Design and Performance Evaluation of Optimum Service Time Concept for Round Robin Algorithm (OSTRR)

F.A. Himanshi Saxena and S.B. Prashant Agarwal

***Abstract*—Problem Statement: Extending the concept of Round Robin algorithm (RR) to incorporate user or system defined priority and consider the case of different arrival times of process and suggesting a novel approach that minimizes context switching overhead, average waiting time and turnaround time. Approach: We calculate Optimum Priority 'OP' for each process which determines the order of execution of processes, Optimum Service Time 'OST' for each process which determines time of execution of process in a single round and execute the processes in RR fashion using the calculated parameters. Results: Based on experiments and calculations, the proposed algorithm is successful in reducing afore mentioned problems. Conclusion: Our proposed algorithm can be effective in priority based systems where burst time and arrival time can be easily predicted.**

***Index Terms*—Scheduling algorithm, context switch, waiting time, arrival time, turnaround time, priority, round robin, time quantum.**

I. Introduction

A CPU scheduling algorithm should focus on maximising throughput and CPU utilization and minimizing turnaround time, waiting time and response time. Apart from these parameters there are other factors to consider like number of context switches, fairness etc. [1] [2] [3] [4] [11] [12]. Existing scheduling techniques based on priority of the process are not fair and responsive and suffer from the problem of starvation. If we use RR algorithm then we can achieve fairness and responsiveness but at the cost of neglecting the effect of user or system defined priority totally and large number of context switches. Neglecting priority is certainly not always good. If we use priority based RR algorithm [9] we still have the problem of large number of context switches. Some algorithms; that have been worked out; do not consider the case of different arrival times of the process. The effect of context switching overhead is considerable which can be checked from [8]. This motivated us to work on a novice algorithm that adopts an approach that is fairer, gives consideration to user defined or system defined priority of the process and minimizes the number of context switches and thus more suitable for priority based systems or soft real time systems. We incorporate the concept of optimum service time rather than time quantum for deciding the time for which a process is allowed to execute when competing for CPU time and optimum priority rather than user or system defined priority of process to decide the order of execution of processes.

The rest of the paper is divided as follows: Section II deals with existing algorithms. Section III describes related work till now. Section IV gives the details of our proposed approach. Section V compares the performance of our proposed scheduling algorithm with DQRRR [10] which considers the case of different arrival times of the process but fails to consider the effect of user or system defined priority of the process. Conclusion and direction of future work is given in section VI.

Manuscript received March 30, 2012. This work was supported by Faculty of Department of Computer Science and Engineering, Madan Mohan Malviya Engineering College, Gorakhpur, India.

Authors are with Computer Science and Engineering from Madan Mohan Malviya Engineering College, Gorakhpur, India (email: himanshi57@gmail.com; pidge.1000@gmail.com)

II. Existing Algorithms

Many scheduling algorithms have been proposed for processor assignment, but there is no suitable algorithm for all purposes. We discuss the most commonly used algorithms Priority based algorithm and RR algorithm in this section and highlight the drawbacks of these two.

Priority scheduling algorithm assigns CPU to competing processes according to the priority of the process. Priority can be determined by the user or by the system. Priority assignment can be fixed or dynamic and the approach adopted can be pre-emptive or non pre-emptive. Priority scheduling algorithm suffers from problem of starvation. They are not fair as they are biased to process of high priority.

Round Robin algorithm is simplest, fairest and most widely used scheduling algorithm especially designed for time sharing systems [5] [6] [7] [9] [10]. A small unit of time called time quantum is defined. All runnable processes are kept in circular queue. The CPU scheduler goes around this queue allocating the CPU to each process for a time interval of one quantum. New processes are added to tail of the queue. It does not suffer from problem of starvation and is fair. Round Robin algorithm, however does not take into account the priority of process at all. It is not desirable if the size of the jobs or tasks are strongly varying [2] [4] [5]. It also has more context switching overhead than other algorithms.

