Volume 2, No. 2, February 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

Experimental Analysis of New Fair-Share Scheduling Algorithm with Weighted Time Slice for Real Time Systems

H.S. Behera*, Rakesh Mohanty, Jajnaseni Panda, Dipanwita Thakur and Subasini Sahoo

Department of Computer Science and Engineering,
Veer Surendra Sai University of Technology, Burla, Odisha, India
[1]hsbehera_india@gmail.com
[2] rakesh.iitmphd@gmail.com
[3] meennu22@gmail.com
[4]dipanwitathakur31@gmail.com
[5]subasini.vssut@gmail.com

Abstract: The performance and efficiency of multitasking operating systems mainly depend upon the used CPU scheduling algorithm. In Time Shared System, Round Robin(RR) scheduling gives optimal solution. But it is not suitable for real time system because it gives more number of context switches, larger waiting and turnaround time. In this paper a new Fair-Share scheduling with weighted time slice is proposed and analyzed which calculates time quantum in each round. Our proposed algorithm is based on a novel approach which makes the time quantum repeatedly adjustable according to the burst time of the currently running processes. This algorithm assigns a weight to each process and the process having the least burst time is assigned the largest weight. The process having largest weight is executed first, then the next largest weight and so on. Experimental analysis shows that our proposed algorithm gives better result, reduces the average waiting time, average turnaround time and number of context switches.

Keywords: Scheduling, Round Robin scheduling, Context Switching, Waiting Time, Turnaround time

INTRODUCTION

CPU scheduling is a very essential task of operating system in multitasking environment. When there is more than one process to be executed, a ready queue is maintained. The operating system follows a predefined procedure for selecting process from a number of processes waiting in the ready queue. The operating system must decide through the scheduler the order of execution and should assign the CPU to the processes. Careful attention is required to assure fairness and avoid starvation during allocation of CPU to the processes. The goal of scheduling is to minimize average waiting time, average turnaround time and number of context switches.

Scheduling Algorithms

There are many well known CPU scheduling algorithms such as First Come First Serve(FCFS), Shortest Job First (SJF), Priority etc. With FCFS, the process that arrives first in the ready queue is allocated the CPU first. In SJF, when the CPU is available, it is assigned to the process that has the smallest next CPU burst. If two processes has same length next CPU burst, FCFS scheduling is used to break the tie. In priority scheduling algorithm, a priority is associated with each process and the process having highest priority is executed first and so on. All the above algorithms are non-preemptive in nature and are not suitable for time sharing systems. The Round Robin (RR) Scheduling is designed especially for time sharing systems.. RR scheduling is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a time quantum or time slice is defined. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time quantum.

Related Work

Recently, a number of CPU scheduling algorithms have been developed for predictable allocation of processor. Self-Adjustment Time Quantum in Round Robin Algorithm [2] is based on a new approach called dynamic time quantum in which, time quantum is repeatedly adjusted according to the burst time of the running processes. Dynamic Quantum with Readjusted Round Robin Scheduling Algorithm[1] uses the job mix order for the algorithm in [2]. According to [1], from a list of N processes, the process which needs minimum CPU time is assigned the time quantum first and then highest from the list and so on till the Nth process. Again in the 2nd round, the time quantum is calculated from the remaining CPU burst time of the processes and is assigned to the processes and so on. Both [1] and [2] are better than RR scheduling and overcomes the limitations of RR scheduling .

Our Contribution

We have proposed a new algorithm in our paper which improves the Dynamic Quantum with Readjusted Round Robin Scheduling Algorithm (DQRRR) in [1]. Instead of taking job mix order, we have taken the processes in ascending order of burst time in the ready queue and the time quantum is calculated using our proposed weighted time slice method which changes with the every round of execution.

Organization of the Paper:

Section II contains the background and preliminaries. Section III presents the pseudo code, flow chart and illustration of our proposed algorithm. In section IV, experimental analysis is performed.. Conclusion and Future work is presented in section V.

