

## Weighted Mean Priority Based Scheduling for Interactive Systems

H.S.Behera<sup>\*1</sup>, Sabyasachi Sahu<sup>2</sup> and Sourav Kumar Bhoi<sup>3</sup>

Department of Computer Science and Engineering

Veer Surendra Sai University of Technology (VSSUT), Burla, Sambalpur, Odisha, India

[hsbehera\\_india@yahoo.com](mailto:hsbehera_india@yahoo.com)<sup>1</sup>

[sabyasachi.sahu1102@gmail.com](mailto:sabyasachi.sahu1102@gmail.com)<sup>2</sup>, [souravbhoi@gmail.com](mailto:souravbhoi@gmail.com)<sup>3</sup>

**Abstract**--- Scheduling in Operating System means determining which tasks are supposed to run when there are multiple tasks to be run. Consequently, the efficiency and performance of a system mainly depends on CPU scheduling algorithm where CPU is considered as one of the primary computer resource. Traditionally, Priority Scheduling Algorithm is used for processes in which priority is the determining factor. This paper proposes a newly improved process scheduling algorithm by using dynamic time quantum along with weighted mean. Experimental analysis demonstrates that this proposed algorithm gives better response time making the algorithm useful for interactive systems.

**Keywords**--- CPU Scheduling, Priority, weighted mean, root mean square, Context Switch, Waiting time, Turn-around time, Response time.

### INTRODUCTION

An Operating System is software consisting of programs and data usually running on systems, controls the system's hardware resources and provides a common platform for various application services. In multitasking and multiprocessing environment the way the processes are assigned to run on the available CPUs is called *scheduling*. The fundamental problem in operating systems (OS) is minimizing the wait for the user when he or she simply wants the execution of a particular set of tasks. Consequently, the resource utilization and the overall performance of the system gets affected.

Hence, the Scheduler determines the assigning of processes in the ready queue to the CPU for processing. The main goal of the scheduling is to maximize the different performance metrics viz. CPU utilization, throughput and to minimize response time, waiting time and turnaround time and the number of context switches<sup>[1]</sup>. Basing on the frequency of scheduling, the scheduler in an OS are of three types viz. Long-term Scheduler, Short-term Scheduler, and Middle-term Scheduler. In the computer system, all processes consist of a number of alternating two burst cycles (the CPU burst cycle and the Input & Output (IO) burst cycle)<sup>[2]</sup>. The 2cycles viz. the CPU and the IO burst cycle execute alternatively during a normal CPU cycle.

The scheduler normally defines three states for a process: RUNNING state (process is running for the CPU), READY state (process is ready to run but isn't actually running on the CPU) and the WAITING state (the process is waiting for some IO to happen). Also the scheduler and/or dispatcher can be: Preemptive, implying that it is capable of forcibly removing processes from a CPU when it decides to give the CPU to another process, or Non-preemptive, in which case the scheduler is unable to "force" processes off

the CPU<sup>[3]</sup>. A number of research works have been carried out on scheduling algorithms hitherto for different applications. Abielmona<sup>[4]</sup> on account of his analytical scrutiny of an innumerable number of scheduling algorithms gives a thorough insight into the factors affecting a CPU scheduling algorithm's performance. Also, Matarneh<sup>[5]</sup>, has used dynamic time quantum in order to remove the limitations featuring in RR on using static time quantum. Previous works by Joseph, Mathai [6], also give insight into the value of response time to improve interactivity of a scheduling algorithm, and Ramamrithan, Krithi<sup>[7]</sup>, enumerate the significance of dynamic priority and its repercussions in a the algorithm.

### SCHEDULING ALGORITHMS

In computer science, a scheduling algorithm is the method by which tasks, processes, threads or data flow are given access to system resources [2]. The need for scheduling algorithms arises from the requirement for many Operating Systems for multiprogramming. The criteria for performance evaluation of a CPU scheduling algorithms viz. the performance metrics are as follows:

- 1). Turnaround Time:** This is the amount of time from submission to completion of process. Usually, the goal is to minimize the turnaround time.
- 2). Waiting Time:** This is the amount of time spent ready to run but not running. It is the difference in start time and ready time. Usually, the goal is to minimize the waiting time.
- 3). Response Time:** It is the amount of time it takes from when a request was submitted until the first response is produced.
- 4). Number of Context Switches:** For the better performance of the algorithm, algorithm, the number of context switches should be less.

