Multi-Queue Priority Based Algorithm for CPU Process Scheduling

Usman Rafi[1], Muhammad Azam Zia[2,3](✉), Abdul Razzaq[4], Sajid Ali[5], and Muhammad Asim Saleem[6]

[1] Department of CS and IT, UOS Lyallpur Campus, Faisalabad, Pakistan
[2] State Key Laboratory of Networking and Switching Technology, BUPT, Beijing, The People's Republic of China
[3] University of Agriculture, Faisalabad, Pakistan
zia_uaf@hotmail.com
[4] Department, NFC Institute of Engineering and Technology, Multan, Pakistan
[5] Department, D.G. Khan Campus, University of Education, Lahore, Pakistan
[6] Department of IT, Government College University, Faisalabad, Pakistan

Abstract. Operating Systems acts as a base software and acts as a driver for both application programs and system programs. All the programs residing in an operating system has to become process for execution. A modern computer system supports multitasking by single user or multiple users. Different processes have different priorities. The major goal of an operating system is to reduce waiting time and enhance throughput by scheduling processes in some way. This paper discusses various scheduling terms and scheduling algorithms. We have proposed a new approach for scheduling. This proposed algorithm is based on the mixture of MQMS, Priority Scheduling mechanism and Round Robin scheduling. The proposed algorithm facilitates operating system by managing separate queue for separate priority of process and manages queue scheduling in round robin fashion with dynamic time slicing. Processes are added to appropriate queue and this decision is based on any user defined or system defined criteria. We have also discussed various case studies regarding this algorithm and compared its results with priority scheduling algorithm. These case studies are limited to two queuing system up till now. We have also proposed multiple queue management (more than 2), dynamic time slicing instead of half execution scheme and varying execution times of queues as future work of this algorithm scheme.

Keywords: Priority scheduling · Multi-queue scheduling · Dynamic time slice execution · Fair priority scheduling · Single processor multi-queue scheduling · Improved priority scheduling

1 Introduction

A computer system comprises of software and hardware. Software includes programs for system and user needs. Hardware is a set of resources including CPU, GPU, RAM and many more of them. The most important software is Operating System that acts as an interfacing layer which serves the users in using hardware effectively. User needs are fulfilled by set of Application Software that requires a base software to run i.e. operating system. Software may be in running state or it may not be running. The former is called a Process while the latter is called Program. A process requires certain resources for execution. CPU and RAM are major resources that each process requires for execution. CPU is the most costly and important resource. It must be utilized properly. CPU should never be left underutilized. Operating System must ensure proper utilization of the CPU.

A computer system may be Single Processor System or it may be Multi-Processor System. Single processor system allows only one process to acquire the CPU and gets execute on it. On the other hand multi-processor system allows more than one processes to be executed with one on each processor at a time.

A process changes its activity from time to time as it executes. Process activity is defined in terms of two-state model or five-state model. Two-state model defines that a process can have two states: running and not running. It gives an abstract idea. Two state model is shown in Fig. 1. Stallings [21] described a more elaborative version of two state model called five-state model. Five state model took into account the reason of process for not being in the running state. So, five-state model described five process activities: new, ready, running, waiting and terminated. Figure 2 shows five state model for process execution states.

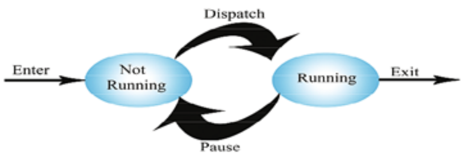

Fig. 1. Two state model for process states

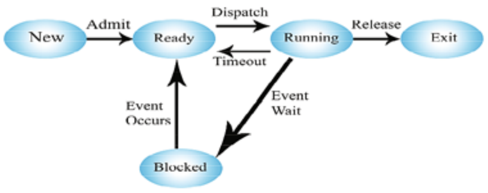

Fig. 2. Five state model for process states

Section 2 of the paper describes CPU Scheduling, its need, various queue management systems, queues involved in scheduling and scheduling criteria. Section 3 is organized to give a brief idea of various scheduling algorithms. Section 4 describes some of related work. Sections 5 and 6 describes our proposed algorithm, its working logic and pseudo code. Section 7 comprises of various case studies and results of our proposed algorithm. The results of our proposed algorithm are compared with the priority scheduling algorithm in Sect. 8.

2 CPU Scheduling

Operating system may be designed to be Single Programmed Systems or Multi Programmed Systems.

