www.ijsrnsc.org

Available online at www.ijsrnsc.org

IJSRNSC

Volume-11, Issue-6, December 2023
Review Paper

Int. J. Sc. Res. in Network Security and Communication

E-ISSN:2321-3256

A Review on New Multilevel Scheduling Algorithm and SJF and Priority Scheduling Algorithms

Irshad Khan[1*], Mohd Owais[2], Saima Aleem[3], Tasleem Jamal[4]

[1,2,3,4]Dept. of CSE Khwaja Moinuddin Chishti Language University, Lucknow, UP, India

**Corresponding Author: irshadkhan05.234@gmail.com, 6306655471*

Received: 22/Sept/2023, Accepted: 09/Nov/2023, Published: 31/Dec/2023

***Abstract*—** The two CPU scheduling algorithms that received the majority of our attention in this paper after reviewing a variety of CPU scheduling algorithms were the shortest job first and priority scheduling algorithms, as well as an improved priority scheduling algorithm that performs better than current scheduling algorithms. Scheduling is the process of assigning tasks to the CPU to optimize use. Because the CPU is the most important resource in a computer system, numerous scheduling approaches aim to maximize its use. The purpose of this study is to explore the CPU scheduler's construction of high-quality scheduling algorithms that meet the scheduling objectives and to study the performance of a multilevel scheduling algorithm that combines two scheduling algorithms.

***Keywords* —** CPU scheduling, Scheduler, Exponential Averaging, burst time, waiting time, turnaround time

I. INTRODUCTION

An essential part of an operating system is scheduling. Scheduling means assigning certain processes, and CPU time so that they get executed. Only one process out of numerous processes may be active at once in a single-processor system, all other processes must wait until the CPU is free and can be rescheduled. To optimize CPU use, multiprogramming aims to keep some processes active at all times. Most computer resources are planned out before usage. Of course, one of the most important computer resources is the CPU. Because of this, operating-system architecture revolves around its scheduling. CPU scheduling decides which run-able processes are executed from numerous run-able processes. CPU scheduling is crucial because it has a significant impact on the consumption of resources and the performance of the overall system. The development and evaluation of CPU scheduling methods, involving the modification and testing of operating system kernel code and assessing performance with real applications under a consistent workload, can be a complex and time-intensive task. Given that the processor is a pivotal resource, effective CPU scheduling is essential for realizing the design objectives of an operating system. The goal is to optimize CPU utilization by accommodating as many operating processes as possible simultaneously. Scheduling is required to ensure maximum CPU utilization. Almost every computer resource is scheduled before use. The main task of a scheduler is to choose which tasks to execute as there can be a number of tasks ready to be executed. Successful CPU scheduling depends on process execution and I/O wait. Process execution starts with a CPU burst followed by an I/O burst and this thing happens rapidly. The last CPU burst ends with a system request to expire execution. An SJF algorithm allocates CPU to the task with the smallest possible burst time. A priority scheduling algorithm assigns certain priority to processes and executes the process with the highest priority. But contemporary algorithms can be improved as we will see in this review.

II. CPU SCHEDULING

A. Scheduling goals

1) *Accountability*: When creating a task scheduling, a system developer must take into account a number of variables, including the type of systems being utilized and user requirements.
2) *Increase throughput*: A scheduler's throughput is increased by serving the most processes possible in a given amount of time.
3) *Prevent an endless blocking phase or starvation:* Preventing a process from being trapped in a waiting state for an indefinite amount of time before or during process servicing.
4) *Reduce overhead:* By effectively using system resources, system overhead may be minimized (system overhead reflects resource wastage). Thus, total system performance significantly increases.
5) *Priorities must be enforced:* In a system based on process priorities, the scheduler must give higher priority processes preference.
6) *Achieve a balance between responsiveness and utilization:* The scheduler must keep the system resources occupied. By prioritizing activities that have a short burst period and can be fulfilled quickly, the scheduler may boost throughput while preventing famine by using the idea of aging. The scheduler will also give preference to processes *whose completion causes other processes to execute to achieve objectives.*

B. Scheduling parameters

CPU scheduling algorithms rely on below mentioned criteria:

