CPUSAS: A Web-Based Interactive Simulator for Teaching CPU Scheduling Algorithms

Isabel Salaberry
Instituto Politécnico de Bragança
Bragança, Portugal
a52983@alunos.ipb.pt

José Rufino
CeDRI, SusTEC, Instituto Politécnico de Bragança
Campus de S. Apolónia, 5300-253 Bragança, Portugal
rufino@ipb.pt

Arnaldo Pereira
Instituto Politécnico de Bragança
Bragança, Portugal
arnaldo@ipb.pt

2025 International Symposium on Computers in Education (SIIE) | 979-8-3315-6257-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/SIIE68580.2025.11368408

***Abstract*—CPUSAS is an interactive web-based educational tool designed to support the teaching of CPU scheduling algorithms in Operating Systems subjects in computer science education. The simulator enables learners to create, edit, and visualize scheduling scenarios using an array of algorithms, including first-come-first-served, shortest job first, shortest remaining time first, round-robin, and both preemptive and non-preemptive priority scheduling. The tool features a multilingual interface, animated visualizations, support for saving and loading scenarios, and a function that enables exporting simulations as video. Distinct user roles for students and instructors allow educators to manage access and customize the solution. Also, the tool features a modular design that supports the addition of new scheduling algorithms if needed. CPUSAS's design and implementation push beyond existing educational simulators, underscoring its potential to enhance conceptual understanding through interactive, learner-centered engagement.**

***Index Terms*—CPU scheduling, operating systems, simulation tools, educational software, computer science education**

I. Introduction

Understanding how an operating system (OS) allocates the central processing unit (CPU) to competing processes is fundamental in computer science education. CPU scheduling is a core topic in most undergraduate Operating Systems courses [1], and mastering it requires students to develop mental models of how scheduling decisions affect performance metrics such as waiting time, turnaround time, and response time. These concepts often involve time-dependent, interrupt-driven behavior that is difficult to visualize through static content alone [2]. Students frequently report difficulties in understanding preemption, context switching, and the effects of different scheduling strategies on system responsiveness [3].

Traditional teaching methods, such as textbook diagrams, blackboard exercises, and manual Gantt chart construction, remain widely used despite notable limitations in helping students visualize dynamic execution behavior. Trying to follow how processes are scheduled mentally can easily lead to misunderstandings or oversimplified thinking [4]. Additionally, working through examples by hand takes time and is often prone to errors, which leaves less room for in-depth discussion or reflection [5]. For instructors, it's not always easy to demonstrate the dynamic nature of scheduling or provide students with personalized feedback during class.

Many simulation tools have been introduced to support teaching CPU scheduling, trying to address these issues [6]. Because current tools offer basic, quasi-static visualizations, they lack flexibility, pedagogical depth, and ease of deployment and use. Some require installation or complex configuration, others are outdated or limited to a small subset of algorithms without the possibility of extension [7]. Few tools offer real-time interaction, customizable scenarios, multilingual access, or instructor control, features that are increasingly important in diverse learning environments [8]. Furthermore, new developers of algorithm visualization tools often lack established guidelines or best practices [9].

CPUSAS (CPU Scheduling Algorithm Simulator), an interactive, browser-based simulator designed for teaching and learning CPU scheduling, aims to fill this gap. It was designed with both students and instructors in mind to enable users to create and edit scheduling scenarios, to observe algorithm behavior through animated Gantt charts, and to evaluate key metrics in real-time. The tool implements widely taught algorithms, supports exporting simulations as video, and includes role-based access to enable instructors to manage classroom use. Built using modern web technologies to ensure accessibility and maintainability, CPUSAS stands on a modular architecture, facilitating the addition of new features.

Educational tools that promote visual reasoning have shown great potential in improving student engagement [10]. In particular, interactive simulators enable students to experiment with different inputs, observe the behavior of algorithms, and receive immediate feedback [11]. CPUSAS builds on these insights to offer a web-based, feature-rich learning environment adapted to modern classrooms.