III. Related Work

Efforts have been made to modify RR in order to give better turnaround time, average waiting time and minimize context switches. Changes to time quantum according to the nature of process have been suggested in [5] [6] [7] [9] [10] [11]. Employing the concept of priority along with RR has been suggested in [9]. The concept of dynamic time quantum has been suggested in [5] [6] [7] [9] [10].

IV. Proposed Algorithm

A. Approach

For achieving low average waiting time and turnaround time it is advisable to execute processes with small burst time early. We also propose that any process that has arrived earlier should not be stalled for long and should be executed as early as possible. Since we consider priority based systems, we employ the concept of Optimum Priority which combines user defined priority, effect of shorter burst time and effect of arrival time in a way so as to achieve better turnaround time and average waiting time.

We assign a weight of 0.5 to user or system defined priority, a weight of 0.3 to burst time and 0.2 to arrival time. This ensures that user defined priority; burst time; and arrival time get consideration while deciding order of execution of processes. Note that a higher priority process gets a higher number; a shorter process also gets a higher number; and a process that arrived earlier also gets a higher number in our numbering scheme. We now calculate Optimum Priority 'OP' which is:

$$OP= 0.5*P + 0.3*BT+0.2*AT \quad (1)$$

Here P is user or system defined priority; BT is priority number assigned according to shorter burst time; and AT is priority number assigned according to early arrival of the process. We round off the value of 'OP' to the nearest integer for rest of the calculations.

We now decide a time quantum which is selected while taking into account the same considerations that is taken while selecting time quantum for RR algorithm. Now we decide the Optimum service time 'OST' for each process. It is given by:

$$OSTi = OPi * q \quad (2)$$

where OSTi is the service time of process with Optimum priority 'OPi'. 'q' is the decided time quantum.

We place all the processes in a priority queue. Priority queue can be implemented as heap. After calculating the service time of each process we assign CPU to the process with highest optimum service time. In case of conflict the process with shorter burst time is given preference. If conflict still persists the process that arrived early is given preference. The process decided executes for a period that is equal to optimum service time of the process OSTi. or its burst time whichever is smallest. In case a process or the set of process with same value of 'OP', which so ever is applicable, finish execution after a single round, we redistribute the value of 'OST' removing the value of OST of finished process and redistributing the value of 'OST' accordingly. For e.g. suppose there are five processes P1, P2, P3, P4, P5 whose 'OP' and 'OST' have been calculated. Value of 'OST' for the processes is 12, 16, 8, 8 and 16 respectively. Suppose P1 finishes execution. Then the value of 'OST' for P2 and P5 will remain 16 but for P3 and P4, it will change to 12 and 12 respectively thus increasing the execution time for rest of the process after the round. The value of 'OST' is self adjusting in this way.

B. Pseudo Code of Algorithm

```
process_priority_queue = empty

while (true) {
     for each new process {
            add process to process_priority_queue
            along with its priority P
            }
scheduler();
if (process_terminate) {
              if (process_priority_queue is empty)
                    wait until new process arrives
               else
                    recalculate_OST ( ); } }

pre_scheduler(q) {
 for each process i   {
     Assign BT to all the processes such that processes
     with shorter burst time get higher number and assign
     AT to all the processes such that processes that
arrived
    earlier get higher number
 OP= 0.5*P + 0.3*BT+0.2*AT//Round of FP to nearest
integer
 OST_i = OP_i * q // calculate OST
                          } }

scheduler ( ) {
   pre_scheduler (time_quantum)
   //process_selector
      assign CPU to process with highest value of OST.
       If more than one process assign CPU to process
       with shorter burst time among them, if conflict
       still persist assign CPU to process that arrived early.
if (burst time < optimum service time)
                    execute process till its burst time
          else
                    execute process till OST_i


    //next_process
    select  next process from process_priority_queue
                   }

recalculate_OST ( ) {
    if (process_terminate)
       redistribute_OST
     else
              do nothing
          }
```

V. Experiments and Illustration

We compare our algorithm with DQRRR [12] in terms of number of context switches, average turnaround time and average waiting time. DQRRR does not take into account the effect of user or system defined priority which gives our algorithm an intrinsic advantage in priority based systems.