BACKGROUND AND PRELIMINARIES

Terminologies:

A *process* is a program in execution. *Ready queue* holds the processes waiting to be executed or to be assigned to the processer. *Burst time* (b_t) is the time, for which a process requires the CPU for execution. The time at which the process arrives is called the *arrival time*(a_t).*Time quantum*(t_q) *or time slice* is the period of time given to each process to have CPU. *Waiting time* (w_t) is the time gap between arrival of a process and its response by the CPU. Average waiting time(a_{wt}) is the ratio of the sum of waiting time of all the processes and the number of processes. *Turnaround time* (t_{at}) is the time gap between the instant of process arrival and the instant of its completion. Average turn around time(a_{tat}) is the ratio of the sum of turn around time of all the processes and the number of processes. The number of times the CPU switches from one process to another is called the *context switches* (c_s)

Dynamic Quantum with Re-adjusted Round Robin[1]

Scheduling Algorithm

The DQRRR scheduling [1] has improved the RR scheduling by improving the turnaround time, waiting time and number of context switches. Processes are arranged in job mix order in the ready queue and time quantum is found using median method. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. Again the time quantum is calculated from the remaining burst time of the processes and so on. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue and allocates the CPU to the process for 1 time quantum. The performance of the DQRRR algorithm can be improved by using the weighted time slice.

OUR PROPOSED APPROACH

Our proposed Fair-share Scheduling with weighted Time Slice(FSWT) algorithm finds the time quantum in an intelligent way which gives better result than Dynamic Quantum with Readjusted Round Robin Scheduling Algorithm [1](DQRRR). This algorithm calculates the time quantum in a dynamic manner in each cycle of execution by using another method other than median method. The time quantum is repeatedly adjusted in every round, according to the remaining burst time of the currently running process. We have taken the weighted time slice method to get the optimal time quantum, where a weight (w) is assigned to each processes. Process having highest burst time is assigned the lowest weight 1, Process having next highest burst time is assigned weight 2 and so on. In this algorithm the shorter processes or the most weighted processes are executed first so it gives better turnaround time and waiting time.

First the processes are arranged in descending order of weight in the ready queue. The weighted time slice (WTS) is calculated as below.

$$WTS = Y_{(n+1)/2} + \{(\sum_{i=1}^{i=n} W_i) / W_{(n+1)/2}\},$$

for odd number of processes.

$$WTS = \frac{[Y_{n/2} + \{(\sum_{i=1}^{i=n} W_i) / W_{n/2}\}] + Y_{1+n/2} + \{(\sum_{i=1}^{i=n} W_i)/W_{1+n/2}\}]}{2}$$

for even number of processes.

Where, $Y_{n/2}$=Burst time of n/2th process
$Y_{1+n/2}$=Burst time of 1+n/2th process
n= number of processes.
$W_{n/2}$= Weight of the n/2th process
$W_{1+n/2}$= Weight of the 1+n/2th process

This time quantum is assigned to each process and after that again the time quantum is recalculated in the second round from the remaining burst time and so on.

Pseudo code

```
Let   n : number of processes
    b[i] : burst time of  i^th  process.
   rb[i] : remaining burst time of i^th process
Initialize: c_s=0, a_wt=0, a_tatt=0.
while (ready queue!=  NULL)
     Sort the processes in ascending order of  b_t
       //find the time quantum using  weighted time slice
     t_q= WTS
      //Sort the processes in ready queue as follows.
     for i=1 to n
       Put the processes with descending order of weight
        in ready queue
      Assign t_q to each process
       end for
     for i=1 to n
         if(b[i] < t_q)
           p[i]=b[i]=t_q and rb[i]=0
         else if (b[i] = = t_q)
           p[i]=t_q and rb[i]=0
```