A single programmed system can only accommodate single process in memory for execution. CPU has to remain idle in the case of process waiting for some I/O operation. This results in underutilization and causes increase in waiting time and turnaround time.

Silberchatz et al. [18] narrated that a multi-programmed system facilitated the efficient utilization of processor by allowing multiple processes or jobs ready/waiting (residing in memory) for execution and switching the CPU among these processes. If an executing process had to wait for some I/O operation, the CPU no longer remained idle. It switched to the next job waiting for CPU allocation to get executed. In this manner proper utilization of CPU was achieved. To achieve this utilization, scheduling was required. Scheduling scheme comprised of criteria and algorithm to implement this criteria. CPU scheduling was of immense importance in multi-programmed systems because CPU was expansive and scarce resource.

2.1 Ready Queue Management

The ready queue is a bulk of processes ready to get executed. Ready queue may be partitioned as it is done in Multilevel Queue Scheduling and Multilevel Feedback Scheduling. There are following two ways to manage ready queue:

(1) SQMS
SQMS (Single Queue Management System) involves no partitioning of ready queue. SQMS consists of single partition ready queue in which all the incoming processes are stored. It is a straight forward queue management scheme. It is due to the fact that processes are to be picked for execution from a single queue. So operating system requires only process scheduling algorithm. Only a single queue has to be managed and scheduled. No overhead is involved in terms of scheduling ready queue partitions or multiple ready queues. In addition to this, operating system doesn't require any overhead for deciding the addition of processes to un-partitioned ready queue.

(2) MQMS
MQMS (Multi-Queue Management System) scheme maintains multiple ready queues of jobs or partitions of ready queue. Processes are added to appropriate ready queue portion depending upon certain criteria. MQMS scheme is complex scheme in terms of implementation, management and scheduling. The selection of process for execution requires two important decisions: (i) appropriate queue selection and (ii) appropriate process selection from the selected queue. Hence MQMS involves both queue scheduling algorithm and process scheduling algorithm. MQMS scheme involves overhead in terms of queue selection.

A process is first added to job queue. After this, it is moved to ready queue. Process can now move itself among ready queue, running state and waiting queue as and when required. All these state changes depend upon a criteria on the basis of which the Operating System decides that which process move to ready queue, which started execution next, which process will be needed to move to waiting queue and which process pre-empted and moved to ready queue. These decisions are taken by intelligent algorithms, executed by CPU Scheduler.

3 Scheduling Algorithms

Tanenbaum [22] described that First Come First Served (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Round Robin (RR) and Priority Scheduling (PS) algorithms were commonly used Scheduling Algorithms. These algorithms were implemented in the form of SQMS. However, priority scheduling may had MQMS version. Mishra and Khan [11] narrated that SJF only permitted process with lowest burst time to acquire CPU for execution and lowered average waiting time. RR scheduling defined a time quantum for assigning CPU fairly to each process present in the ready queue for specific amount of time. We discuss Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Round Robin (RR) and Priority Scheduling algorithms below.

3.1 Shortest Job First (SJF)

Shortest Job First is based on the Burst Time of processes. SJF is based on a simple idea derived from FCFS, that is, executing smaller processes before larger ones lowers the Average Waiting Time (AWT) of process set. If a sub set of processes have same burst time, then they are executed in FCFS fashion. The processes are arranged in a queue on the base of their Burst Times in a way that process with smallest burst time is placed at the Head/Front of queue while the process with largest burst time is placed at Rear of queue. In this way, CPU Scheduler always picks the process with smallest burst time from ready queue. It's a non-preemptive scheduling algorithm. The main advantage of SJF is to decrease the Average Waiting Time of process set. However, this algorithm does not take into account process priorities. SJF may result in starvation of larger processes.

3.2 Shortest Remaining Time First (SRTF)

Shortest Remaining Time First scheduling algorithm is simply preemptive version of SJF. In SRTF, the main idea is simply the same, that is, execute small process before large one. But in this algorithm, if a process is under execution and during course of its execution a new process enters the system. Then the burst time of newly entered process is compared with remaining burst time of currently executing process. If the comparison results in the fact that entering process has smaller burst time, then CPU is preempted and allocated to the newly entering process. However, if the new entering process has same burst time as that of already executing process, then FCFS manner is used for their scheduling. The major advantage is the exploitation of phenomenon to lower Average Waiting Time of process set. The demerit of SRTF is that, the processes in real environment have priorities and SRTF does not support priorities. Moreover, it is not necessary that smaller processes are important while larger ones are less important. This algorithm also includes high overheads for queue managements, queue rearrangements, burst time comparisons and context switches. It may also give rise to starvation for large processes.

