Heliyon 10 (2024) e29959

Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon

Research article

Study and evaluation of CPU scheduling algorithms

Miguel González-Rodríguez, Lorena Otero-Cerdeira *, Encarnación González-Rufino, Francisco Javier Rodríguez-Martínez **

Escuela Superior de Ingeniería Informática, Campus As Lagoas, University of Vigo, Spain

ARTICLE INFO

Keywords:
Educational simulators
Evaluation methodologies
Improving classroom teaching

ABSTRACT

In the teaching of the operating systems course, which is part of computer engineering degrees, a thorough understanding of processor scheduling algorithms is crucial. However, it has been identified that the current knowledge of classical algorithms is insufficient in the present context. Therefore, it is proposed to conduct a review of the state of the art in the field to identify new trends and algorithms that can enhance the teaching of the subject and improve student training. As a result, the state of the art is thoroughly reviewed, and study sheets are designed to facilitate the comprehension of the algorithms. Additionally, a software simulator is developed to compare different algorithms in a controlled environment, allowing for the validation of the most promising ones for classroom teaching.

1. Introduction

An important aspect of the operating systems course, which is part of the computer engineering degree at the University of Vigo, is the theoretical and practical understanding of processor scheduling algorithms. Despite the significance and teaching focus given to this topic, students have shown difficulties in comprehending it, leading to negative outcomes in their overall performance in the course.

Specifically, in recent years, a correlation has emerged between the failure rate of students in activities related to process scheduling and the final percentage of students failing the course on their first attempt. This relationship is depicted in Fig. 1.

In an effort to address this issue, it has been proposed to enhance the course not only by covering classical algorithms but also by incorporating more contemporary algorithms employed in current operating systems or specific environments. Additionally, to facilitate the understanding of these algorithms, concise study sheets will be developed and provided alongside the teaching materials and current activities.

To achieve this objective, our initial proposal involves conducting a comprehensive study on the current state of the art in processor scheduling algorithms. This will enable us to select algorithms that are academically interesting and relevant.

The following sections describe the process of achieving this objective. Initially, in section 2, a contextualization of the subject matter and basic concepts is introduced. In section 3, we present a compilation of current studies on process planning and analyze the research trends in this field. In section 4, thanks to this review of the state of the art, some CPU scheduling algorithms are selected

* Corresponding author.
** Principal corresponding author.
E-mail addresses: migonrod@gmail.com (M. González-Rodríguez), locerdeira@uvigo.es (L. Otero-Cerdeira), nrufino@uvigo.es (E. González-Rufino), franjrm@uvigo.es (F.J. Rodríguez-Martínez).

https://doi.org/10.1016/j.heliyon.2024.e29959
Received 12 July 2023; Received in revised form 14 April 2024; Accepted 18 April 2024

Available online 24 April 2024

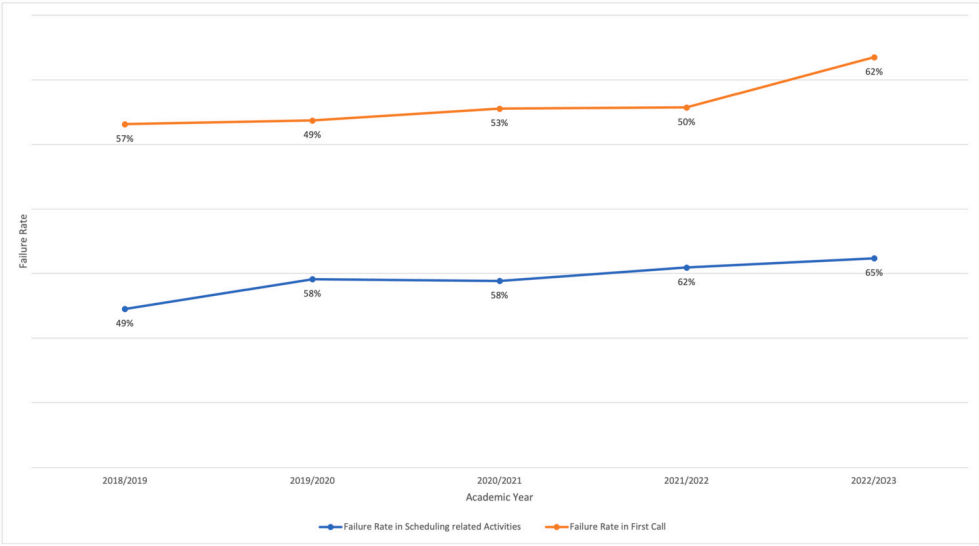


Fig. 1. Evolution of failure rate in the past years.

for further examination of their characteristics and performance. This will help determine the most interesting and appropriate ones for the subject, and they will be studied from a theoretical, practical, and comparative point of view.

2. Background and context

In this section, we review some fundamental concepts and classic scheduling algorithms that provide background information for this article. They also serve to contextualize the evaluations and comparisons presented in Section 4.

Programming languages allow the development of programs in which a series of activities are specified. The execution of these activities is known as a process, and the operating system is responsible for its management, making it one of the most crucial components. The operating system must efficiently handle tasks such as resource allocation, inter-process communication, control of data structures, and process execution. Among these tasks, one important aspect is the control of process execution itself. The operating system needs to employ an algorithm to determine which process will utilize the CPU and proceed with its execution. This is the purpose of CPU scheduling.

As hardware has advanced, operating systems have also adapted to leverage improvements in processing power. For instance, modern computers often feature multi-core processors, allowing for some degree of parallelism in process execution, as shown in [1]. Mobile devices, on the other hand, incorporate specialized processors to enhance performance and battery life.

Operating systems have had to evolve to meet the growing needs of users engaged in increasingly diverse activities. As a result, CPU scheduling algorithms, as an integral part of operating systems, must also evolve. Therefore, it is crucial for computer engineers to be familiar with classical processor scheduling algorithms. Additionally, staying informed about new proposals and algorithms integrated into current devices is equally important.

2.1. Process concept

An operating system has the task of resource management. One of the most crucial resources in a computer system is the CPU (Central Processing Unit), which is responsible for executing program instructions. In addition to the program's source code, there is a set of associated data. The sequence of actions or activities resulting from the execution of a predefined set of instructions is what we refer to as a process.

With this in mind, we can define a process as an instance of a running program.

From its creation to its termination, a process can go through three different basic states:

- **Active:** the process is using the CPU.
- **Ready:** the process is executable but another process is using the CPU. The operating system maintains a list that points to all processes with this state, the list of processes in the ready state. This will be sorted according to some criteria, depending on the CPU scheduling algorithm being used by the operating system.
- **Blocked:** the process is not executable because it is waiting for a certain event to occur, usually an I/O operation.

Therefore, a process may require the CPU as a resource. Fig. 2 shows how transitions between states of a process can occur.

To determine which process in the ready state will utilize the CPU and continue its execution, the operating system employs a process scheduler.

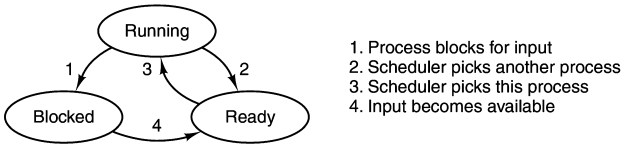


Fig. 2. Transitions between process states [2].

2.2. Objectives of CPU scheduling algorithms

Selecting or comparing scheduling algorithms is a highly complex task because different operating systems may be tailored for specific uses or environments (such as batch processing, general-purpose, or real-time). As a result, there is no singular criterion to evaluate the quality of scheduling. However, we can assert that the primary objective of short-term scheduling is to optimize a particular aspect of the system when assigning the CPU to a process. Stallings [3], lists some key criteria in short-term planning, indicating that it is impossible to optimize all of them simultaneously.

- *User-oriented*
  - **Turnaround Time (TT):** The time that elapses between the launching of a process and its termination.
  - **Response Time (RT):** In interactive processes, it is the time that elapses between launching a request and starting to receive the response. It is usually limited by the output device.
  - **Waiting Time (WT):** It is the sum of all the periods in which the process is in the queue of ready processes.
  - **Deadlines:** When it is possible to specify it, the number of deadlines that can be achieved should be maximized.
  - **Predictability:** Time required and the cost for a job should be approximately the same, regardless of the system load.
- *System-oriented*
  - **Throughput:** measure of work performed, which is equal to the number of processes completed per unit of time. It is also called processing rate.
  - **CPU utilization:** percentage of time the CPU is busy. It is desirable for it to be as busy as possible.
  - **Fairness:** processes should be treated equally, and none should suffer starvation. A process is said to suffer from starvation when it is not possible to limit the waiting time for the process to be assigned a resource. Therefore, we say that a scheduler is fair when it grants each process a fair share of the CPU.
  - **Priority imposition:** in the case where processes have different priorities, the scheduler should favor those with higher priorities.
  - **Resource balancing:** processes that under-utilize resources that are currently over-utilized should be favored.

As a general rule, the objective of scheduling is to maximize CPU utilization and processing rate while minimizing execution time, waiting time, and response time [4].

2.3. Important concepts of scheduling algorithms

In the following section, we will discuss some concepts related to processes and their scheduling. This will aid in our understanding of how CPU scheduling algorithms function.

Process behavior. Processes usually alternate between computation time (CPU bursts) and I/O requests, in which the process blocks while waiting for an external device to complete its work.

Depending on the investment of their time between these two actions, we can classify processes into two types:

1. CPU-bound processes: They invest most of their time performing computations, with long CPU bursts.
2. I/O-bound processes: They spend most of their time waiting for I/O operations.

Nowadays, in interactive operating systems, processes are mostly I/O-bound, so their scheduling becomes more important [2].

Appropriative and non-appropriative scheduling. There are five circumstances in which the scheduler must make a decision on which process will occupy the CPU [4]:

1. When a process that was running goes to the blocked state, for example, due to an I/O request or other event.
2. When a process that was running goes to the ready state, such as upon completion of the maximum time allowed to make use of the CPU.
3. When a new process is added to the list of processes in the ready state.
4. When a process that was blocked enters the ready state, for example due to the completion of an I/O operation.
5. When a process terminates.

In the first and last case, the scheduler must simply select the next ready process to grant it the CPU. However, depending on the scheduler's possibility of action in the other three cases, we will have two types of scheduling:

- In a non-appropriative (non-preemptive) schedule, the currently running process would continue to run until it terminates or blocks.

- In a preemptive schedule, the scheduler may decide that another process should take the CPU, so it could eject the currently running process and place it in the ready list.

Priorities. In a system, there may be several processes performing different tasks, which may be of greater or lesser importance to the user at any given time. For the scheduler, the processes are equal entities, since he does not know which task each one represents. The mechanism used for the scheduler to take this factor into account is priorities.

In a priority-based scheduling algorithm, each process is assigned a qualifier that determines the attention it will receive from the system. The qualifier is usually a number within a range defined by the system, which is called the process priority. It also defines whether the higher the priority value, the more important the process is, or the lower the priority value, the more important it is. When choosing the next process to be executed, the scheduler will choose the ready process with the highest priority.

Depending on the system, priorities can be assigned by two different agents:

- The operating system is the one that assigns the priorities (internal priorities).
- Users, other processes, etc., can assign priorities (external priorities).

When using priorities, the problem can arise that processes with lower priorities are never able to use the CPU, resulting in a case of starvation. This occurs in operating systems that do not change the priority value of the process since its creation (static priorities). The solution is to vary the priority of processes during their execution, for example, according to whether they have used the CPU or if they have been waiting for a long time (dynamic priorities) [3].

Some operating systems group processes according to their orientation (e.g., CPU-bound or I/O-bound). In these cases, a priority class is associated with them. For each of these classes, the operating system can use a different process scheduling [2].

Quantum. Some operating systems do not want the same process to hog the CPU indefinitely, so they check from time to time whether there is a higher-priority process that should be given the CPU. This granted time is called quantum (timeslice).

The quantum can be fixed (generally, the operating system assigns it based on the system clock) or variable during the execution of the process, and even different for each process [3].

As we will see later, a key element in the design of many algorithms is the length of the quantum, and many studies focus on selecting the optimal one for each case.

2.4. Parameters for performance assessment

When determining the most suitable scheduling algorithm, we utilize a range of parameters, some of which have been previously discussed in the criteria section, to assess their effectiveness. Below, we elaborate on the parameters that are commonly employed [5]:

- Burst Time (BT): All processes need a certain amount of time to run, in which the computation time and the time consumed by I/O operations are added together. When performing studies and comparisons of scheduling algorithms, the I/O time is generally ignored and only the CPU time required by the process is used.
- Arrival Time (AT): Moment in the course of time at which a process enters the list of processes ready to take the CPU.
- Completion Time (CT): Time at which the process finishes its execution and exits the system.
- Turnaround Time (TT): Time that elapses between the launching of a process and its termination. TT = CT - AT.
- Waiting Time (WT): The sum of all the periods in which the process is in the queue of ready processes. WT = TT - BT.
- Response Time (RT): In interactive processes, it is the time that elapses between the launching of a request and the start of the response. It is usually limited by the output device.

2.5. Classic algorithms

This section provides a brief overview of the key CPU scheduling algorithms. These algorithms serve as the foundation for scheduling in research studies and current operating systems. Additionally, they form the basis for teaching process scheduling in operating systems courses. While only the classical variants of these algorithms are presented here, alternative versions are often proposed in educational settings solely for theoretical purposes, as they have no practical use.

First Come, First Served: The First In, First Served (FCFS or FIFO - First In, First Out) algorithm is the simplest, and follows a strict queuing system. The list of processes in ready state is ordered according to the arrival of the process in the system, with the oldest process at the head, which will be the first to be allocated the CPU.

A version of this algorithm selects the next process based on its arrival in the list of ready processes. When the scheduler must select a process to grant it the CPU, it chooses the process that has been on this list the longest [4].

The FCFS algorithm is not preemptive and does not use quantum or priorities.

Shortest Job First: The Shortest Job First (SJF) algorithm associates to each process the duration of its next CPU burst, to allocate the available CPU to the ready process with the shortest CPU burst [4].