1) *Processor Utilization:* The ratio of the processor's busy time to the overall amount of time needed for a process to be completed. The goal is to maintain the processor's maximum amount of activity.
   Processor Utilization = (Processor buy time) / (Processor busy time + Processor idle time)
2) *Throughput:* The number of tasks accomplished within a specified time frame,
   Throughput = (Number of completed processes) / (Time Unit)
3) *Turnaround Time (tat):* Total time spent by processes waiting to get into the ready queue, time for execution, and time spent on input/output operations
   tat = t (process completed) – t (process submitted)
4) *Waiting Time (wt):* Time spent by a process in the ready queue The duration occupied in the ready queue is influenced solely by processor scheduling algorithms, as they impact the waiting time specifically. Therefore, focusing on waiting time, rather than turnaround time, is usually adequate.
5) *Response Time (rt):* The time required to initiate a response to a request.
   rt = t (first response) – t (submission of request)

In interactive systems, this criterion holds significance. Typically, the goal is to optimize processor utilization and throughput while minimizing turnaround time (tat), waiting time (wt), and response time (rt). However, in certain situations, alternative combinations may be necessary based on specific process requirements.

C. Scheduling strategies

There are two different kinds of scheduling.

1) *Non-preemptive:* A non-preemptive scheduling algorithm selects a process to execute and allows it to continue running until it either becomes blocked or voluntarily releases the CPU. In other words, it lets the chosen task or job complete its execution without interruption. Examples of non-preemptive scheduling algorithms include First-Come-First Serve (FCFS) and Shortest Job Next (SJF).
2) *Preemptive:* Within that type of scheduling, a process's execution may be interrupted before the end of its burst time and may be replaced by another process to whom the priority is greater than the first process to arrive in the CPU. Examples of this kind of scheduling include Round Robin and Priority Driven.

D. First come first serve (FCFS)

FCFS is not pre-emptive. It employs the First in-First Out (FIFO) approach to give processes attention in the order for which they submit requests to the CPU. The process or job that wants the CPU first gets it, and any other processes or jobs in the queue must wait for the CPU to be available. FCFS, also known as First-In-First-Out (FIFO), is the most basic scheduling method [1]. Its first-in, first-out implementation is straightforward to understand and put into practice, but because the average wait time is so long, it performs poorly [3].

E. Shortest job first (SJF) scheduling

The criterion of the Shortest Job Next (SJF) algorithm is to allocate the CPU to the process with the smallest CPU burst. In cases where two processes share the same CPU burst time, First-Come-First-Serve (FCFS) is employed to resolve the tie. SJF can operate in either pre-emptive or non-preemptive mode, depending on the arrival and burst times of the processes. When compared to FCFS, SJF typically reduces the average waiting time for processes.

In contrast to FCFS, SJF prioritizes shorter processes over longer ones, which can be seen as an overhead. It selects the task with the shortest burst time, freeing up the CPU for subsequent processes immediately after the current process is completed. This prevents smaller processes from getting delayed in the ready queue for an extended duration behind larger processes. The objective is to enhance overall efficiency by swiftly accommodating shorter processes. It's straightforward to implement in Batch systems, because the amount of CPU time needed is foreseen in advance, and it's the best method for reducing wait times. However, because the amount of CPU time needed is unknown in interactive systems, its implementation is impractical. In this case, the processor should be aware of the expected processing time.[3].

F. SJF algorithm

1) Sort all the processes according to their arrival time.
2) Select the process with a minimum arrival time as well as minimum burst time.
3) After completion of the process, select from the ready queue the process that has the minimum burst time.
4) Repeat the above processes until all processes are terminated.

G. Prediction of SJF burst time

SJF burst time prediction is a critical aspect of CPU scheduling algorithms, involving the anticipation of a process's next CPU burst duration. Utilizing historical data, statistical methods, and adaptive models contributes to refining these predictions, and addressing the challenges posed by the dynamic and unpredictable nature of computer systems.

The concept of predicting burst times in Shortest Job First (SJF) is an integral part of CPU scheduling algorithms, aiming to anticipate the duration of a process's upcoming CPU burst. The accuracy of burst time predictions holds significant importance in optimizing the operating system's scheduling decisions, allowing for more informed selections of processes for execution.