3.3 Round Robin (RR)

Round Robin scheduling algorithm is same as that of FCFS with certain innovation. This innovation includes preemption and Time Quantum. Time Quantum is a small amount of time. Processes are arranged in a circular queue. CPU is allocated a process from the queue; CPU executes the allocated process for a time period equal to that of time quantum. When the time quantum expires, the CPU is preempted and next process from the queue is selected. Next process is also executed for a time length equal to time quantum. This algorithm continues to schedule processes in this fashion until queue is empty. The major advantage of this scheme is that each of the process gets fair share and equal opportunity of execution. The major drawback is that, if time quantum is very small, then too many context switches are involved. On the other hand, if time quantum is large, then RR may perform same as FCFS. Moreover, RR also does not consider priorities of processes. Siregar [23] narrated that starvation could be avoided if CPU was allocated fairly to processes. Time quantum was an important factor in RR scheduling scheme for lowering waiting time of process set. Time quantum could take any value. But best time quantum should be discovered for minimizing average waiting time.

3.4 Priority Scheduling (PS)

Priority Scheduling s involves assigning a Priority to each process. A Priority is either indicated by an integer number (0, 1, 2, 3, etc.) or a string (low, medium, high etc.). In the case of numeric priority, two choices exist: higher number may indicate a higher priority (our proposed algorithm consideration) or lower priority (other algorithm consideration). This scheduling algorithm is implemented using a Priority Queue. Processes are arranged in decreasing order of priority (Front to Rear) with process having highest priority placed at front while process having second highest priority placed at a place next to front and so on. The CPU Scheduler picks process with highest priority and allocates CPU for execution. If a number of processes have same priority then they are scheduled in FCFS fashion as of in SJF and RR case. PS may be preemptive or non-preemptive. If non-preemptive version is being used, then no matter how much higher priority process enters the queue, if CPU is executing a process then it will not be preempted unless the process is completed. On the other hand, the preemptive version preempted the CPU if such a situation occurs. The main attraction in PS is realistic approach about priorities. However, PS can lead to starvation situation preventing lower priority processes from execution. Its implementation involves high overheads of queue rearrangements and priority comparisons in case of preemptive version. PS can be implemented in the form of either SQMS (Traditional Priority Scheduling) or MQMS (Multi-Level Queue Scheduling). Rajput and Gupta [14] narrated that in priority scheduling, processes were arranged in queue according to their fixed priority values. A high priority incoming process interrupted low priority process. Waiting time of low priority processes were thus increased and hence starvation occurred.

Adekunle et al. [2] analysed the performance of FCFS, SJF, RR, SRTF and Lottery scheduling algorithms. SJF showed higher while RR showed medium CPU utilization. SJF also showed lower average waiting time and good response time as compared to RR. However, RR algorithm was found to be fair and starvation free.

Almakdi et al. [4] simulated FCFS, SJF, RR and SRT algorithms for analysing scheduling algorithm performance. The results showed that the lowest average waiting time and the lowest average turnaround time were showed by SJF. However, high average core utilization and high average throughput were resulted by RR.

4 Related Work

Rao et al. [15] elaborated that Multilevel Feedback Queue Scheduling involved use of multiple queues. Process could be migrated (promoted/demoted) from one queue to another. Each queue had its own scheduling algorithm. Similarly, criteria were also defined for promotion and demotion of processes. PMLFQ resulted in starvation because of execution of high priority process queues before low priority process queues.

Goel and Garg [5] explained fairness was an important scheduling criteria. Fairness ensured that each process got fair share of CPU for execution. Further, higher priority processes exhibited smaller waiting time and starvation could take place for lower priority process.

Shrivastav et al. [16] proposed Fair Priority Round Robin with Dynamic Time Quantum (FPRRDQ). FPRRDQ calculated time quantum on the basis of priority and burst time for each individual process. Experimental results when compared with Priority Based Round Robin (PBRR) and Shortest Execution First Dynamic Round Robin (SEFDRR) scheduling algorithms, showed that FPRRDQ had improved performance considerably.