A. Assumptions

All experiments are assumed to be performed in uniprocessor environment and all the processes are

independent from each other. The attributes of all the processes like burst time and user or system priority are known before submitting the process. All processes are CPU bound. No process is I/O bound.

B. Experiment

Case 1) Suppose there are five processes with following burst time, user priority and increasing order of arrival time as shown in Table I:

TABLE I: PARAMETERS FOR CASE I

Process	Burst time	User Priority 'P'	Arrival Time 'AT'
P1	28	5	0
P2	35	4	2
P3	50	3	6
P4	82	2	6
P5	110	1	8

Burst time and arrival time have been assumed to be in milliseconds. Assuming time quantum to be 25 millisecond. We assign value of BT in such a way that shorter process gets higher number and AT such that process that arrived earlier gets higher number as shown in Table II:

TABLE II: CALCULATION OF OP AND OST FOR CASE II

Process	AT	BT	P	'OP'	'OST' for 1st round
P1	5	5	5	5.0	5*25 = 125
P2	4	4	4	4.0	4*25 = 100
P3	3	3	3	3.0	3*25 = 75
P4	3	2	2	2.2	2*25 = 50
P5	2	1	1	1.2	1*25 = 25

We begin with P1. After 1st round of execution P1, P2 and P3 have finished executing. So we assign redistribute 'OST' and now 'OST' of P4 becomes 125 and 'OST' of P5 becomes 100. In 2nd round P4 and P5 finish execution as shown in Table III. Scheduling with DQRRR is shown in Table IV.

TABLE III: OSTRR

Process	OST	
	1st round	2nd round
P1	125	0
P2	100	0
P3	75	0
P4	50	125
P5	25	100

Gantt chart according to our proposed algorithm is shown in Fig.1.

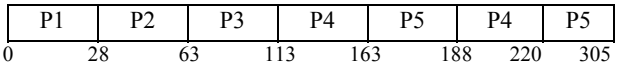

Fig. 1. Gantt chart with OSTRR for case 1

TABLE IV: DQRRR

Algorithm	Time Quantum
DQRRR	28, 66, 30, 14

Gantt chart of DQRRR is shown below in Fig2:

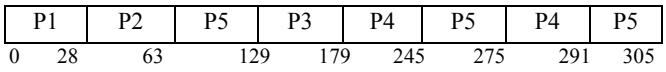

Fig. 2. Gantt chart with DQRRR for case 1

To calculate the average waiting time, turnaround time and number of context switches we have performed simulations in C on gcc compiler. The snapshot of results of simulation has been shown. We have shown the comparison of several parameters for OSTRR and DQRRR in Table V and through a chart in Fig 4.

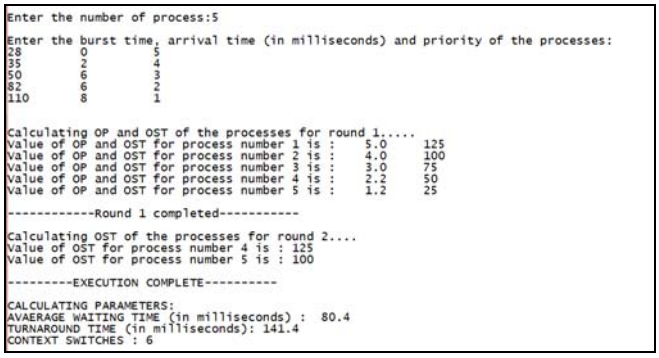

Fig.3: Snapshot of simulation of OSTRR for case 1

TABLE V: COMPARISON FOR CASE I

Algorithm	Turnaround time (TAT)	Average waiting time (AWT)	Context Switches (CS)
DQRRR	209.8	147.8	7
OSTRR	141.4	80.4	6