There are four well known algorithms predominantly used in CPU scheduling briefly discussed below:

**First-Come, First-Served (FCFS):** This algorithm is preemptive in nature and allocates the CPU to the process that requests the CPU first. This algorithm is implemented using FIFO queue. This scheduler runs each task until it either terminates or leaves the task due to an IO interrupt. The processes are allocated to the CPU on the basis of their arrival at the queue. The FCFS is simple and fair but is unsatisfactory for time sharing systems since it favors long tasks.

**Shortest-Job-First (SJF):** The SJF algorithm is primarily non-preemptive. It associates the length of the next CPU burst with each process such that the process with the smallest next CPU burst is allocated to the CPU. The SJF uses the FCFS to break tie i.e. when there are two processes having the same CPU burst). The SJF algorithm can also be implemented as a preemptive algorithm. When the execution of a process that is currently running is interrupted in order to give the CPU to a new process with a shorter next CPU burst, it is called a preemptive SJF. On the other hand, the non-preemptive SJF will allow the currently running process to finish its CPU burst before a new process is allocated to the CPU.

**Priority Scheduling (PrS):** The PrS algorithm associates with each process a priority. The tasks are sorted according to their priorities and CPU is allocated to the process based on their priorities. Usually, lower numbers are used to represent higher priorities. The process with the highest priority is allocated first and those with the same priorities are scheduled by FCFS policy. The methods of determining priorities are done by some default mechanisms basing on time limits, memory requirements and other resource usages. The PrS algorithm runs high risks of starvation because of it favoring jobs on the basis of their priorities rather than their burst times.

**Round Robin (RR):** The RR algorithm is designed especially for time-sharing systems. Here, a small unit of time (called time quantum or time slice) is defined, its range generally varying from 10-100 milliseconds. The RR algorithm allows the first process in the queue to run until it expires its quantum, then run the next process in the queue for the duration of the same time quantum. The RR keeps the ready processes in a FIFO queue. In a situation where the process need more than a time quantum, the process runs for the full length of the time quantum and then it is preempted and added to the tail of the queue again but with its CPU burst now a time quantum less than its previous CPU burst. This continues until the execution of the process is completed. The RR algorithm is naturally preemptive.

## PROPOSED WMPrS ALGORITHM

In our work, the Priority Scheduling algorithm is improvised by an judicious distribution of time quantum of processes, and making the priority dynamic repeatedly over the whole Round Robin cycle. Static time quantum being a limitation of RR algorithm, we have used the concept of dynamic time quantum. For efficient priority calculation and time quantum distribution we use the concept of weighted mean i.e. to calculate  $Ta_w$  we use priority as the weight and for calculating  $Pr_{wm}$  we use time quantum as the weight. Subsequently, we calculate  $TQ_{rms}$  and  $Pr_{rms}$  using the values found above and also calculate  $Pr_{avg}$  and  $TQ_{avg}$ . In every cycle, the algorithm groups burst times and priorities into two basing on whether burst time of processes is  $>$  or  $\leq$   $BT_{avg}$ . If  $BT_i \leq BT_{avg}$  then the process  $P_i$  is allocated to the CPU with time quantum =  $TQ_{wm}$ , otherwise  $P_i$  is allocated to the CPU with time quantum =  $TQ_{wm} + TQ_{rms}$ . Similarly, the priorities are also changed. The processes are then updated with the remaining burst time and their new priorities at the completion of each round robin cycle.

```

1. Sort the n processes according to their priority.
   while (ready queue! =NULL)
2. Find the Weighted Mean ( $TQ_{wm}$ ).
    $TQ_{wm}$  = Priority Weighted Mean Time Quantum of
   all the processes.
3. Find the Weighted Mean ( $Pr_{wm}$ ).
    $Pr_{wm}$  = Burst time Weighted Mean priority of all the
   processes.
4. Calculate the  $TQ_{rms}$ .
    $TQ_{rms}$  = Root Mean Square Time Quantum
5. Calculate  $Pr_{rms}$ 
    $Pr_{rms}$  = Root Mean Square Priority
6. Find  $BT_{avg}$  and  $Pr_{avg}$ .
    $BT_{avg}$  = Average of the burst time of the processes
    $Pr_{avg}$  = Average of the priorities of the processes
7. if ( $BT_i \leq BT_{avg}$ )
   Assign  $P_i \leftarrow TQ_{wm}$ 
   else
   Assign  $P_i \leftarrow TQ_{wm} + TQ_{rms}$ 
8. while( a cycle of Round Robin is completed )
   if (  $Pr_i \leq Pr_{avg}$  )
    $Pr_i \leftarrow Pr_{rms} + Pr_i$ 
   else
    $Pr_i \leftarrow Pr_i - Pr_{rms}$ 
   end of while
9. Next , update the table for remaining processes by
   new priority and remaining burst time and then goto
   step 1.
10. End