CPUSAS aims to enhance student understanding through interactive learning and facilitate more effective teaching practices by providing an intuitive tool, significantly improving OS proficiency. The simulator is live, maintained in a continuous deployment model, and has undergone thorough testing for correctness, with iterative refinements made based on feedback from educators and informal interactions with students.

The remainder of the paper is structured as follows: Section II provides background on CPU scheduling algorithms and visualization tools. Subsequently, Section III details the system's design. Section IV describes the overall development and technical implementation of CPUSAS. Section V evaluates the tool's functionality and correctness. Finally, Section VI concludes the paper and lays out future work directions.

II. Background

Understanding CPU scheduling algorithms is crucial for guiding the design and contributions of CPUSAS. This context extends to a systematic evaluation of existing visualization tools, assessing their current capabilities and limitations to identify areas for innovation and improvement.

A. CPU Scheduling Algorithms

CPU scheduling algorithms determine the order in which processes are executed by the processor, directly affecting system performance. They can be classified as preemptive or non-preemptive: in the 1st case, a running process can be suspended to make room for another, more prioritized process to execute; in the 2nd, the process remains in the CPU until it completes its execution or voluntarily blocks. Below, the most frequently referred classic algorithms are briefly presented:

- *First-Come-First-Served (FCFS)*: executes processes in arrival order, without preemption.
- *Shortest Job First (SJF)*: selects the process with the shortest estimated execution time. It can be preemptive or non-preemptive. The preemptive version is known as *Shortest Remaining Time First (SRTF)*.
- *Round Robin (RR)*: uses a fixed time quantum to alternate CPU between processes, promoting greater fairness.
- *Priority Scheduling (PS)*: assigns priorities to processes and selects the one with the highest priority (with or without preemption). The preemptive version is known as *Preemptive Priority Scheduling (PPS)*.

These algorithms are frequently used in academic environments and simulators due to their didactic importance and historical relevance in the study of Operating Systems, and are therefore fundamental components in CPUSAS.

B. CPU Scheduling Visualization Tools

CPU scheduling visualization tools were identified by querying the GitHub REST API with '`cpu-scheduler forks:>1`', which returned repositories having at least one fork, indicating some evidence of community interest. This initial filtering helped ensure a focus on actively maintained and potentially influential projects. After removing false positives, only projects that were sufficiently documented and deployable were considered. This systematic analysis revealed a spectrum of simulators, which was subsequently categorized by input modes (I-codes), visualization and interaction richness (V-codes), added key features (F-codes), and high-impact limitations (L-codes), as presented in Table I. The identified CPU scheduling visualization tools were then categorized based on these criteria, with their detailed classification provided in Table II. Notice that the repository URL is obtained by prefixing the name with `https://github.com/`. For transparency and independent verification of this study's findings, a replication package [12], comprising scripts, collected data, and classification output, is available as supplementary material.

Table I
Classification codes.

Input modes	Visualization and interaction	Other key features	High–impact limitations
I1 – On-screen table editor	**V1** – Static Gantt chart	**F1** – Video/image export	**L1** – Desktop-only install
I2 – File import/export	**V2** – Animated autoplay	**F2** – Multilingual support	**L2** – Outdated/unmaintained
I3 – Random scenario generator	**V3** – Step-by-step controls	**F3** – Role-based access	**L3** – Narrow algorithm set
I4 – Session restore	**V4** – Metrics panel	**F4** – Save/load scenarios	**L4** – Not extensible (cannot add new algorithms)

Table II
CPU scheduling visualization tools classification.