A prevalent approach to SJF burst time prediction involves utilizing historical data. The system observes the past CPU burst times of a specific process and leverages this information to estimate its future behaviour. This method

operates on the assumption that a process's future performance aligns with its historical patterns, providing the basis for predictions.

However, accurately predicting burst times faces challenges due to the dynamic nature of computer systems. Factors such as changes in input data, fluctuations in system load, and the unpredictable nature of certain applications can impact a process's behaviour. Consequently, SJF burst time prediction methods typically incorporate a level of uncertainty.

Diverse algorithms and techniques are applied to SJF burst time prediction, encompassing statistical methods, exponential averaging, and adaptive models. Statistical methods may entail computing the mean or weighted average of past burst times. Exponential averaging gives more weight to recent burst times, assuming that recent behaviour holds greater relevance for predicting future behaviour.

We can classify these algorithm as:

1) Static techniques

a) Process size: This categorization pertains to static techniques reliant on process size. In the context of CPU scheduling algorithms, process size denotes the quantity of CPU time or resources required for a specific process to finalize its execution. Algorithms falling under the umbrella of process size static techniques make scheduling decisions by taking into account the predetermined or known size of processes.

b) Process type: Within the realm of static techniques, the process type classification encompasses algorithms that base scheduling decisions on the specific type or category of processes. Distinct processes may possess unique characteristics or priorities, and this categorization integrates these attributes into the scheduling process.

2) Dynamic techniques

a) Simple Averaging

Simple Averaging is a basic statistical technique employed in different domains, including its application in predicting Shortest Job First (SJF) burst times within CPU scheduling algorithms. In this particular scenario, simple averaging entails the computation of the mean or average based on the past CPU burst times of a specific process, serving as an estimation for its future behavior.

The procedure is straightforward: the system observes the historical CPU burst times of a designated process, aggregates them, and subsequently divides the sum by the total number of observations. This calculation results in the average burst time, which is then regarded as a forecast for the forthcoming CPU burst duration of the process.

b) Exponential Averaging or Aging

The Shortest Job First (SJF) algorithm, being optimal, proves challenging for implementation in CPU scheduling due to the inherent uncertainty in predicting the length of the next CPU burst. Typically, the next CPU burst is estimated using an exponential average derived from the measured lengths of prior CPU bursts, and the formula for this exponential average is as follows. Let t_n be the length of the nth CPU burst and let τ_{n+1} be the predicted value of the next CPU burst. Then, for α, $0 \le \alpha \le 1$, define.

$$\tau_{n+1}=\alpha t_n+(1-\alpha)\,\tau_n \quad (1)$$

τ_{n+1}=expected time for n+1

t_n=actual burst time of the process

Let's expand the formula (1) to understand the behavior of the exponential average:

$$\tau_{n+1}=\alpha t_n(1-\alpha)st_{n-1}+\ldots(1-\alpha)^j t_{n-j}+\ldots(1-\alpha)^{n+1}\tau_{0\,(2)}$$

Let's say we have 4 processes with known burst times as mentioned in "Table I", and we must calculate the burst time for the fifth process: -

Table 1. Process and their burst time

Process Id	Time (ms)
	CPU burst duration
P1	3
P2	4
P3	2
P4	1
P5	?

$\alpha = 0.2$, $\boldsymbol{\tau_1} = 3$, $\boldsymbol{\tau_5}$ =?

Using formula (1):