Some authors differentiate this algorithm from the Shortest Process Next (SPN), in which what is associated is the Burst Time of the process, selecting also the shortest one.

The SJF algorithm is not preemptive and does not use quantum or priorities.

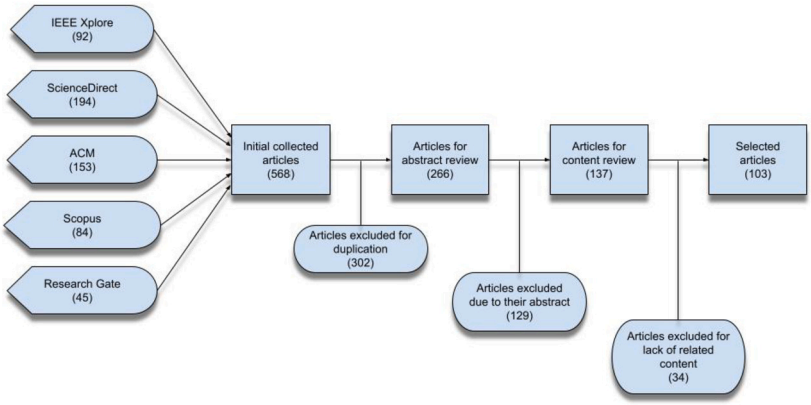


Fig. 3. Procedure followed to obtain the articles for the review.

Shortest Remaining Time First: The Shortest Remaining Time First (SRTF) algorithm is an appropriate version of SJF. The system must first estimate the processing time required by each process, for example, by keeping an average of its CPU burst time. In case a process is received whose remaining execution time is less than that of the currently executing one, it would have to be ejected from the CPU to be given to the new process [3].
Therefore, the SRTF algorithm is preemptive, but does not use quantum or priorities.

Round Robin: Round Robin (RR) takes advantage of the quantum to reduce the penalty caused by the FCFS algorithm to the shortest processes. This algorithm also uses a circular queue for ready processes. When the running process exhausts its quantum, the scheduler places it at the end of the queue of ready processes, and assigns the CPU to the first in the queue.
The RR algorithm is preemptive and uses quantum, but has no priorities.

Multi-Level Queues: The Multi-Level Queuing (MLQ) algorithm first assigns a priority to each process. It uses a data structure consisting of several queues; ready processes with the same priority are stored in each queue. When allocating the CPU, the first process in the highest priority queue that is not empty will be chosen [4].
This algorithm is preemptive, and uses both quantum and priorities.

Multi-Level Feedback Queues: In the MLQ algorithm, processes have the same priority throughout their runtime, so they always enter and leave the same queue. This is a drawback as it can lead to starvation of processes in the last queues. In the Multi-Level Feedback Queues (MLFQ) algorithm, the priority of a process can vary, so it can move between different queues, reflecting changes in the behavior of the processes. For example, if a CPU-limited process starts to perform many I/O operations, its priority will be increased and it will move up the queues. Conversely, if an I/O-limited process starts to perform a lot of computation, it will decrease its priority and move down the queues [4].
This algorithm is preemptive, and uses both quantum and priorities.

Earliest Deadline First: In real-time systems, absolute time limits are set that must be met (hard real-time) or at least must not generally be violated (soft real-time). One of the algorithms used in real-time systems is the Earliest Deadline First (EDF) algorithm. In this algorithm, when a process is created, the system assigns it a priority based on its deadline, with the process that is closest to its deadline having the highest priority [3].
This algorithm is preemptive, and uses both quantum and priorities.

3. Literature review on CPU scheduling

This section is dedicated to evaluating trends in the field of process scheduling with the goal of identifying current interests within the domain and potential algorithms that could enhance the teaching of operating systems courses. We will describe the process of collecting diverse data points and extracting statistical values, which will serve as the foundation for identifying contemporary trends, as demonstrated in the section 4.

3.1. Procedures

To compile articles for this literature review, we conducted searches across various online databases to retrieve those relevant to the field of CPU scheduling. Initially, we retrieved over 500 articles, which then underwent a meticulous filtering process to narrow down the selection to the final 103 articles that were included in this review. The screening process involved manual examination and spanned over a period of more than two months due to multiple iterations and the substantial number of articles reviewed. The procedure employed to identify and filter the publications is illustrated in Fig. 3. It is important to note that this process exclusively considered open-access journal articles and conference papers, as these represent the primary avenues through which researchers disseminate their research findings. Consequently, other forms of publications, such as books, newspapers, doctoral dissertations, essays, etc., were not included.

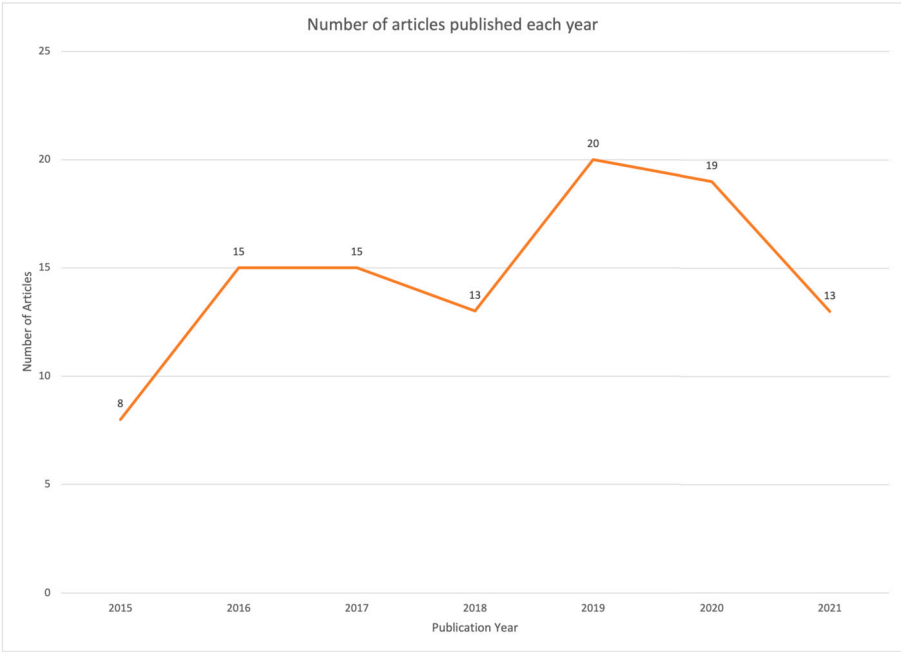


Fig. 4. Number of articles published each year.

To explore the latest advancements in CPU scheduling, we conducted searches in several scientific research databases. Specifically, we queried the following search engines: *IEEE Xplore* [6], *ACM Digital Library* [7], *Scopus* [8] and *Research Gate* [9].

We used the following search strings to query these databases: *CPU Scheduling*, *Scheduling Algorithm*, *Process Scheduling*, *Job Scheduling* and *Task Scheduling*. Additionally, we applied a time interval to limit the articles to those published between 2015 and 2021.

When sorting the data sources by the number of articles obtained, we found the following counts: Science Direct (194), ACM Digital Library (153), IEEE Xplore Digital Library (92), and Scopus (84), totaling 523 articles initially obtained. The next step involved removing duplicated articles that were obtained through two or more databases, resulting in the elimination of 302 articles.

Subsequently, we analyzed the remaining articles to exclude those unrelated to CPU Scheduling. This analysis took into consideration the articles' keywords and abstracts. Any articles that did not specifically mention the CPU scheduling field in their keywords or did not address research related to this field in their abstracts were excluded. For example, articles like *Characterizing and Optimizing the Performance of Multithreaded Programs Under Interference* [10] or *Olympian: Scheduling GPU Usage in a Deep Neural Network Model Serving System* [11] were excluded. By doing so, we dismissed 129 articles.

The 137 remaining articles underwent a careful review to exclude those that were less relevant to the research's purpose. For instance, *Hybrid CPU-GPU scheduling and execution of tree traversals* [12] was among those excluded. As a result, 103 articles remained for this study. The complete list of the 103 reviewed articles can be found in the final annex 6.

3.2. Statistical results

This section presents statistical results and assessments of the articles collected for the review. Before delving deeper into the publications, we will begin with an initial analysis by examining the publication dates, as illustrated in Fig. 4.

The graph illustrates a growing interest in CPU scheduling within the scientific community in recent years. In 2015, only 8 articles were published, but this number nearly doubled in the subsequent three years. The years 2019 and 2020 recorded the highest number of publications, with 20 and 19 articles, respectively. However, there was a slight decline to 13 articles in 2021. It is important to note that this decrease could potentially be attributed to the effects of the pandemic, so it might be premature to interpret it as a new trend at this stage.

Analyzing the selected articles provides us with several insights into the future of the CPU scheduling field.

First, we can classify the publications into one of the three following main categories:

- *New versions of CPU scheduling algorithms or policies*. These works primarily focus on proposing new CPU scheduling algorithms or enhancing previously established ones, as demonstrated in Section 3. For instance, works like [13] and [14] fall under this category.
- *Performance comparisons between various CPU algorithms.* These papers aim to present performance comparisons of different scheduling algorithms using various scenarios or simulation tools, as exemplified by [15].

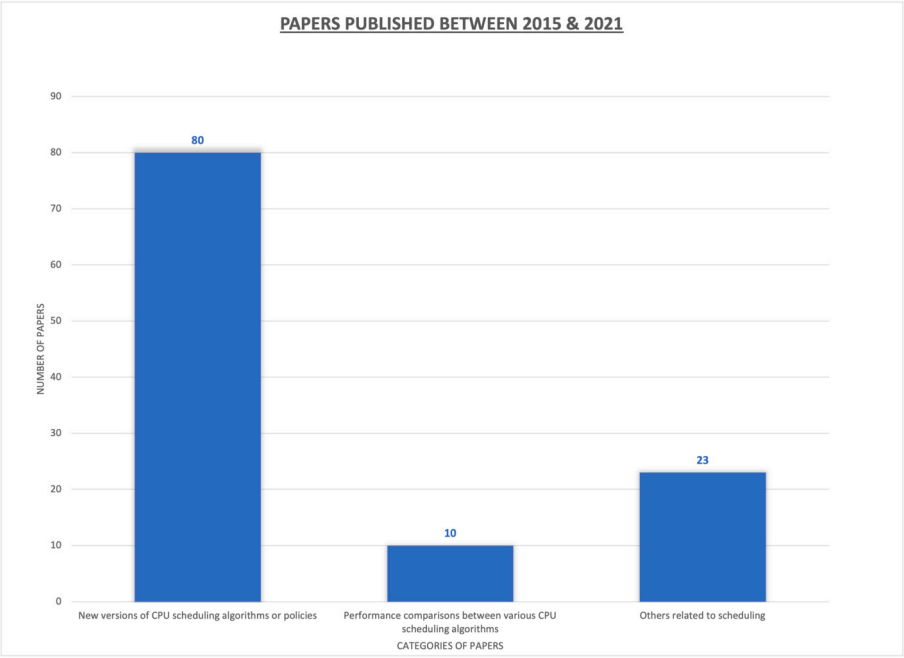


Fig. 5. Contents of analyzed publications.

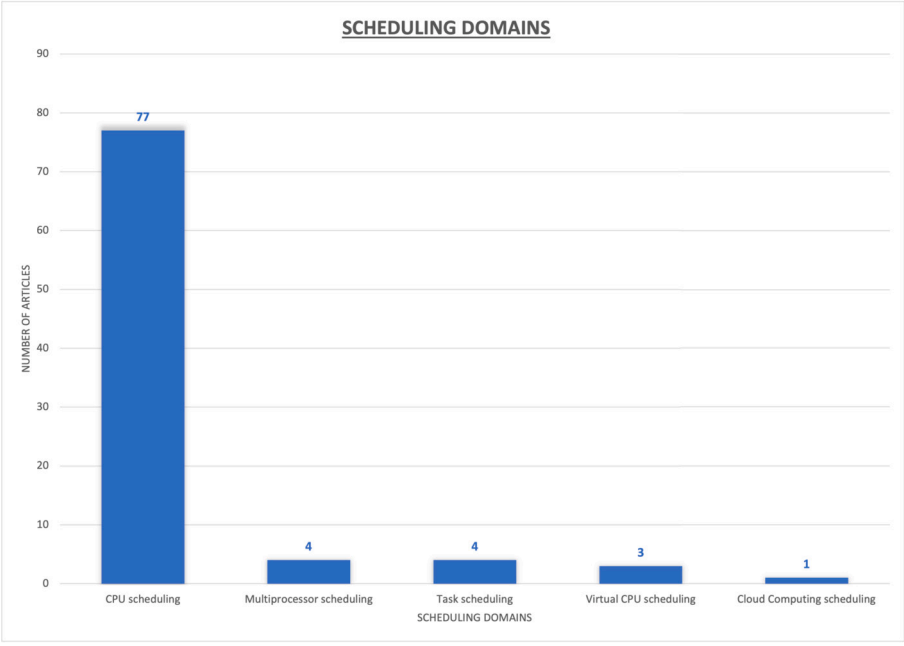


Fig. 6. Types of scheduling in the collected publications.

- *Others related to scheduling.* These articles may discuss concepts related to scheduling without making specific algorithmic proposals. An example is the paper titled *Comparative Analysis of CPU Scheduling Algorithms: Simulation and Its Applications* by [16].

In this regard, the majority of publications, specifically 71%, are articles that introduce new CPU scheduling algorithms, as illustrated in Fig. 5.