Patel and Patel [13] proposed a Shortest Job Round Robin (SJRR) scheduling algorithm. SJRR suggested to arrange processes in SJF fashion. The shortest process's burst time was designated as time quantum. Case study showed reduction in average waiting time of process set.

Abdulrahim et al. [1] proposed New Improved Round Robin (NIRR) algorithm for process scheduling. The algorithm involved two queues ARRIVE and REQUEST. In this algorithm, the time quantum was calculated by finding the ceiling of average burst time of process set. First process in the REQUEST queue was allocated to CPU for one time quantum. In any case, the executing process had burst time less than or equal to half time quantum then CPU reallocated for remaining CPU burst time. Otherwise, process was removed from REQUEST and added to ARRIVE. NIRR algorithm improved scheduling performance and also lowered number of context switches as the case of RR algorithm.

Mishra and Rashid [12] introduced Improved Round Robin with Varying Time Quantum (IRRVQ) scheduling algorithm. IRRVQ proposed arrangement of processes (in ready queue) in SJF fashion and setting time quantum equal to burst time of first process. After a complete execution of first process, next processes were selected and assigned one time quantum. If finished, processes were removed from ready queue otherwise, processes were preempted and added to tail of ready queue.

Sirohi et al. [20] described an improvised round robin scheduling algorithm for CPU scheduling. This algorithm calculated the time quantum by finding the average of the burst times of processes. The processes were allocated to ready queue in SJF order. CPU was allocated to first process for one time quantum. The reallocation of CPU was based on remaining burst time of process under execution.

Akhtar et al. [3] proposed a preemptive hybrid scheduling algorithm based on SJF. Initially, it showed similar steps as that of SJF. The algorithm reduced number of context switches considerably but only showed minor change in average waiting time. The algorithm could exhibit starvation.

Joshi and Tyagi [7] explained Smart Optimized Round Robin (SORR) scheduling algorithm for enhancing performance. At first step, the SORR algorithm arranged processes in increasing order of burst times. In the second step, the mean burst time was calculated. Third step involved calculation of Smart Time Quantum (STQ) after which the CPU was allocated to first process in the ready queue for a time quantum equal to calculated STQ. If the allocated process was remained with burst time less or equal to 1 time quantum, CPU was reallocated for the burst time. Otherwise process was preempted, next process was selected from ready queue and CPU was allocated to the newly selected process for STQ.

Lulla et al. [10] devised a new algorithm for CPU scheduling. The algorithm introduced mathematical model for calculating initial time quantum and dynamic time quantum. Dynamic time quantum assisted in lowering number of context switches. The algorithm proposed execution of top two high priority processes by assigning calculated initial time quantum followed by the calculation and assignment of dynamic time quantum for execution of top two high priority processes.

Goel and Garg [6] proposed Optimized Multilevel Dynamic Round Robin Scheduling (OMDRRS) algorithm. This algorithm suggested the allocation of dynamic time slices. Factor analysis based on arrival time, burst time and priority was calculated for sorting processes in ready queue. Burst values of first and last process were used for time slice calculation. Simulated results, statistically analysed using ANOVA and t-test, showed considerable performance improvement.

Kathuria et al. [8] devised Revamped Mean Round Robin (RMRR) scheduling algorithm. RMRR involved two queues ReadyQueue (RQ) and Pre-ReadyQueue (PRQ). Time quantum was calculated by taking mean of the burst time of processes. The proposed algorithm lowered average waiting time, average turnaround time and number of context switches as compared to RR scheduling. RMRR showed the best response time when compared with FCFS, SJF and RR scheduling algorithms.

Singh and Singh [19] proposed a hybrid scheduling algorithm based on Multi Level Feedback Queue (MLFQ) and Improved Round Robin (IRR) scheduling principles. The proposed algorithm suggested the sub-division of memory in three queues as in MLFQ. Fonseca and Fleming's Genetic Algorithm (FFGA) was implemented for optimization. Results showed optimized performance in comparison to various algorithms (Fig. 3).