```

Fig.1: Pseudocode of WMPrS Algorithm

The following formulae are used in the pseudo-code of the algorithm.

$$TQ_{wm} = \frac{\sum_{i=1}^n [ ( Pri * TQi ) / \sum ( Pri ) ]}{no.ofprocesses}$$

$$Pr_{wm} = \frac{\sum_{i=1}^n [ ( Pri * TQi ) / \sum ( TQi ) ]}{no.ofprocesses}$$

$$TQ_{rms} = \frac{\sqrt{\sum_{i=1}^n [ ( TQi - TQwm )^2 ]}}{no.ofprocesses}$$

$$Pr_{rms} = \frac{\sqrt{\sum_{i=1}^n [ ( Pri - Prwm )^2 ]}}{no.ofprocesses}$$

$$BT_{avg} = \sum (BT of all processes) / \text{number of processes.}$$

$$Pr_{avg} = \sum (\text{priorities of all processes}) / \text{no. of processes}$$

**Illustration**

Given the burst time sequence: 91 , 67 ,32 ,28, 97 and the priorities as 6 , 4 , 3 , 7 ,1 respectively for five processes. Initially the processes are sorted in ascending order of their priorities. Then we find the  $TQ_{wm}$  and  $Pr_{wm}$  and it is found to be 57 and 4 (rounded off to the nearest integer) respectively. Then we calculate  $TQ_{rms}$  and  $Pr_{rms}$  and we get 13 and 1(rounded off to the nearest integer) respectively. After that we calculate  $BT_{avg}$  and  $Pr_{avg}$  and we get 63 and 4(rounded off to the nearest integer) respectively. Now the main scheduling begins by assigning the time quantum dynamically to the processes. The process with burst time 97 will be executed first because of its higher priority, which is greater than  $BT_{avg}$  so we assign 70 as the time quantum ( addition of 57 and 13) . Then after it we go to next process with burst time 32 which is lower than the average so we assign 57 as the time quantum. Then we continue like this up to the completion of first cycle . In the next cycle we see that P5 and P1 left with 27 and 21 as the remaining burst time with priorities 1 and 6 respectively. Then we increase the priorities of these processes by applying certain steps, so from this example we see that priority of process P1 is less than  $Pr_{avg}$ , so new priority for P1 is equal to 2 (addition of 1 and 1) and new priority for process P2 is 5 (subtraction of 6 and 1).Hence after this we apply the same steps(goto step 1) for scheduling of these two processes.

**PERFORMANCE EVALUATION**

**Assumptions**

1. There is a pool of processes in the ready queue contending for the allocation of CPU.
2. The processes are independent, running in a single processor environment and compete for resources.
- 3.All basic attributes like burst time, priorities number of processes of all the processes are known before submitting the processes to the processor.
4. All processes are CPU bound and none I/O bound.