Continuing with the analysis, we have observed that the collected publications encompass papers on scheduling across various domains, as depicted in Fig. 6. While the majority of the reviewed papers, specifically 87%, are focused on CPU scheduling, others have

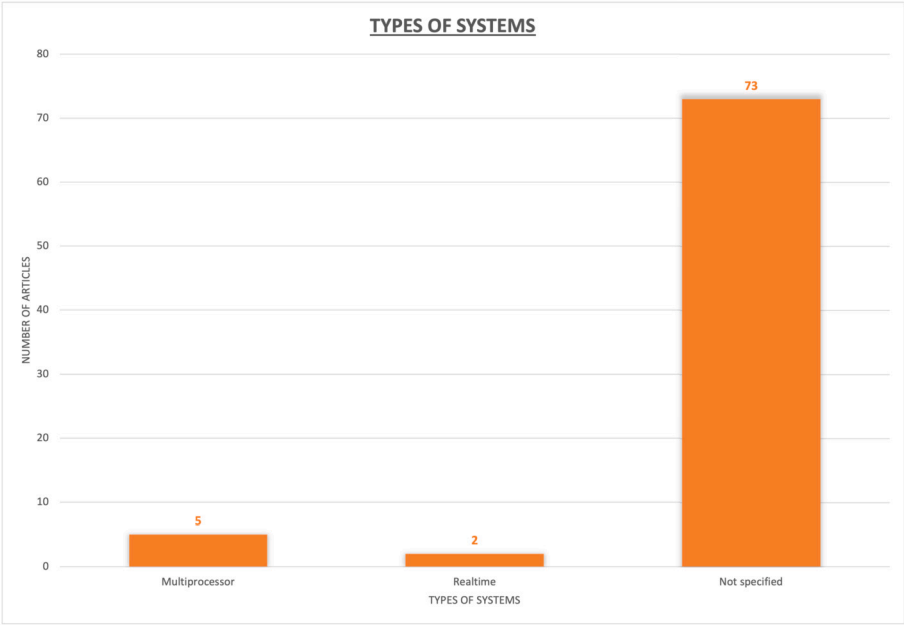


Fig. 7. Types of systems in the collected publications.

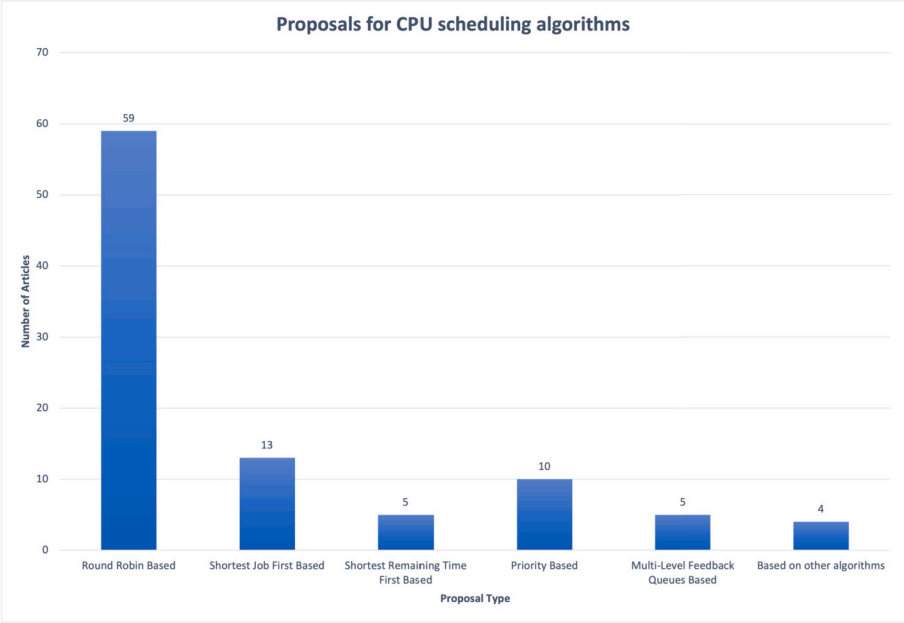


Fig. 8. Reference algorithms in the collected publications.

been included to assess whether they propose new algorithms that could be applicable. For instance, the paper by [17] introduces a novel algorithm for task scheduling in heterogeneous systems, which could potentially be adapted for process scheduling.

In 91% of the collected publications, the specific operating system in which the scheduling is implemented is not specified, with the assumption being that they pertain to multiprogrammed systems, typically with a single processor. However, a subset of studies addresses multiprocessor systems (6%) or focuses on real-time systems (3%), as illustrated in Fig. 7.

With regard to the proposals for new CPU scheduling algorithms, we have categorized the studies based on the classical algorithms upon which they are built, yielding the results depicted in Fig. 8. It's worth noting that an algorithm can draw inspiration from multiple classical algorithms simultaneously. Among the articles studied in this research, 57% of the new algorithms are rooted in the classical Round Robin (RR) algorithm. In these specific papers, we observe that many of these proposals revolve around determining the most suitable time quantum.

We also observe a notable number of algorithms that draw inspiration from the Shortest Job First algorithm and its Shortest Remaining Time First variant. To be precise, 17% of the publications are associated with either of these two algorithms. After reviewing all these findings, we can confidently assert that current research predominantly centers on making incremental refinements to classical algorithms with the aim of enhancing CPU scheduling

3.3. Limitations

Conducting a literature review in the field of CPU scheduling presents significant challenges due to its extensive history and the broad scope of research within this domain. In this study, we gathered articles through online searches, utilizing various search terms and applying specific content-based criteria. While our initial collection exceeded 500 articles, and we conducted in-depth analyses of over 100, it's important to acknowledge the possibility of inadvertently omitting relevant articles.

While the databases we consulted are well-recognized and frequently used by researchers, we acknowledge that other databases might have yielded additional results. Furthermore, our search scope was limited to articles written in English, which is the predominant language in scientific research. As a result, potentially valuable papers in other languages may not have been included in this review.

Despite these limitations, this paper offers an extensive review of articles published between 2015 and 2021 in the realm of CPU scheduling. We have meticulously scrutinized and analyzed articles within this timeframe, allowing us to identify various research trends and highlight specific areas of interest among researchers in recent years. This analysis helps uncover emerging trends and novel algorithms, enriching the educational resources available for teaching this subject.

4. Newest trends for CPU planning

As a result of our literature review, it is evident that the majority of the collected publications focus on classical algorithms and their modifications, all aimed at improving practical experimental results. In this section, we will highlight some of the noteworthy proposals that have emerged from this extensive review.

To spotlight the most exceptional examples, we filtered algorithms based on two well-established classics: Round Robin and Shortest Job First. We made this choice due to the significant number of algorithms rooted in one of these two classics. Additionally, for algorithms based on Round Robin, we recognized the critical importance of selecting an appropriate time quantum, a parameter central to the Round Robin scheduling algorithm. The time quantum plays a pivotal role in determining how CPU time is allocated among different processes in a multitasking environment. The selection of the right time quantum in Round Robin scheduling depends on specific system requirements. Typically, time quantum values are set to be sufficiently small to ensure responsiveness and prevent process starvation, yet large enough to mitigate excessive context switching. The decision on an appropriate time quantum can be influenced by factors such as workload nature and desired system performance.

At this stage of our research, our initial selection comprises four algorithms, chosen to establish the study's structure and utility. Furthermore, as discussed in Section 4.2, when comparing and evaluating these selected algorithms and to ensure the utmost objectivity, we implemented a simulator. The algorithms chosen for this initial assessment were those that provided sufficient information in their respective articles for correct implementation and subsequent performance evaluation.

In the following section, we will provide a comprehensive review of the latest trends and highlight some of the most promising algorithms identified during our review. This review will be presented using specially created study sheets tailored to the purpose of this work. Following this review, we will present a comparison and evaluation of the described algorithms.

4.1. Studied algorithms

After selecting the algorithms, we developed a set of standardized study sheets, one for each algorithm. These study sheets serve as valuable tools for collecting essential information about each algorithm, streamlining the review process, facilitating a deeper understanding of the algorithms, enabling straightforward comparisons, and assisting in their evaluation.

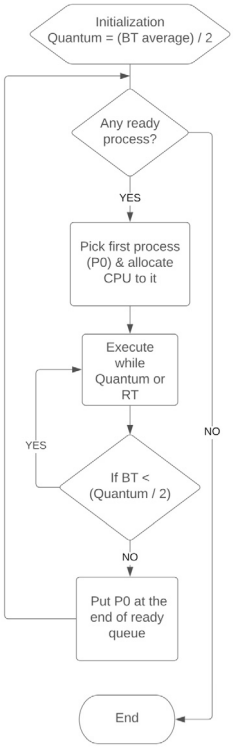
Each study sheet contains essential information to provide a quick and comprehensive overview of the algorithm, ensuring easy comprehension and initial assessment. It includes a concise summary of how the algorithm functions, a visual flowchart illustrating the algorithm's steps, and a table summarizing its key characteristics. This table offers insights into whether the algorithm is suitable for use, whether it employs a time quantum, priorities, and/or multiple queues. It also specifies whether the article provides an implementation of the algorithm and whether it is tailored for a specific operating system.

By utilizing these study sheets, the process of comprehending, comparing, and evaluating the algorithms becomes significantly more efficient and manageable.

Below is a list of the algorithms featured in this work, accompanied by their respective study sheets.

1. Round Robin based algorithms
   - *Augmented Dynamic Round Robin* [18]
   - *Modified Median Round Robin Algorithm* [19]
2. SJF based algorithms
   - *Factor K* [20]
   - *Multilevel Dynamic Feedback Queues* [21]

Augmented Dynamic Round Robin



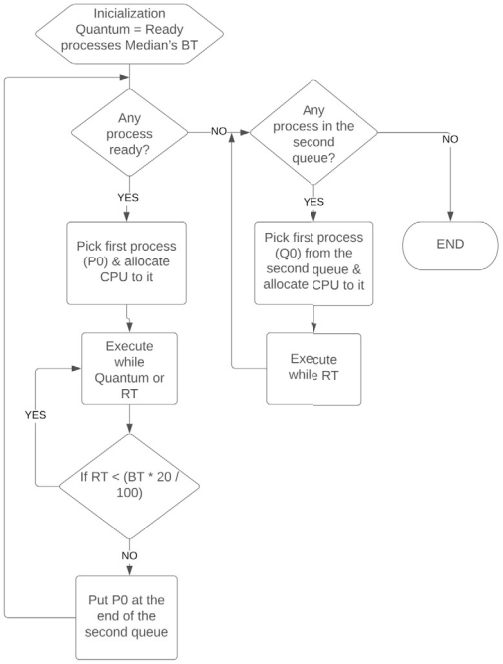
This algorithm is based on the traditional Round Robin, but the decision on the next process to be executed depends on the estimated execution times, favoring the reduction of the number of context switches.

When a running process exhausts its quantum, if its remaining time is less than half of the quantum, it can continue to run.

The algorithm has been designed to perform better than the classical Round Robin even with different static quantum times.

Appropriative	Yes
Quantum	Fixed
Priorities	No
Multiple Queues	No
Implemented	No
O.S	No

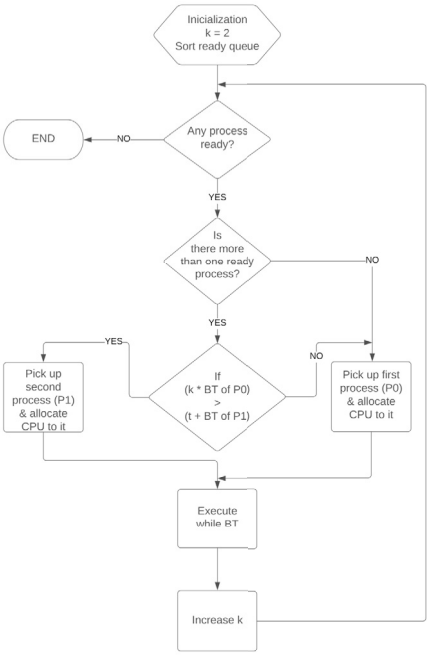
Modified Median Round Robin Algorithm



This algorithm is based on Round Robin and its main objectives are to improve the average response time and the number of context switches of other versions of Round Robin. The main modification is the calculation of quantum, which is obtained from the median execution time of the processes in the queue of processes in the ready state. In addition, processes that run out of quantum and have much more execution time pending (more than 20%), are moved to a second queue where they wait until the original ready queue has been emptied.

Appropriative	Yes
Quantum	Fixed
Priorities	No
Multiple Queues	Yes
Implemented	No
O.S	No

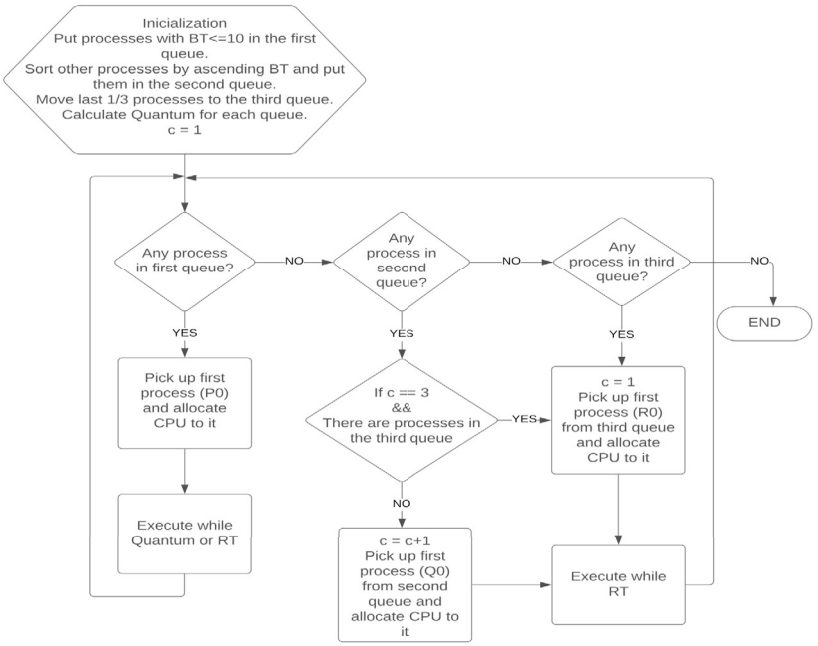
K-FACTOR



The main objective of this algorithm is to provide an alternative to SJF that minimizes its most significant disadvantages. Specifically, it seeks to treat all processes fairly, trying to avoid starvation of the processes with the longest execution time. For this purpose, a variable (K-Factor) is used, which is modified over time, and serves to evaluate which will be the next process to be executed. The results obtained in the tests do not show that this algorithm is better than SJF in terms of average TT and AT, but it does modify the selection of processes to make it a little fairer.

