Journal of Interdisciplinary Innovational Letters
Vol 2 No 1 (2025): 01-22

Empowering OS Efficiency: A Contemporary Investigation of Paradigms of Process Scheduling Algorithms

Ayaz Ali Mandan[1*], Zaeem Nazir[2], Syed Pervez Hussnain Shah[3], Muhammad Haris[4], Muhammad Akram Mujahid[5]

[1]Assistant Professor, Department of Smart Engineering & Advanced Technology, Faculty of Artificial Intelligence, Universiti Teknologi Malaysia (UTM), Malaysia.

[2]*Lecturer, Faculty of Computer Science, University of Narowal, Narowal, Punjab, Pakistan.

[3]Lecturer, Department of Computer Science, Lahore Leads University, Lahore, Punjab, Pakistan.

[4]BSCS Scholar, Department of Information Science, DSNT, University of Education, Lahore, Punjab, Pakistan.

[5]Assistant Professor, Department of Information Science, DSNT, University of Education, Lahore, Punjab, Pakistan.

*** Corresponding author:**zaeem.nazir@gmail.com

Abstract

In task scheduling, every process that has a processor must be supported in parallel processing. in numerous circumstances. Not every algorithm is more effective on important tasks. Occasionally, different algorithms do work better than others. Scheduling algorithms used in contemporary operating systems are compared and discussed. This study's goal is to assess the efficiency and resource consumption of various scheduling algorithms used in the two most popular operating systems, Windows and Linux. Scheduling algorithms are essential for controlling tasks and allocating system resources like memory and CPU time. Effective resource utilization, fairness between many activities, and optimal system performance should all be guaranteed by a smart scheduling algorithm. The project shows a thorough literature assessment of operating system scheduling algorithms, outlining each algorithm's key characteristics, benefits, and drawbacks. The paper also discusses the techniques for assessing the scheduling algorithms' performance, resource use, and example test questions. The study's findings and analysis are included in the report, along with comparisons and contrasts between the scheduling algorithms employed by Windows and Linux. Both operating systems use priority-based scheduling algorithms, but they operate differently in prioritizing threads, sharing time, and scheduling real-time operations. This ends with a discussion of potential future research topics in operating system scheduling algorithms. As technology develops, new difficulties in workload diversity, hardware heterogeneity, and system scalability will emerge. Therefore, it is crucial to keep researching new scheduling algorithms and guidelines that can handle these issues and enhance the functionality, fairness, and resource efficiency of systems. In conclusion, this work gives important insights into the performance, fairness, and resource utilization of scheduling algorithms seen in contemporary operating systems. The study builds on current research in operating system scheduling algorithms and lays the groundwork for more studies in this field. The results of this study may be used for the selection and enhancement of scheduling algorithms for better system performance and resource utilization by operating system designers and researchers.

Keywords: OS Efficiency; Paradigms of Process Scheduling; Algorithms

1. Introduction

It is described as the procedure to check which bulk processes, each of which may be designated a resource safely, will be given access to the resource in operating systems. The algorithm may take into consideration the importance of the individual participating in the process, their desire to maintain high system utilization of the resources, and also meet the job's deadline (Johnson et al., 2023).For example, A priority time-slicing system divides the processes waiting to be executed into several queues, with the higher priority queues receiving a shorter time slice. Once a processor becomes available for scheduling, the oldest process that is present to run in the highest-priority queue is carried out (Al-Fareed et al., 2022).

2. Background History And Literature Review

Computer operating systems depend heavily on scheduling algorithms, which have been created and improved over many years. Scheduling algorithm development may be categorized into several stages, each of which is characterized by distinct conceptual and technological advancements (Fonseca et al., 2022). The development of the hardware for electronic computing devices was the main emphasis of the early computing period (1940s–1950s). Since computer systems were typically single-user systems with few tasks at this point, scheduling methods were not a major problem. Batch processing gained popularity in the late 1950s, and scheduling algorithms were created to control the execution of several user-submitted jobs (Ainsworth, 2023).

First-Generation Operating Systems (1960s): The first-generation operating systems were created in the 1960s, and scheduling algorithms gained importance. There was a finite amount of memory available currently, and computers were still comparatively sluggish. Simple round-robin algorithms that gave every work an equal number of time slices were the first scheduling algorithms to be created. Later, priority scheduling algorithms that assign time slices to jobs depending on their priority levels were established (Gen & Lin, 2023).

Second-Generation Operating Systems (1970s): The second-generation operating systems were created at this time, and scheduling algorithms improved. The emphasis was on creating interactive computer environments for several people at this point when time-sharing started to gain popularity. Algorithms for scheduling were created that could react swiftly to user input and prioritize tasks according to how interactive they were (Chen, 2023) (Hamid, Muhammad, et al., 2022) (Hamid et al., 2023) (Hamid, Iqbal, Muhammad, Fuzail, et al., 2022).

Third-Generation Operating Systems(1980s) The 1980s saw the development of third-generation operating systems and the advancement of scheduling algorithms. At this point, computers were getting more potent and had access to further memory. To accommodate several processes operating formerly and distribute the workload across several processors, cataloging algorithms were created (Asselineau et al., 2022) (Gozdz & Miller, 2023). Real-Time Operating Systems (the 1990s) Real-time operating systems gained fissionability during this period, and real-time task-handling scheduling ways were created. The scheduling algorithms needed to make sure that these jobs were completed on time since they had strict time constraints (Sinha & Sinha, 2022).

2.1 Operating Systems from the 2000s through the 2020s

Algorithms for scheduling still play a crucial role in operating systems in the ultramodern era. Still, the emphasis has turned to tailoring scheduling algorithms for particular workloads, such as those associated with high-performance computing, cloud computing, and mobile computing. To increase the efficiency and responsiveness of computing systems, cataloging methods are continuously improved (Sakshi et al., 2022a). Recent developments in AI-driven compliance systems, such as those used in Anti-Money Laundering (AML) for real-time threat detection and adaptive decision (Rajpoot & Raffat, 2024) making, offer promising insights for enhancing the intelligence and adaptability of modern process scheduling algorithms. Furthermore, post-pandemic research has underscored the transformative role of big data analytics in driving performance and resilience across business systems (Rafi & Sulman, 2025), highlighting its potential to inform more data-aware and context-sensitive scheduling paradigms within operating systems.

Computer System Evaluation, from clean group processing, approaches for a single user to sophisticated multi-user environments with real-time circumstances, is reflected in the history of scheduling algorithms. Cataloging algorithms will continue to be an essential component of operating systems as calculating systems develop, ensuring that calculating funds are utilized effectively and efficiently (Alpala et al., 2022; Hamid et al., 2022a; Hamid et al., 2022b; Muhammad et al., 2022; Hamid et al., 2022c).

2.2 Basic Terminologies

To start with the **CPU Scheduling** we should better know about the basic terminologies which are the following

2.2.1 Process ID

The process ID should be the first thing to solve any kind of problem. The process ID acts as the name of the process being solved. It is usually represented with numbers or the letter P with the number (Wang et al., 2022).

2.2.2 Example

Table No 1: Binary Nos

P0	0
P1	1
P2	2
P3	3

Now it is based on us, we can start with the number 0 or either with 1 or with the letters with the number,

2.2.3 Arrival Time

The time when the process is about to be executed or in the ready queue. It is represented by **AT**. The arrival time is **non-negative, positive,** or **zero** (Salem et al., 2022).