else

$p[i]=t_q$ and $rb[i]=b[i]-t_q$

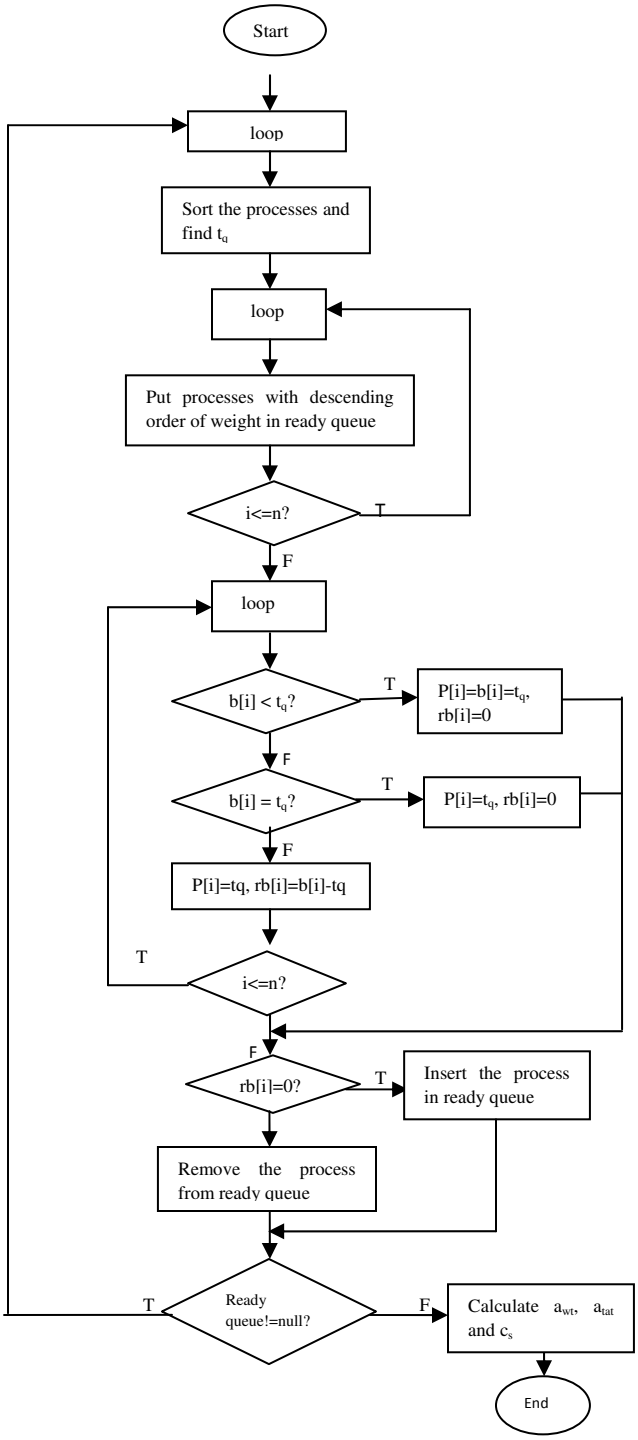
end of for

if rb[i]=0, remove the process from the ready queue

if rb[i] > 0, insert the process in the ready queue with rb[i]

end of while

a_{wt} , a_{tat} and c_s are calculated.



Flowchart of fair-share scheduling with weighted time slice

Illustration

The burst time of five processes P1, P2, P3, P4, P5 are 33, 68, 57, 85, 49 along with weights 5, 2, 3, 1, 4 respectively. Here the arrival time is assumed to be zero. First, all the processes are sorted in ascending order of burst time such as 33, 49, 57, 68, 85. Then the time quantum is calculated through weighted time slice. Here t_q=62. In the next step, processes are rearranged in ready queue in ascending order of burst time. i.e. P1 with bt=33, P5 having bt=49, P3 with bt=57, P2 with bt=68, P4 with bt=85. After assigning tq to each process the remaining burst time of all processes are P1=0, P5=0, P3=0, P2=6, P4=23. Once a process completes its execution, it is automatically deleted from ready queue. Again the next time quantum is calculated from the remaining burst time as per the algorithm. Here t_q=16. So the remaining burst times are P4=7, P2=0.. According to algorithm the next t_q will be 7 and in the last step P4 will complete its execution and will be deleted from the ready queue.