ID	Repository	Codes	ID	Repository	Codes
1	brilacasck/CPU-Scheduler	I1, I2, I3; V2, V4; —; L1, L2, L4	**15**	vivek9patel/CPU-Scheduling-APP-React-Native	I1; V1, V2, V4; —; L2, L4
2	marvinjason/CPUSchedule	I1; V1, V4; —; L1, L2, L4	**16**	Maharshi-Pandya/os-lab-project	I1; V4; —; L2, L4
3	boonsuen/process-scheduling-solver	I1; V1, V4; —; L2, L4	**17**	nadaamohhamed/CPU-Schedulers-Simulator	I1; V1, V4; —; L1, L2, L4
4	ammarlodhi255/cpu-scheduling-simulator	I2; V2, V4; —; L1, L2, L4	**18**	PrinceSinghhub/CPU-SCHEDULING-ALGORITHM-VISUALISER	I1; V2, V4; —; L2, L4
5	mukul2310/cpu-scheduler-visualiser	I1; V1, V4; —; L2, L4	**19**	AbdelrahmanBayoumi/CPUScheduler-FX	I1; V4; —; L1, L2, L4
6	EL-SHREIF/CPU-Scheduler	I1; V1, V4; —; L1, L2, L4	**20**	elzoughby/RMS-scheduling	I1; V1; —; L1, L2, L3, L4
7	hirushacooray/cpu-scheduling-sim	I1; V4; —; L2, L4	**21**	MariamSafwat/CPU_Scheduler	I1; V1, V4; —; L1, L2, L4
8	Ravipatel1309/CPUScheduler	I1; V1, V4; —; L2, L4	**22**	Locality-PH/cpu_scheduler	I1; V1, V4; —; L2, L3, L4
9	Ahmedkhalifa1999/CPU-Scheduler-Visualizer	I1; V1, V4; —; L1, L2, L4	**23**	Devyanshu/cpu-scheduling-algorithms-python	I1; V1, V4; —; L1, L2, L4
10	Nada-Nasser/CPU-Schedulers-Simulator	—[a]; V1; —; L1, L2, L4	**24**	Seif-Ibrahim1/CPU-Schedulers-Simulator	I1; V1, V4; —; L1, L2, L4
11	dat911zz/CPU-Scheduling-Algorithms	I1; V1, V4; —; L1, L2, L4	**25**	sukhum29/CPU-Scheduler-Project	I1; V1, V4; —; L1, L4
12	vasu-gondaliya/cpu-scheduling-algorithms	I1; V1, V4; —; L2, L4	**26**	ottmartens/cpu-algorithms	I1; V2, V4; —; L2, L4
13	aitikgupta/interactive_cpu_scheduler	I1, I3; V4; —; L1, L2, L4	**27**	andrewi66doe/Reduced-Energy-Scheduling-Simulation	I2; V1; —; L1, L2, L3, L4
14	hemanth-07-11/CPU-Scheduling-Visualisation	I1; V1, V4; —; L2, L4	**28**	nayera540/OS-Scheduler-Project	I1; V1, V4; —; L1, L2, L4

[a] Tool 10 only accepts console input, which none of the four I-codes cover.

The input-mode codes (I1–I4) indicate how a scenario initially enters the simulator, whether it is typed in a table, loaded from a file, generated randomly, or restored from a previous session. The visualization/interaction codes (V1–V4) describe the richness of the output display, ranging from a single static Gantt chart to fully animated playback with pausing and showing metrics (e.g., waiting time). Other key features (F1–F4) summarize capabilities that add value, such as export options, multilingual interfaces, role separation, and scenario persistence. Finally, the high-impact limitations (L1–L4) identify constraints such as desktop-only deployment, outdated code, limited algorithm coverage, or a lack of extensibility, any of which may prevent adoption in OS classrooms.

The classification presented in Table II categorizes 28 CPU scheduling visualization tools. A notable observation is the prevalence of the I1 input mode (89.28% of tools), while I2 and I3 are relatively niche, and I4 is conspicuously absent. Similarly, the visualization and interaction richness lean heavily towards V4 (89.28%) and V1 (71.42%), often combined. This high frequency of particular I- and V-codes suggests a largely homogeneous approach to fundamental design and user interaction across the analyzed tools, indicating an established, albeit somewhat limited, standard in their development.

However, the analysis also shows some areas of deficiency. For instance, there is a complete absence of any added key features (F-codes) across all 28 tools, which indicates a lack of innovation and more advanced functionalities beyond the basic simulation and visualization features. The prevalence of high-impact limitations reinforces this, specifically L4 (100%), L2 (96.42%), and L1 (60.71%). Therefore, it is reasonable to conclude that i) current tools fall short in providing more advanced features, which limits their effectiveness, and ii) there is a clear need for the development of more feature-rich and less constrained CPU scheduling visualization tools, such as the one proposed in this paper.