2.2.4 Burst Time

The period that is required by the process to complete the specific task is known as **Burst Time**. The Burst time can be represented as **BT**. The Burst time shouldn't be less than zero, it is always greater than zero

BT > 0 (Bukhari et al., 2022)**.**

2.3 Time of Completion

The total time required by the **CPU** to complete the process is the completion time. Represented as **CT.** It should always be greater than zero (Almhanna et al., 2023).

2.4 Turn Around

The time taken by the **CPU** when the process is ready to be executed or in the **Ready Queue** is called **Turn Around Time.** Represented as TAT. It is calculated by the difference between the Arrival time and completion time in any kind of operating system (Dalmia et al., 2022).

2.4.1 Formula

The difference between the Completion and arrival Times is

$$\boxed{CT-AT=TAT}$$

2.4.2 Waiting Time

The time when the process is waiting for completion is called **Waiting time**. Represented by **WT**. It is calculated with the help of **Turn Around and Burst Time** in specific operating systems (Sakshi et al., 2022b).

2.4.3 Formula

The waiting Time can be calculated as the difference b/w the Turn Around Time & Burst Time is:

$$\boxed{WT= TAT -BT}$$

2.4.4 Ready Queue

The Ready queue is defined as where all the process is stored before the execution of the previous process. The ready queue plays an important role when two processes are about to be carried out. It restrains the CPU when the two same kinds of processes are executed and come into place and then, the duty is fulfilled in an operating system scheduling (Muneeswari & R, 2022).

2.5 Gant Chart

This is a place, chart where all the executed processes are stored. The **Gant Chart** is very useful for calculating the **Waiting**, **Completion**, **Turn Around Time,** etc.

3. Methodologies

Modern computers have multicore processors that use context switching to run multiple programs at once. We compare these CPU scheduling strategies based on their average wait time, response time, and turn-around times. These are scheduled using several computer chip scheduling algorithms, which will be reviewed. Implemented a bar chart and C++ algorithms and code to compare the results of several algorithms to determine the top CPU scheduling algorithm.

4. Analytical Review of Algorithms

The CPU needed some system-determined algorithm when loading from the ready queue to run the next process. The main purpose of scheduling is to keep the CPU busy every second so that processes do not have to wait for long periods. OS processes run in two variants kernel and user. OS tasks involve allocating resources to various processes. After creation, a process may be in various states (fresh, executing, Wait, Prepared, terminated). A fresh state when the process is generated, a running state when the process is running, a waiting condition when it is waiting for some reason, a process when it is ready and waiting on the CPU. is in the Ready state at this point, and when the process has completed its task and left the CPU, the CPU is in the Terminated state. Some processes are ready and need only the CPU to perform and execute their tasks. These processes are held in the ready queue. Multi-threading keeps multiple processes in a prepared queue waiting for the CPU to finish the running process. In the old programming environment, he had a single process running at a time, and if those processes had to wait due to any issue, CPU idle was not a good approach, so CPU idle scheduling algorithms were used to reduce CPU usage. maximize rate. The CPU algorithm keeps track of processes and puts them on the ready queue is executed when the CPU has finished working with the current worker process. If some processes are called at the same or different time intervals and are in the ready queue. According to Sindhu, the short-term scheduler decides which process to run next and allocates CPU, while the long-term scheduler decides which jobs are ready enough to be put in the ready queue. When the short-lived scheduler hands off a terminated process to the CPU dispatcher, we call it here to perform a context switch to save the current process and start a new one.

The FCFS algorithm may outperform other algorithms for short burst times, but the round-robin is better for multiple processes each time. Average latency is a standard measure for evaluating scheduling algorithms. Index Terms - FCFS, SJF, Round Robin, Schedule, OS. In scheduling, there are several methods used to execute queued processes arriving at the processor. Several algorithms are common, such as first-come, first-served, shortest job first, and round-robin. The purpose of this comparison is to check which algorithm is good for some processes in the ready queue. Job scheduling aims to shift CPU work between processes. If processes arrive at the same time, the services they perform are queued in order. Processes in the ready state are queued in FCFS according to their arrival time. Here, however, processes with short burst times must wait until other processes have finished. Each process is assigned a priority number which is the burst period. The FCFS algorithm is used when multiple processes have the same priority. Let's say P1 is given more importance than P0. When the CPU burst is over, the non-preemptive fixed algorithm finishes P0, moving P1 to the front of the queue. The arrival timings for a completed queue will be the same as for P1–P3, P4–P6, and so on.

The algorithm under the round-robin notion involves time sharing. In contrast, if a process's CPU burst exceeds the time quantum, the process will be paused when the time quantum is achieved and be forced to wait until the end of the available queue position before continuing. Quantum ready queue for q Time. The CPU schedules up to q time units at a moment, allocating 1/n of CPU time to each task. If one process is running on the CPU, the other process waits until the CPU is free. A multi-programmed operating system runs processes concurrently to increase CPU utilization. An entire process requires CPU cycles and I/O cycles to complete execution. There is no useful work to do during this time, as other processes waiting for the CPU may notice longer wait times. In multiprogramming, the operating system manages the scheduling of processes, and runnable processes are kept in memory at the same time. Applying this method can increase CPU utilization and minimize process latency. In this paper, researchers presented a comparative analysis of traditional CPU scheduling algorithms. CPU scheduling is therefore an internal part of the OS design. When multiple processes become available to run in the ready queue, the operating system uses a scheduler (using scheduling algorithms) to determine which process to run. These scheduling algorithms are used to minimize throughput time, response time, and latency, and avoid context switching. Scheduling Criteria There are many CPU scheduling algorithms with varying characteristics, and choosing a particular algorithm may favor one class of processes over another. The scheduling algorithm is therefore designed to minimize the number of context switches. Therefore, the scheduling algorithms are designed to use the CPU as much as possible. Planning algorithms are therefore designed to minimize throughput time. This is the sum of all wait times performed by processes in the ready-to-run queue. Scheduling algorithms are therefore designed to minimize latency.

The operating system "OS" depends on CPU scheduling since the processor is a crucial resource. aims in design. The CPU should be able to run as many tasks at once for the best efficiency. This white paper presents a state diagram that represents a comparative study of different scheduling algorithms for a single CPU and shows the best algorithm for each situation. With this representation, it's much easier to understand what's going on in the system and why different sets of processes are candidates for CPU allocation at different times. The purpose of this research is to analyze efficient CPU schedulers to design high-quality scheduling algorithms that meet scheduling goals. Only one process can be active at once on a single-processor machine. The rest of the group must hold off until the CPU can reschedule. To get the most out of the CPU, multiprogramming aims to always keep processes active. The operating system's essential task is scheduling. Virtually all computer resources are planned before being used. Therefore, the operating system's design is centered on that strategy. Because it may significantly affect resource use and overall system performance, CPU scheduling is crucial. The rest of the book is organized as follows. The long-term or admission scheduler determines which jobs or processes are placed in the ready queue. That is, when a process is attempted to run, its inclusion in the set of currently running processes is either approved or delayed by the long-term scheduler. This scheduler thus determines the processes that the system needs to run and the degree of concurrent parallelism it supports Half-time scheduler Processes are momentarily moved from main memory to secondary storage (such as a disc drive) by the half-time scheduler. a short-term planner The CPU scheduler, is also referred to as the short-term scheduler in an operating system.