5. A large number of processes is assumed in the ready queue for better efficiency.
6. The Context Switching Time is equal to zero i.e. there is no Context Switch Overhead incurred in transferring from one job to another

**Data Set and Framework**

To demonstrate the applicability of and performance of the Weighted MeanPriority Scheduling (WMPPrS) algorithm, it is compared with Priority Scheduling(PrS) algorithm and three case studies are taken, depending on the variance of time quantum and priorities.

The input parameters consist of burst time, time quantum and the number of processes. The output parameters consist of average waiting time, average turnaround time, number of context switches and response time.

**Case Study 1:** We Assume five processes with priorities6, 4 , 3 , 7 , 1 respectively and with increasing burst time (P1= 91, P2 = 67, P3 = 32, P4 = 28, P5= 97) as shown in Table-1(upper). The Table-1(lower) shows the output using PrS and WMPPrS algorithm.Table-2, Table-3 and Table 4 shows the priority in each cycle for WMPPrS and Table-5 shows the comparison of response time among PrS and WMPPrS . Figure-2, Figure-3 , Figure-4 shows Gantt chart for WMPPrS and Figure-5 shows Gantt chart for PrS respectively.

Processes	Priority	Burst Time
P1	6	91
P2	4	67
P3	3	32
P4	7	28
P5	1	97

Algorithm	Avg. TAT	Avg.WT	CS
PrS	224.8	141.8	4
WMPPrS	233.2	170.2	7

Table 1: Comparison between PrS algorithm and WMPPrS Algorithm( case 1 ).

Processes	Priority	Burst Time
P5	1	97
P3	3	32
P2	4	67
P1	6	91
P4	7	28

Table 2: Priority table for WMPPrS after sorting priorities(case 1).

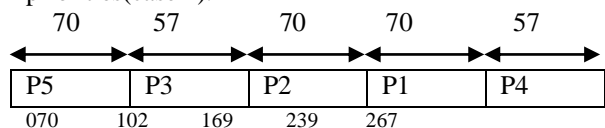


Fig. 2: Gantt chart for WMPPrS in 1<sup>st</sup>cycle(case1).

Processes	Priority	Burst Time
-----------	----------	------------

P5	2	27
P1	5	21

Table 3: Priority table for WMPPrS in 2<sup>nd</sup> cycle (case 1).

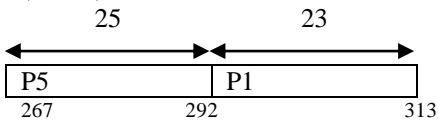


Fig. 3: Gantt chart for WMPPrS in 2<sup>nd</sup> cycle(case1).

Processes	Priority	Burst Time
P5	2	2

Table 4: Priority table for WMPPrS in 3<sup>rd</sup> cycle (case 1).

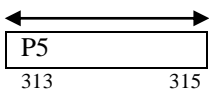


Fig. 4: Gantt chart for WMPPrS in 3<sup>rd</sup> cycle(case 1).

P5	P3	P2	P1	P4	
0	97	129	196	287	315

Fig. 5: Gantt chart for PrS(case 1).

Processes	Response time through PrS	Response Time through WMPPrS
P1	196	169
P2	129	102
P3	97	70
P4	287	239
P5	0	0

Table 5: Comparison of Response times of each processes by using Prs and WMPPrs (case1).

**Case Study 2:** We Assume five processes with priorities 5 , 4 , 8 , 7 , 1 respectively and with increasing burst time (P1= 52, P2 = 87, P3 = 72, P4 = 13, P5= 21) as shown in Table-6(upper). The Table-6(lower) shows the output using PrS and WMPPrS algorithm. Table-7, Table-8 and Table 9 shows the priority in each cycle for WMPPrS and Table-10 shows the comparison of response time among PrS and WMPPrS . Figure-6 , Figure-7 , Figure-8 shows Gantt chart for WMPPrS and Figure-9 shows Gantt chart for PrS respectively.