Appropriative	No
Quantum	No
Priorities	No
Multiple Queues	No
Implemented	No
O.S	No

Dynamic Multilevel Feedback Queues



Multiple queue implementation, combining Round Robin and SJF to select processes. It consists of a queue for the processes with the shortest execution time, which will be executed first and through Round Robin. The others are distributed in two other queues, using SJF to select the next process. To prevent starvation problems of the processes with the longest execution time, the CPU is granted to one of them 1 out of 3 times.

Appropriative	Yes
Quantum	Fixed
Priorities	Yes
Multiple Queues	Yes
Implemented	No
O.S	No

4.2. Evaluation and comparison

Evaluating the performance of various proposals does not permit a direct comparison based solely on the results provided in the original works. This is because the proposed environments often exhibit uniqueness and specificity in each case. Therefore, a pivotal aspect of our work involves verifying whether the improvement objectives articulated in the research are genuinely achieved.

In all the studies we've gathered, the authors affirm the validity of their algorithms based on tests they conducted themselves. However, instead of solely relying on their conclusions, we have opted to independently implement and execute these algorithms. This approach allows us to make more effective and accurate comparisons, as the evaluation environment remains consistent throughout the assessments.

Furthermore, these tests complement the information found in the study sheets, enabling us to objectively contrast these new algorithms with the classic ones they build upon. This comparison facilitates the evaluation of algorithm performance based on the parameters outlined in Section 2.4.

To accomplish this objective, we developed a Java application to simulate the operation of a CPU and the process scheduler, emulating the behavior of an actual operating system. This application allows us to generate processes within a system and have the scheduler manage their CPU time allocations according to the commands of the various algorithms previously described.

There are other simulators in the field of process planning; some, such as *CPU-OS Simulator* [22] or *AnimOS CPU-Scheduling*[23], are comprehensive tools. In the case of *CPU-OS Simulator*, it is a complex tool that combines a CPU simulator and an OS simulator. In this context, both simulators work together, forming a robust tool that facilitates the simulation of diverse scheduling algorithms, showcases program execution behavior, observes the progression of queues in ready and blocked states, and explores elements like the utilization of CPU registers and their content. Despite the tool's substantial capabilities and the results it produces, it does not facilitate the retrieval of comparative data for the execution of different algorithms. The same limitation applies to the *AnimOS CPU-Scheduling* tool. The latter is a tool that enables real-time and graphical simulation of different processes' behavior based on various classical scheduling algorithms and their variants. It provides instant metrics, both general and specific to each process. Despite the insightful information offered by the obtained metrics, the tool's primary purpose is to simulate the behavior of a specific algorithm, rather than comparing the performance of different algorithms.

While these options are highlighted, others are described in [24], [25], [26], [27], [28]. The shared characteristic among all these works is their focus on simulating diverse scheduling algorithms, rather than comparing the performance of different algorithms against each other. Furthermore, in some of these cases, although the tool and its results are outlined, the software itself remains inaccessible.

Further details about the functioning of this simulator are elaborated in the following Section 4.2.1.

4.2.1. Simulator

Within the developed simulator, users have the capability to specify the number of processes they wish to create for execution, along with their estimated maximum execution times (BT) and maximum arrival times to the operating system (AT). The application takes this input and generates the specified number of processes, randomly assigning BT and AT values to each one while ensuring that these values do not exceed the user-defined maximum limits. In cases where algorithms utilize a time quantum, this quantum is generated based on the average burst time of the processes present in the ready queue. This approach ensures that the simulator effectively replicates real-world scenarios.

The class diagram shown in Fig. 9 illustrates the Java class structure implemented for this simulator, including its structure, methods, and established relationships. The main class, *App*, allows for the configuration and initiation of the simulation for the various types of algorithms implemented.

Fig. 10 shows the application's interface where the user has entered the requested data.

The application simulates the scheduling of the generated processes using each of the algorithms mentioned in the study sheets. Additionally, the classic Round Robin and Shortest Job First algorithms have been implemented to facilitate comparisons with the new algorithms. Upon completion of a process, the program collects a set of data that can be utilized to evaluate the efficiency of the algorithm. Fig. 11 shows the results obtained in one program execution.

When a process has completed, the program has collected a set of data that we can use to evaluate the efficiency of the algorithm. It is possible to modify the program's code to display these details for each process, as we can see in Fig. 12.

4.2.2. Testing conditions

For each algorithm, two different tests are conducted. In the first test, all processes are created at the initial execution time (AT = 0). The estimated execution time (BT) of each process is randomly generated within the range of 1 to 100 ms. Each algorithm is tested on systems with 100, 500, and 1000 running processes.

In the second test, the same conditions are applied to the system and estimated execution time, but a random arrival time is generated for each process within the range of 0 to half the number of processes. This ensures varying arrival times for the processes.

These two tests provide different scenarios to evaluate the performance and efficiency of each algorithm under different load and arrival time conditions.

Once the processes are executed using an algorithm, the program calculates the values of the following parameters:

- Average execution time of all processes (Mean TT; **MTT**).
- Average waiting time of all processes (Mean WT; **MWT**).
- Number of times the scheduler must take CPU away from a process to give it to a different one, either because the running process has terminated or because its quantum has been terminated (Number of context switches) (**NCS**).
- Number of times the operating system must sort a list according to the criteria decided by the scheduling algorithm (Number of sorts)(**NS**).

The first two parameters, which are commonly used in the studies we have collected, serve as metrics to evaluate and compare the algorithms. Lower values indicate higher efficiency among the algorithms.

The last two parameters are employed to simulate the overhead generated when the operating system performs various operations unrelated to the actual execution of processes. It is intuitive that minimizing these operations reduces the overhead and improves overall efficiency.

By considering these parameters, we can effectively assess the performance of the algorithms and compare their efficiency in terms of execution time, overhead reduction, and overall effectiveness.

However, it is worth noting that the results may be influenced by the choice of input parameters such as burst time (BT), arrival time (AT), and time quantum, as these parameters significantly impact the performance and outcomes of various algorithms. The selection of these parameters can affect the behavior and results of the simulation, thus influencing the comparison of different

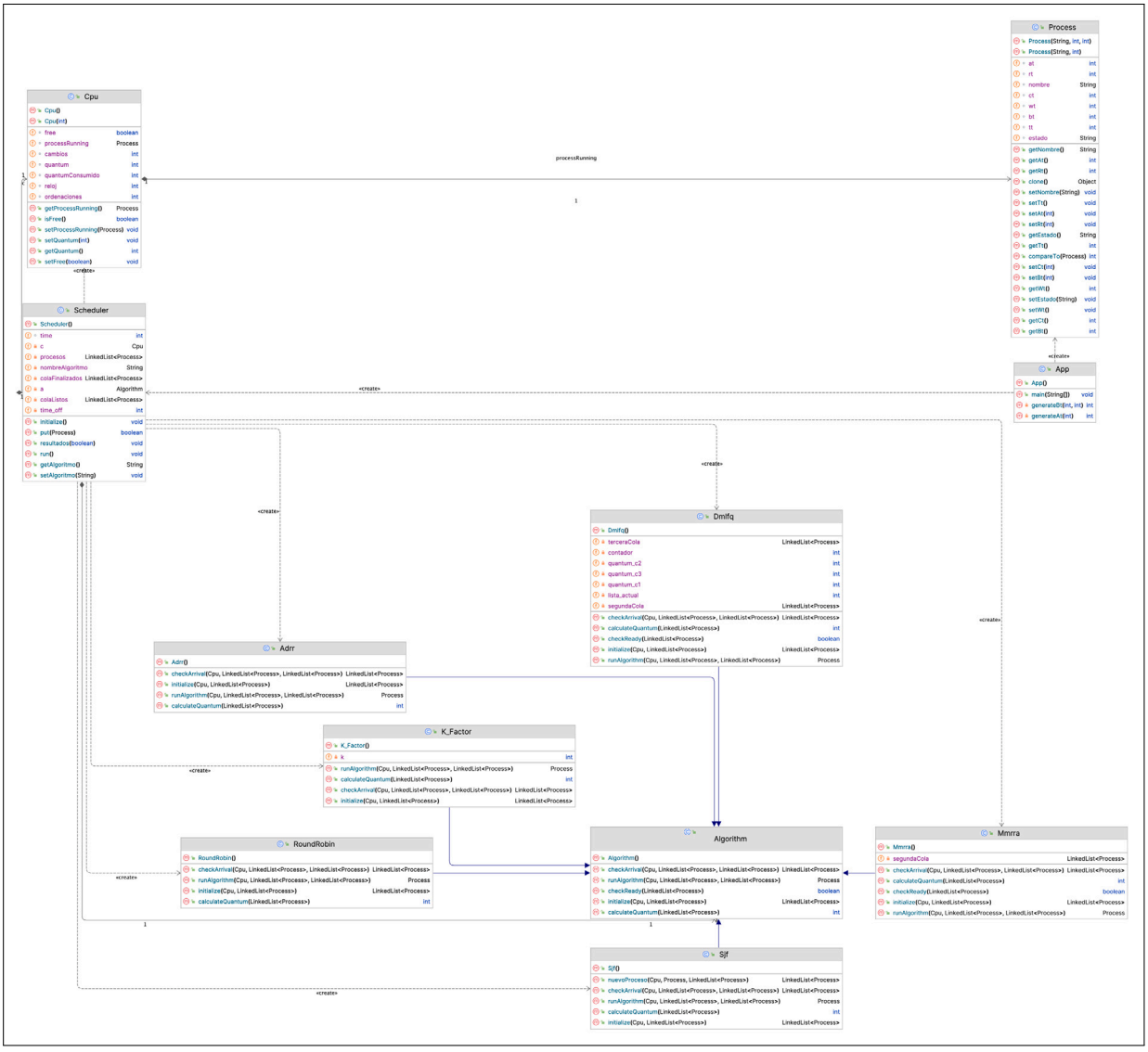


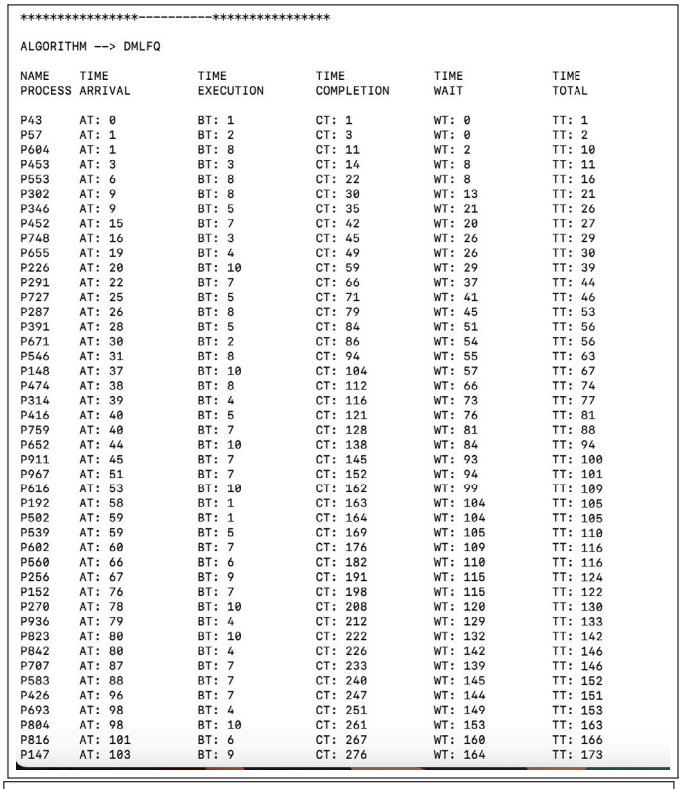
Fig. 9. Simulator Class Diagram.

```
run:

Enter the number of processes:1000

Enter the maximum execution time:100

Enter the maximum arrival time:250

CREATING PROCESSES

STARTING SIMULATION
```

Fig. 10. Data Input Interface in the Simulator.

```
i301@iMac-de-Despacho301 dist % java -jar Simulador.jar

Enter the number of processes:1000

Enter the maximum execution time:100

Enter the maximum arrival time:250

CREATING PROCESSES

STARTING SIMULATION

ALGORITHM --> RR


Average TT = 31590.057
Average WT = 31539.32
Number of Context Switches = 1395
Number of times the queue has been sorted = 0

**********----------**********

ALGORITHM --> ADRR


Average TT = 26484.8
Average WT = 26434.115
Number of Context Switches = 1138
Number of times the queue has been sorted = 0

**********----------**********

ALGORITHM --> MMRRA


Average TT = 28725.123
Average WT = 28674.414
Number of Context Switches = 1643
Number of times the queue has been sorted = 0

**********----------**********

ALGORITHM --> SJF


Average TT = 16973.18
Average WT = 16922.484
Number of Context Switches = 1001
Number of times the queue has been sorted = 246

**********----------**********

ALGORITHM --> KFACTOR


Average TT = 16988.18
Average WT = 16937.482
Number of Context Switches = 1001
Number of times the queue has been sorted = 246

**********----------**********

ALGORITHM --> DMLFQ


Average TT = 16979.623
Average WT = 16928.928
Number of Context Switches = 1001
Number of times the queue has been sorted = 246

**********----------**********
```