.

EXPERIMENTAL ANALYSIS

Assumptions

The environment where all the experiments are performed is a single processor environment and all the processes are independent. Time quantum is assumed to be not more than the maximum burst time of the given processes. Here we have taken 'n' processes and all these processes are independent from each other. All the attributes like burst time, number of processes, weight of each process and the time quantum of all the processes are known before submitting the processes to the processor. All the processes are CPU bound.

Experimental Frame Work

Our experiments consists of several input and output parameters. The input parameters consist of burst time, arrival time, weight, time quantum and the number of processes. The output parameters consist of average waiting time, average turnaround time and number of context switches.

Data set

We have taken two cases, i.e. Case 1 is for processes without arrival time (a_t=0, here each process arrives at the same time) and Case 2 is for processes with arrival time (here processes arrive at different time). Under these two cases we have performed three different experiments taking three different types of data sets (in increasing order, decreasing order and random order).

Experiments Performed

To evaluate the performance of our proposed FSWT algorithm, we have taken a set of processes in six different cases. This algorithm can work effectively with large number of data. In each case we have compared the experimental results of algorithm with the scheduling algorithm DQRRR [1].

Case 1: Arrival Time equal to zero

Increasing Order

We consider six processes p1, p2, p3, p4, p5 and p6 arriving at time 0 with burst time 30, 42, 50, 85, 97, 120 respectively shown in Table I. Table II shows the comparing result of DQRRR algorithm and our proposed FSWT algorithm.

Table I: Data in Increasing Order

No. of process	at	bt	Weight(wt)
P1	0	30	6
P2	0	42	5
P3	0	50	4
P4	0	85	3
P5	0	97	2
P6	0	120	1

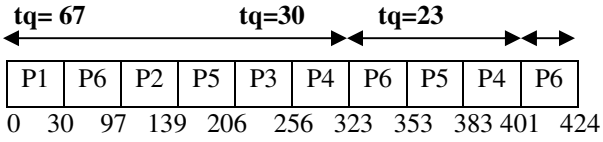


Fig I: Gantt chart for DQRRR in Table I

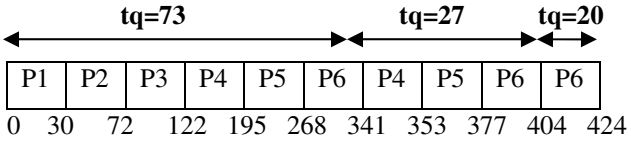


Fig II: Gantt chart for FSWT in Table I

Table II: Comparison between DQRRR and FSWT

Algorithms	DQRRR	FSWT
t_q	67,30,23	73,27,20
c_s	9	9
a_{wt}	201.5	171.1
a_{tat}	272.16	229

Decreasing Order

We consider six processes p1, p2, p3, p4, p5 and p6 arriving at time 0 with burst time 85, 73, 65,54,42 respectively shown in Table III. Table IV shows the comparing result of DQRRR and our proposed FSWT algorithm.

Table III. Data in decreasing Order

No. of process	at	bt	Weight
P1	0	85	1
P2	0	73	2
P3	0	65	3
P4	0	54	4
P5	0	42	5

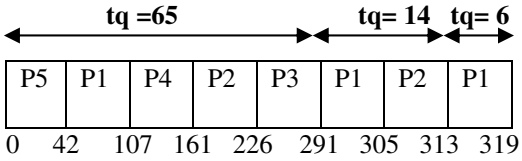


Fig.III: Gantt chart for DQRRR

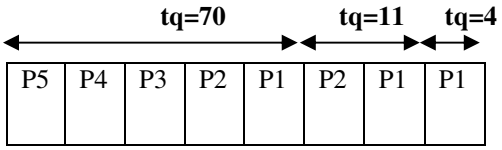


Fig. IV Gantt chart for FSWT

Table IV. Comparison between DQRRR and FSWT

Algorithms	DQRRR	FSWT
t_q	65,14,6	70,11,4
c_s	7	7
a_{wt}	161.4	120.6
a_{tat}	225.2	184.4

Random Order

We consider six processes p1, p2, p3, p4, p5 and p6 arriving at time 0 with burst time 35, 92, 68,86,49,83 respectively shown in Table V. Table VI shows the comparing result of DQRRR algorithm and our proposed algorithm FSWT.