4.1 Results and Analysis

The function then prints out this information for each process and calculates the average waiting time. Finally, the main function creates a vector of processes and calls the FCFS function to run the simulation.

4.2 First Come First Serve

It is the most basic kind of scheduling algo & as implied by the name, a FIFO queue since any process that arrives first would be run first. There are also several issues related to this kind, such as the fact that if the process is serviced first is too long subsequent, shorter processes will have to wait for an extended period, which will increase the average waiting time. The effect of convoys is a different term for this problem. The process doesn't stop using the computer chip until it is completed.

4.3 FCFS Processing Steps

1. P1 enters the ready queue first, and the immediate-term scheduler allows CPU to P1
2. P2 reaches the ready queue after P1, and P1 must wait for P2 to complete running before it can release the CPU.
3. Following that, P3 shows up, and it too must wait for the CPU to get up.
4. After P1 leaves the CPU, P2 and P3 each begin their computations and finish their respective tasks.

Table No 2: Arrival Period & Burst Period

ID	Arrival Period	Burst Period
P1	0	20
P2	1	6
P3	2	3

Figure No 1: Gant Chart

Scenario # 1

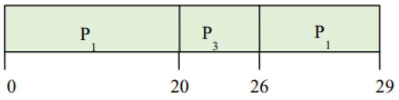


Fig. 2. FCFS Processes Sequence (1st case)

Scenario # 2

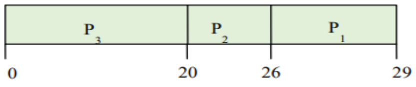


Fig. 3. FCFS Processes Sequence (2nd case)

Table No 3: Calculation

Calculation # 1	**Calculation # 2**
Beginning time minus time of arrival equals wait time. Total W/T = (0-0) + (20+1) + (26-2) Total Time Waited = 43 Wait Time on Average = 14.33 Finish T – arrival T= total turnaround T. Total Turnaround Time = (20) + (25) + (27) Turnaround Time Overall = 72 Response Time on Average = 24	Beginning time minus time of arrival equals wait period. Total W/T = (0-0) + (20+1) + (26-2) Total Time Waited = 43 Wait Time on Average = 14.33 Finish T – arrival T= total turnaround T Total Turnaround Time = (20) + (25) + (27) Turnaround Time Overall = 72 Response Time on Average = 24

Figure No 2: FCFS

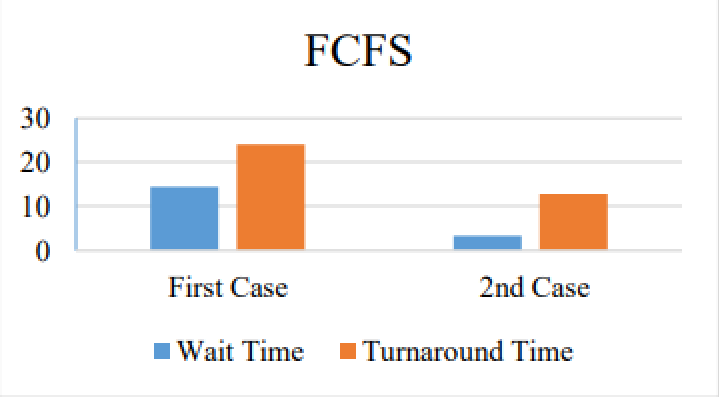


Fig. 4. FCFS 1st case vs 2nd case

4.4 Shortest Job First

The selection of the process in this scheduling method is based on the process execution's smallest burst time. This scheduling method is superior to FCFS scheduling algorithms since we encounter the least amount of waiting and turnaround time. However, it also has some flaws, like how challenging it is to foresee the next memory chip burst time demand. Furthermore, the CPU does not use this algo for scheduling at the most minimal level. Finally, this type's main flaw is the starving process.

Table No 4: Arrival Period & Burst Period

ID	Arrival Period	Burst Period
P1	5	10
P2	0	9
P3	2	6
P4	3	4

Figure No 3: Gant Chart

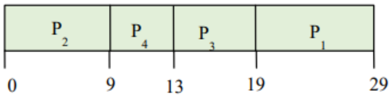
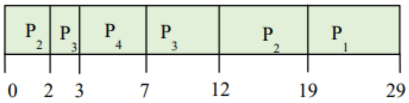
Scenario # 1	Scenario # 2
 Fig. 8. SJF processes sequence (1st case)	 Fig. 9. Priority processes sequence (2st case)

Table No 5: Calculation

Calculation # 1	Calculation # 2
Starting time - arrival time = wait / t Total Wait/T = (6.5) + (0-0) + (16.3) Total W/T =14 Average W/T = 4.66 Finish T – arrival T = total turn-around T. Turnaround Total = (16-5) + (6-0) + (24-3) Average = 12.66 Total Turnaround Time + 38	Start time – arrival time = Wait time Total W/T = (0-5), (0-0), 21-3, 3-3, and 15-5. Average W/T = 9.33 Total Wait/T = 28 Finish T – arrival T = total turn-around T. Turn-around T.= (15-5) + (24-0) + (21-3) Turnaround Time Overall = 52 Average = 17.33
Steps of execution Case 1: 1. P2 reaches 0 in this situation, the short-term scheduler assigns computer chip resources to it because there is no better alternative. 2. Because It is a non-primitive instance, the CPU can become available after the job is completed, therefore as soon as P2 is finished, every other procedure arrives in the ready queue. 3. The scheduler for short- now chooses the Job that received the least amount of burst a period in this case P4. 4. Following P4, P3, and finally P1 will receive CPU to finish their tasks.	**Steps of execution Case 2:** 1. P2 arrives at 0 in this scenario, and the CPU is assigned to it. 2. When P3 arrives at 2 and P2 still has a longer burst duration than P3, the dispatcher switches context, and the CPU is now assigned to P3. 3. After P4 comes at 3 and the CPU switches to it, P1 follows at 3 but has a longer burst duration than P4 so the CPU stays assigned to P4. 4. After P4's job was completed, the short-term scheduler allotted CPU to P2's and P1's shortest jobs.

Figure No 4: FCFS

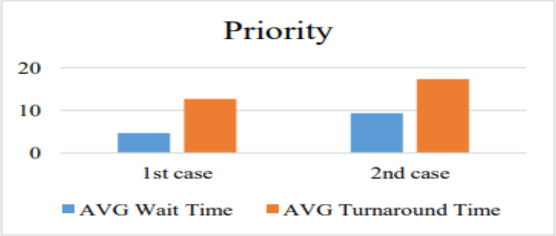


Fig. 7. Priority 1st case Vs 2nd case

4.5 Priority Based Scheduling

According to the kind of process, a priority scheduling method classifies processes using a set of system criteria. A group of actions defines the process's priority, and every step that adds itself to the ready queue has its value as a top priority. The only priority number is whether to allocate a process to the CPU in such its high-value priority will reach the CPU first or second. Preemptive and non-preemptive variants of this algorithm are also available. If an instance with a high priority continues to appear in the ready queue, an algorithm's preemptive type may cause the least important process to starve to death.