Fig. 11. Example of Algorithm Simulator Execution.



```
**********----------**********

ALGORITHM --> DMLFQ

NAME    TIME            TIME            TIME            TIME            TIME
PROCESS ARRIVAL         EXECUTION       COMPLETION      WAIT            TOTAL

P43     AT: 0           BT: 1           CT: 1           WT: 0           TT: 1
P57     AT: 1           BT: 2           CT: 3           WT: 0           TT: 2
P604    AT: 1           BT: 8           CT: 11          WT: 2           TT: 10
P453    AT: 3           BT: 3           CT: 14          WT: 8           TT: 11
P553    AT: 6           BT: 8           CT: 22          WT: 8           TT: 16
P302    AT: 9           BT: 8           CT: 30          WT: 13          TT: 21
P346    AT: 9           BT: 5           CT: 35          WT: 21          TT: 26
P452    AT: 15          BT: 7           CT: 42          WT: 20          TT: 27
P748    AT: 16          BT: 3           CT: 45          WT: 26          TT: 29
P655    AT: 19          BT: 4           CT: 49          WT: 26          TT: 30
P226    AT: 20          BT: 10          CT: 59          WT: 29          TT: 39
P291    AT: 22          BT: 7           CT: 66          WT: 37          TT: 44
P727    AT: 25          BT: 5           CT: 71          WT: 41          TT: 46
P287    AT: 26          BT: 8           CT: 79          WT: 45          TT: 53
P391    AT: 28          BT: 5           CT: 84          WT: 51          TT: 56
P671    AT: 30          BT: 2           CT: 86          WT: 54          TT: 56
P546    AT: 31          BT: 8           CT: 94          WT: 55          TT: 63
P148    AT: 37          BT: 10          CT: 104         WT: 57          TT: 67
P474    AT: 38          BT: 8           CT: 112         WT: 66          TT: 74
P314    AT: 39          BT: 4           CT: 116         WT: 73          TT: 77
P416    AT: 40          BT: 5           CT: 121         WT: 76          TT: 81
P759    AT: 40          BT: 7           CT: 128         WT: 81          TT: 88
P652    AT: 44          BT: 10          CT: 138         WT: 84          TT: 94
P911    AT: 45          BT: 7           CT: 145         WT: 93          TT: 100
P967    AT: 51          BT: 7           CT: 152         WT: 94          TT: 101
P616    AT: 53          BT: 10          CT: 162         WT: 99          TT: 109
P192    AT: 58          BT: 1           CT: 163         WT: 104         TT: 105
P502    AT: 59          BT: 1           CT: 164         WT: 104         TT: 105
P539    AT: 59          BT: 5           CT: 169         WT: 105         TT: 110
P602    AT: 60          BT: 7           CT: 176         WT: 109         TT: 116
P560    AT: 66          BT: 6           CT: 182         WT: 110         TT: 116
P256    AT: 67          BT: 9           CT: 191         WT: 115         TT: 124
P152    AT: 76          BT: 7           CT: 198         WT: 115         TT: 122
P270    AT: 78          BT: 10          CT: 208         WT: 120         TT: 130
P936    AT: 79          BT: 4           CT: 212         WT: 129         TT: 133
P823    AT: 80          BT: 10          CT: 222         WT: 132         TT: 142
P842    AT: 80          BT: 4           CT: 226         WT: 142         TT: 146
P707    AT: 87          BT: 7           CT: 233         WT: 139         TT: 146
P583    AT: 88          BT: 7           CT: 240         WT: 145         TT: 152
P426    AT: 96          BT: 7           CT: 247         WT: 144         TT: 151
P693    AT: 98          BT: 4           CT: 251         WT: 149         TT: 153
P804    AT: 98          BT: 10          CT: 261         WT: 153         TT: 163
P816    AT: 101         BT: 6           CT: 267         WT: 160         TT: 166
P147    AT: 103         BT: 9           CT: 276         WT: 164         TT: 173
P411    AT: 150         BT: 96          CT: 45983       WT: 45737       TT: 45833
P415    AT: 186         BT: 96          CT: 46079       WT: 45797       TT: 45893
P117    AT: 190         BT: 96          CT: 46175       WT: 45889       TT: 45985
P419    AT: 9           BT: 97          CT: 46272       WT: 46166       TT: 46263
P672    AT: 12          BT: 97          CT: 46369       WT: 46260       TT: 46357
P868    AT: 16          BT: 97          CT: 46466       WT: 46353       TT: 46450
P263    AT: 34          BT: 97          CT: 46563       WT: 46432       TT: 46529
P65     AT: 99          BT: 97          CT: 46660       WT: 46464       TT: 46561
P963    AT: 126         BT: 97          CT: 46757       WT: 46534       TT: 46631
P120    AT: 157         BT: 97          CT: 46854       WT: 46600       TT: 46697
P303    AT: 175         BT: 97          CT: 46951       WT: 46679       TT: 46776
P146    AT: 180         BT: 97          CT: 47048       WT: 46771       TT: 46868
P293    AT: 215         BT: 97          CT: 47145       WT: 46833       TT: 46930
P34     AT: 39          BT: 98          CT: 47243       WT: 47106       TT: 47204
P444    AT: 46          BT: 98          CT: 47341       WT: 47197       TT: 47295
P994    AT: 64          BT: 98          CT: 47439       WT: 47277       TT: 47375
P651    AT: 97          BT: 98          CT: 47537       WT: 47342       TT: 47440
P849    AT: 155         BT: 98          CT: 47635       WT: 47382       TT: 47480
P184    AT: 161         BT: 98          CT: 47733       WT: 47474       TT: 47572
P171    AT: 193         BT: 98          CT: 47831       WT: 47540       TT: 47638
P965    AT: 206         BT: 98          CT: 47929       WT: 47625       TT: 47723
P582    AT: 242         BT: 98          CT: 48027       WT: 47687       TT: 47785
P550    AT: 243         BT: 98          CT: 48125       WT: 47784       TT: 47882
P629    AT: 17          BT: 99          CT: 48224       WT: 48108       TT: 48207
P90     AT: 130         BT: 99          CT: 48323       WT: 48094       TT: 48193
P311    AT: 138         BT: 99          CT: 48422       WT: 48185       TT: 48284
P508    AT: 155         BT: 99          CT: 48521       WT: 48267       TT: 48366
P458    AT: 158         BT: 99          CT: 48620       WT: 48363       TT: 48462
P649    AT: 195         BT: 99          CT: 48719       WT: 48425       TT: 48524
P992    AT: 205         BT: 99          CT: 48818       WT: 48514       TT: 48613
P247    AT: 208         BT: 99          CT: 48917       WT: 48610       TT: 48709
P112    AT: 218         BT: 99          CT: 49016       WT: 48699       TT: 48798
P320    AT: 239         BT: 99          CT: 49115       WT: 48777       TT: 48876
P612    AT: 245         BT: 99          CT: 49214       WT: 48870       TT: 48969
P423    AT: 24          BT: 100         CT: 49314       WT: 49190       TT: 49290
P204    AT: 24          BT: 100         CT: 49414       WT: 49290       TT: 49390
P118    AT: 27          BT: 100         CT: 49514       WT: 49387       TT: 49487
P987    AT: 42          BT: 100         CT: 49614       WT: 49472       TT: 49572
P522    AT: 50          BT: 100         CT: 49714       WT: 49564       TT: 49664
P854    AT: 79          BT: 100         CT: 49814       WT: 49635       TT: 49735
P68     AT: 93          BT: 100         CT: 49914       WT: 49721       TT: 49821
P430    AT: 99          BT: 100         CT: 50014       WT: 49815       TT: 49915
P310    AT: 114         BT: 100         CT: 50114       WT: 49900       TT: 50000
P605    AT: 122         BT: 100         CT: 50214       WT: 49992       TT: 50092
P864    AT: 201         BT: 100         CT: 50314       WT: 50013       TT: 50113

Average TT = 16773.484
Average WT = 16723.172
Number of Context Switches = 1001
Number of times the queue has been sorted = 246
```

Fig. 12. Example of Simulator Execution with detailed data for each process.

Table 1
Test results with Round Robin-based algorithms.

ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NCS
RR	100	100	0	3797,78	3738,92	142
ADRR	100	100	0	2962,79	2903,93	107
MMRR	100	100	0	3265,51	3206,65	126
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NCS
RR	500	100	0	15555,51	15504,93	693
ADRR	500	100	0	13215,91	13165,33	570
MMRR	500	100	0	13240,43	13189,85	594
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NCS
RR	1000	100	0	32211,37	32160,46	1405
ADRR	1000	100	0	27035,6	26984,66	1139
MMRR	1000	100	0	29207,19	29156,23	1572
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NCS
RR	100	100	50	3464,02	3409,31	142
ADRR	100	100	50	2806,2	2751,49	113
MMRR	100	100	50	2871,09	2816,38	123
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NCS
RR	500	100	250	16315,92	16263,94	701
ADRR	500	100	250	13614,43	13562,45	568
MMRR	500	100	250	14090,03	14038,05	630
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NCS
RR	1000	100	500	29675,32	29626,71	1381
ADRR	1000	100	500	25229,44	25180,81	1143
MMRR	1000	100	500	26628,98	26580,33	1321

algorithms. To establish an objective environment, the values of AT (arrival time) and BT (burst time) are randomly generated, while the value of Q (quantum) is calculated based on the mean BT of processes already in the queue.

When considering burst time (BT), selecting a high value can have adverse effects on system responsiveness and increase average waiting times. For instance, if a process with an exceptionally high burst time is placed at the front of the queue, it can lead to significant waiting times for other processes, ultimately impacting overall system efficiency. Conversely, very low BT values would result in frequent context switches, potentially introducing overhead and hindering system performance. In such scenarios, algorithms like Shortest Job First (SJF) or SJF-based ones may perform well by prioritizing shorter tasks, thereby reducing waiting times.

Turning our attention to arrival time (AT), high arrival times mean that processes enter the queue later, resulting in reduced waiting times due to less competition for CPU time. However, this approach may also lead to processes missing deadlines or failing to respond promptly to external events. On the other hand, low arrival times imply that processes are introduced early into the queue. While this can improve responsiveness, it may also lead to increased contention for CPU resources, causing longer waiting times.

Regarding the time quantum, a longer time quantum in algorithms like Round Robin (RR) and RR-based ones can decrease the frequency of context switches. Nevertheless, it might result in processes monopolizing the CPU for extended periods, impacting fairness. Conversely, shorter time quanta can lead to more frequent context switches, ensuring fairness but introducing scheduling overhead. This can also have implications for the overall system throughput.

4.2.3. Tests on Round Robin-based algorithms

In the publications discussing the Augmented Dynamic Round Robin (ADRR) and Modified Median Round Robin (MMRR) algorithms, the authors compare their algorithms with the classical Round Robin (using a fixed quantum) and other variations of it. Their test environments typically involve a fixed number of processes with specific, non-random BT and AT values. To evaluate the results, they examine the average execution and waiting times.

In our tests, we also consider these average values. However, as mentioned in the previous section, we generate processes with random characteristics. The results obtained from our tests are presented in Table 1.

These results demonstrate that both the Augmented Dynamic Round Robin (ADRR) and Modified Median Round Robin (MMRR) algorithms outperform the classical Round Robin in terms of average execution time, waiting time, and the number of context switches. This improvement is consistent across different variables such as the number of processes and arrival times. Notably, the improvement is more pronounced when processes have varying arrival times.

When comparing the two algorithms, it can be observed that ADRR yields better results in the tests, particularly as the number of processes in the system increases. The difference in favor of ADRR becomes more significant, especially when the arrival times of the processes differ from zero. This suggests that ADRR is more effective in handling larger numbers of processes and varying arrival patterns, offering improved performance compared to MMRR.

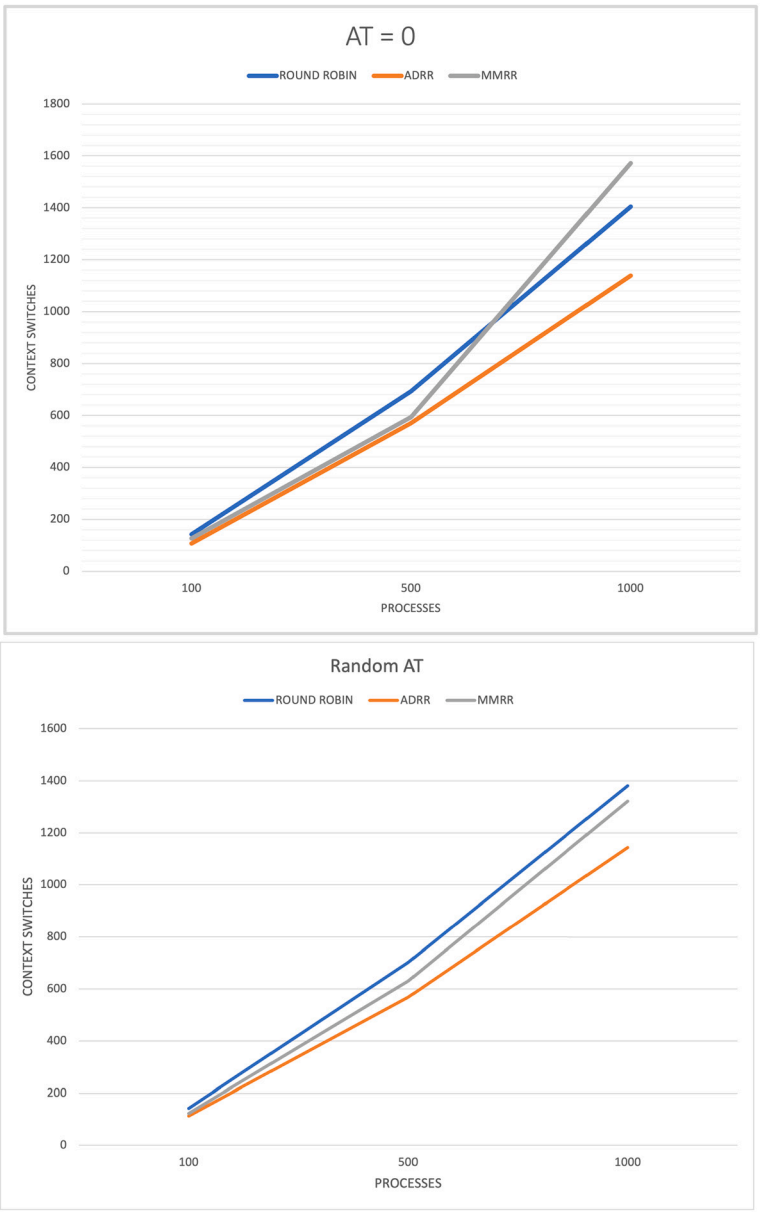


Fig. 13. Results of number of context switches in Round Robin-based algorithms.