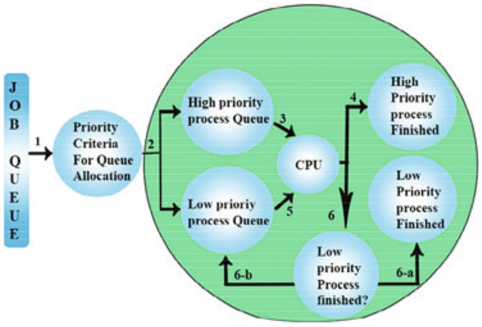
JOB QUEUE
1
Priority Criteria For Queue Allocation
2
High priority process Queue
Low prioriy process Queue
3
5
CPU
4
6
High Priority process Finished
Low Priority process Finished
Low priority Process finished?
6-a
6-b

Fig. 3. Schedulers and scheduling queues

5 Proposed Algorithm

In this paper, we proposed a new scheduling algorithm as an extension of priority scheduling. Here, we proposed it only for single processor system (neither multiple processors nor multi-core processors). It is based on hybrid concepts of multiple queue management, priority scheduling and round robin scheme. In our algorithm, multiple queue management scheme facilitates subdivision of ready queue in multiple portions. Shukla et al. [17] described the use of multi-level queuing approach. Multi-level queuing involved partitioning of ready queue into various sub queues. In such scheduling, processes were assigned permanently to specific queue. Memory size, process type and priority were some of parameters used for such assignments.

In our proposed algorithm, each job is allocated to specific queue according to criteria predefined criteria. The criteria consists of two factors: (1) Priority of individual process and (2) Burst Time of process. Firstly, a priority criteria is defined in a set of processes. The processes are allocated after comparing process priorities to priority criteria. Our proposed algorithm assumes that the largest number represented highest priority and smallest number represented lowest priority. Shortest job first scheme is applied while arranging processes when they are transferred from the job pool to the ready queue partitions. Jobs inside a specific queue are scheduled as defined by priority scheduling scheme. Khan and Kakhani [9] suggested SJF based PS was proposed for performance improvement. Processes were placed in ready queue in higher to lower order of priority. SJF was employed for arranging process in case of more than one processes having similar priority. Results of SJF based PS were compared to FCFS based PS and results showed considerable decrease in average waiting and average turnaround times. Moreover, the queues are scheduled in a round robin fashion from the highest priority queue to the lowest priority queue. Round robin scheme is based on dynamic time slicing. Whenever a queue is given time slice it is according to the process burst time and some variations of it. Our algorithm has following main steps:

Step 1. Create Multiple Queues (one for high priority processes and other for low priority processes) based on Priority Scheme. These queues are designated as High Priority and Low Priority Queues.

Step 2. Create Processes in Job Pool.

Step 3. Define Priority Criteria.

Step 4. Allocate processes to queues based on priority criteria and burst time. Processes are arranged in decreasing order of priority from front to rear. Additionally following are considered:

- **Step 4.1.1** If process priority is less than or equal to priority criteria then add it to the Low Priority Queue. Otherwise add it to High Priority Queue.
- **Step 4.1.2** If multiple processes have same priority then arranges them in a way that the process with shortest burst time comes first in the queue.

Step 5. Select first process from High Priority Queue and execute it fully (for entire burst time by assigning time slice equal to the burst time) and remove it from queue head.

Step 6. Select first process from Low Priority Queue and execute for a time equal to half that of burst time (assign time slice equal to half the burst time). Important thing to consider is that, in this case, each low priority process completed in two halves.
Step 7. Repeat Steps 5–6 until both queues are empty.

One of the interesting things to note is that, CPU is alternated between high priority and low priority processes in order to increase the fairness. Our approach lowers the starvation faced by low priority processes in the presence of high priority processes. Another point may arise that why low priority processes are executed in two halves? The answer of this question is that, it is done so that if a low priority process is executed fully, then the purpose of two queues is unclear. Similarly, if high priority process is executed for half time, then the use of priorities becomes meaningless.

6 Pseudo Code

Proposed Algorithm (P[1, $\cdots$, n], Priority Criteria) (Fig. 4)

```
begin
Create Queue Portions QH[1....nh], QL[1....nl]
For each P[i] in P[1.......n]
        If (P[i].Priority <= PriorityCriteria) Then
                Add P[i] to QL in such a way that processes are arranged in
                decreasing order of priority and increasing order of burst time
        Else
                Add P[i] to QH in such a way that processes are arranged in
                decreasing order of priority and increasing order of burst time
             End If
Repeat following steps while QH and QL are not empty
        Select a process from QH head, allocate CPU, execute completely and
        remove it from QH
        Select a process from QL head, allocate CPU, execute for the time half to
        its original burst time
                If (P[i].RBT) = 0 Then
                    Removes it from QL
                Else
                    place it back to QL head
            End If
                Goto step 4
            End while
Completion of process set
End Algorithm
*RBT = Remaining Burst Time
```

Fig. 4. The proposed algorithm

7 Case Studies

7.1 Case Study 1

Gantt Chart of above process data when executed by Priority Scheduling is as Fig. 5 (Table 1).