Table No 6: Arrival Period, Launch Period & Priority

ID	Arriving Period	Launch Period	Priority
P1	5	10	1
P2	0	6	5
P3	3	8	4

Figure No 5: Gant Chart

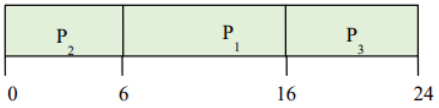
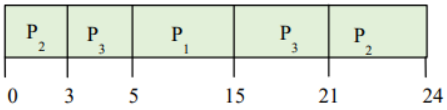
Scenario # 1	Scenario # 2
 Fig. 5. Priority Processes sequence (1st case)	 Fig. 6. Priority processes sequence (2st case)

Table No 7: Calculation

Calculation # 1	**Calculation # 2**
Start time - arrival time = wait time Total = (6.5) + (0) + (16.3) Total =14 Average W/T = 4.66 End T - arrival = total turn-around time. Turnaround Time in Total = (16-5) + (6-0) + (24-3) Average Turn-around T= 12.66 Total = 38	Start time - arrival time = wait time Total = (5), (0-0), 21-3, 3-3, and 15-5. Average Wait Time = 9.33 Total Wait/Time = 28 Finish time - arrival = total turn-around. Turnaround Time = (15.5) + (24.0) + (21.3) Turnaround Overall = 52 Average = 17.33
Steps of execution Case 1: 1. Despite having the lowest priority, P2 is the sole process to arrive at time 0 in this situation, therefore the STS allocates processing to the 2nd process. 2. Because of a non-primitive case, the CPU is allotted and is only released after the job is complete. Like P2 just ended, P1 % P3 is added to the queue of people who are ready to work. 3. Next, the short-term scheduler makes decisions based on priority, with P1 having the greatest priority followed by P3. Therefore, the CPU is assigned to P1, and after its work is finished, the CPU is allocated to P3.	**Steps of execution Case 2:** 1. Like the previously mentioned situation, P2 receives CPU. 2. In This step, the short-term scheduler resumes its search for the procedure in the ready queue with the greatest priority after commencing execution. then the currently active process because of the straightforward circumstance. 3. The short-term scheduler will assign the central processing unit to P3 when it arrives with a priority greater than the process that is currently running. P2 still has work to do, so its dispatcher must store its current progress and send it to the ready queue once more. which just invoke, complete, and dispose of. 4. When P1 shows up, the CPU is given to P1, which has a high priority, and P3's progress is stored before being sent into the ready queue to wait. P1 has now finished working and released the CPU. 5. Once the dispatcher has finished loading the P3 and the short-term scheduler has given the CPU to P3 for its job, the CPU is then allotted to P2.

4.6 Shortest Remaining Time

In this case, the scheduling method creates the ready queue based on the process' burst times. The processes that take a brief period for completion are moved toward the front of the queue. Preemption causes the process to be divided into two parts, which results in more context-switching. Then following all-time units, the process's burst time is tracked. Check the technique that results in the following scheduled task having the shortest burst duration after executing one unit. The Shortest Time Remaining Time Scheduling-Preemptive type is another name for this type. Input Processes and Shortest Remaining Time are called by the main function to input the processes and perform the SRT algorithm, respectively.

4.7 Multilevel Feedback Queues Scheduling

This implementation stores the Round Robin algorithm's Round Robin ID, arrival time, burst time, priority, waiting time, turnaround time, and remaining time in a struct named "Process." The 'MLFQ' function uses three queues—one for each level of priority—to implement the Multilevel Feedback Queues Scheduling method. The round-robin algorithm with a predetermined quantum is used in the first queue, which has the highest priority. The third queue has the lowest priority, followed by the second queue with a lower priority. A process is rammed to a lower priority queue each time it is preempted. The process ID, arrival time, burst time, priority, completion time, turnaround time, and waiting time for each process are all printed out by the function. Furthermore, it computes the average, all processes must wait their turn. The ‘main' function defines the processes and the quantum and invokes the 'MLFQ' function to schedule the processes.

4.8 Highest Response Ratio Next

The Process struct in this implementation represents a single process and has the following fields: ID, arrival time, burst time, waiting time, and turnaround time. The processes are sorted by arrival time using the cmp function, followed by the greatest response ratio using the hrrn_cmp function. The processes are then ordered according to arrival time, a queue is initialized to contain the processes that are prepared to execute, and the time is set to 0. The main loop of the program runs until all processes have been completed. Each iteration of the loop first adds any processes that have arrived in the queue and then checks if the queue is empty. If the queue is empty, the program increments the current time and continues to the next iteration of the loop. If the queue is not empty, the program sorts the processes in the queue by the highest response ratio next using the hrrn_cmp function. It then selects the process with the highest response ratio, calculates its waiting time and turnaround time, updates the current time, and prints the waiting time.

4.9 Round Robin

It is a scheduling method used by network and process schedulers in computers. Time slices, or time quanta as they are more often known, are allocated to each process in equal chunks and circular order, treating all processes equally and without regard to priority (also known as cyclic executive). Scheduling using a round-robin is straightforward, simple to implement, and starvation-free. Other scheduling issues, such as data packet scheduling in computer networks, can be solved with round-robin scheduling. It is an idea for an operating system.

Table No 8: Arrival Period, Launch Period & Priority

ID	Arrival Period	Launch Period	Priority
P1	2	5	3
P2	0	10	1
P3	4	15	5
P4	1	20	2
P5	3	25	4

Figure No 6: Gant Chart

RR with first come first serve.

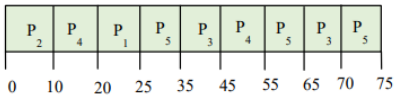


Fig. 11. RR (FCFS) processes Sequence

RR with SJF.

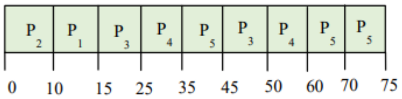


Fig. 12. RR (SJF) processes Sequence

RR with Priority Sch.

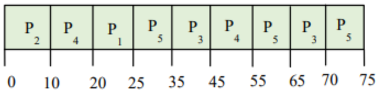


Fig. 13. RR (Priority) processes Sequence

Table No 9: Calculation

Scenario # 1	**Scenario # 2**	**Scenario # 3**
Starting T - arrival T = wait T Total Wait Time = (20, 2) + (0, 0) + [(35, 4), (+(65, 45)] +[(10- 1)+(45-20)]+[(25-3)+(55-35)+(70-45)] Total = 180 Average wait Time = 36 Finish time - arrival time = total turnaround time. Total Turnaround Time = (25-2) + (10-0) + (70.4), (55.1), (75.2), and (75.3) Total Turnaround Time= 225 Average Turnaround Time= 45	Starting T - arrival T = wait T Wait Time = (10-2 + 0) + [(15-4) + (45-25)]+[(25- 1)+(50-35)]+[(35-3)+(60-45)] Waiting T Overall = 125 The typical = 25. Finish time - arrival time = total turnaround time. Total Turnaround Time = (15.2) + (10.0) +(50.4) +(60.1) Total Turnaround Time = 205 Average Turnaround Time = 41	Starting T - arrival T = wait T Wait Time = (18) + [(31) + (20)]+[(9)+(25)]+[(22)+(20)+(15)] Average W/T=30 Total = 150 Finish time - arrival time = total turnaround time. Total Turnaround Time = (25-2) + (10-0) + (70.4) + (55.1) + (75.2) + (75.3) Total Turnaround Time: 225 Average Turnaround Time =45