In fact, we see in Fig. 13 that MMRR has a clear disadvantage when increasing the number of processes in the system, since the increase in the number of context switches it performs is more significant than in ADRR.

4.2.4. Tests on SJF-based algorithms

The SJF-based algorithms studied do not primarily aim to improve the average execution and waiting times achievable with the classical Shortest Job First (SJF) algorithm. Instead, their focus lies in addressing the main drawback of SJF, which is the unfair treatment of processes with longer estimated execution times. Algorithms such as K-Factor and Dynamic Multilevel Feedback Queues (DMLFQ) are specifically designed to provide more equitable opportunities for such processes.

Due to the nature of these algorithms, queuing operations are necessary to select the next process. Evaluating their performance based solely on average times would overlook crucial aspects. Therefore, the performance assessment should consider the efficiency of these queuing operations. After conducting a single run, we have obtained the results presented in Table 2.

Based on the results of our tests, it is observed that these two algorithms, K-Factor and Dynamic Multilevel Feedback Queues (DMLFQ), do not exhibit improvements over the classical Shortest Job First (SJF) algorithm in terms of execution times and waiting times. The K-Factor algorithm achieves comparable results to SJF, indicating that it does not provide significant enhancements in

Table 2
Test results with SJF-based algorithms.

ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NS
SJF	100	100	0	1280,69	1239,67	1
K-FACTOR	100	100	0	1293,2	1252,18	1
DMLFQ	100	100	0	1515,62	1474,6	1
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NS
SJF	500	100	0	8580	8528,95	1
K-FACTOR	500	100	0	8592,18	8541,14	1
DMLFQ	500	100	0	9929,76	9878,72	1
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NS
SJF	1000	100	0	17272,21	17221,27	1
K-FACTOR	1000	100	0	17286,59	17235,65	1
DMLFQ	1000	100	0	20039,12	19988,18	1
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NS
SJF	100	100	50	1882,4	1828,79	44
K-FACTOR	100	100	50	1900,52	1846,91	44
DMLFQ	100	100	50	1882,74	1829,13	43
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NS
SJF	500	100	250	8632,64	8581,07	215
K-FACTOR	500	100	250	8644,88	8593,31	215
DMLFQ	500	100	250	8634,7	8583,13	210
ALGORITHM	Processes	Max BT	Max AT	MTT	MWT	NS
SJF	1000	100	500	16131,58	16081,93	451
K-FACTOR	1000	100	500	16147,63	16097,99	451
DMLFQ	1000	100	500	16137,92	16088,27	427

terms of performance. However, DMLFQ performs notably worse, especially when dealing with groups of processes with the same arrival time, as depicted in Fig. 14.

These findings suggest that, in the specific context of execution times and waiting times, the studied SJF-based algorithms do not outperform the classical SJF algorithm. It highlights the importance of considering other factors, such as fairness in process scheduling, when evaluating and comparing these algorithms.

On the contrary, when examining the results obtained with processes having different arrival times, it becomes evident that the performance of Dynamic Multilevel Feedback Queues (DMLFQ) algorithm significantly improves. In fact, it becomes nearly equal to the performance of the classical Shortest Job First (SJF) algorithm and outperforms the K-Factor algorithm, as illustrated in Fig. 15.

This observation indicates that the effectiveness of DMLFQ is highly dependent on the arrival time distribution of the processes. When faced with varying arrival times, DMLFQ demonstrates improved performance, potentially even comparable to the well-established SJF algorithm. These findings emphasize the importance of considering the specific characteristics of the workload and arrival time patterns when evaluating and selecting scheduling algorithms.

Furthermore, it is worth noting that the number of times Dynamic Multilevel Feedback Queues (DMLFQ) algorithm performs queue reordering is lower compared to the classical Shortest Job First (SJF) algorithm, as depicted in Fig. 16. This reduction in queue reordering implies a decrease in system overhead, indicating that DMLFQ is more efficient in managing and organizing the process queues compared to the other two algorithms.

This observation highlights an additional advantage of DMLFQ in terms of system efficiency and resource utilization. By minimizing the frequency of queue reordering operations, DMLFQ reduces the overhead associated with managing the scheduling queues, contributing to improved overall system performance.

4.3. Discussion

Based on the analysis of the collected publications and the performance experiments conducted with the selected algorithms, several observations can be made regarding the future of CPU scheduling algorithms.

Firstly, the continuous contribution of new proposals in recent years, as evident from the number of publications ranging between 13 and 20 per year, highlights the sustained interest and importance of CPU scheduling algorithms. This suggests that researchers and developers recognize the ongoing relevance and significance of improving scheduling strategies, particularly in the context of mobile device operating systems. As mobile devices continue to evolve and become more prevalent, efficient CPU scheduling algorithms will play a crucial role in optimizing system performance and resource utilization.

It can be anticipated that the advancements in CPU scheduling algorithms will continue to drive innovation and improvements in the field. The ongoing research and development efforts signify the recognition of the challenges and opportunities associated with

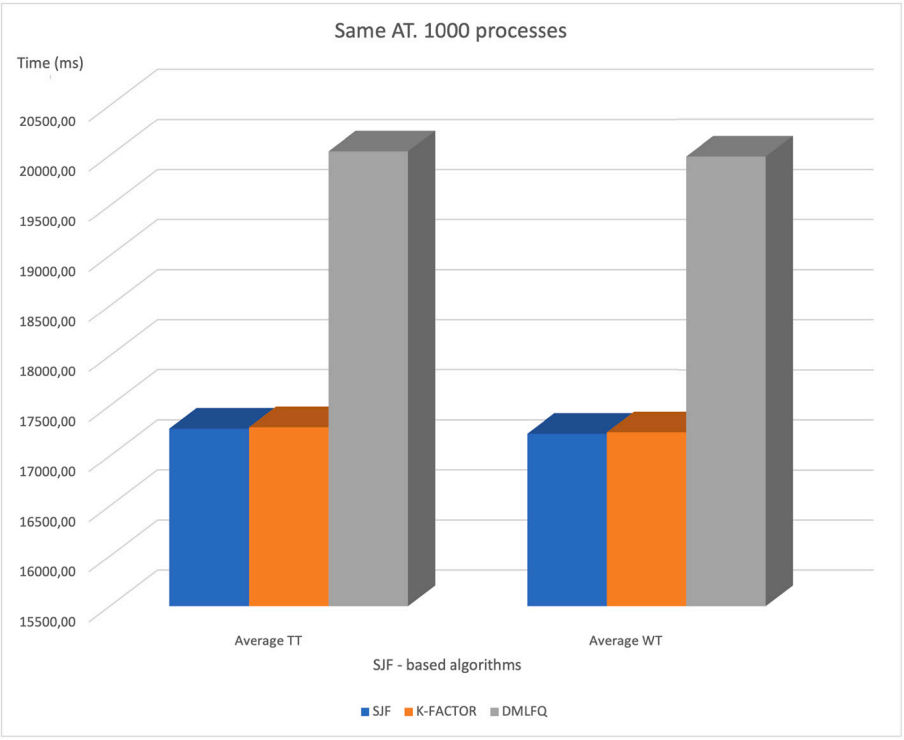


Fig. 14. Results of average run time (TT) and waiting time (WT) with 1000 processes for Shortest Job First-based algorithms.

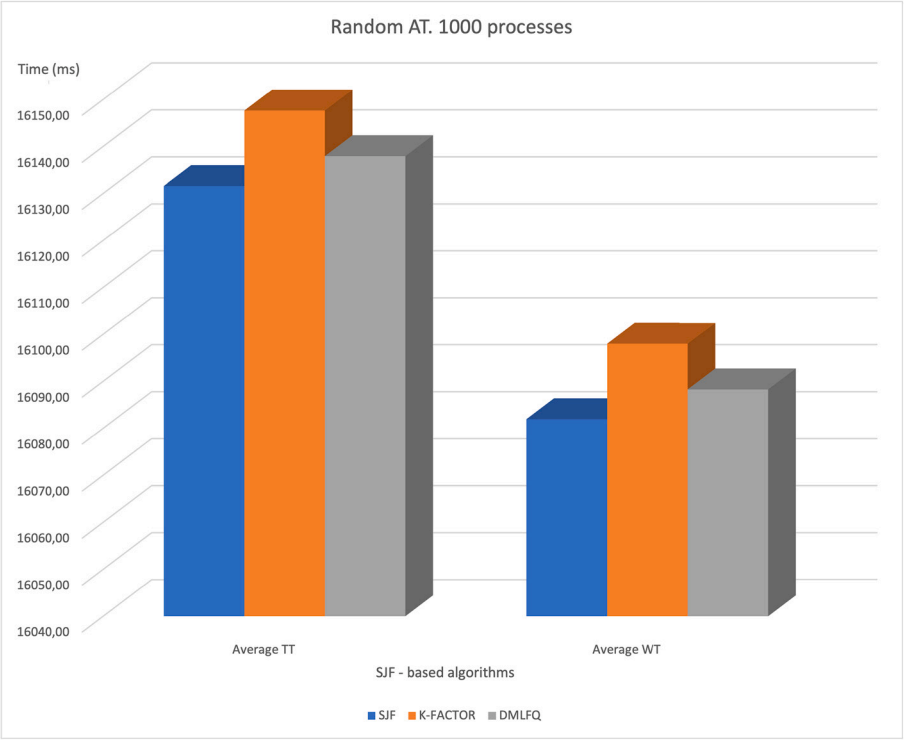


Fig. 15. Results of running time (TT) and waiting time (WT) averages for processes with random arrival times, using Shortest Job First-based algorithms.

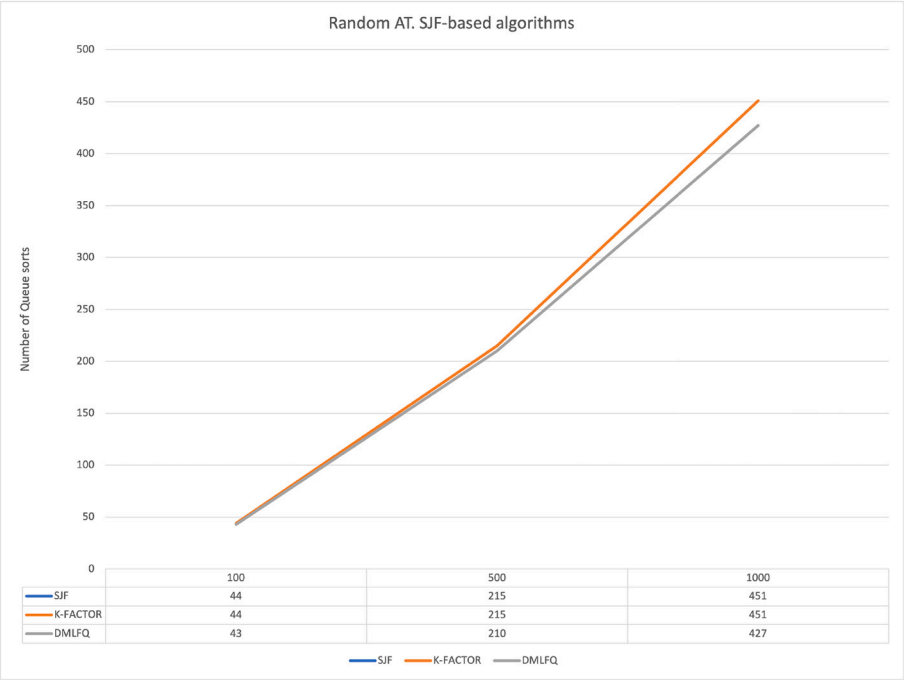


Fig. 16. Results of number of queue sorts for processes with random arrival times, using Shortest Job First-based algorithms.

efficient task scheduling in modern computing systems. Consequently, the future of CPU scheduling algorithms appears promising, with further enhancements expected to address emerging needs and complexities in diverse computing environments.

Based on our analysis of articles published since 2015 on CPU scheduling, we have identified several important lines of work that could be pursued in the future:

- Classic Scheduling Algorithms: The importance of classical scheduling algorithms is evident in the continued research and proposals based on their procedures and data structures. Future work can focus on further optimizing these algorithms, improving their efficiency, and exploring variations or enhancements to address specific system requirements.
- Dynamic Quantum Selection: The selection of the quantum in algorithms that utilize it is a crucial factor in process scheduling. Future research can investigate dynamic approaches where the operating system dynamically adjusts the quantum value based on the system's workload, resource availability, or other factors. This adaptive approach can lead to improved efficiency and responsiveness in task scheduling.
- Fairness and Starvation Prevention: Estimating the execution time of processes is essential for optimizing average execution and waiting times. However, it can pose challenges in terms of fairness, particularly for processes with high estimated execution time values. Future algorithms should incorporate mechanisms to ensure fairness among processes and minimize the chances of starvation, ensuring that all processes receive appropriate CPU time.
- Balancing Efficiency and Fairness: Achieving a balance between efficiency and fairness is crucial in CPU scheduling. Future research can focus on developing algorithms that strike the right balance, providing efficient execution while ensuring fairness among processes. This could involve incorporating priority-based mechanisms, considering the urgency or importance of tasks, or exploring multi-objective optimization techniques.
- Real-Time Scheduling and Guarantees: Real-time systems require stringent timing guarantees, and future research can focus on developing real-time scheduling algorithms that can meet these requirements. This may involve integrating real-time constraints, such as task deadlines and response times, into the scheduling decisions to ensure timely and predictable execution.
- Novel Scheduling Strategies: Beyond the classical algorithms, there is room for innovation and the exploration of new scheduling strategies. This could include leveraging machine learning, AI, or data-driven approaches to develop intelligent scheduling algorithms that can adapt to dynamic system conditions, workload variations, and user preferences.