$$\boldsymbol{\tau_{n+1}=\alpha t_n+(1-\alpha)\,\tau_n}$$

$\boldsymbol{\tau_1 = 3}$

$\boldsymbol{\tau_2}$ = 0.2 x 3 + 0.8 x $\boldsymbol{\tau_1}$

= 0.2 x 3 + 0.8 x 3

$\boldsymbol{\tau_2 = 3}$

$\boldsymbol{\tau_3}$ = 0.2 x 4 + 0.8 x $\boldsymbol{\tau_2}$

= 0.2 x 4 + 0.8 x 3

$\boldsymbol{\tau_3 = 3.2}$

$\boldsymbol{\tau_4}$ = 0.2 x 2 + 0.8 x $\boldsymbol{\tau_3}$

= 0.2 x 2 + 0.8 x 3.2

$\boldsymbol{\tau_4 = 2.96}$

$\boldsymbol{\tau_5}$ = 0.2 x 1 + 0.8 x $\boldsymbol{\tau_4}$

= 0.2 x 1 + 0.8 x 2.96

$\boldsymbol{\tau_5 = 2.568}$

Hence the expected Burst Time for the fifth process is **2.568 milliseconds.**

H. Round robin (RR):

Round Robin (RR) Algorithm has been one of the earliest, simplest, fairest, and most ubiquitously used scheduling algorithms. It was made for time-sharing processes [4] In this algorithm, each process gets a piece of time called "quantum time" to use while the CPU is running. Every process has the same priority here, and each is assigned a time limit after which it is preempted. Usually, it lasts between 10 and 100 milliseconds [5]. If the current process's limit has been reached, it will temporarily stop

running and go to the finish of the ready queue. When employing RRS, the operating system selects the first task from the ready list, sets a timer to stop after a one-time quantum, and then assigns the CPU to that process. The process releases the processor freely, either by quitting or by sending an I/O request, if its processor burst time is less than the time quantum. The OS will then carry out the following procedure in the ready list. The timer will activate when a one-time quantum has passed if the process, on the other hand, has a processor burst time longer than the time quantum. This interrupts (preempts) the running process and moves its Program Control Block towards the end of the ready list. [4][2]

I. Priority scheduling

One of the most popular scheduling methods in batch systems is priority scheduling, a non-preemptive technique. Priority is given to each procedure. The highest priority process must be carried out first and as quickly as possible. Processes of the same priority are carried out in the order in which they are received. Based on memory needs, time needs, or any additional resource needs, priority may be determined. Priority is generally stated using a fixed range of numbers like 0 to 7 or 0 to 4,095 etc. Generally, 0 is considered the highest priority but there is no fixed rule. The Operating System assigns a fixed priority rank to each process. Processes with Lower priority get interrupted by incoming higher-priority processes.

III. LITERATURE SURVEY

This section includes a quick assessment of some of the most important studies conducted to improve the efficiency of task scheduling in operating system environments. The author presented a novel method to enhance the RR algorithm's time quantum computation by taking the processes' arrival times into account [6]. The following algorithms were outperformed by this new strategy. Lowering the average WT, average TAT, and quantity of context switch CS, DABRR, S.R.R, DQRRR, and SARR may be achieved.

The author introduced a novel method in [7] that combines the SJF algorithm with the RR algorithm to reduce the waiting time for activities. The core concept of the novel approach involves dynamically calculating the time quantum for every job in each round. Moreover, the task-ready queue has been categorized into short and long jobs, utilizing the median as the determining factor. The results of the experiment using Cloud Sim demonstrate that the unique approach successfully addressed the issue of famine in lengthy jobs. Additionally, compared to SJF, RR, and Second Interval Priority Based RR [2], it has shown excellent performance in lowering waiting time and reaction time (TSPBRR).

The Author [8] introduced the PIMRR method in [14]. Each process's priority is first determined via a modulo operation. The procedures are then organized in the ready state based on their priority. The quantum time, which varied dynamically between each round, is then set as the average of their individual burst times. The findings show excellent progress in lowering waiting times, turnaround times, and the frequency of context switches.

[11] Jinkyu Lee and colleagues introduced control pre-emption (CP-EDF) in their work [44]. This method manages instances of pre-empting processes, contrasting with existing approaches that organize pre-empted processes. The technique is specifically designed for a single-processor platform, enabling the execution of at most one process at a time.

In their work [12], O. Hani and M. Dorgham introduced an innovative approach to enhance the round-robin (RR) algorithm. Their method involves utilizing the geometric median to determine the time quantum. The application of this method extends to two distinct scenarios. In the first case, when the arrival times of processes are known, they are allocated to the CPU in a First-Come-First-Serve (FCFS) manner.

If the initial process requires more than one quantum, a comparison with other processes in the ready queue is made, and the Shortest Job First (SJF) algorithm is employed to select the subsequent process. In the second case, where the arrival times are unknown, the proposed method exclusively utilizes the SJF algorithm for process selection. The outcomes of this approach demonstrate a notable enhancement in terms of average waiting time (WT) and average turnaround time (TAT).