Processes	Priority	Burst Time
P1	5	52
P2	4	87
P3	8	72
P4	7	13
P5	1	21

Algorithm	Avg. TAT	Avg. WT	CS
PrS	141.4	92.4	4
WMPPrS	159.2	110.2	7

Table 6: Comparison between PrS algorithm and WMPPrS Algorithm ( case 2).

Processes	Priority	Burst Time
P5	1	21
P2	4	87
P1	5	52
P4	7	13
P3	8	72

Table 7: Priority table for WMPPrS after sorting priorities (case 2).

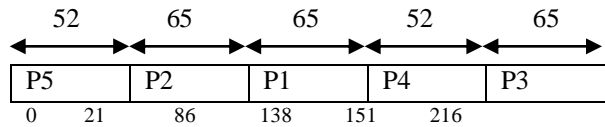


Fig. 6: Gantt chart for WMPPrS in 1st cycle(case 2).

Processes	Priority	Burst Time
P2	5	22
P3	7	7

Table 8: Priority table for WMPPrS in 2nd cycle(case 2).

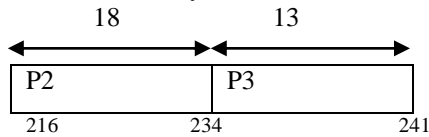


Fig. 7: Gantt chart for WMPPrS in 2nd cycle(case 2).

Processes	Priority	Burst Time
P2	5	4

Table 9: Priority table for WMPPrS in 3<sup>rd</sup> cycle(case 2).

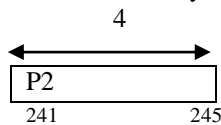


Fig. 8: Gantt chart for WMPPrS in 3rd cycle(case 2).

P5	P2	P1	P4	P3	
0	21	108	160	173	245

Fig. 9: Gantt chart for PrS(case 2).

Processes	Response Time through PrS	Response Time through WMPPrS
P1	108	86
P2	21	21
P3	173	151
P4	160	138
P5	0	0

Table 10: Comparison of Response times of each processes by using Prs and WMPPrs (case2).

**Case Study 3:** We Assume five processes with priorities 4 , 8 , 2 , 5 , 10 respectively and with increasing burst time (P1= 49, P2 = 60, P3 = 38, P4 = 54, P5= 63) as shown in

TABLE-11(upper). The Table-11(lower) shows the output using PrS and WMPPrS algorithm. Table-12 and Table-13 shows the priority in each cycle for WMPPrS and Table-14 shows the comparison of response time among PrS and WMPPrS. Figure-10 and Figure-11 shows Gantt chart for WMPPrS and Figure-12 shows Gantt chart for PrS respectively.

Processes	Priority	Burst Time
P1	4	49
P2	8	60
P3	2	38
P4	5	54
P5	10	63

Algorithm	Avg. TAT	Avg.WT	CS
PrS	146.2	93.4	4
WMPPrS	146.2	93.4	5

Table 11: Comparison between PrS algorithm and WMPPrS Algorithm( case 3).

Processes	Priority	Burst Time
P3	2	38
P1	4	49
P4	5	54
P2	8	60
P5	10	63

Table 12: Priority table for WMPPrS after sorting priorities(case 3).

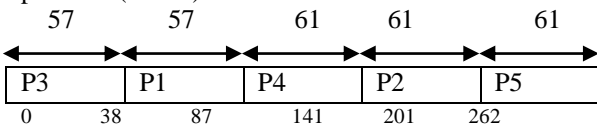


Fig. 10: Gantt chart for WMPPrS in 1st cycle(case 3).

Processes	Priority	Burst Time
P5	10	2

Table 13: Priority table for WMPPrS in 2nd cycle(case 3).

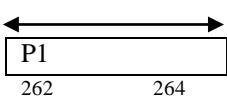


Fig. 11: Gantt chart for WMPPrS in 2nd cycle(case 3).

Processes	Priority	Burst Time
P3	2	38
P1	4	49
P4	5	54
P2	8	60
P5	10	63

Fig. 12: Gantt chart for PrS(case 3).

Processes	Response Time through PrS	Response Time through WMPPrS
P1	38	38
P2	141	141
P3	0	0
P4	87	87
P5	201	201

Table 14: Comparison of Response times of each processes by using PrS and WMPPrS (case3).