In conclusion, our study highlights the importance of considering multiple factors when evaluating the performance of CPU scheduling algorithms. Relying on a single parameter is insufficient, and it is crucial to align the evaluation criteria with the objectives of the specific algorithm under consideration.

Based on our findings and the identified lines of work, we can summarize the future directions for CPU scheduling algorithms as follows:

- Building upon Classical Algorithms: Future algorithms are likely to continue building upon the foundation of classical scheduling algorithms. By leveraging the knowledge and insights gained from these algorithms, researchers can develop more efficient and effective scheduling strategies.
- Quantum Calculation and Management: Improving the efficiency of quantum calculation and management is an important area of focus. Dynamic approaches that adapt the quantum value based on system conditions can lead to better scheduling outcomes.
- Multiple Queue Management: Algorithms utilizing multiple queues should explore techniques to effectively manage these queues. This may involve optimizing queue selection, task prioritization, and resource allocation among queues to enhance overall performance.
- Algorithm Combination: Implementing a single algorithm may not be suitable for all operating system environments. Instead, a combination of multiple algorithms, tailored to the characteristics of the managed processes, can provide better results. This approach allows for flexibility and adaptability in handling diverse workloads.
- Real-World Implementation: Considering the practical implementation of CPU scheduling algorithms in real operating systems, it is essential to evaluate their performance in conjunction with other system components. Integration with other modules, such as memory management and I/O scheduling, should be considered to achieve holistic system optimization.

5. Conclusions

In this article, our primary objective was to assess the current state of process scheduling and identify algorithms that can enhance the teaching of operating systems. We began by scrutinizing existing CPU scheduling methods and conducted a thorough analysis of recent research trends to identify promising areas for exploration.

Throughout our analysis, specific challenges emerged. Managing information from various databases proved to be laborious and intricate, given the need to sift through an abundance of similar publications to select relevant and informative articles. Additionally, some articles lacked clarity regarding algorithm operation or the specific conditions under which evaluation tests were conducted.

Despite these challenges, we persevered and selected several recent algorithms for in-depth study, resulting in various practical utilities. The algorithm study sheets we devised can serve as valuable templates for future projects. We envision compiling a curated library of algorithms for use in operating systems courses. Another significant result of this work is the scheduling algorithm comparison simulator, designed to simplify the processes of testing and evaluating new algorithms. Despite being a versatile tool, its initial purpose was to serve as a proof of concept and remains open to diverse enhancements, including interface refinements and the incorporation of new comparison metrics.

The feedback received in class so far indicates that students have a better understanding of the internal functioning of algorithms, thanks to the flowchart included in the worksheets. Additionally, they benefit from comparing Java implementations of different algorithms and their variants. Similarly, the ability to run different simulations, modifying the input parameter values for each algorithm, enables them to contrast the effects of these parameters on algorithm behavior. This practical experimentation addresses issues such as the impact of quantum or priorities.

Despite the promising results and positive feedback obtained in class, it is important to highlight two fundamental issues. Firstly, the feedback gathered so far is informal. To provide rigor to these responses, there is a plan to frame them within a new assessment where students record their activity when using the worksheets and simulator. This information can then be subsequently compared with the results obtained. Secondly, it is crucial to consider that this academic year marks the inaugural use of the new materials in teaching the course. While initial test results are emerging, it is believed that a minimum of 3 years should elapse before these results can be deemed reliable.

Nevertheless, we would like to point out that the initial assessments for the course were conducted in November 2023, with 79.05% of the 148 students who participated successfully completing the activities related to process planning. These results correspond to the first partial exams of the course, and while they are encouraging, the outcomes regarding the final passing percentage in the first round will not be available until January.

We firmly believe that the testing environment developed through this work will prove invaluable for future research and the creation of new algorithms. This is especially true considering the complexities encountered while examining the results presented in some of the reviewed publications. It became evident that certain authors had analyzed results obtained under highly specific conditions that favored the algorithms they proposed.

In addressing these challenges and offering valuable resources and tools, we aim to make a substantial contribution to the advancement of process scheduling research and the enrichment of educational materials in the field of operating systems.

6. Annex: list of reviewed publications

Publication Title	Authors	Year
A Novel Intelligent Round Robin Cpu Scheduling Algorithm	Prem Sagar Sharma, Sanjeev Kumar, Madhu Sharma Gaur, Vinod Jain	2021
Ats: A Novel Time-Sharing Cpu Scheduling Algorithm Based On Features Similarities	Samih M. Mostafa, Sahar Ahmed Idris, Manjit Kaur	2021
Orr: Optimized Round Robin Cpu Scheduling Algorithm	Amit Kumar Gupta, Priya Mathur, Carlos M. Travieso-Gonzalez, Muskan Garg, Dinesh Goyal	2021

An Efficient Customized Round Robin Algorithm For Cpu Scheduling	Priyanka Sharma, Yatendra Mohan Sharma	2021
Cpu Scheduling In Operating System: A Review	Parag Agrawal, Amit Kumar Gupta, Priya Mathur	2021
Finding The Optimum Parameter Values Of The Round Robin Cpu Scheduling Algorithm With Genetic Algorithm	S. Ökdem, B. Kosmaz	2021
Modified Pre-Emptive Priority-Based Round Robin Scheduling Algorithms With Dynamic Time Quantum: Mppbrracbdq And Mppbrralbtdq	Sudharshan Nagarajan, R. Akhil Krishna, M. Sivagami, N. Maheswari	2021
Process Scheduling Model Based On Random Forest	Xiafei Lv, Lijun Chen	2021
Improved Round Robin Algorithm For Effective Scheduling Process For Cpu	Rahul Mishra, Gourav Mitawa	2021
Improved Variable Round Robin Scheduling Algorithm	Ayush Mangukia, Mohammed Ibrahim, Soorya Golamudi, Nishant Kumar, M. Anand Kumar	2021
Relative Time Quantum-Based Enhancements In Round Robin Scheduling	Sardar Zafar Iqbal, Hina Gull, Saqib Saeed, Madeeha Saqib, Mohammed Alqahtani, Yasser A. Bamarouf, Gomathi Krishna, May Issa Aldossary	2021
Algorithm Visualizer: Its Features And Working	Barnini Goswami, Anushka Dhar, Akash Gupta, Antriksh Gupta	2021
Improved Analysis And Optimal Priority Assignment For Communicating Threads On Uni-Processor.	Qingling Zhao, Yecheng Zhao, Minhui Zou, Zhigang Gao, Haibo Zeng	2021
A Novel Amended Dynamic Round Robin Scheduling Algorithm For Timeshared Systems	Uferah Shafi, Munam Shah, Abdul Wahid, Kamran Abbasi, Qaisar Javaid, Muhammad Asghar, Muhammad Haider	2020
Dynamic Round Robin Cpu Scheduling Algorithm Based On K-Means Clustering Technique	Samih M. Mostafa, Hirofumi Amano	2020
An Adjustable Variant Of Round Robin Algorithm Based On Clustering Technique	Samih M. Mostafa, Hirofumi Amano	2020
An Architecture For Declarative Real-Time Scheduling On Linux	Gabriele Serra, Gabriele Ara, Pietro Fara, Tommaso Cucinotta	2020
Fluctuating Time Quantum Round Robin (Ftqrr) Cpu Scheduling Algorithm	Chhaya Gupta, Kirti Sharma	2020
An Efficient Threshold Round-Robin Scheme For Cpu Scheduling (Etrr)	Khaldun Arif, Mohammed Morad, Mohammed Mohammed, Mohammed Subhi	2020
Impact Of Quantum Values On Multilevel Feedback Queue For Cpu Scheduling	David Mulder, Joseph Ssempala, Todd Walton, Ben Parker, Stephen Brough, Sam Bush, Soha Boroojerdi, Jingpeng Tang	2020
Refining The Round Robin Algorithm Using A Recomputed Time Quantum: A Comparison	Neal Oyam, Lester Joshua Pidlaoan, Rey Benjamin Baquirin, Eugene Frank Garcia Bayani, Roma Joy Dela Cruz Fronda	2020
Utilizing Dynamic Mean Quantum Time Round Robin To Optimize The Shortest Job First Scheduling Algorithm	Aaron Kyle G. Butangen, Calvin E. Velasco, Jethro Czar B. Codmos, Eugene Frank Bayani, Rey Benjamin Baquirin	2020
A Comparative Analysis Of Quantum Time Based On Arithmetic, Geometric, And Harmonic Mean For Dynamic Round Robin Scheduling	Fillian Janine G. Marcelino, Mayumi T. Escubio, Rey Benjamin M. Baquirin	2020
Comparative Analysis Of Cpu Scheduling Algorithms: Simulation And Its Applications	Ali A. Al-Bakhrani, Abdulnaser A. Hagar, Ahmed A. Hamoud, Seema Kawathekar	2020
Performance Analysis Of Fifo And Round Robin Scheduling Process Algorithm In Iot Operating System For Collecting Landslide Data	H. Hayatunnufus; Mardhani Riasetiawan; Ahmad Ashari	2020
Selection Of Cpu Scheduling Dynamically Through Machine Learning	Sara Tehsin, Yame Asfia, Naeem Akbar, Farhan Riaz, Saad Rehman, Rupert Young	2020
An Experimental Study On The Shortest Fittest Job First (Sfjf) Scheduling Algorithm Using Dynamic Queues	Karel Cassandra Cruz, Lance Justin Raymond Talon, Virgelio Ostria, Michael Rayden Dicang, Rey Benjamin Baquirin	2020
Efficient Process Scheduling Algorithm Using Rr And Srtf	Preeti Sinha, B. Prabadevi, Sonia Dutta, N. Deepa, Neha Kumari	2020
End To End Dynamic Round Robin (E-Edrr) Scheduling Algorithm Utilizing Shortest Job First Analysis	Renz Rallion T. Gomez, Christopher M. Bermudez, Vily Kaylle G. Cachero, Eugene G. Rabang, Rey Benjamin Baquirin, Roma Joy D. Fronda, Eugene Frank G. Bayani	2020
Improved Round Robin Scheduling Algorithm With Varying Time Quantum	Fiad Alaa; Mekkakia Maaza Zoulikha; Bendoukha Hayat	2020
Integrating User-Defined Priority Tasks In A Shortest Job First Round Robin (Sjfrr) Scheduling Algorithm	Hannah S. Caranto, Wyli Charles L. Olivete, Jeune Vincent D. Fernandez, Cecilia Agatha R. Cabiara, Rey Benjamin Baquirin, Eugene Frank G. Bayani, Roma Joy D. Fronda	2020
Performance Improvement Of Linux Cpu Scheduler Using Policy Gradient Reinforcement Learning For Android Smartphones	Junyeong Han, Sungyoung Lee	2020
Priority Based Round Robin (Pbrr) Cpu Scheduling Algorithm	Sonia Zouaoui, Lotfi Boussaid, Abdellatif Mtibaa	2019
Comparative Analysis Of Cpu Scheduling Algorithms And Their Optimal Solutions	M. Rajasekhar Reddy, V.V.D.S.S. Ganesh, S. Lakshmi, Yerramreddy Sireesha	2019

Fittest Job First Dynamic Round Robin (Fjfdrr) Scheduling Algorithm Using Dual Queue And Arrival Time Factor: A Comparison	Jezreel Ian Cruz Manuel, Rey Benjamin Baquirin, Kier Sostenes Guevara, Dionisio R. Tandingan	2019
A Novel Cpu Scheduling Algorithm Based On Ant Lion Optimizer	Shail Kumar Dinkar, Kusum Deep	2019
A New Improved Round Robin-Based Scheduling Algorithm-A Comparative Analysis	Yosef Hasan Jbara	2019
Sorted Round Robin Algorithm	R. Srujana, Y. Mohana Roopa, M. Datta Sai Krishna Mohan	2019
Performance Evaluation Of Dynamic Round Robin Algorithms For Cpu Scheduling	Abdulaziz A. Alsulami, Qasem Abu Al-Haija, Mohammed I. Thanoon, Qian Mao	2019
Cyclic Scheduling Algorithm	R. Kumar	2019
Round Robin Algorithm With Average Quantum Dynamic Time Based On Multicore Processor	Emma Rosita Simarmata, Gortap Lumbantoruan, Rena Nainggolan, Junika Napitupulu	2019
Round Robin Scheduling Algorithm Based On Dynamic Time Quantum	R. Dhruv	2019
Utilization Of Fuzzy Logic In Cpu Scheduling In Various Computing Environments	Bailey Granam, Hala Elaarag	2019
An Efficient Parallel Implementation Of Cpu Scheduling Algorithms Using Data Parallel Algorithms	Suvigya Agrawal, Aishwarya Yadav, Disha Parwani, Veena Mayya	2019
Towards An Educational Simulator To Support The Cpu Scheduling Algorithm Education	Leo Natan Paschoal, João Paulo Biazotto, Ana C. F. Spengler, Myke Morais De Oliveira, Katia Felizardo, Elisa Yumi Nakagawa	2019
A Modified Priority Preemptive Algorithm For Cpu Scheduling	Kunal Chandiramani, Rishabh Verma, M. Sivagami	2019
Efficient Round Robin Algorithm (Erra) Using The Average Burst Time	Masroor Aijaz, Ramsha Tariq, Maheen Ghori, Syeda Wasma Rizvi, Engr. Farheen Qazi	2019
Mean Threshold Shortest Job Round Robin Cpu Scheduling Algorithm	Pragati Pathak, Prashant Kumar, Kumkum Dubey, Prince Rajpoot, Shobhit Kumar	2019
Modified Concept To Achieve Maximum Efficiency Of Cpu Scheduling Algorithm	Himani Chauhan, Anunaya Inani	2019
Smart Round Robin Cpu Scheduling Algorithm For Operating Systems	Samkit Mody, Sulalah Mirkar	2019
Container-Based Real-Time Scheduling In The Linux Kernel	Luca Abeni, Alessio Balsini, Tommaso Cucinotta	2019
Preemptive Multi-Queue Fair Queuing	Yong Zhao, Kun Suo, Xiaofeng Wu, Jia Rao	2019
A New Cpu Scheduling Algorithm Using Round-Robin And Mean Of The Processes	N. Sujith Kumar Reddy, H. Santhi, P. Gayathri, N. Jaisankar	2018
A Novel Method Based On Priority For Enhancement Round-Robin Scheduling Algorithm	Ahmed Subhi Abdalkafor, Hadeel M. Taher, Khalid W. Al-Ani	2018
Design And Evaluation Of Cpu Scheduling Algorithms Based On Relative Time Quantum: Variations Of Round Robin Algorithm	Hina Gull, Sardar Zafar Iqbal, Saqibm Saeed, Mohammad Abdulrahman Alqatani, Yasser Bamarouf	2018
An Improved Round Robin Cpu Scheduling Algorithm Based On Priority Of Process	Govind Prasad Arya, Kumar Nilay, Siva Prasad	2018
A Study On Performance Analysis Of Multi-Level Feedback Queue Scheduling Approach	Sanjeeva Polepaka, R. P. Ram Kumar	2018
A Hybrid Of Round Robin And Shortest Job First Cpu Scheduling Algorithm For Minimizing Average Waiting Time	Tanuja Jha, Tanupriya Choudhury	2018
A K-Factor Cpu Scheduling Algorithm	Sumedha Garg, Deepak Kumar	2018
A Novel And Efficient Round Robin Algorithm With Intelligent Time Slice And Shortest Remaining Time First	Saqib Ul Sabha	2018
A Novel Hybrid Scheme To Magnify Performance Of Cpu Scheduling	M. Shalini, Amit Kumar Gupta, Yatendra Mohan Sharma	2018
An Intelligent Mlfq Scheduler Prediction Using Neural Approach	G. Siva Nageswara Rao, J. Ram Kumar, T. Srinivasa Rao	2018
An Interactive, Graphical Cpu Scheduling Simulator For Teaching Operating Systems	Joshua W. Buck, Saverio Perugini	2018
Rtvirt: Enabling Time-Sensitive Computing On Virtualized Systems Through Cross-Layer Cpu Scheduling	Ming Zhao, Jorge Cabrera	2018
Improvising Process Scheduling Using Machine Learning	N. Nikhil Jain, Srinivas Kumar, Syed Akram	2018
Cpu Scheduling With A Round Robin Algorithm Based On An Effective Time Slice	Mohammad M. Tajwar, Md. Nuruddin Pathan, Latifa Hussaini, Adamu Abubakar	2017
Optimal Round Robin Cpu Scheduling Algorithm Using Manhattan Distance	N. Srilatha, M. Sravani, Y. Divya	2017
Quantum Operating Systems	H. Corrigan-Gibbs, D.J. Wu, D. Boneh	2017
Enhanced Round Robin Cpu Scheduling With Burst Time Based Time Quantum	J. R. Indusree, B. Prabadevi	2017