Table V. Data in Random Order

No. of process	at	bt	Weight
P1	0	35	6
P2	0	92	1
P3	0	68	3
P4	0	86	2
P5	0	49	5
P6	0	83	4

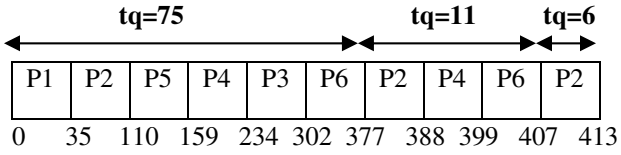


Fig V: Gantt chart for DQRRR

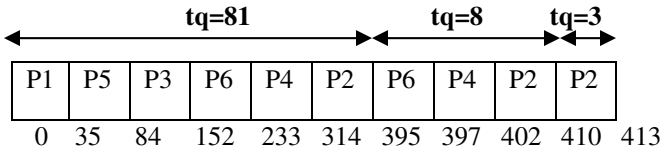


Fig.VI: Gantt chart for FSWT

Table VI. Comparison between DQRRR and FSWT

Algorithms	DQRRR	FSWT
t_q	75,11,6	81,8,3
c_s	9	9
a_{wt}	242.6	178.3
a_{tat}	285.8	247.1

Case 2: Arrival Time not equal to zero

Increasing Order

We consider six process p1,p2,p3,p4,p5 and p6 arriving at time 0,2,3,4,5,6 and burst times 32,48,57,69,73 and 80 respectively shown in the table VII. Table VIII shows the comparing result of DQRRR alogorithm and our proposed algorithm FSWT.

Table VII. Data in Increasing Order

No.of process	at	bt	Weights
P1	0	32	6
P2	2	48	5
P3	3	57	4
P4	4	69	3
P5	5	73	2
P6	6	80	1

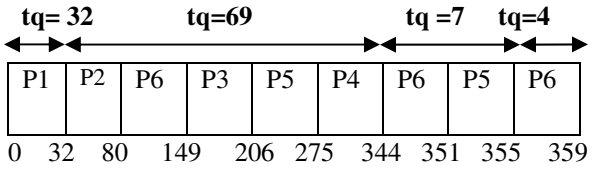


Fig.VII Gantt chart for DQRRR

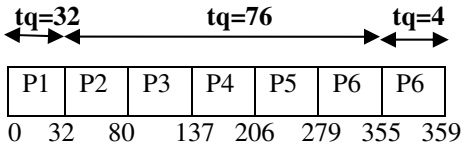


Fig VIII: Gantt chart for FSWT

Table VIII. Comparison between DQRRR and FSWT

Algorithms	DQRRR	FSWT
t_q	32,69,7,4	32,76,4
c_s	8	6
a_{wt}	165.8	119
a_{tat}	226	178.8

Decreasing Order

We consider six processes p1, p2, p3, p4, p5 and p6 arriving at time 0,2,4,6,6,8 and burst time 116,97,75,64,45,35 respectively shown in table IX. Table X shows the comparing result of DQRRR algorithm and our proposed algorithm FSWT.

