$ onwards, we need to calculate $Q_{RR}(P_i, r_i - 1)$. We must therefore evaluate the number of time slices that have been used by the CPU in each round, before P_i is executed in $(r_i - 1)^{th}$ round under RR policy. Let t_r and $t_{i,r}$ respectively denote the number of processes terminated up to r^{th} round and the number of processes P_j (having index j < i), terminated up to r^{th} round under conventional RR policy .

In first round all *n* processes will be executed by the CPU, so total number of processes executed in first round = *n*
Total number of processes to be executed in the second round = *n - the number of processes terminated up to first round*
= $n - t_1$
Total number of processes to be executed in the third round= *n - number of processes terminated up to second round*
= $n - t_2$
……....
Total number of processes to be executed in the $(r_i - 2)^{th}$ round= *n – number of processes terminated up to* $(r_i - 3)^{th}$ *round*
= $n - t_{r_i - 3}$
In the $(r_i - 1)^{th}$ round we are concerned with only those processes which are executed prior to P_i in this round. It is obvious that the processes terminating after the P_i^{th} process in $(r_i - 2)^{th}$ round will not contribute to the order of P_i in $(r_i - 1)^{th}$ round.
Therefore the number of processes executed up to P_i in the $(r_i - 1)^{th}$ round = *i – (number of processes* P_j *(having index* $j < i$ *), terminated up to* $(r_i - 2)^{th}$ *round)* = $i - t_{i,r_i-2}$
Summing up all above expressions and using $t_0 = 0$,

$$Q_{RR}(P_i, r_i - 1) = (n - t_0) + (n - t_1) + (n - t_2) + \ldots + (n - t_{r_i-3}) + i - t_{i,r_i-2}$$

$$= \sum_{k=0}^{r_i-3} (n - t_k) + i - t_{i,r_i-2}, \text{ for } r_i \geq 3. \quad \ldots.(9)$$

From (8) and (9), following expression is obtained for $Q(P_i)$; $r_i \geq 3$.

$$Q(P_i) = Q_{RR}(P_i, r_i - 1) - (p_{r_i-3,i} + p_{i,r_i-2})$$

$$= \sum_{k=0}^{r_i-3} (n - t_k) + i - t_{i,r_i-2} - (p_{r_i-3,i} + p_{i,r_i-2}),$$

$$= \sum_{k=0}^{r_i-3} (n - p_{k-1}) + i - p_{i,r_i-3} - (p_{r_i-3,i} + p_{i,r_i-2}),$$

substituting $p_{i,r-1} = t_{i,r}$, where $p_{i,0} = t_{i,1}$ = number of processes terminating before P_i in the first round and $p_{r-1} = t_r$, where $p_{-1} = t_0 = 0$ and $p_0 = t_1$ = number of processes terminating in the first round.

$$= \sum_{k=0}^{r_i-3} (n - p_{k-1}) + i - p_{r_i-3} - p_{i,r_i-2},$$

since $p_{i,r_i-3} + p_{r_i-3,i} = p_{r_i-3}$. ……(10)

To understand the working of expression (10), let us consider an example of following five processes functioning under RR policy having a quantum of 50 time units : $P_1(350)$, $P_2(125)$, $P_3(475)$, $P_4(250)$, $P_5(75)$.
Table 1, demonstrates determination of $Q(P_i)$ according to (10).

Table 1

$\mathbf{P_i}$	$\mathbf{r_i}$	$\sum_{k=0}^{r_i-3}(n - p_{k-1})$	p_{r_i-3}	p_{i,r_i-2}	$Q(P_i)$
P_1	7	20	3	0	18
P_2	3	5	0	0	7
P_3	10	25	4	2	22
P_4	5	14	2	1	15
P_5	2	5	0	0	5

Determination of $T(P_i)$

The actual order of termination $T(P_i)$ for the process P_i (needing r_i rounds for completion) depends not only on $Q(P_i)$ but also on the number of processes already present in *Q2* at the arrival of P_i. Therefore a tentative value $T'(P_i)$ for the order of termination is first obtained, which is later finalized according to the situation. In the determination of $T'(P_i)$, we take into account following facts:

1. IRRP adopts a cycle of *two* processes first from *Q1*, followed by *one* process from *Q2*, for the purpose of allocation to CPU.
2. All processes needing two or more rounds for completion will be terminated only through *Q2*.
3. Processes terminating in their first round of execution do not move to *Q2*.

