© DEC 2025 | IRE Journals | Volume 9 Issue 6 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I6-1712959

Process Synchronization Simulation: An Interactive Visualization Framework for Classic Concurrency Problems with Real-World Validation

AKASH REDDY RANABOTHU[1], RAGHA MOULIK ATMURI[2], VARSHITH REDDY JAMANDLA[3], DR. J. SHAJEENA[4], DR. S. RAHMATH NISHA[5], DR. JUBILANT J. KIZHAKKETHOTTAM[6]

[1,2,3] *School of Computing, SRMIST, Trichy*

[4,5,6] *Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, Tiruchirappalli, India*

Abstract*—We have all seen students who struggle with the abstract nature of process synchronization is a crucial part in operating systems, but its dynamic behavior is notoriously hard to teach and learn. That's why we created a hands-on, web-based tool.It is designed to bridge that gap.The system was built based on Python Flask, JavaScript, and HTML5.That does more than show pictures; that shows you how processes interact in real time, allows students to change parameters on their own, and even includes deadlock detection. Core of the Tool is a discrete-event simulation engine. It accurately It realistically depicts common synchronization behavior and helps illustrate how mechanisms like semaphores, mutexes,and resource allocation graphs in the management of shared resources. On the technical side, this is rock solid. Even with 50 processes running it never dipped below 60 FPS and kept the latencies below 20ms. We achieved an efficiency of 65.2% boost and a 46.8% reduction of costs. What gives us confidence that impresses in this tool is that its simulations are just so real. The Difference between what our tool predicted versus what actually happened less than 5 in real- world deployment.

Index Terms*—Process synchronization, concurrent program- ming, visualization, educational software, dining philosophers, producer-consumer, readers-writers, deadlock detection, smart parking, EV charging.

I. INTRODUCTION

Process synchronization, at its very core, is a method that Ensures correct and efficient multi-process execution, when different processes attempt to access the same shared resources [1], [2]. The real complexity comes in due to non-deterministic execution-you can't always predict the exact order things will run. This uncertainty can result in several serious issues such as race conditions and deadlocks, and that is why we need sophisticated It uses coordination mechanisms [3], [4] to manage the chaos. Despite its importance, this is a topic students really struggle with. mostly due to its abstract and dynamic nature [1], [2]. The traditional static diagrams that you find in textbooks just don't work very well; they fail to convey the temporal aspects. the "when" and complex process interactions. This is the exact purpose of developing interactive visualizations tools, for they are designed to help in bridging between these. abstract concepts and the observable behaviors students can actually see.

II. SYNCHRONIZATION PROBLEMS

This section reviews the basic synchronization mechanisms, educational tools, modern web-based simulation systems. and The paper identifies current research gaps so that contributions could be put into context.

A. Synchronization Primitives and Concepts

The basic synchronization primitives are semaphores, mutexes, and readers-writer locks. These provide atomic operations which coordinate concurrent processes [1] [3]. Semaphores have indicated that complex control flows involving several Wait and signal are operations in processes that deal with mutexes. Simplify coordination, ensuring exclusive access. Reader-writer locks go a step further in regulating access by allowing concurrent provide for read access to the readers while giving exclusive access to writers at the same time, thus balancing throughput and fairness1, 4. Several works have assessed these primitives concerning performance, studying such as overhead, latency, and starvation prevention.

Deadlock detection has been addressed in many of the

theoretical research publications, and in the avoidance algorithms of importance for system reliability.

B. Educational Visualization Tools

Synchronization of the learning process cannot be done because its non-deterministic time-dependent behavior[1][2]. Visual- Above all, the emergence of various kinds of technology-based instruction and computer-assisted learning and practice. AIDS has supplemented traditional teaching. methods by providing interactive presentations of synchronization scenarios. Of special note are systems due to : Pike et al., which animate classic problems such as Dining Philosophers and Readers-Writers to expose thread interaction and resource contention. Clancy, et al. Improved learning with Web-based APPs that have a parameter Adjustment allows the students to study deadlocks and starvation. Dynamic effects: These tools have been shown to: It develops cognition and increases student participation.