Table 1. Process set with five processes

Process	Burst time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

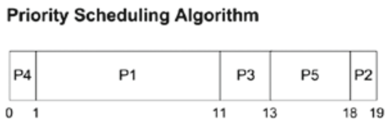

Fig. 5. Gantt chart for priority scheduling algorithm

Average Waiting Time = 8.6 ms.
Average Turnaround Time = 12.4 ms.
Now we define certain parameters related to proposed algorithm.
(1) Priority Criteria: Process with Priority > 2 is placed in High Priority Queue while process with Priority <= 2 is placed in Low Priority Queue.
(2) Ready Queue Partitions: As Fig. 6.
The proposed algorithm on given data results in Fig. 7.

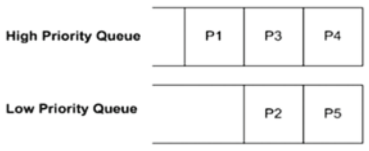

Fig. 6. Scheduling queues

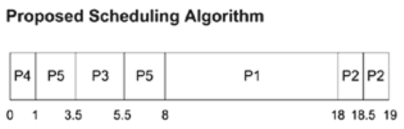

Fig. 7. Gantt chart for proposed algorithm

Average Waiting Time = 6.5 ms
Average Turnaround Time = 10.3 ms

7.2 Case Study 2

Gantt Chart of above process data when executed by Priority Scheduling is as Fig. 8 (Table 2).

Average Waiting Time = 7.0 ms.
Average Turnaround Time = 10.8 ms.
Now we define certain parameters related to proposed algorithm.

Table 2. Process set with five processes

Process	Burst time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

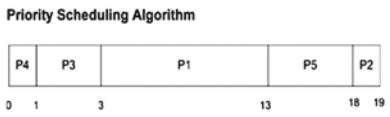

Fig. 8. Gantt chart for priority scheduling algorithm

(1) Priority Criteria: Process with Priority $>$ 2 is placed in High Priority Queue while process with Priority $\leq$ 2 is placed in Low Priority Queue.
(2) Ready Queue Partitions: As Fig. 9.
The proposed algorithm on given data results in Fig. 10.

Average Waiting Time = 6.5 ms.
Average Turnaround Time = 10.3 ms.

7.3 Case Study 3

Gantt Chart of above process data when executed by Priority Scheduling is as Fig. 11 (Table 3).

Average Waiting Time = 33.8 ms.
Average Turnaround Time = 49.8 ms.
Now we define certain parameters related to proposed algorithm.
(1) Priority Criteria: Process with Priority $>$ 2 is placed in High Priority Queue while process with Priority $\leq$ 2 is placed in Low Priority Queue.
(2) Ready Queue Partitions: As Fig. 12
The proposed algorithm on given data results in Fig. 13.

Average Waiting Time = 33.5 ms
Average Turnaround Time = 49.5 ms

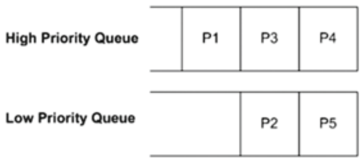

Fig. 9. Scheduling queues

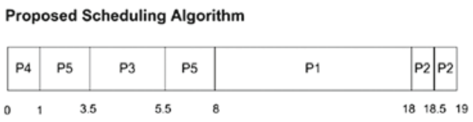

Fig. 10. Gantt chart for proposed algorithm

Table 3. Process set with five processes

Process	Burst time	Priority
P1	12	3
P2	19	3
P3	21	5
P4	13	2
P5	15	3

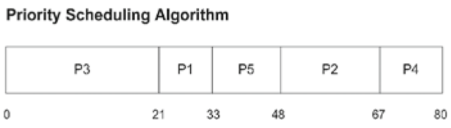

Fig. 11. Gantt chart for priority scheduling algorithm

7.4 Case Study 4–Special Case

This case deals with a situation involving same priority processes (high priority). Similarly, inverse case of this can also be assumed. In such cases all the processes added to a single queue either high or low priority (Table 4).

Gantt Chart of above process data when executed by Priority Scheduling is as Fig. 14.