In [13], S. Saeidi proposed a novel mathematical model designed for computing the optimal quantum of the round-robin (RR) algorithm. The aim is to minimize the average waiting time of processes. The experimental results, conducted using Lingo 8.0 software, demonstrated superior performance compared to alternative approaches.

The authors presented an effective scheduling approach in their paper [14], aimed at improving the performance of both the Round Robin (RR) algorithm and the Robin Cloudlet Scheduling Algorithm (IRRCSA). In this strategy, tasks are allocated to suitable Virtual Machines (VMs), each equipped with its own local queue (LQ).

The scheduling of tasks in these LQs is accomplished using the Round Robin (RR) algorithm with an optimal quantum, determined based on the median value of the average execution time of tasks and the highest execution time among them. The evaluation results demonstrate notable enhancements in resource utilization, average waiting time, and average turnaround time.

The studies highlight diverse innovations in task scheduling, including optimizing the RR algorithm with arrival times, combining SJF with RR, and introducing PIMRR and CP-EDF methods. These approaches consistently reduce waiting times, and context switches,

and enhance prioritization, leading to notable improvements in system efficiency.

IV. PRIORITY SCHEDULING ALGORITHM BASED ON EXISTING FCFS

The present algorithm follows a prioritized execution sequence, giving precedence to the process with the highest priority. Waiting time and turnaround time are calculated after the execution of each task. However, in situations where priorities are equal, the algorithm resorts to First-Come, First-Served (FCFS) ordering. The method for priority Scheduling, in which, if two or more processes have the same priority, the process that comes first is run first.

J. Problem statement

As comparable priority tasks occur, the priority scheduling algorithm uses FCFS, this results in an increase in average wait and turnaround times. Regardless of how long it takes the CPU, the process that reaches first gets performed first. Therefore, under this scenario, if large burst period processes are executed early, other processes will spend a lot of time in the waiting queue. The average waiting time and turnaround time are increased as a consequence of the way the procedures are organized in the ready queue.

Objective: The purpose of this study is to:

a) To analyze the SJF priority-based and FCFS priority-based scheduling algorithms.

b) To lower the average CPU waiting and turnaround time.

K. Analysis and Experimental Results

One of the primary responsibilities of the process manager is scheduling, which entails determining the allocation of the CPU to processes in the ready queue. Different scheduling algorithms are available for decision-making. Priority Scheduling Algorithm is one of them; it is based on the priority given to each process. Priority scheduling prioritizes the execution of each process, starting with the one with the greatest priority [9]. When multiple processes share the same priority, allocate the CPU to the process with the briefest task duration (lowest burst time). In cases where processes have equal priority and burst time, prioritize them based on the First Come First Serve (FCFS) principle.

Consider the set of 5 processes whose arrival time and burst time are given below in table -

Table 2. Arrival Time and Burst Time

Process Id	**Time**	
	Arrival Time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turnaround time (In this process, when one process is completed, only then does the other start, and no one can interrupt).

Gantt Chart-

P4	P1	P3	P5	P2

0 6 10 13 14 16

Now, we know that: -

• Turn Around time = Exit time – Arrival time

• Waiting time = Turn Around time – Burst time

We have calculated turn around time and waiting time for processes P1-P5 in "Table II."

Table 3. Turn Around Time and Waiting Time of processes in Table 2

Process Id	Time	
	Turn Around Time	Waiting time
P1	6	0
P2	10	6
P3	13	10
P4	14	13
P5	16	14

Now,

• Average Turn Around time = (6 + 10 + 13 + 14 + 16) / 5 = 40 / 5 = 11.8 unit

• Average waiting time = (0 + 6 + 10 + 13 + 14) / 5 = 24 / 5 = 8.6 unit

For Pre-emptive: -

Consider the set of 5 processes whose arrival time and burst time are given in "Table 2.".

Considering that the CPU scheduling policy is SJF preemptive, we will calculate the average waiting time and average turnaround time.

Gantt chart

P4	P2	P1	P2	P3	P5	P4

0 1 3 4 6 8 11 16

In this process, one or more processes can interrupt each other. The calculated Exit time, Turn Time, and Waiting Time are mentioned in "table 4."

Table 4. Exit time, Turn Around Time, and Waiting Time of the process are mentioned in Table 2.