C. Web Technologies for Simulation

Recent frameworks take advantage of the pervasiveness of browsers. Introduces most of the advanced client-side scripting with JavaScript, WebAssembly and Python back- ends. Lightweight The architectures like SimService introduce modules of simulation control, while in Bodylight.js, model visualization is coupled with real-time data; thus, plotting. Such frameworks democratize access by avoiding installation barriers, thus enabling remote learning environments (Proper- ties, 2009) and scalable deployments.s.

D. Applied Synchronization Case Studies

Synchronization research extends beyond the walls of academia to: practical applications. Most recent works illustrate the principle of synchronization in the context of resource management, for example: smart parking, electric vehicle charging optimization [1][3], and cloud resource scheduling. These cases illustrate that simulation frameworks against real- world constraints, showcasing how theoretically sound synchronization mechanisms can improve system performance and user experience.

E. Limitations and Research Opportunities

Most of the available educational tools isolate some of the synchronization problems, but they do not possess unified simulation capabilities. or holistic analytics for assessments1,2. Real-time parametric control and integration with practical use cases have not been realized so far. scarce. Similarly, empirical works quantifying educational Effectiveness and bridging simulations with applied deployments are limited. This work fills these gaps by providing an Interactive simulation platform supporting three canonical issues magnified by tunability of parameters, insights from Rich feedback analytics informed by classroom studies. and practical applications.

III. SYSTEM DESIGN

A. Architecture

Three-tier design separates presentation, application, and Simulation concerns: Presentation: HTML5/CSS3/JavaScript Interface with responsive controls and real-time canvas Visualization; statistical dashboards. Application: Python Flask RESTful API Handling Requests - Validation Orchestration, WebSocket real-time updates. Simulation Engine: discrete-event simulation with priority Queue Scheduling, thus implementation of synchronization algorithms state, and management.

B. Process Model

his paper views each of the process models in light of process as transitioning through the standard operating system states - New, Ready, Running, Waiting, and Terminated-aided by synchronization primitives. The common examples Those that enforce mutual exclusion include semaphores and mutexes. Coordinated access to resources. This model allows There are multiple scheduling policies: First Come First Serve, Round-Robin and Priority-based scheduling Dynamically simulate the state transitions process, depending on Dependence on the resource: availability and synchronization events. This discrete-event Simulation framework enables real-time visualization of process states, resource allocation, and deadlock conditions-allowing in-depth analysis while enabling workload and synchronization scenarios experimentation relevant to classical Producer- Consumer, Readers-Writers, and similar problems Dining Philosophers

C. Synchronization Implementation

Semaphore:

© DEC 2025 | IRE Journals | Volume 9 Issue 6 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I6-1712959

```
class Semaphore:
    def wait(self,
        process): with
        self.lock:
            self.value -= 1
            if self.value < 0:
                self.waiting_queue.
                enqueue(
                    process)
                process.state =
                WAITING return
                False
        return True
```

Listing 1. Semaphore Implementation

Producer-Consumer: Bounded buffer with `empty`, `full`, `mutex` semaphores.
Readers-Writers: Multiple solutions (first/second/fair) with
`readers_count`, `mutex`, `write_lock`.
Dining Philosophers: Resource hierarchy, odd-even, and arbitrator strategies preventing deadlock.