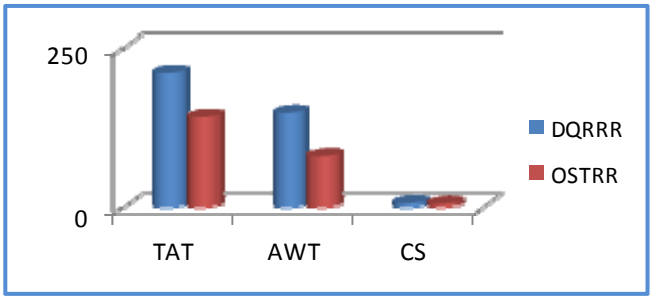

Fig.4: Comparison of DQRRR and OSTRR for case1

Case 2) Suppose there are 5 processes with following burst time, user priority and decreasing order of arrival time as shown in Table VI:

TABLE VI: PARAMETERS FOR CASE II

Process	Burst time	User Priority 'P'	Arrival Time 'AT'
P1	80	5	0
P2	72	4	2
P3	65	3	3
P4	50	2	4
P5	43	1	5

Time quantum is 25 milliseconds. Calculating 'OP' and 'OST' we get as shown in table VII:

TABLE VII: CALCULATION OF OP AND OST FOR CASE II

Process	BT	UP	AT	'OP'	'OST' for 1st round
P1	1	5	5	3.8	4*25=100
P2	2	4	4	3.4	3*25=75
P3	3	3	3	3.0	3*25=75
P4	4	2	2	2.6	3*25=75
P5	5	1	1	2.2	2*25=50

Scheduling with OSTRR is shown in Table VIII:

TABLE VIII: OSTRR

Process	OST
	1st Round
P1	100
P2	75
P3	75
P4	75
P5	50

Gantt chart for OSTRR is shown in Fig 5:

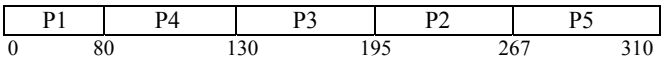

Fig 5: Gantt chart for OSTRR case 2

Scheduling with DQRRR is shown in Table IX:

TABLE IX: DQRRR

Algorithm	Time Quantum
DQRRR	80, 57, 11, 4

Gantt chart for DQRRR is shown in Fig 6:

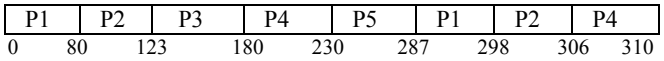

Fig 6: Gantt chart for DQRRR case 2

The snapshot of results of simulation has been shown in Fig 7.

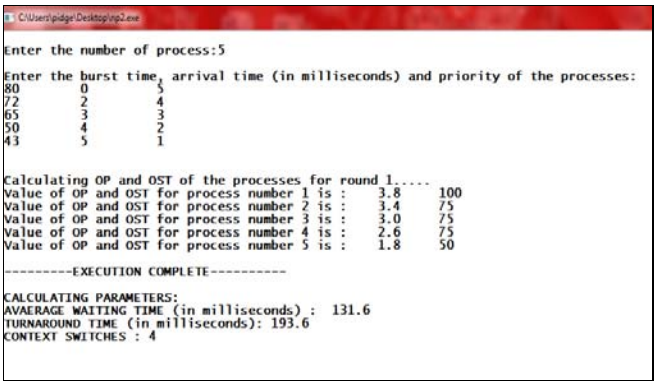

Fig.7: Snapshot of simulation of OSTRR for case 2

Comparison between both the algorithms is shown in Table X and Fig 8.