Development Of A Cpu Scheduling Algorithms Visualizer Using Javascript	Kyohei Nishiyama, Koji Kagawa, Yoshiro Imai	2017
An Advanced Approach To Traditional Round Robin Cpu Scheduling Algorithm To Prioritize Processes With Residual Burst Time Nearest To The Specified Time Quantum	Mythili N. Swaraj Pati, Pranav Korde, Pallav Dey	2017
Performance Analysis Of Cpu Scheduling Algorithms With Dfrrs Algorithm	C. Jayavarthini, Angsuman Chattopadhyay, Pratik Banerjee, Shounak Dutta	2017
Intelligent Srtf: A New Approach To Reduce The Number Of Context Switches In Srtf	Shoba Bindu Chigarapalle, A. Yugandhar Reddy, P. Dileep Kumar Reddy	2017
An Efficient Dynamic Round Robin Algorithm For Cpu Scheduling	Muhammad Umar Farooq, Aamna Shakoor, Abu Bakar Siddique	2017
Improved Group Based Time Quantum (Igbtq) Cpu Scheduling Algorithm	Hadiza Musa Mijinyawa, Saleh E. Abdullahi	2017
Improved Optimum Dynamic Time Slicing Round Robin Algorithm	Shihab Ullah	2017
Modified Median Round Robin Algorithm (Mmrra)	Hafsatu Mora, Saleh E. Abdullahi, Sahalu B. Junaidu	2017
Modified Round Robin Algorithm (Mrr)	Montek Singh, Rohit Agrawal	2017
A Fair Scheduling Algorithm For Multiprocessor Systems Using A Task Satisfaction Index	Jinman Jung, Jongho Shin, Jiman Hong, Jinwoo Lee, Tei-Wei Kuo	2017
A Virtual Cpu Scheduling Model For I/O Performance In Paravirtualized Environments	Jinman Jung, Jisu Park, Seoyeon Kim, Mhanwoo Heo, Jiman Hong	2017
A Novel Fuzzy Decision-Making System For Cpu Scheduling Algorithm	Muhammad Arif Butt, Muhammad Akram	2016
Fairness-Oriented Os Scheduling Support For Multicore Systems	Changdae Kim, Jaehyuk Huh	2016
Real Time Scheduling For Dynamic Process Execution Using Round Robin Switching	G. Siva Nageswara Rao	2016
An Efficient Round Robin Cpuscheduling Algorithm Using Dynamic Time Slice	G. Siva Nageswara Rao, S.V.N. Srinivasu	2016
Scheduling Techniques For Tinyos: A Review	Anita Patil, Rajashree V. Biradar	2016
Least-Mean Difference Round Robin (Lmdrr) Cpu Scheduling Algorithm	D. Rohith Roshan, K. Subba Rao	2016
A Proposed Priority Dynamic Quantum Time Algorithm To Enhance Varying Time Quantum Round Robin Algorithm	Maysoon A. Mohammed, Mazlina Abdulmajid, Balsam A. Mustafa	2016
A Novel Cpu Scheduling With Variable Time Quantum Based On Mean Difference Of Burst Time	Sushil Kumar Saroj, Aravendra Kumar Sharma, Sanjeev Kumar Chauhan	2016
S2R2 - An Enhanced Method To Improve The Performance Of Cpu Scheduling Using Sort-Split Round-Robin Technique For Load Balancing	Padmashree M. G., Savita Shridharamuthy	2016
Efficient Implementation Of Multilevel Feedback Queue Scheduling	Malhar Thombare, Rajiv Sukhwani, Priyam Shah, Sheetal Chaudhari, Pooja Raundale	2016
Improved Round Robin Cpu Scheduling Algorithm: Round Robin, Shortest Job First And Priority Algorithm Coupled To Increase Throughput And Decrease Waiting Time And Turnaround Time	Harshal Bharatkumar Parekh, Sheetal Chaudhari	2016
An Enhanced Round Robin Cpu Scheduling Algorithm	Jayanti Khatri	2016
Adaptive Fair Scheduler: Fairness In Presence Of Disturbances	Enrico Bini	2016
Cpu Scheduling Algorithms: Case & Comparative Study	Sonia Zouaoui, Lotfi Boussaid, Abdellatif Mtibaa	2016
Modified Round Robin Algorithm For Resource Allocation In Cloud Computing	Pandaba Pradhan, Prafulla Kumar Behera, B. N. Bhramar Ray	2016
A New Proposed Dynamic Dual Processor Based Cpu Scheduling Algorithm	G. Siva Nageswara Rao, S.V.N. Srinivasu, N. Srinivasu, O. Naga Raju	2015
Enhanced Precedence Scheduling Algorithm With Dynamic Time Quantum (Epsadtq)	G. Siva Nageswara Rao, S.V.N. Srinivasu, N. Srinivasu, G. Ramakoteswara Rao	2015
The Adaptive80 Round Robin Scheduling Algorithm	Christopher Mcguire, Jeonghwa Lee	2015
An Augmented Dynamic Round Robin Cpu Scheduling Algorithm	N. Srinivasu, A.S.V. Balakrishna, Durga Vijaya Lakshmi	2015
Using An Interactive Animated Tool To Improve The Effectiveness Of Learning Cpu Scheduling Algorithms	Sukanya Suranauwarat	2015
A Task Set Based Adaptive Round Robin (Tarr) Scheduling Algorithm For Improving Performance	Neeraj Kumar Rajput, Ashwani Kumar	2015
Cpu Task Scheduling Using Genetic Algorithm	Abhineet Kaur, Baljit Singh Khehra	2015
Prioritized Fair Round Robin Algorithm With Variable Time Quantum	Arfa Yasin, Ahmed Faraz, Saad Rehman	2015

Ethics declaration

Informed consent was not required for this study because all the information used is publicly available, both within research repositories and in the University's archives

Funding

Funding for open access charge: University of Vigo/CISUG/GID SIAX.

CRediT authorship contribution statement

Miguel González-Rodríguez: Writing – review & editing, Writing – original draft, Software, Investigation. **Lorena Otero-Cerdeira:** Writing – review & editing, Writing – original draft, Validation, Supervision. **Encarnación González-Rufino:** Writing – review & editing, Writing – original draft, Investigation, Formal analysis, Data curation. **Francisco Javier Rodríguez-Martínez:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All data associated with the research presented in this article have not been deposited in any public repository at this time. However, we are committed to making these data available upon request. To access the data associated with this research, please contact us via the email address provided in this article, and we will be happy to provide you with access to the relevant data.

References

[1] David A. Patterson, John L. Hennessey, Computer Architecture. A Quantitative Approach, 6th edition, Morgan Kaufmann, 2019.
[2] Herbert Tanembaum, Andrew S. Bos, Modern Operating Systems, 4th edition, Pearson, 2009.
[3] William Stallings, Operating Systems. Internals and Design Principles, 7th edition, Prentice Hall, 2012.
[4] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, 10th edition, Wiley, 2018.
[5] Nilam Choudhary, Geetika Gautam, Yukti Goswami, Soumil Khandelwal, A comparative study of various cpu scheduling algorithms using moos simulator, J. Adv. Shell. Program. 5 (2018) 1–5.
[6] IEEEXplore Digital Library, http://ieeexplore.ieee.org/Xplore/home.jsp, October 2022.
[7] ACM Digital Library, http://dl.acm.org/, October 2022.
[8] Scopus, http://www.scopus.com/, October 2022.
[9] Research Gate, https://www.researchgate.net, October 2022.
[10] Yong Zhao, Jia Rao, Qing Yi, Characterizing and optimizing the performance of multithreaded programs under interference, in: 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT), 2016, pp. 287–297.
[11] Yitao Hu, Swati Rallapalli, Bongjun Ko, Ramesh Govindan, Olympian: scheduling gpu usage in a deep neural network model serving system, in: Proceedings of the 19th International Middleware Conference, Middleware '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 53–65.
[12] Jianqiao Liu, Nikhil Hegde, Milind Kulkarni, Hybrid cpu-gpu scheduling and execution of tree traversals, in: Proceedings of the 2016 International Conference on Supercomputing, ICS '16, Association for Computing Machinery, New York, NY, USA, 2016.
[13] Uferah Shafi, Ali Munam, Munam Shah, Abdul Wahid, Kamran Abbasi, Qaisar Javaid, Muhammad Asghar, Intsab Haider, A novel amended dynamic round Robin scheduling algorithm for timeshared systems 1, Int. Arab J. Inf. Technol. 16 (2019) 03.
[14] Prem Sharma, Sanjeev Kumar, Madhu Gaur, Vinod Jain, A novel intelligent round Robin cpu scheduling algorithm, Int. J. Inf. Technol. 14 (2021) 03.
[15] M. Rajasekhar Reddy, Vijay Ganesh, S. Sudha Lakshmi, Yerramreddy Sireesha, Comparative analysis of cpu scheduling algorithms and their optimal solutions, in: 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), 2019, pp. 255–260.
[16] Ali Al-Bakhrani, Abdulnaser Hagar, Ahmed Hamoud, Seema Kawathekar, Comparative analysis of cpu scheduling algorithms: simulation and its applications, Int. J. Adv. Sci. Technol. 29 (2020) 483–494.
[17] Jinman Jung, Jongho Shin, Jiman Hong, Jinwoo Lee, Tei-Wei Kuo, A fair scheduling algorithm for multiprocessor systems using a task satisfaction index, in: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, RACS '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 269–274.
[18] R. DurgaLakshmi, N. Srinivasu, A.S.V. Balakrishna, An augmented dynamic round Robin cpu scheduling algorithm, J. Theor. Appl. Inf. Technol. 76 (1) (2015) 118–126.
[19] Hafsatu Mora, Saleh E. Abdullahi, Sahalu B. Junaidu, Modified median round Robin algorithm (mmrra), in: 2017 13th International Conference on Electronics, Computer and Computation (ICECCO), 2017, pp. 1–7.
[20] Sumedha Garg, Deepak Kumar, A k-factor cpu scheduling algorithm, in: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2018, pp. 1–6.
[21] Malhar Thombare, Rajiv Sukhwani, Priyam Shah, Sheetal Chaudhari, Pooja Raundale, Efficient implementation of multilevel feedback queue scheduling, in: 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), 2016, pp. 1950–1954.
[22] Besim Mustafa, Cpu-os simulator, https://teach-sim.com/, December 2023.
[23] Gregor Kotainy, Olaf Spinczyk, Animos cpu-scheduling, https://ess.cs.uni-osnabrueck.de/software/AnimOS/CPU-Scheduling/index.html, December 2023.
[24] Ahmad Yahya, Hamed Hijazi, Operating system scheduling, https://github.com/AhmadYahya97/OperatingSystemScheduling, December 2023.
[25] Martin Ivanchev, Simulation and visualization tool for short-term process scheduling, Master's thesis, Ogden Honors College, 2022.
[26] Daw Khin Po, Simulation of process scheduling algorithms, Int. J. Trend Sci. Res. Develop. 3 (4) (2019) 1629–1632.
[27] Hirusha Cooray, Cpu scheduling simulator, https://cpu-scheduling-sim.netlify.app/, December 2023.
[28] Boonsuen Oh, Process scheduling solver, https://boonsuen.com/process-scheduling-solver, December 2023.