Figure No 7: FCFS

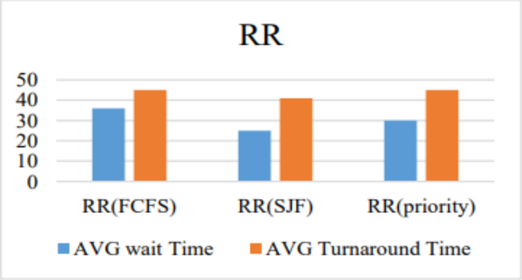


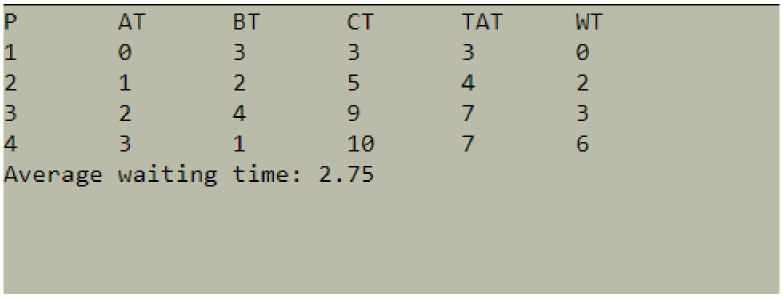
Fig. 14. Comparison of RR (FCFS), RR (SJF) and RR (priority)

4.10 Runtime Codes Output

4.11 First Come First Serve

In this algorithm tasks or processes are executed according to their arrival order, without any priority. The first task in the arrival list is executed first and this is the output of an FCFS algorithm.

Table No 10: First Come First Serve

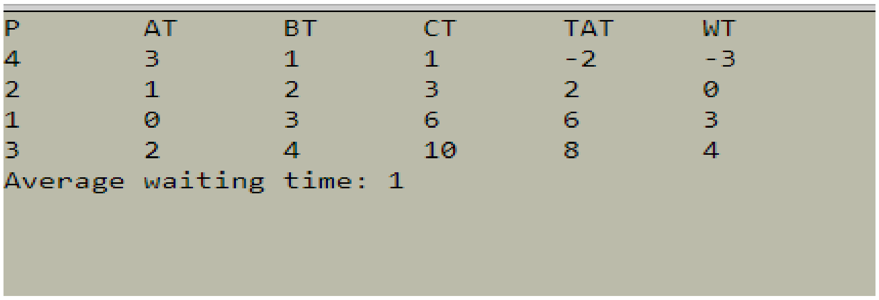


```
P       AT      BT      CT      TAT     WT
1       0       3       3       3       0
2       1       2       5       4       2
3       2       4       9       7       3
4       3       1       10      7       6
Average waiting time: 2.75
```

4.11 Shortest Job First

In this algorithm, a task or a process having a short execution time is executed first and the ones having a longer execution period will be set for later execution. If a longer task is running and a short task arrives, this algorithm stops the longer task and starts executing the short task, this is an output of the SJF algorithm.

Table No 11: Shortest Job First

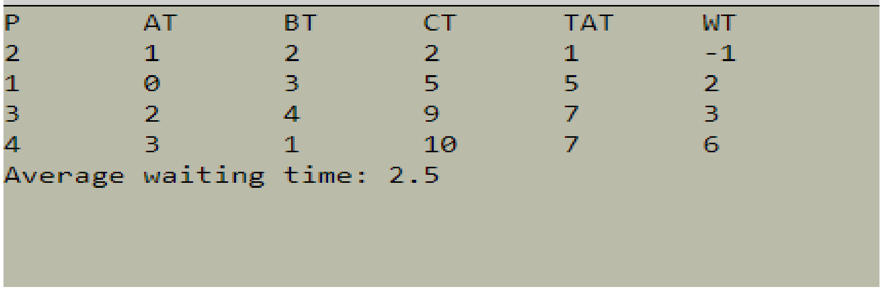


```
P         AT        BT        CT        TAT       WT
4         3         1         1         -2        -3
2         1         2         3         2         0
1         0         3         6         6         3
3         2         4         10        8         4
Average waiting time: 1
```

4.12 Priority Based Scheduling

In this algorithm tasks are executed according to the level of their priority, the tasks with higher priority levels are executed before the ones having a low level of priority. If a task with a high priority level comes during the execution of a low-priority task, it stops the execution of the low-level task and starts executing the high-level priority task. Here is an output of the PBS algorithm. This scheduling algorithm can either be preemptive or non-preemptive.

Table No 12: Priority Based Scheduling

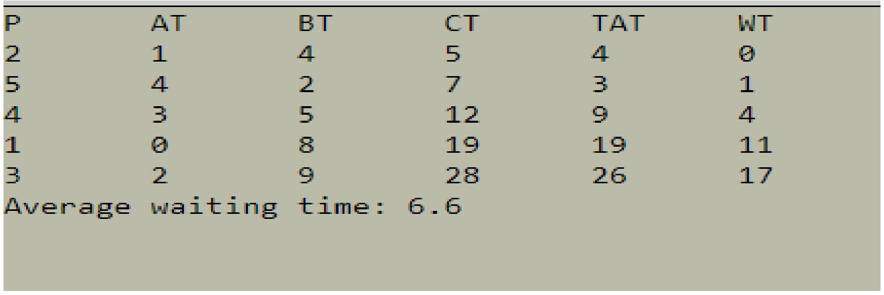


```
P         AT        BT        CT        TAT       WT
2         1         2         2         1         -1
1         0         3         5         5         2
3         2         4         9         7         3
4         3         1         10        7         6
Average waiting time: 2.5
```

4.13 Shortest Time Remaining

In this algorithm, the tasks or processes are executed according to their remaining processing time. The tasks with a high remaining process time are executed later and the ones having a small remaining process time are executed first. Moreover, it is a preemptive scheduling algorithm. The output of a STR algorithm is given below.

Table No 13: Shortest Time Remaining

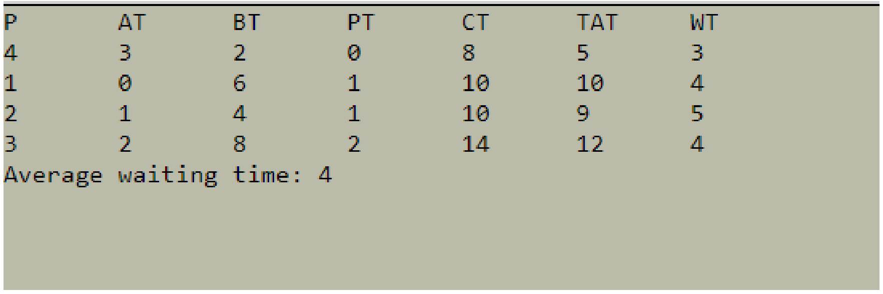


```
P        AT       BT       CT       TAT      WT
2        1        4        5        4        0
5        4        2        7        3        1
4        3        5        12       9        4
1        0        8        19       19       11
3        2        9        28       26       17
Average waiting time: 6.6
```

4.14 Multilevel Feedback Queue Scheduling

In this algorithm, multiple queues with different priority levels are used to schedule tasks. Tasks are shifted between queues based on their characteristics like priority or CPU burst duration, and each queue has a different priority level. The output for the MLFQ algorithm is given as follows.