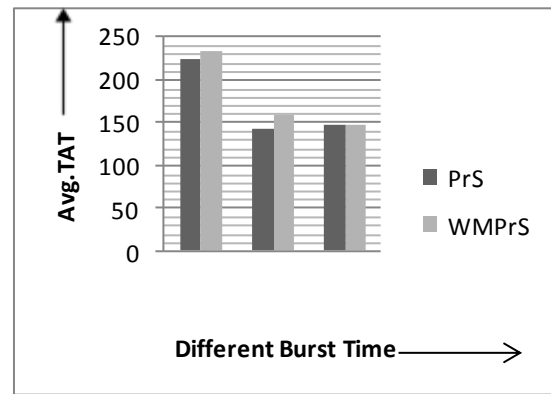


Fig 13: Comparison between PrS Algorithm and WMPPrS Algorithm by considering Average Turnaround Time for case 1, case 2 and case 3 respectively.

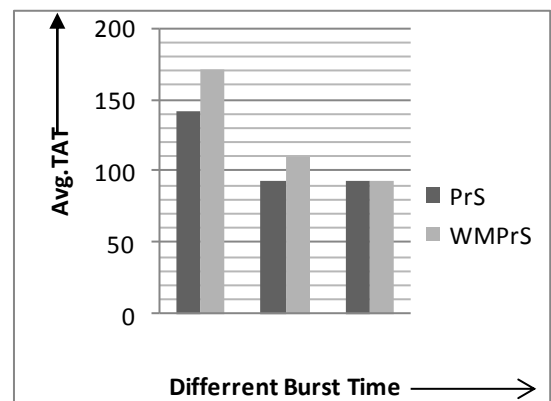


Fig 14: Comparison between PrS Algorithm and WMPPrS Algorithm by considering Average Waiting Time for case 1, case 2 and case 3 respectively.

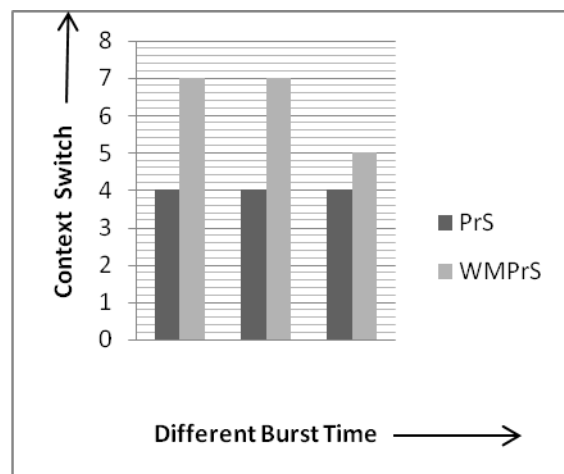


Fig 15: Comparison between PrS Algorithm and WMPPrS Algorithm by considering context switches for case 1, case 2 and case 3 respectively.

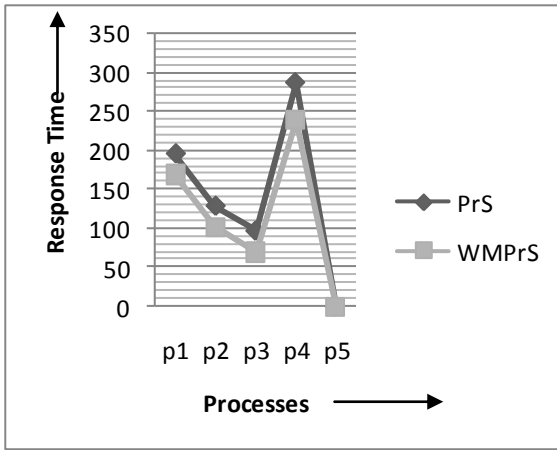


Fig 16: Comparison of Response Time of PrS and WMPPrS in case 1.