Average Waiting Time = 3.67 ms
Average Turnaround Time = 9.33 ms
Now we define certain parameters related to proposed algorithm.
(1) Priority Criteria: Process with Priority $>$ 2 is placed in High Priority Queue while process with Priority $\leq$ 2 is placed in Low Priority Queue.
(2) Ready Queue Partitions: As Fig. 15
The proposed algorithm on given data results in Fig. 16.

Average Waiting Time = 3.67 ms
Average Turnaround Time = 9.33 ms

In the case with all the processes getting place in a single queue, the proposed algorithm gives an average waiting time and average turnaround time exactly equal to priority scheduling algorithm.

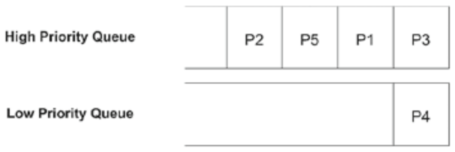

Fig. 12. Scheduling queues

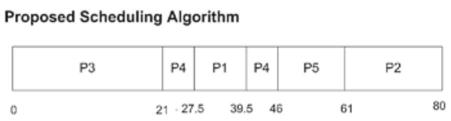

Fig. 13. Gantt chart for proposed algorithm

Table 4. Process set with three processes

Process	Burst Time	Priority
P1	10	5
P2	3	6
P3	4	7

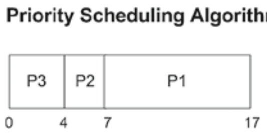

Fig. 14. Gantt chart for proposed scheduling algorithm

8 Results and Discussion

8.1 Comparison of Average Waiting Time

In this section we compare average waiting time of our proposed algorithm and traditional priority scheduling algorithm. This comparison is based on case studies discussed in the previous sections.

The average waiting time is minimized by our proposed algorithm as compared to the baseline scheduling algorithm as shown in Fig. 17. However, the last case showed same average waiting time. This was because, the last case comprised of special dataset involving all the processes with same priority (high). In this case, only one priority queue was formed rather two queues. So, in this case it exhibited same behaviour as that of traditional priority scheduling algorithm.

8.2 Comparison of Average Turnaround Time

In this section we presented comparison of average turnaround time of our proposed algorithm with traditional priority scheduling algorithm. This comparison based on case studies discussed in the previous sections.

Figure 18 clearly showed that in first three case studies the average turnaround time of our proposed scheduling algorithm was less than traditional priority scheduling algorithm. However, the last case showed same average turnaround time. This was because, the last case comprised of special dataset having

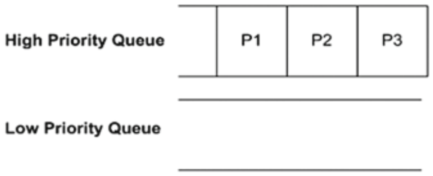

Fig. 15. Scheduling queues

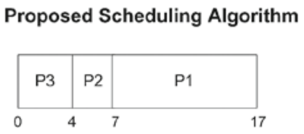

Fig. 16. Gantt chart for proposed algorithm

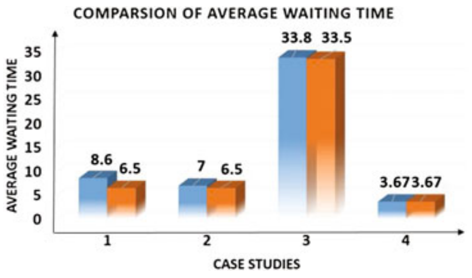

Fig. 17. Comparison of average waiting time

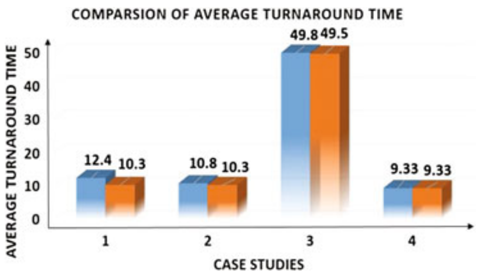

Fig. 18. Comparison of average turnaround time

all the processes having high priority. In this case only one priority queue was formed rather two queues.