Table No 14: Multilevel Feedback Queue Scheduling

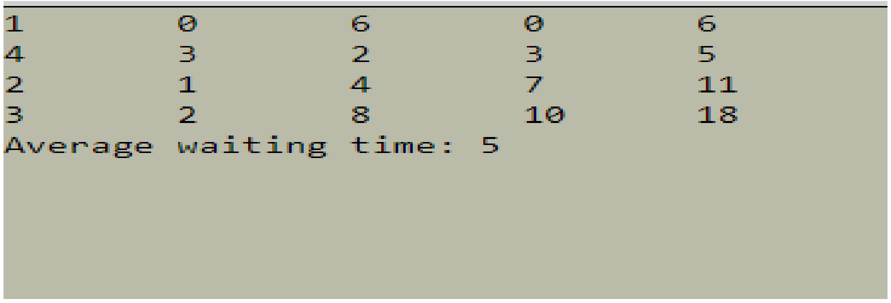


```
P       AT      BT      PT      CT      TAT     WT
4       3       2       0       8       5       3
1       0       6       1       10      10      4
2       1       4       1       10      9       5
3       2       8       2       14      12      4
Average waiting time: 4
```

4.15 Highest Response Ratio Next

In this algorithm, tasks are selected with the highest response ratio. It is the ratio of the task's expected processing time to the total of the task's awaiting period and time for processing. Using that approach, it is ensured that tasks with a higher response ratio and hence, a stronger sense of urgency are carried out first. The code output for the HRRN algorithm is given below.

Table No 15: Highest Response Ratio Next

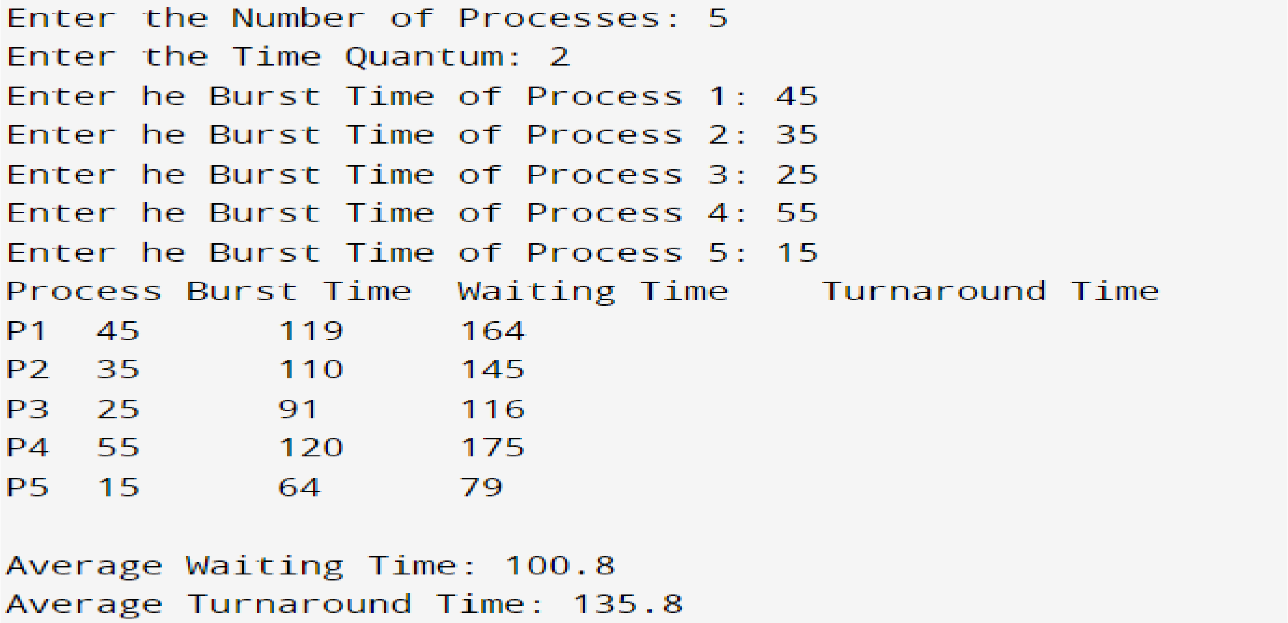


```
1        0        6        0        6
4        3        2        3        5
2        1        4        7        11
3        2        8        10       18
Average waiting time: 5
```

4.16 Round Robin

It is a preemptive scheduling algorithm that assigns a fixed time quantum to each task in a cyclic order. Every process is executed till the duration of that time quantum and if its execution is not complete it is moved to the end of the queue and then the next task is executed in the same order. A coded output of the RR algorithm is given below.

Table No 16: Round Robin

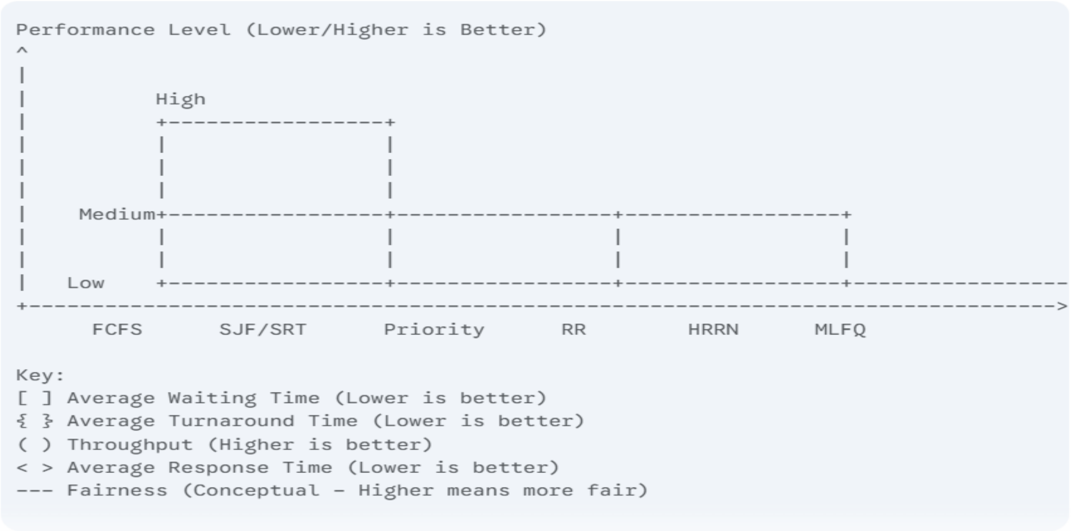


```
Enter the Number of Processes: 5
Enter the Time Quantum: 2
Enter he Burst Time of Process 1: 45
Enter he Burst Time of Process 2: 35
Enter he Burst Time of Process 3: 25
Enter he Burst Time of Process 4: 55
Enter he Burst Time of Process 5: 15
Process Burst Time  Waiting Time    Turnaround Time
P1   45       119      164
P2   35       110      145
P3   25       91       116
P4   55       120      175
P5   15       64       79

Average Waiting Time: 100.8
Average Turnaround Time: 135.8
```