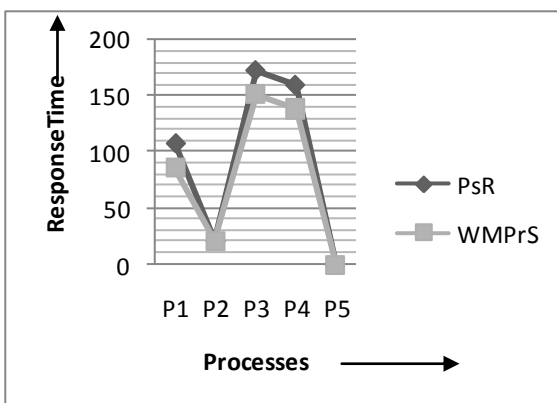


Fig 17: Comparison of Response Time of PrS and WMPPrS in case 2.

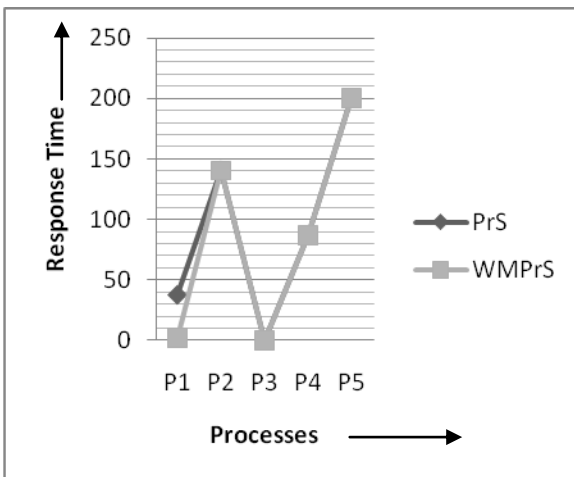


Fig 18: Comparison of Response Time of PrS and WMPPrS in case 3.

### CONCLUSION

Methodologies employed in a multitude of priority scheduling algorithms are based on an efficient distribution of priorities to reduce starvation of low priority processes and increase the fairness of the scheduling algorithms. This method, eventually results in the algorithm becoming more interactive. Likewise, this proposed algorithm proposes a method in which we take the weighted mean of the priorities and the burst times, so that we can get a closer relation between the burst times and priorities.

The approaches' significance is observed when two or more processes have a massive difference between their burst times but have modest difference in their priorities. In PrS, this might lead to starvation, but WMPPrS through a variant aging method checks this starvation to certain extent. Although the WMPPrS has higher avg. waiting time and turn-around time than the PrS, the response times for each process is noticeably lesser. The proposed algorithm performs efficiently, provided there are surplus processes in the ready queue and the processes have considerably larger difference in their burst times as compared to their difference in their priorities.

### REFERENCES

- [1] TarekHelmy, AbdelkaderDekdouk, "Burst Round Robin: As a Proportional-Share Scheduling Algorithm", IEEE Proceedings of the fourth IEEE-GCC Conference on towardsTechno-Industrial Innovations, pp. 424-428, 11- 14 November,2007
- [2] Silberschatz, A., P.B. Galvin and G.Gagne, *Operating Systems Concepts*.7th Edn., John Wiley and Sons, USAISBN:13:978- 0471694663, pp: 944,
- [3] Stallings,W.:*Operating Systems Internals and Design Principles*; 5<sup>th</sup>edn. Prentice Hall, Englewood Cliffs(2004)
- [4] Rami Abielmona, *Scheduling Algorithmic Research*, Department of Electrical and Computer Engineering Ottawa-Carleton Institute, 2000.
- [5] 2004Rami J. Matarnah , *Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the now Running Processes*, Department of Management Information Systems, American Journal of Applied Sciences 6 (10):1831-1837, 2009, ISSN 1546-9239..
- [6] Joseph, Mathai. Fixed Priority Scheduling – A Simple Model: Real-time Systems Specification, verification and Analysis, Prentice-Hall International, London, (2001).
- [7]Ramamrithan, Krithi, Dynamic Priority Scheduling – A Simple Mode: Real-time Systems Specification, verification and Analysis, Prentice-Hall International. London, (2001).