Process Id	Time		
	Exit Time	Turn Around Time	Waiting Time
P1	4	1	0
P2	6	5	1
P3	8	4	2
P4	16	16	10
P5	11	9	6

Average Turn Around time = (1 + 5 + 4 + 16 + 9) / 5 = 35 / 5 = 7 unit

Average waiting time = (0 + 1 + 2 + 10 + 6) / 5 = 19 / 5 = 3.8 units.

Now that we are aware of how to plan tasks using SJF while taking into account their burst duration. However, the issue of how to forecast the amount of burst time needed for a certain operation emerges. since we are unable to predict precisely how long a work will take. So, the following are various burst time determination methods employed in contemporary operating systems.

For example, let P0, P1, P2, P3, and P4 Priority be assigned for each process as mentioned in "table 5".

Table 5. Burst time and Priority.

Process Id	Time(ms) and Priority	
	Burst Time	Priority
P0	12	3
P1	2	1
P2	3	3
P3	2	4
P4	6	2

Gantt Chart:

P1	P4	P0	P2	P3	
0	2	8	20	23	25

The described algorithm represents a modern priority scheduling approach but comes with certain drawbacks:

- There's a potential for indefinite blocking or starvation in low-priority processes.
- In cases where two processes share the same priority, the tie is resolved on a First Come First Serve (FCFS) basis, leading to a gradual increase in waiting time for processes with equivalent priorities.

Consider the following processes with equal priority:

Table 6. Burst Time and Priority of Processes

Process Id	Time	
	Burst time	Priority
P1	12	3
P2	8	3
P3	3	3
P4	3	3
P5	3	3

Gantt Chart Using Normal Algorithm:

P1	P2	P3	P4	P5	
0	12	20	23	26	29

The waiting time and the average waiting time is (0+12+20+23+26)/5 = 16.2ms.

Table 7. Waiting Time of processes in Table 6 Using a normal algorithm

Process Id	Time (ms)
	Waiting time
P1	0
P2	12
P3	20
P4	23
P5	26

Table 8. Turnaround Time of processes in Table 6 Using a normal algorithm

Process Id	Time
	Turnaround time (ms)
P1	12
P2	20
P3	23
P4	26
P5	29

The average turnaround time is (12+20+23+26+29)/5 = 29ms

We can improve these results by slightly changing the algorithm. Using the algorithm mentioned in the paper [10].

V. PROPOSED PRIORITY SCHEDULING ALGORITHM [10]

The algorithm [10] integrates the operational principles of both Shortest Job First (SJF) and the standard priority scheduling algorithm. The procedure is outlined as follows:

1. Assign priorities to processes in the ready queue.
2. Allocate the CPU to the process with the highest priority.

If two or more processes have the same priority, then:

{
Assign the CPU to the process with the shortest job, characterized by the minimum burst time.
}

If two or more processes have the same priority and equal burst time, then:

{
Assign the CPU to the processes following a First Come First Serve (FCFS) basis.
}

Applying comparison and implementation methods used in "A New Multilevel CPU Scheduling Algorithm" on the processes taken above:

The Gantt Chart for the Processes mentioned in "Table 6.":

P3	P4	P5	P2	P1	
0	3	6	9	17	29

The average waiting time is (17+9+0+3+6)/5= **7ms**

Table 9. The waiting time of processes in Table 6 using the proposed algorithm [10]

Process Id	Time(ms)
	Waiting time
P1	17
P2	9
P3	0
P4	3
P5	6

Comparison Of Scheduling Policies(Proposed and Existing)

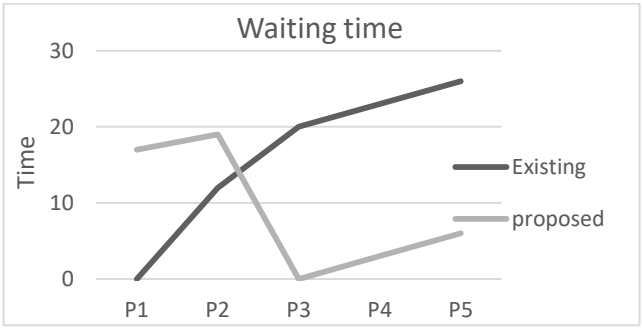

Figure 1. Waiting Time Comparison of existing algorithm vs proposed algorithm.