4.17 Comparative Analysis

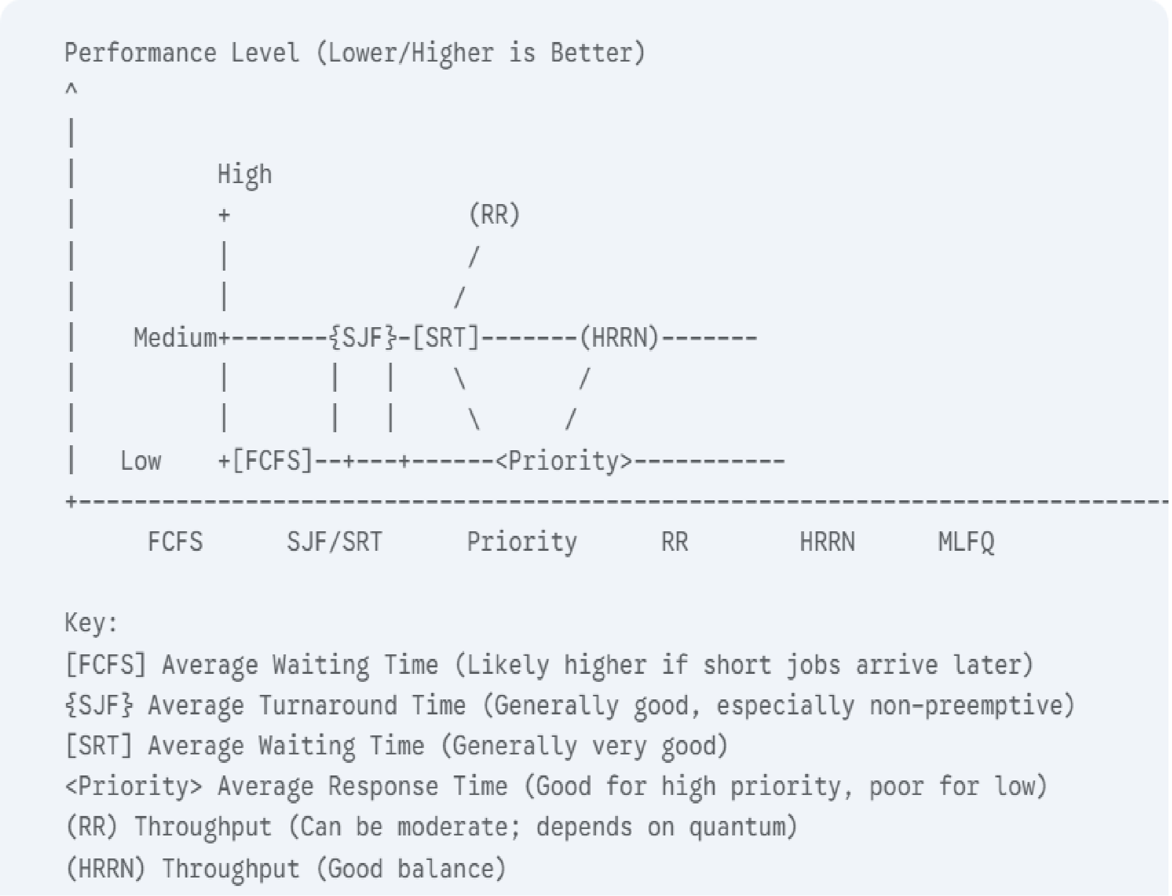
Table No 17(a): Comparative Analysis



```
Performance Level (Lower/Higher is Better)
^
|
|          High
|          +-----------------+
|          |                 |
|          |                 |
|          |                 |
|    Medium+-----------------+-----------------+-----------------+
|          |                 |                 |                 |
|          |                 |                 |                 |
|   Low    +-----------------+-----------------+-----------------+-----------------
+---------------------------------------------------------------------------------->
      FCFS      SJF/SRT      Priority      RR        HRRN      MLFQ

Key:
[ ] Average Waiting Time (Lower is better)
{ } Average Turnaround Time (Lower is better)
( ) Throughput (Higher is better)
< > Average Response Time (Lower is better)
--- Fairness (Conceptual - Higher means more fair)
```

Table No 17(b): Comparative Analysis



```
Performance Level (Lower/Higher is Better)
^
|
|          High
|          +                 (RR)
|          |                 /
|          |                /
|    Medium+-------{SJF}-[SRT]-------(HRRN)-------
|          |       |   |    \        /
|          |       |   |     \      /
|   Low    +[FCFS]--+---+------<Priority>-----------
+-------------------------------------------------------------------------------
      FCFS      SJF/SRT      Priority      RR        HRRN      MLFQ

Key:
[FCFS] Average Waiting Time (Likely higher if short jobs arrive later)
{SJF} Average Turnaround Time (Generally good, especially non-preemptive)
[SRT] Average Waiting Time (Generally very good)
<Priority> Average Response Time (Good for high priority, poor for low)
(RR) Throughput (Can be moderate; depends on quantum)
(HRRN) Throughput (Good balance)
```

FCFS: Low (Long jobs can block short ones)

SJF: Low (Starvation possible for long jobs)

SRT: Low (Starvation possible for long jobs)

Priority: Low (Low priority can starve)

RR: High (Each process gets a fair share of time)

HRRN: Moderate to High (Attempts to address starvation)

MLFQ: Can be high if configured properly

{SJF} Average Turnaround Time (Generally good)

{SRT} Average Turnaround Time (Generally good)

{Priority} Average Turnaround Time (Depends on priority mix)

[RR] Average Waiting Time (Can be higher with small quantum due to context switching)

[HRRN] Average Waiting Time (Attempts to be good)

[MLFQ] Average Waiting Time (Depends on configuration)

<FCFS> Average Response Time (Can be high for short jobs arriving after long ones)

<SJF> Average Response Time (Good for short jobs)

<SRT> Average Response Time (Excellent for short jobs)

(Priority) Throughput (Depends on priority mix and preemption)

(MLFQ) Throughput (Depends on configuration

5. Conclusion

In conclusion, scheduling algorithms play an important part in determining the effectiveness and performance of operating systems. Each algorithm has its strengths and weaknesses, and the choice of algorithm depends on the nature of the system and the tasks being performed. Round Robin is a simple and fair algorithm that distributes CPU time inversely among processes. First come first serve is also a simple algorithm that's easy to apply but can affect long delay times for high- precedence tasks. Priority-grounded scheduling is a useful algorithm that prioritizes tasks grounded on their significance. Multilevel feedback queue scheduling is a complex algorithm that stoutly adjusts the precedence of tasks grounded, which can ameliorate overall system performance. The shortest job first and shortest remaining time are effective algorithms that prioritize short tasks to minimize delay times. Eventually, the loftiest response rate next is an algorithm that balances the precedence and length of tasks to ensure effective use of system coffers. Overall, opting for the applicable scheduling algorithm is pivotal for achieving optimal system performance, and it's important to consider the unique characteristics of each algorithm before deciding.

References

Ainsworth, B. (2023). Trevor Pearcey and the development of CSIRAC – An Australian first-generation computer. *IEEE Annals of the History of Computing*, 1–13. IEEE Annals of the History of Computing. https://doi.org/10.1109/MAHC.2023.3306780

Al-Fareed, H., Alghamdi, O., Alshuraya, A., Alqahtani, M., Alwasfer, S., Aljomea, A., Rahman, A., Aljameel, S., & Krishnasamy, G. (2022). Simulator for Scheduling Real-Time Systems with Reduced Power Consumption. *Mathematical Modelling of Engineering Problems*, *9*, 1225–1232. https://doi.org/10.18280/mmep.090509

Almhanna, M. S., Al-Turaihi, F. S., & Murshedi, T. A. (2023). Reducing waiting and idle time for a group of jobs in the grid computing. *Bulletin of Electrical Engineering and Informatics*, *12*(5), Article 5. https://doi.org/10.11591/eei.v12i5.4729

Alpala, L. O., Quiroga-Parra, D. J., Torres, J. C., & Peluffo-Ordóñez, D. H. (2022). Smart Factory Using Virtual Reality and Online Multi-User: Towards a Metaverse for Experimental Frameworks. *Applied Sciences*, *12*(12), Article 12. https://doi.org/10.3390/app12126258