In view of the above points, it can easily be observed that if a process P_i is assumed to be the first process going to *Q2* then its order in the sequence for termination will be an odd number with respect to $Q(P_i)$. Therefore to obtain $T'(P_i)$, either 1 or 2 must respectively be added to $Q(P_i)$ according to its even or odd value. Further, if P_i is not the first process to go to *Q2*, then the number of processes already gone to *Q2* before P_i must also be added in the earlier sum in order to get the value of $T'(P_i)$. Assuming n_i to be the number of processes gone to *Q2* before P_i, following expression is obtained for $T'(P_i)$.

$$T'(P_i) = \begin{cases} Q(P_i) + 1 + n_i, & \text{if } Q(P_i) \text{ is even} \\ Q(P_i) + 2 + n_i, & \text{otherwise} \end{cases} \quad \ldots\ldots(11)$$

where n_i denotes the number of processes P_j satisfying $Q(P_j) < Q(P_i)$.

The tentative value $T'(P_i)$ given by (11) may differ from the actual value $T(P_i)$ in some special situations, for example, when *Q1* falls short of the requisite number of processes to be dispatched according to IRRP cycle or *Q1* being emptied or the process under consideration being the last, second last or third last etc.. It must however be noted that in all the cases, the order of the last terminating process is always equal to $\sum_{i=1}^{n} r_i$. These situations require following modifications in $T'(P_i)$ to arrive at the actual termination point $T(P_i)$.

Case I: $T'(P_i) < \sum_{i=1}^{n} r_i - 2$.

In view of the dispatch cycle adopted in IRRP, the difference between the termination of two processes must atleast be three, with an exception of some special cases referred earlier. To ascertain it, following modifications can be done:

$$T(P_i) = \begin{cases} T(P_j) + 3, & if\ T'(P_i) - T(P_j) < 3 \\ T'(P_i), & otherwise, \end{cases} \quad \text{.....(12)}$$

provided the modified termination number $T(P_i)$ also remains within the bounds of this case, that is, $T(P_i) < \sum_{i=1}^{n} r_i - 2$, where $T(P_j)$ denotes the actual termination number of the process terminated immediately preceding P_i (for the first terminating process, $T(P_j) = 0$). The case where $T(P_i) \geq \sum_{i=1}^{n} r_i - 2$ is dealt according to case II.

Case II : $T'(P_i)$ (or $T(P_i)$ obtained from Case I) $\geq \sum_{i=1}^{n} r_i - 2$.

$$T(P_i) = \begin{cases} \sum_{i=1}^{n} r_i - 2, & if\ n - n_i = 3; \\ \sum_{i=1}^{n} r_i - 1, & if\ n - n_i = 2; \\ \sum_{i=1}^{n} r_i, & if\ n - n_i = 1. \end{cases} \quad \text{.....(13)}$$

To understand the working of expressions (11), (12) and (13), let us consider the example of ten processes functioning under RR policy with a quantum of 50 time units.
Table 2 given immediately before section IV, demonstrates the functioning of the IRRP in terms of *Q1* and *Q2* and determines the values of $Q(P_i), T'(P_i)$ and $T(P_i)$ with the help of (11), (12) and (13).

Waiting Time of a Process

The order of termination $T(P_i)$ of a process determines the number of time slices during which, CPU was in use, till the termination of P_i. These time slices may include the following:

1. All time slices consumed for the process P_i;
2. Number of time slices of full quantum consumed for processes other than P_i;
3. Number of time slices of full or part of the quantum, consumed in the last round of the processes terminated before P_i.