III. System Design

This section describes the main features required for the tool, and the architecture based on which it was later deployed.

A. Requirements

The foundational design of the system was shaped by a set of key requirements, based on the educators' teaching experience, ensuring both functionality and overall performance. These requirements fall in several categories, next laid out.

Core simulation and visualization features:

- *Algorithm simulation* - Simulate the functioning of classic CPU scheduling algorithms, including FCFS, SJF, SRTF, RR, PS, and PPS.
- *Graphical visualization* - Generate graphical representations (e.g., Gantt Charts) that illustrate the execution of scheduling algorithms.
- *State visualization* - Use distinct visual indicators (e.g., colors) to represent the different states of processes (e.g., executing, waiting/ready, terminated).

Scenario management (input and persistence):

- *Scenario definition* - Allow the explicit input of process parameters and the definition of specific preemption/interruption instants to simulate preemption or external events.
- *Random scenario generation* - Include a mechanism to generate simulation scenarios with random process parameters.
- *Save/load scenarios* - Implement functionality to save user-defined simulation scenarios and to load previously saved scenarios (using a descriptive/serializable format).

Simulation interaction and advanced output:

- *Step-by-step tracking* - Allow visual tracking of the simulation step-by-step (tick-by-tick), showing the evolution of the scheduling.
- *Automatic animation* - Provide an automatic animation mode that shows the evolution of scheduling over time without manual user intervention.
- *Simulation control* - Include controls similar to a multimedia player (play, pause, step forward/backward, timeline bar) to control step-by-step visualization or animation.
- *Video generation* - Implement the ability to generate short videos that demonstrate the simulation of a specific algorithm for a particular scenario.

User and administrator management:

- *User and profile management* - Implement user registration and login, and an "admin" profile distinct from the normal user profile.
- *Administrator features* - Validate, approve, or reject user registrations; delete user accounts; and make specific algorithms invisible in the main interface.

Operational requirements:

- *Web platform* - The platform must be web-based and accessible via a browser.
- *Flexibility* - The simulator must be flexible in accepting parameters and scenarios.
- *Dynamic timeline*: The simulator timeline representation should not have a fixed, predefined limit.

These requirements established the fundamental scope and functional objectives of CPUSAS, serving as a blueprint for its design and implementation. The iterative nature of the requirements formulation process enabled continuous refinement and adaptation, ensuring that the system remained aligned with the evolving needs of future users. Ultimately, the successful realization of these requirements culminated in an intuitive and effective educational tool that facilitates the learning process.

B. System Architecture

The high-level system architecture of CPUSAS is represented in Fig. 1, as a cloud-backed application that leverages client-side processing for its core simulations.

The *Browser* component hosts the main web application, providing an interactive user interface (UI) with distinct areas

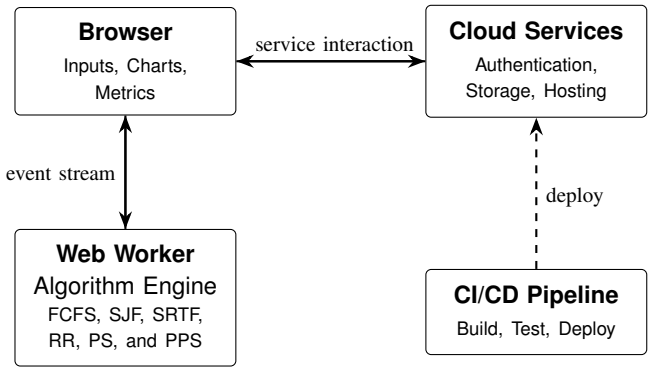

Figure 1. High-level architecture of CPUSAS.

for inputs, charts, and metrics. A dedicated *Web Worker* subcomponent operates within the browser environment, isolating the CPU scheduling algorithm computations and communicating with the main browser thread via an event stream to ensure UI responsiveness. Backend functionalities are provided by *Cloud Services*, including authentication for user identity, storage, and hosting for application delivery. The *Browser* interacts directly with these *Cloud Services* through API communication for data exchange and service requests. A *CI/CD Pipeline* should manage the system's continuous evolution, automating the building, testing, and deployment processes (a feature not yet implemented in the current CPUSAS version).