Asselineau, C.-A., Pye, J., & Coventry, J. (2022). Exploring efficiency limits for molten-salt and sodium external cylindrical receivers for third-generation concentrating solar power. *Solar Energy*, *240*, 354–375. https://doi.org/10.1016/j.solener.2022.05.001

Bukhari, M. M., Ghazal, T. M., Abbas, S., Khan, M. A., Farooq, U., Wahbah, H., Ahmad, M., & Adnan, K. M. (2022). An Intelligent Proposed Model for Task Offloading in Fog-Cloud Collaboration Using Logistics Regression. *Computational Intelligence and Neuroscience*, *2022*, e3606068. https://doi.org/10.1155/2022/3606068

Chen, X. (2023). Research on Human Resource Allocation of Vulnerable Groups in Enterprises Based on a Resource Scheduling Algorithm. *Journal of The Institution of Engineers (India): Series C*, *104*(2), 339–344. https://doi.org/10.1007/s40032-023-00911-6

Dalmia, R., Sinha, A., Verma, R., & Gupta, P. K. (2022). *Dynamic Ready Queue Based Process Priority Scheduling Algorithm* (arXiv:2205.07314). arXiv. https://doi.org/10.48550/arXiv.2205.07314

Fonseca, T., Ferreira, L. L., Landeck, J., Klein, L., Sousa, P., & Ahmed, F. (2022). Flexible Loads Scheduling Algorithms for Renewable Energy Communities. *Energies*, *15*(23), Article 23. https://doi.org/10.3390/en15238875

Gen, M., & Lin, L. (2023). Nature-Inspired and Evolutionary Techniques for Automation. In S. Y. Nof (Ed.), *Springer Handbook of Automation* (pp. 483–508). Springer International Publishing. https://doi.org/10.1007/978-3-030-96729-1_21

Gozdz, K., & Miller, R.-E. L. (2023). *Developing Third-Generation Learning Organizations: A Heuristic Discovery Process*. Cambridge Scholars Publishing.

Hamid, K., Iqbal, M. waseem, Ashraf, M. U., Alghamdi, A., Bahadad, A., & Almarhabi, K. (2022). Optimized Evaluation of Mobile Base Station by Modern Topological Invariants. *Computers, Materials and Continua*, *74*, 363–378. https://doi.org/10.32604/cmc.2023.032271

Hamid, K., Iqbal, M. waseem, Muhammad, H., Basit, M., Fuzail, Z., † Z., & Ahmad, S. (2022). *Usability Evaluation of Mobile Banking Applications in Digital Business as Emerging Economy*. 250. https://doi.org/10.22937/IJCSNS.2022.22.2.32

Hamid, K., Iqbal, M. waseem, Muhammad, H., Fuzail, Z., & Nazir, Z. (2022a). ANOVA BASED USABILITY EVALUATION OF KID'S MOBILE APPS EMPOWERED LEARNING PROCESS. *Qingdao Daxue Xuebao(Gongcheng Jishuban)/Journal of Qingdao University (Engineering and Technology Edition)*, *41*, 142–169. https://doi.org/10.17605/OSF.IO/7FNZG

Hamid, K., Iqbal, M. waseem, Nazir, Z., Muhammad, H., & Fuzail, Z. (2022b). USABILITY EMPOWERED BY USER'S ADAPTIVE FEATURES IN SMART PHONES: THE RSM APPROACH. *Tianjin Daxue Xuebao (Ziran Kexue Yu Gongcheng Jishu Ban)/Journal of Tianjin University Science and Technology*, *55*, 285–304. https://doi.org/10.17605/OSF.IO/6RUZ5

Hamid, K., Muhammad, H., Iqbal, M. waseem, Bukhari, S., Nazir, A., & Bhatti, S. (2022c). ML-BASED USABILITY EVALUATION OF EDUCATIONAL MOBILE APPS FOR GROWN-UPS AND ADULTS. *Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)*, *41*, 352–370. https://doi.org/10.17605/OSF.IO/YJ2E5

Hamid, K., Muhammad, H., Iqbal, M. waseem, Nazir, A., shazab, & Moneeza, H. (2023). ML-BASED META MODEL EVALUATION OF MOBILE APPS EMPOWERED USABILITY OF DISABLES. *Tianjin Daxue Xuebao (Ziran Kexue Yu Gongcheng Jishu Ban)/Journal of Tianjin University Science and Technology*, *56*, 50–68.

Johnson, E., Laufer, E., Zhao, Z., Gohman, D., Narayan, S., Savage, S., Stefan, D., & Brown, F. (2023). WaVe: A verifiably secure WebAssembly sandboxing runtime. *2023 IEEE Symposium on Security and Privacy (SP)*, 2940–2955. https://doi.org/10.1109/SP46215.2023.10179357

Muhammad, H., Basit, M., Hamid, K., Iqbal, M. waseem, Shahzad, S., Muneem, F., & Shaheryar, M. (2022). *Usability Impact of Adaptive Culture in Smart Phones*.

Muneeswari, G., & R, R. S. (2022). Agent Based Queue Aware Scheduling for Distributed Multicore System. *2022 IEEE 7th International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, *7*, 1–6. https://doi.org/10.1109/ICRAIE56454.2022.10054343

Sakshi, Sharma, C., Sharma, S., Kautish, S., A. M. Alsallami, S., Khalil, E. M., & Wagdy Mohamed, A. (2022a). A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time. *Alexandria Engineering Journal*, *61*(12), 10527–10538. https://doi.org/10.1016/j.aej.2022.04.006

Sakshi, Sharma, C., Sharma, S., Kautish, S., A. M. Alsallami, S., Khalil, E. M., & Wagdy Mohamed, A. (2022b). A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time. *Alexandria Engineering Journal*, *61*(12), 10527–10538. https://doi.org/10.1016/j.aej.2022.04.006

Salem, I., Mijwil, M., Wagih, A., & M. Ismaeel, M. (2022). Flight-schedule using Dijkstra's algorithm with comparison of routes findings. *International Journal of Electrical and Computer Engineering*, *12*, 1675–1682. https://doi.org/10.11591/ijece.v12i2.pp1675-1682

Sinha, P. K., & Sinha, P. (2022). *Foundations of Computing: Essential for Computing Studies, Profession And Entrance Examinations - 5th Edition*. BPB Publications.

Wang, G.-G., Gao, D., & Pedrycz, W. (2022). Solving Multiobjective Fuzzy Job-Shop Scheduling Problem by a Hybrid Adaptive Differential Evolution Algorithm. *IEEE Transactions on Industrial Informatics*, *18*(12), 8519–8528. IEEE Transactions on Industrial Informatics. https://doi.org/10.1109/TII.2022.3165636

Rajpoot, M. H., & Raffat, M. W. (2024). The AI-Driven Compliance and Detection in Anti-Money Laundering: Addressing Global Regulatory Challenges and Emerging Threats: AI-Driven AML: Compliance Threat Detection. *Journal of Computational Science and Applications (JCSA)*, ISSN: 3079-0867 (Onilne), 1(2).

Rafi, S., & Sulman, M. (2025). Post-Pandemic Insights: Evaluating the Impact of Big Data Analytics, Circular Economy Practices, and Digital Marketing on Firm Performance. *Journal of Computational Informatics & Business*, 2(1), 8-16.