In the determination of total waiting time for the process P_i, we need to identify the time slices of each of the above category and compute their number. The expression to calculate total waiting time of a process is as follows:

$P_i(TWT)$ = CPU time consumed before the termination of P_i, for processes other than P_i

= Quantum × (number of full quantum slices used by other processes before the termination of P_i) + total CPU time used by the processes terminated before P_i, in their last rounds.

= Quantum × [$T(P_i)$ - number of rounds needed for P_i - number of time slices used in the last rounds of the processes terminating before P_i] + sum of (burst time – time consumed up to penultimate round) for all processes that have terminated before P_i.

$$= (T(P_i) - r_i - n_i) * q + \sum_{\substack{P_k \\ T(P_k) < T(P_i)}} [x_k - (r_k - 1) * q)]$$

Thus

$$P_i(TWT) = (T(P_i) - n_i - r_i) * q + \sum_{\substack{P_k \\ T(P_k) < T(P_i)}} [x_k - (r_k - 1) * q)] \quad \text{.....(14)}$$

The average waiting time $AWT = \sum_{i=1}^{n} P_i(TWT) / n$

..……(15)

Table 2

$P_1(80), P_2(175), P_3(200), P_4(245), P_5(125), P_6(220), P_7(140), P_8(230), P_9(150), P_{10}(120)$

ROUND	ORDER OF PROCESSES											$Q(P_i)$	$T'(P_i)$	$T(P_i)$
1	$Q1$	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_1	P_1	P_1
	$Q2$	P_1										1	3	3
2	$Q1$	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}		$P_5\ P_7\ P_9\ P_{10}$	$P_5\ P_7\ P_9\ P_{10}$	$P_5\ P_7\ P_9\ P_{10}$
	$Q2$				P_5		P_7		P_9	P_{10}		14 16 18 19	16 19 22 25	16 19 22 25
3	$Q1$	P_2	P_3	P_4	P_6	P_8						$P_2\ P_3$	$P_2\ P_3$	$P_2\ P_3$
	$Q2$	P_2	P_3									20 21	26 29	28 31
4	$Q1$	P_4	P_6	P_8								$P_4\ P_6\ P_8$	$P_4\ P_6\ P_8$	$P_4\ P_6\ P_8$
	$Q2$	P_4	P_6	P_8								25 26 27	34 35 38	34 36 37

IV. RESULTS AND CONCLUSIONS

We worked on several randomly generated data sets in an attempt to compare performance of proposed IRRP with conventional RR policy with respect to average waiting time. Results obtained on different data sets are presented in Table 3.

Table 3

Data sets Size / Quantum	Range of CPU Burst	Average Waiting Time	
		RR	IRRP
set 1 50/10	10-100 tu	1886	1649
set 2 50/30	25-134 tu	2764	2159
set 3 50/20	11-150 tu	3042	2595
set 4 50/50	36-539 tu	8119	7072
set 4 50/40	36-539 tu	8183	7306
set 4 50/30	36-539 tu	8223	7505
set 4 50/20	36-539 tu	8197	7719
set 4 50/10	36-539 tu	8222	7982

Data set 4 has been worked on five different values of time quantum. IRRP shows better average waiting time than RR in all the cases. This would, in turn, amount to better results toward average turnaround time also. In terms of throughput, IRRP serves better than the conventional RR because of its policy of assigning priority to the processes nearing completion. Those processes that require only two rounds of CPU service, shall slightly delay the first CPU response of the processes following them but in all other situations, the first CPU response time of IRRP and conventional RR remains the same.

For the visual perception of the difference in the performance of improved round robin policy against the conventional round robin policy we present the comparative bar-graphs for average waiting time and average turnaround time of the two policies in Figures 2 and 3.