9 Conclusion and Future Work

It is concluded that our proposed algorithm used minimum waiting and turnaround time as compared to baseline Priority Scheduling algorithms. Our proposed algorithm also supported multiple queue supporting operating systems using different priorities. Each type of priority can be managed with a separate queue. The proposed algorithm provided fairness to both high priority and low priority jobs and thus saved low priority jobs from starvation. The importance is given to high priority process within both high priority and low priority list by our proposed algorithm. Our algorithm also enhanced throughput by lowering average waiting and average turnaround time. As compared to Round Robin scheduling with small time slice, proposed algorithm used a scheme requiring only two context switches per process. In this way, our proposed algorithm limited the number of context switches required for a job. This algorithm is a good mixture of priority scheduling, shortest job first and round robin scheduling.

As a future work, we will try out different execution lengths or time slices for high priority processes and low priority processes to find optimized solution providing optimized results of scheduling criteria. We will also extend this algorithm in a way that for each priority type a single queue may be used.

References

1. Abdulrahim A, Abdullahi SE, Sahalu JB (2014) A new improved round robin (nirr) cpu scheduling algorithm. Int J Comput Appl 90(4):27–33
2. Adekunle O (2014) A comparative study of scheduling algorithms for multiprogramming in real-time systems. Int J Innov Sci Res 12:180–185
3. Akhtar M, Hamid B et al (2015) An optimized shortest job first scheduling algorithm for cpu scheduling. J Appl Environ Biol Sci 5:42–46
4. Almakdi S (2015) Simulation and Performance Evaluation of CPU Scheduling Algorithms. LAP LAMBERT Academic Publishing, Saarbrücken
5. Goel N, Garg RB (2013) A comparative study of cpu scheduling algorithms. Int J Graph Image Process 2(4):245–251
6. Goel N, Garg RB (2016) Performance analysis of cpu scheduling algorithms with novel omdrrs algorithm. Int J Adv Comput Sci Appl 7(1):216–221
7. Joshi R, Tyagi SB (2015) Smart optimized round robin (sorr) cpu scheduling algorithm. Int J Adv Res Comput Sci Softw Eng 5:568–574
8. Kathuria S, Singh PP et al (2016) A revamped mean round robin (rmrr) cpu scheduling algorithm. Int J Innov Res Comput Commun Eng 4:6684–6691
9. Khan R, Kakhani G (2015) Analysis of priority scheduling algorithm on the basis of fcfs and sjf for similar priority jobs. Int J Comput Sci Mob Comput 4:324–331
10. Lulla D, Tayade J, Mankar V (2015) Priority based round robin cpu scheduling using dynamic time quantum. Int J Emerg Trends Technol 2:358–363
11. Mishra MK (2012) An improved round robin cpu scheduling algorithm. J Glob Res Comput Sci 3(6):64–69
12. Mishra MK, Rashid F (2014) An improved round robin cpu scheduling algorithm with varying time quantum. Int J Comput Sci Eng Appl 4(4):1–8
13. Patel R, Patel M (2013) Sjrr cpu scheduling algorithm. Int J Eng Comput Sci 2:3396–3399
14. Rajput G (2012) A priority based round robin cpu scheduling algorithm for real time systems. Int J Innov Eng Technol 1:1–10
15. Rao MVP, Shet KC, Roopa K (2009) A simplified study of scheduler for real time and embedded system domain. Comput Sci Telecommun 12(5):1–6
16. Shrivastav MK, Pandey S et al (2012) Fair priority round robin with dynamic time quantum: Fprrdq. Int J Mod Eng Res 2:876–881
17. Shukla D, Ojha S, Jain S (2010) Data model approach and markov chain based analysis of multi-level queue scheduling. J Appl Comput Sci Math 8(4):50–56
18. Silberschatz A, Gagne G, Galvin PB (1983) Operating System Concepts, 8th edn. Addison-Wesley Pub. Co, Boston Binder Ready Version
19. Singh N, Singh Y (2016) A practical approach on mlq-fuzzy logic in cpu scheduling. Int J Res Educ Sci Methods 4:50–60
20. Sirohi A, Pratap A, Aggarwal M (2014) Improvised round robin (cpu) scheduling algorithm. Int J Comput Appl 99(18):40–43
21. Stallings W (2011) Operating Systems–Internals and Design Principles, 7th edn. DBLP
22. Tanenbaum AS, Tanenbaum AS (2001) Modern Operating Systems, 2nd edn. Prentice-Hall, Upper Saddle River
23. Ulfahsiregar M (2012) A new approach to cpu scheduling algorithm: Genetic round robin. Int J Comput Appl 47(19):18–25