IV. Tool Development

Modern web technologies and best software engineering practices guided the development of the system. A live version is accessible at https://cpusas.web.app, and the source code is also available at https://github.com/isalaberry/CPUSAS.

A. Application Interface and Core Features

Fig. 2 shows the CPUSAS user interface. A fixed header ① presents the simulator's title, a language toggle (Portuguese/English), and a user avatar icon that opens the login/register panel. Immediately beneath, a flat navigation strip ② lists the available schedulers, acting as tabs that reload the page state without a complete refresh. When RR is active, a small "Quantum" spinner ③ appears above the scenario table ④, allowing users to adjust the time slice. The table itself serves as the primary input mode: users enter or edit arrival and burst times in number-field cells, add or delete rows with one-click buttons ⑤ and ⑥, or click button ⑦ to populate the grid with randomly generated data instantly. The "Import" and "Export" buttons ⑧ read or write the current scenario in a plain-text descriptor, while the "Add Interruption" band

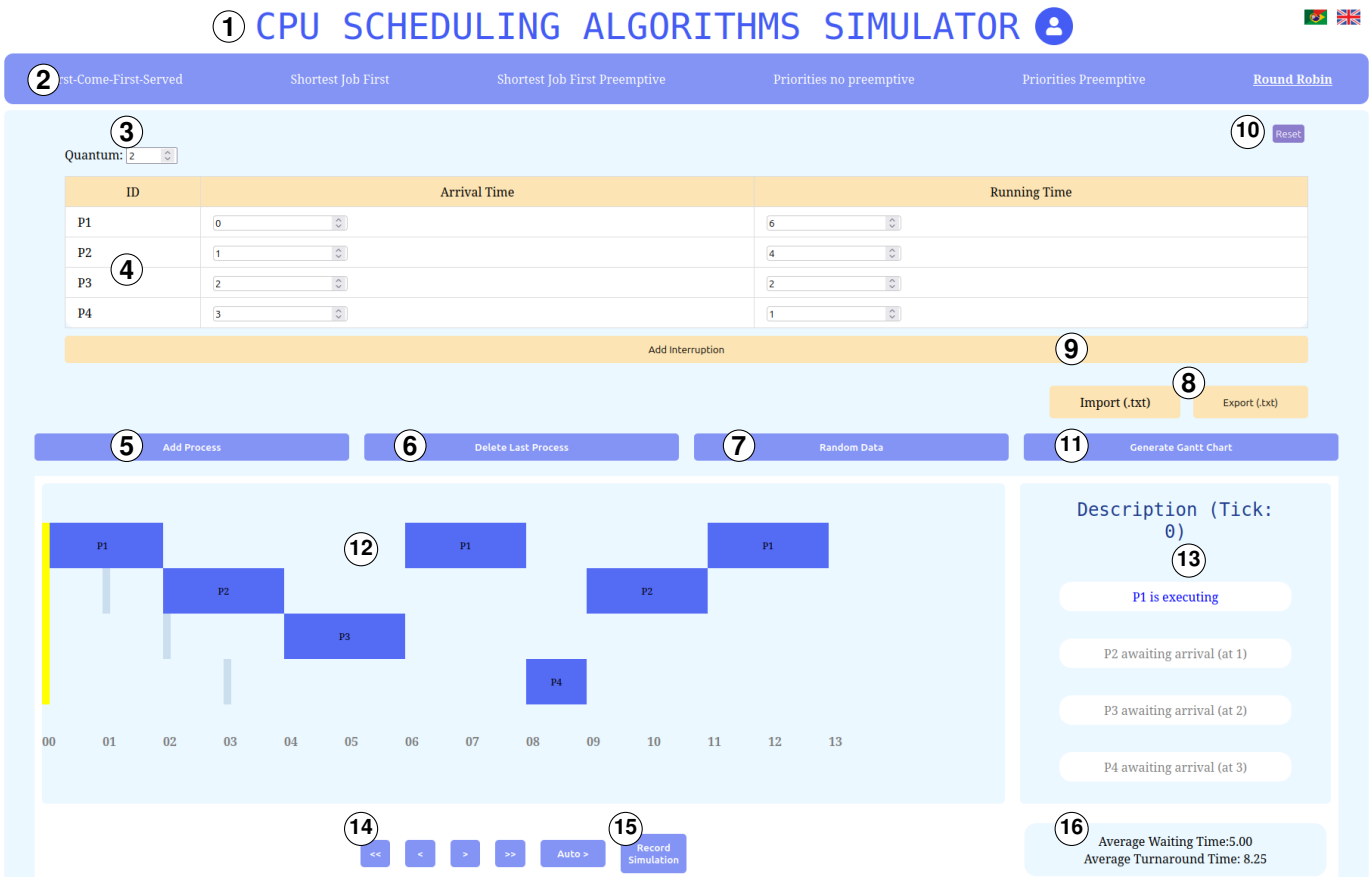

Figure 2. CPUSAS user interface.

⑨ allows for inserting explicit preemption points. A "Reset" button ⑩ clears everything in one action.

Below the editor, after pressing the "Generate Gantt Chart" button ⑪, the screen splits into two synchronized panels. On the left, a light blue canvas displays the Gantt timeline ⑫. Dark blue bars indicate CPU bursts, while lighter markers denote waiting intervals. A thin yellow stripe serves as the time cursor. On the right, a textual "Description" panel ⑬ explains the state of each process at the current tick, updating in real-time as the simulation progresses. Playback controls ⑭, including rewind, step, play/pause, fast-forward, and an "Auto-play" toggle, as well as a "Record Simulation" button ⑮ that exports the session to MP4/WebM, sit directly below the canvas, mirroring the layout of a media player. A metrics footer ⑯ in the lower-right corner displays average waiting and turnaround times, providing users with immediate quantitative feedback as they experiment with different inputs or algorithms.

B. Frontend Implementation