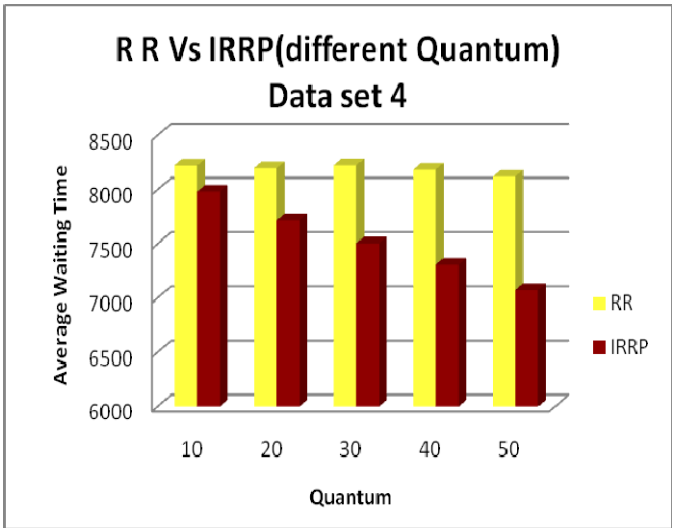

Figure 2

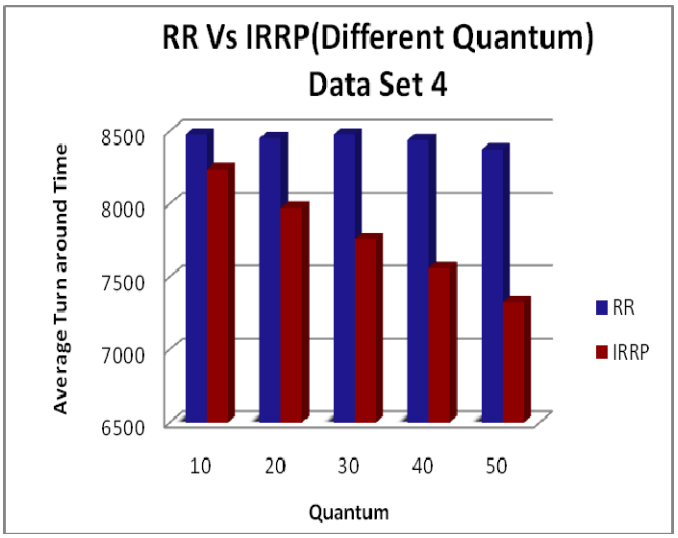

Figure 3

The scheduling policy discussed in this work is competitive to the conventional Round Robin scheme in fairness, in the first assignment time of the processor and in respects of the overheads due to preemption, but it certainly performs better, in terms of average waiting time, average turnaround time and the throughput of the system. Moreover, the derivation of related mathematical expressions is useful, not only in reducing the simulation efforts but also in making the policy transparent to the extent that for each process it provides actual order of its being transferred to the second queue and also its order of termination. The method is different from the one suggested by Kleinrock [2] using multi-level feedback queues; however, a cross-application of the structure of multi-level feedback queues with the basic approach of the new policy may be worked in future, to provide priority at different levels to the processes nearing completion.

REFERENCES

[1] Silberschatz, A., Galvin P.B., and Gagne, G. *Operating System Concepts*, India, Wiley Student Edition, John Wiley, 2004.

[2] D.M. Dhamdhere Operating Systems: A Concept-Based Approach, New Delhi: Tata McGraw-Hill 2002.

[3] L. Kleinrock, "A Continuum of Time Sharing Scheduling Algorithms", *Proceedings of AFIPS*, SJCC 453-458, 1970.

[4] D. Pandey, Vandana and M. K. Sharma "CPU Scheduling: FCFS with Shorter Processes First", *MR Int. J. of Engg. and Tech*. 1 (2):11-17,2008.

AUTHORS PROFILE

D. Pandey is Professor of Mathematics and Dean Faculty of Engineering & Technology in Ch. Charan Singh University, Meerut (India). He obtained his Bachelors and Masters degrees from Allahabad University (India) and doctorate from Meerut University Meerut, (India). His research interests include Reliability theories, Fuzzy Sets and Operating systems. He has published more than 45 research papers in various National and International Journals of repute in different areas of Mathematics and Computer Science.

Vandana, an Asst. Prof., Dept of Computer Applications in BIT college, Meerut (India), is a doctoral candidate in Computer Science. She has published 3 research papers in computer science. Her research interest includes CPU scheduling theories, and Natural Language Processing. She is life member of International Academy of Physical Sciences.