TABLE X: COMPARISON OF PARAMETERS FOR CASE II

Algorithm	Turnaround time (TAT)	Average waiting time (AWT)	Context Switches (CS)
DQRRR	209.8	147.8	7
OSTRR	193.6	131.6	4

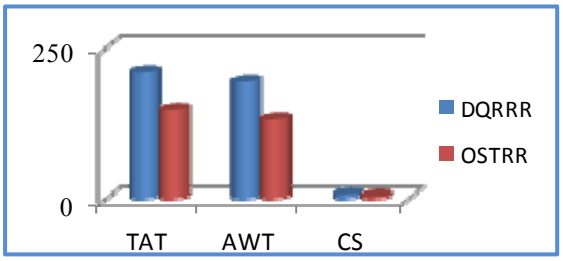

Fig.8: Comparison of DQRRR and OSTRR for case2

Case 3) Suppose there are 5 processes with following burst time, user priority and random order of arrival time as shown in Table XI:

TABLE XI: PARAMETERS FOR CASE III

Process	Burst time	User Priority 'P'	Arrival Time 'AT'
P1	26	5	0
P2	82	4	1
P3	70	3	2
P4	31	2	5
P5	40	1	7

Time quantum is 25 milliseconds. Calculating 'OP' and 'OST' as shown in Table XII we get:

TABLE XII: CALCULATION OF OP AND OST FOR CASE III

Process	BT	UP	AT	'OP'	'OST' for 1st round
P1	5	5	5	5.0	5*25=125
P2	1	4	4	3.1	3*25=75
P3	2	3	3	2.7	3*25=75
P4	4	2	2	2.6	3*25=75
P5	3	1	1	1.6	2*25=50

Scheduling with OSTRR is shown in Table XIII:

TABLE XIII: OSTRR

Process	OST
	1st Round
P1	125
P2	75
P3	75
P4	75
P5	50

Gantt chart for OSTRR is shown in Fig 9:

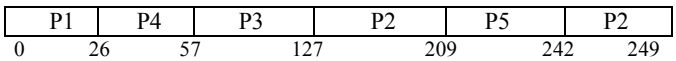

Fig 9: Gantt chart for OSTRR case 3

Scheduling with DQRRR is shown in Table XIV:

TABLE XIV: DQRRR

Algorithm	Time Quantum
DQRRR	26, 55, 21, 6

Gantt chart for DQRRR is shown in Fig 10:

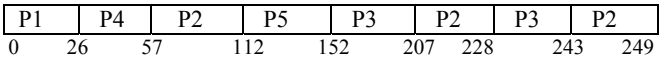

Fig 10: Gantt chart for DQRRR case 3

The snapshot of results of simulation has been shown in Fig 11:

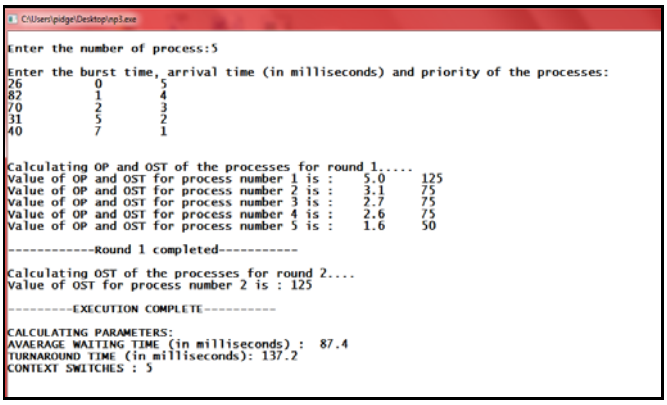

Fig 11: Snapshot of simulation of OSTRR for case 3

Comparison between both the algorithms is shown in Table XV and Fig 12:

TABLE XV: COMPARISON FOR CASE III

Algorithm	Turnaround time (TAT)	Average waiting time (AWT)	Context Switches (CS)
DQRRR	145.4	95.6	7
OSTRR	137.2	87.4	4