The single-page client-side application is built upon the *React*[1] library. This choice enables the creation of a component-based architecture, promoting code reusability, maintainability, and scalability. Key interactive elements, such as the process input table[2] and the dynamic Gantt chart visualization[3], are encapsulated as independent *React* components. The codebase is *TypeScript*[4], which provides static typing to improve code quality and reduce runtime errors. Communication with the *Browser* uses an event stream for data exchange with the client-side *Web Worker* responsible for simulation computations, and via API communication for interactions with the *Cloud Services*, which handle authentication and data persistence.

C. Simulation Engine

As CPU scheduling simulations are demanding computations, they are offloaded to a dedicated *Web Worker*. This design choice ensures the main browser thread remains responsive, even during complex or lengthy simulations. All communication is done using only asynchronous message passing. Input parameters, including process data and interruption events, are sent to the worker, which then executes the selected scheduling algorithm. As a simulation progresses, intermediate results are received by the main thread to incrementally render the Gantt chart and update the performance metrics without blocking user interaction.

D. Cloud Backend Services

Firebase[5] is a mature platform for developing web and mobile applications, and provides CPUSAS with its backend infrastructure. *Firebase Authentication*[6] manages user identity, supporting secure registration and login functionalities. The system is tier-based to manage user access. Newly registered student accounts initially begin in a 'pending' state, and administrators (teachers) must explicitly approve them before they can access advanced features, such as saving scenarios. *Cloud Firestore*[7], a NoSQL document database, manages the persistent data storage solution for user-defined simulation scenarios, user profiles, and global application configurations. *Firestore Security Rules*[8] enforce server-side access control, and *Firebase Hosting*[9] is the global content delivery network for the CPUSAS application, providing users worldwide with reliable access.