Table IX. Data in Decreasing Order

No of processes	at	bt	weights
P1	0	116	1

P2	2	97	2
P3	4	75	3
P4	6	64	4
P5	6	45	5
P6	8	35	6

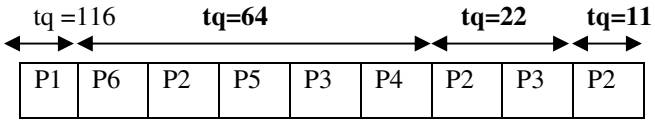


Fig IX**:** **G**antt chart for DQRRR

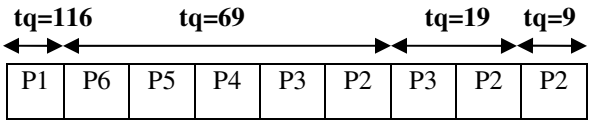


Fig X: Gantt chart for FSWT

Table X. Comparison between DQRRR and FSWT

Algorithms	**DQRRR**	**FSWT**
$\mathbf{t_q}$	116,64,22,11	116,69,19,9
cs	8	8
$\mathbf{a_{wt}}$	218.3	184.3
$\mathbf{a_{tat}}$	290.5	255.5

Random Order

We consider six processes p1, p2, p3, p4, p5 and p6 arriving at 0,2,2,3,5,6 and burst time 92,70,35,40,53,82 respectively shown in Table XI. Table XII shows the comparing result of DQRRR algorithms and our proposed algorithm FSWT.

Table XI. Data in Random Order

No. of processes	**at**	**bt**	**Weights**
P1	O	92	1
P2	2	70	3
P3	2	35	6
P4	3	40	5
P5	5	53	4
P6	6	82	2

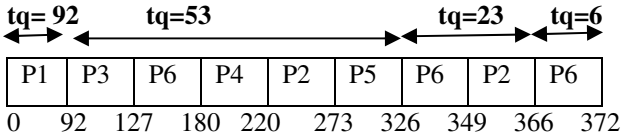


Fig XI: Gantt chart for DQRRR

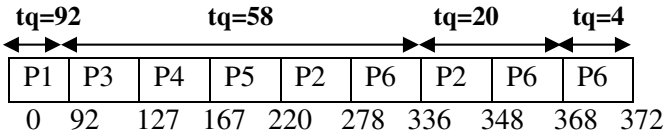


Fig XII: Gantt chart for FSWT

Table XII: Comparison between DQRRR and FSWT

Algorithm	**DQRRR**	**FSWT**
$\mathbf{t_q}$	92,53,23,6	92,58,20.4
$\mathbf{c_s}$	8	8
$\mathbf{a_{wt}}$	202.16	156
$\mathbf{a_{tat}}$	247.5	218