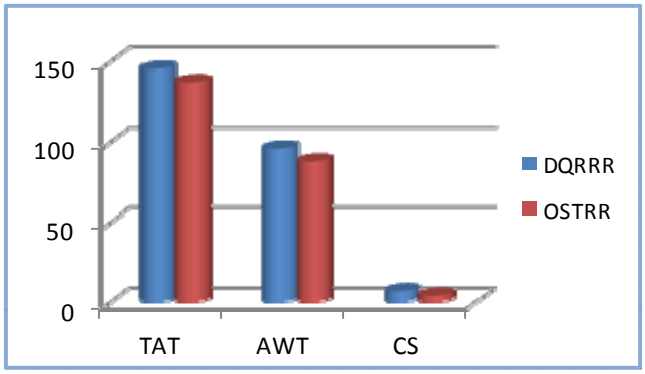

Fig 12: Comparison of DQRRR and OSTRR for case 3

VI. CONCLUSION

From the experimental results we found that our proposed algorithm performs better than DQRRR in terms of decreasing the number of context switches, turnaround time and average waiting time. This has been possible by prefering process with shorter burst time in addition to their priority and considering the effect of arrival time on priority. Future work can be based on this algorithm that includes the effect of deadlines for real time systems.

REFERENCES

[1] William Stallings, "*Operating Systems: internals and design principles*", 6th edition, Prentice Hall, ISBN-13:978-0136006329.

[2] Tanenbaum, Andrew S., "*Modern Operating Systems*", 3rd edition, Prentice Hall, ISBN: 13:9780136006633.

[3] A. Silberschatz, P.B. Galvin and G.Gagne, "*Operating Systems Concepts",* 7th edition, John Wiley and Sons, ISBN: 13:978-0471694663.

[4] C. Yaashuwanth and R.Ramesh "A New Scheduling Algorithm for Real Time System", *International Journal of Computer and Electrical Engineering (IJCEE),* Vol.2, No.6, pp 1104 -1106, December (2010).

[5] H.S. Behera et al. "Comparative and Performance Analysis of Multi-Dynamic Time Quantum Round Robin (MDTQRR) Algorithm with Arrival Time", *Indian Journal of Computer Science and Engineering (IJCSE)*, Vol.2 No.2 Apr-May 2011, ISSN: 0976 – 5166.

[6] Rami J. Matarneh "Self-Adjustment Time Quantum in Round Robin Algorithm depending on Burst Time of Running Process", *American Journal of Applied Sciences,* ISSN 1546-9239, 6 (10):1831-1837, 2009.

[7] H.S. Behera and et. al. "A new Dynamic Round Robin and SRTN Algorithm with Variable Original Time Slice for Soft Real Time Systems", *International Journal of Computer Applications 16 (1):54-60, February 2011.*

[8] Chuapeng Li, Chen Ding and Kai Shen "*Quantifying The Cost of* Context *Switch*" ACM 978-1-59593-751 Exp CS June-2007.

[9] Rakesh Mohanty and H.S. Behera "Priority based Dynamic Round Robin with Intelligent Time Slice for Soft Real Time Systems" , *International Journal of Advanced Computer Science and Applications, Vol.2, No.2, pp. 46- 50*, February 2011.

[10] H.S. Behera and Rakesh Mohanty, "A new proposed Dynamic Quantum with Re-adjusted Round Robin Scheduling Algorithm and its Performance", *International Journal of Computer Applications* (0975-8887) Vol.5- No.5, August 2010.

[11] Moonju Park, Hong Jin Yoo, Jinseok Chae "Quantum Based Fixed Priority Scheduling", *International Conference on Advanced Computer Theory and Engineering (2008).*

[12] Rami Abielmona, *Scheduling Algorithmic Research*, Department of Electrical and Computer Engineering Ottawa-Carleton Institute, 2000.