E. Other Functionalities

Beyond the core simulation, CPUSAS incorporates an animation recording feature, allowing users to generate video records of their simulations. This functionality is particularly valuable for pedagogical purposes, enabling students to review complex scheduling behaviors at their own pace, share specific scenarios with peers or instructors, or integrate simulation outputs into presentations. Technically, the recording process leverages *html2canvas*[10] to capture the current state of the DOM as a series of images, which are then efficiently encoded into a video format (e.g., WebM) using the *MediaRecorder*[11] API. This provides a direct, high-fidelity capture of the interactive simulation as it unfolds.

To ensure universal accessibility and cater to a diverse user base, the platform offers robust internationalization support, currently providing both Portuguese and English interfaces. This is achieved using the *react-i18next*[12] library, which manages translation keys and dynamically renders textual content across all UI elements based on the user's language selection. Furthermore, the *i18next-browser-languagedetector* plugin enhances usability by automatically detecting the user's preferred browser language upon initial access and persisting this preference for future sessions, ensuring a seamless localized experience.

V. Evaluation

The primary methodology for evaluating the simulation logic of the tool was a systematic comparison of the tool's output against pre-prepared, manually derived solutions for specific test cases. This approach provided an irrefutable baseline for verification, as each manual calculation represented the ground truth for the corresponding scenario. As an illustrative example, consider the CPU scheduling problem for the SRTF algorithm, defined by the input parameters shown in Fig. 3.

[1]https://react.dev
[2]src/components/InputTable.jsx
[3]src/components/GridProcess.jsx
[4]https://www.typescriptlang.org
[5]https://firebase.google.com/
[6]https://firebase.google.com/docs/auth
[7]https://firebase.google.com/docs/firestore
[8]https://firebase.google.com/docs/firestore/security/get-started
[9]https://firebase.google.com/docs/hosting
[10]https://html2canvas.hertzen.com
[11]https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder
[12]https://react.i18next.com

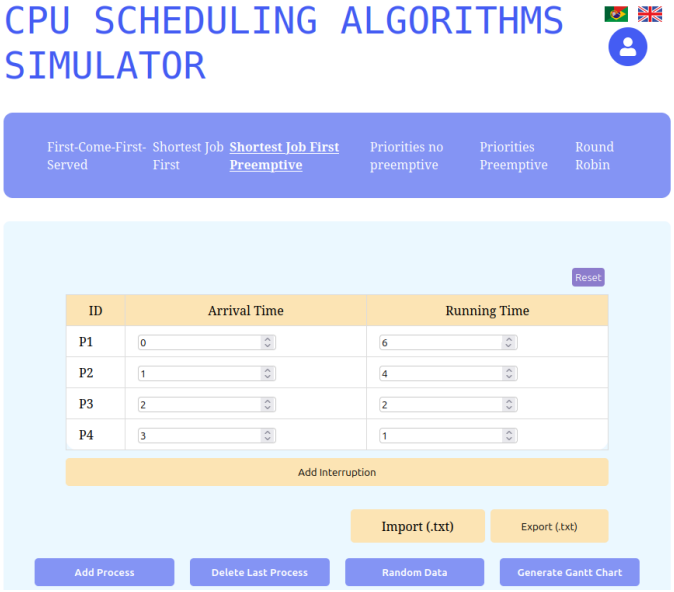

Figure 3. CPUSAS SRTF simulation input.

For this problem instance, the expected Gantt chart and corresponding key performance metrics (such as average waiting time and average turnaround time) were first rigorously calculated by hand, as illustrated by Fig. 4.

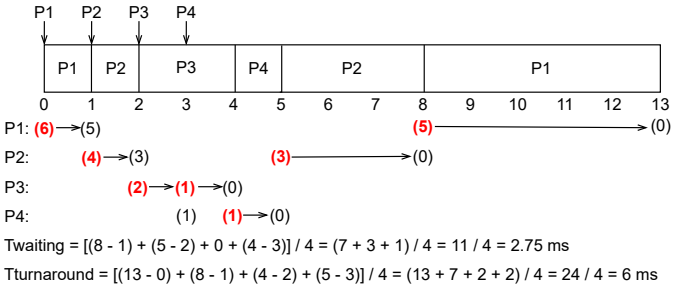

Figure 4. SRTF manual calculation example: Gantt chart and metrics.