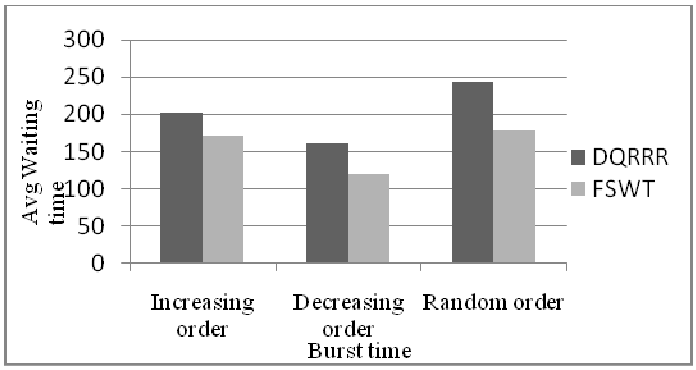


Fig XIII: Comparison of average waiting time between DQRRR and FSWT

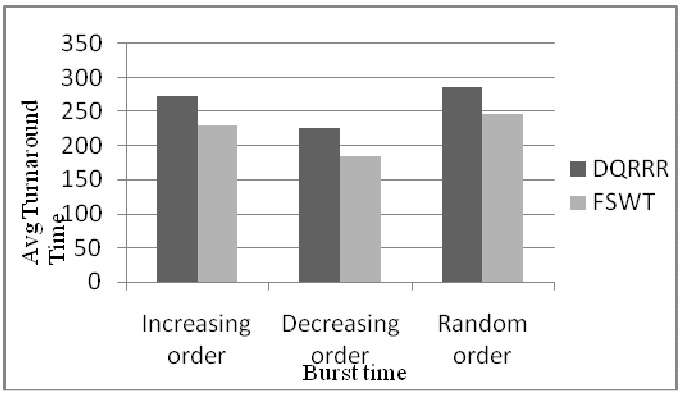


Fig. XIV: Comparison of average turnaround time between DQRRR and FSWT

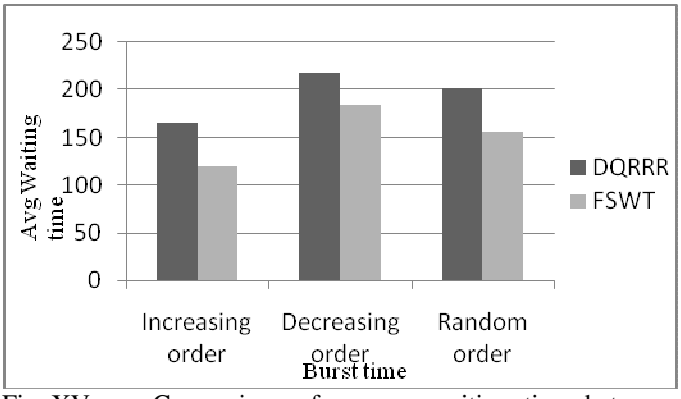


Fig XV: Comparison of average waiting time between DQRRR and FSWT

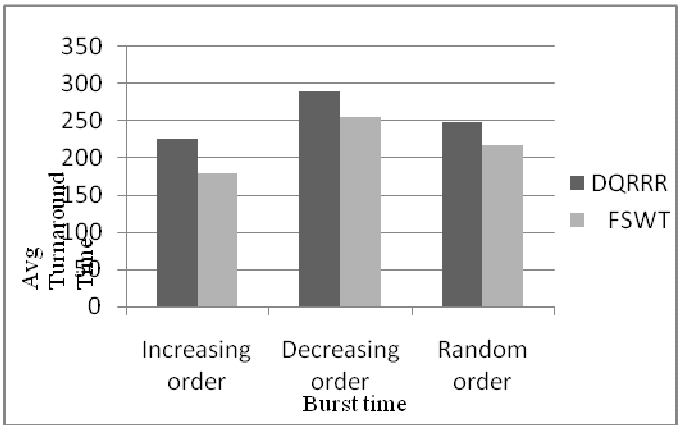


Fig XVI: Comparison of average turnaround time between DQRRR and FSWT

CONCLUSION

The above comparisons show that the proposed Fair-share Scheduling with Weighted Time slice provides much better results than the algorithm proposed in [1] and in some cases perhaps more than other approaches based on fixed time quantum in terms of average waiting time, average turnaround time and number of context switches. This algorithm can be further investigated to be useful in providing more and more task oriented results in future.

REFERENCES

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

[2] Rami J. Matarneh. "Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of Now Running Processes", American J. of Applied Sciences 6(10):1831-1837,2009.

[3] Silberschatz,A.,P.B.GalvinandG.Gange,2004."Operating systems concepts". 7th Edn.,John wiley and Sons, USA. ,ISBN:13:978-0471694663,pp:944.

[4] Tanebun, A.S., 2008,"Modern Operating Systems", 3rd Edition. Prentice Hall , ISBN: 13:9780136006633, pp:1104.

[5] C. Yaashuwanth, Dr. R. Ramesh. "A New Scheduling Algorithms for Real Time Tasks " , (IJCSIS)International Journal of Computer Science and Information Security, Vol.6, No.2, 2009.

[6] Abhishek Chandra, Micah Adler, Pawan Goyaly and Prashant Shenoy "Surplus Fair Scheduling: A Proportional-Share CPU Scheduling Algorithm for Symmetric Multiprocessors", 4th USENIXOSDI Symposium 23 Oct 2000 pp. 45-58.

[7] Tarek Helmy**,** Abdelkader Dekdouk**,** "Burst Round Robin as a Proportional-Share Scheduling Algorithm", IEEE proceding of the 4th IEEE-GCC Conference on Techno Industrial Inovations, pp 424-428, at the Gulf International Convention Center, Bahrain, 2007.