Now the revised turnaround time and the average turnaround time is (29+17+3+6+9)/5 = 12.8ms.

Table 10. Turnaround time of process in Table 6 According to our proposed algorithm.

Process Id	Time (ms)
	Turnaround time
P1	29
P2	17
P3	3
P4	6
P5	9

We can see that this algorithm reduces ((16.2-7)/14) * 100 = 56.8% waiting time for the above example in comparison with the existing algorithm. We can also see a gradual decline in the turnaround time of the processes as more and more processes get executed. The decline can also be seen in the waiting time of processes as the CPU executes more and more processes. The intended and existing algorithms are contrasted in the "Fig 2" below.

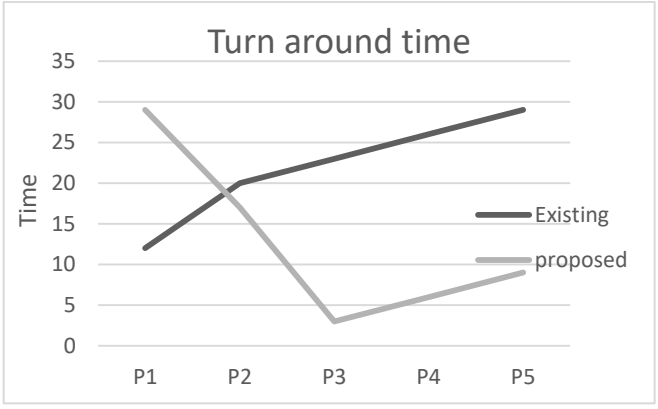

Figure 2. Turn Around time Comparison of existing algorithm vs proposed algorithm.

VI. CONCLUSION

Utilizing a CPU scheduling algorithm in a multiprogram operating system is the single determinant of a computer's efficiency. In this paper, we examine the priority scheduling method to lower both the average waiting time and the average processing time. In the existing priority scheduling system, tasks of the same priority exhibit relatively elevated average waiting and turnaround times. Nevertheless, the investigated study suggests the potential for implementing a priority scheduling system that significantly reduces waiting and turnaround times for processes. The SJF scheduling's shortest procedures result in longer wait times for longer processes. Even though the lengthy procedure results in minimal average waiting times and average turnaround times, it may never be used. Low-priority tasks that may be addressed by aging may suffer from starvation as a result of priority scheduling algorithms. For procedures with equal priority, the waiting time steadily grew. The waiting time for processes as well as their execution time are both dramatically reduced when using changed algorithms.

REFERENCES

[1] D. Goyal, "Comparative Analysis of Various Scheduling Algorithms," International Journal of Advanced Research in Computer Engineering & Technology, Volume 2, Issue 4, pp. **1488-1491, 2013.**

[2] Sonia Zouaoui, Lotfi Boussaid, Abdellatif Mtibaa, "Priority-based round robin (PBRR) CPU scheduling algorithm", International Journal of Electrical and Computer Engineering (IJECE) Vol. 9, No. 1, February 2019, pp. 190~202

[3] C. Shekar, Karthik, "Analysis of Priority Scheduling Algorithm based on FCFS & SJF for Similar Priority Jobs [1]," pp.1-4, **2017.**

[4] P. Surendra Varma, "A FINEST TIME QUANTUM FOR IMPROVING SHORTEST REMAINING BURST ROUND ROBIN (SRBRR) ALGORITHM," Journal of Global Research in Computer Science, Volume 4, No.3, pp.**10-15, 2013.**

[5] O. Hani, M. Dorgham, "Improved Round Robin Algorithm: Proposed Method to Apply SJF using Geometric Mean" International Journal of Computer Science and Mobile Computing, Vol. 5, Iss. 11, pp.**112-119, 2016.**

[6] A. Alsheikhy, R. Ammar, and R. Elfouly, "An improved dynamic Round Robin scheduling algorithm based on a variant quantum time," In the Proceedings of the 2015 11th International Computer Engineering Conference, Egypt, pp.98-104, **2015.**