Upon executing the input within the CPUSAS tool, the generated Gantt chart and all calculated performance metrics were observed to match the manually computed values precisely, as illustrated in Fig. 5.

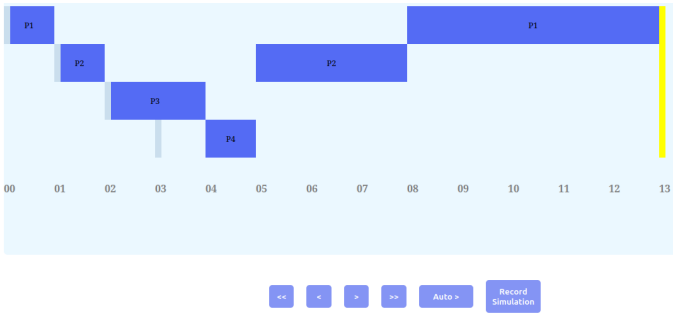

Figure 5. CPUSAS SRTF simulation output Gantt chart and metrics.

This perfect match across all tested scenarios, from simple FCFS to more intricate preemptive algorithms like SRTF, provides strong empirical evidence of the platform's accuracy and reliability.

VI. Conclusions

CPUSAS provides a practical and dynamic platform for engaging students in core CPU scheduling concepts through interactive simulation. Its accessibility, extensibility, and pedagogical orientation position it as a valuable addition to Operating Systems education. Future work encompasses rigorous quantitative and qualitative pedagogical validation, aiming to empirically measure its learning impact and effectiveness in diverse educational settings, as well as broader classroom deployment to reach a wider student audience. Furthermore, the full implementation of the CI/CD pipeline will be pursued.

Acknowledgment

This work was supported by FCT - Fundação para a Ciência e Tecnologia, I.P. by projects: CeDRI, UID/05757/2025 (DOI: 10.54499/UID/05757/2025) and UID/PRR/05757/2025 (DOI: 10.54499/UID/PRR/05757/2025); SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020).

References

[1] A. N. Kumar, R. K. Raj, S. G. Aly, M. D. Anderson, B. A. Becker, R. L. Blumenthal, E. Eaton, S. L. Epstein *et al.*, *Computer Science Curricula 2023*. New York, NY, USA: Association for Computing Machinery, 2024.

[2] F. Giraldeau, M. R. Dagenais, and H. Boucheneb, "Teaching operating systems concepts with execution visualization," in *ASEE Annual Conference and Exposition, Conference Proceedings*, 2014, pp. 1–15.

[3] S. Pamplona, N. Medinilla, and P. Flores, "Exploring misconceptions of operating systems in an online course," in *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, 2013, pp. 77–86.

[4] R. Duran, A. Zavgorodniaia, and J. Sorva, "Cognitive load theory in computing education research: a review," *ACM Transactions on Computing Education*, vol. 22, no. 4, pp. 1–27, 2022.

[5] P. A. Kirschner, J. Sweller, and R. E. Clark, "Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching," *Educational Psychologist*, vol. 41, no. 2, pp. 75–86, 2006.

[6] Y. H. Jbara, "A new visual tool to improve the effectiveness of teaching and learning CPU scheduling algorithms," in *2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, 2017, pp. 1–6.

[7] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *ACM Transactions on Computing Education*, vol. 13, no. 4, pp. 1–64, 2013.

[8] D. H. Rose and A. Meyer, *Teaching Every Student in the Digital Age: Universal Design for Learning*. Alexandria, VA, USA: Association for Supervision and Curriculum Development, 2002.

[9] C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards, "Algorithm visualization: The state of the field," *ACM Transactions on Computing Education*, vol. 10, no. 3, 2010.

[10] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages & Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[11] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi *et al.*, "Exploring the role of visualization and engagement in computer science education," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, 2002, p. 131–152.

[12] A. Pereira, "Replication package for "CPUSAS: A Web-Based Interactive Simulator for Teaching CPU Scheduling Algorithms"," https://osf.io/47xy5/files/osfstorage, 2025, accessed: July 19, 2025.