[7] S. Elmougy, S. Sarhan, and M. Joundy, "A novel hybrid of Shortest job first and round Robin with dynamic variable quantum time task scheduling technique," Journal of Cloud Computing, vol. 6, no. 1, Dec. 2017

[8] G. Siva Nageswara Rao, R. Jayaraman, and S. V. N. Srinivasu, "Efficient PIMRR algorithm based on scheduling measures for improving real Time systems," International Journal of Engineering and Technology(UAE), vol. 7, no. 2.32 Special Issue.32, pp.**275–278, 2018.**

[9] M. R. Khan and G. Kakhani, "Analysis of Priority Scheduling Algorithm on the Basis of FCFS & SJF for Similar Priority Jobs," International Journal of Computer Science and Mobile Computing, Vol. 4, Issue. 9, pp.**324 – 331, 2015.**

[10] M. M. Rashid, M. N. Akhtar, "A new multilevel CPU Scheduling algorithm," Journal of Applied Sciences, vol. 6, no. 9, pp. **2036-2039, 2006.**

[11] J. Lee and K. G. Shin, "Preempt a job or not in EDF scheduling of uniprocessor systems," IEEE Trans. Comput, vol. 63, no. 5, pp.**1197–1206, 2014.**

[12] O. Hani and M. Dorgham "Improved Round Robin Algorithm : Proposed Method to Apply SJF using Geometric Mean" in the proceedings of 2019 2nd International Conference on Computer Applications & Information Security, vol.5, no.11, pp.112–120, 2016.

[13] S. Saeidi "Determining the Optimum Time Quantum Value in Round Robin Process Scheduling Method", I.J. Information Technology and Computer Science, vol. 4, pp.**67–73, 2012.**

[14] S. Banerjee, A. Chowdhury, S. Mukherjee, and U. Biswas "An Approach Towards Development of a New Cloudlet Allocation Policy with Dynamic Time Quantum" Aut. Control Comp. Sci., vol. 52, no. 3, pp.**208–219, 2018.**

Authors Profile

Mr. Irshad khan is a 4th-year B.tech computer science student at Khwaja Moinuddin Chishti Language University in Lucknow. He is Consistently maintaining a high GPA in computer science courses. a strong understanding of data structures and algorithms, coupled with practical experience in web development.
Additionally, His familiarity with databases, including both SQL and MongoDB, reflects a comprehensive approach to handling and managing data. He possesses a strong aptitude for mathematics and a keen proficiency in problem-solving, He brings a solid foundation to tackle complex challenges and contribute to analytical problem-solving scenarios. He is well-equipped to navigate the dynamic landscape of computer science and contribute effectively to innovative solutions.

Mr. Mohd Owais has been a student of Computer Science and Engineering at Khwaja Moinuddin Chishti Language University in Lucknow since July 2020. He is currently working as an intern in the Research and Development department of Bluebook Solutions Private Limited. He possesses a strong groundwork to address intricate challenges and play a valuable role in analytical problem-solving situations. With the ability to adeptly navigate the ever-changing terrain of computer science, he is well-prepared to make meaningful contributions to innovative solutions.

Mrs. Saima Aleem serves as an Assistant Professor at Khwaja Moinuddin Chishti Language University in Lucknow, a role she has undertaken since July 2020. Currently, she is actively engaged as a co-project investigator in a research initiative supported by the Department of Higher Education, Uttar Pradesh Government. In her academic journey, she is pursuing a Ph.D. scholar at Integral University. Saima Aleem's areas of research expertise span across domains, encompassing Computer Networks, Artificial Intelligence, e-services, and e-governance.

Mr. Tasleem Jamal is an Assistant Professor in the Department of Computer Science Engineering, Faculty of Engineering & Technology, Khwaja Moinuddin Chishti Language University in Lucknow, India, Tasleem Jamal's research is focused on LTE, sensor networks, MANET, IoT, Artificial Intelligence, Machine Learning and big data. He has a B.Tech and M.Tech from Zakir Hussain College of Engineering & Technology, AMU, Aligarh. Tasleem Jamal has over 7 years of teaching and research experience, He has published many papers in reputed journals. Additionally, he has presented papers at international conferences. Tasleem Jamal has also attended several short-term courses, workshops, and seminars on topics such as Research Methodology, blockchain, Artificial Intelligence, Machine Learning, and more.