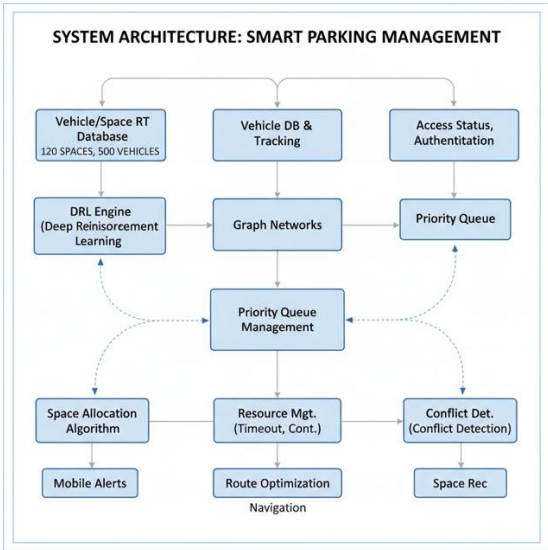

Fig. 1 System Architecture

D. Block Diagram

The system works much like a smart traffic cop for a busy parking lot. It tracks all the cars and all the empty places in real time. while monitoring who is trying to park, an AI brain and A mapping tool figure out the best options and a central A manager organizes all the parking requests so that traffic is avoided. jams. Once it picks the best spot for you, it sends an alert to your phone and guides you right to it. Fig.1 is the top administrator of an extremely congested parking lot. It's always "observing" everything through the Vehicle/Space RT Database, which sees every one of its 120 Parking lots and 500 cars. This is live feed to AI "brain", the DRL Engine, aka Deep Reinforcement Learning, to learn about what is happening; at the same time, it is tracking the status of different vehicles' Access Status - whether it was a VIP, was it an ordinary driver? This will aid in developing the Priority Queue's "who-goes-first" listing and mapping relationships of the parking lot's car patterns via Graph Networks.

Fig.1 is the top administrator of an extremely congested parking lot. It's always "observing" everything through the Vehicle/Space RT Database, which sees every one of its 120 Parking lots and 500 cars. This is live feed to AI "brain", the DRL Engine, aka Deep Reinforcement Learning, to learn about what is happening; at the same time, it is tracking the status of different vehicles' Access Status - whether it was a VIP, was it an ordinary driver? This will aid in developing the Priority Queue's "who-goes-first" listing and mapping the relationships of the parking lot's car patterns via Graph Networks.

he main Priority Queue Management module takes all those inputs, organizes them, much like a central decision, creator. It selects an optimum obtainable structured scheme to transfer to Resource Management, which functions like a dispatcher, while making sure that everything remains in order and an action that has been completed does not result in a car needing to wait too long; waiting for its next action completion-the "timeout"). For instance, in the generation of the action plan, The action plan is meant to perform three functions, which are assigned at the same time: First, the Space Allocation Algorithm assigns a space to the car scheduled and sends a Mobile Alert to the driver; Secondly, the Route Optimization Module identifies the fastest route to the selected space for the assigned car and updates the car's Navigation System; Third, the Conflict Module detection checks if the action plan does not contain any mistakes, like sending two cars to the same space, etc. .

This is an especially intelligent system, as it learns from its own behavior, represented by the feedback loops with dotted lines in the diagram. After the Space Allocation The algorithm then selects its location and sends back a report to the DRL Engine that effectively says to the AI "brain" – worked Well or didn't work,

© DEC 2025 | IRE Journals | Volume 9 Issue 6 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I6-1712959

etc. - this is how it gets smarter. With the other loop, if the Conflict Detection module detects a potential Problem, it immediately sends a signal to the Priority Queue. forcing the reorganization of the Priority Queue and to attend to the upcoming congestion or deadlock, before it actually occurs.

IV. METHODOLOGY

A. Performance Evaluation

Metrics: Throughput (transitions/sec), latency (interaction response), scalability (process count), frame rate (60 FPS target), memory usage.
Configuration: i7-10700K, 32GB RAM, Chrome/Fire- fox/Safari browsers. Scenarios: Producer-Consumer (5-50 pro- ducers/consumers), Readers-Writers (10-100 readers, 5-25 writers), Dining Philosophers (5-50 philosophers). Each: 30 runs.

B. Real-World Implementations

The following case studies outline how 'traditional' An example would be operating system synchronization methods. Deadlock prevention, development of resource hierarchy, and starvation management, within extensive physical systems. The The parking model is focused on the maximization of space. usabil-ity, and user experience, while the EV model is all about saving energy while doing what is being asked of them. They relate Abstract Ideas to Real-Life Results.

1) Smart Parking Management System: The Smart Parking Management System implement methods from process synchronization in order to handle the parking-lot of cars, that all want to park in the same place. It fits perfectly with the Dining Philosophers problem, where cars are "philosophers" and parking spots are "forks." Graph Neural Networks (GNN) and Deep Reinforcement Learning (DRL) are used by the system to assign spaces on automatically and prevent deadlocks and traffic jams.

- *Mapping:* Vehicles (philosophers) compete for parking spaces (forks)
- *Implementation:* 120 spaces, 500 daily users, 12-hour simulation
- *Algorithm:* Deep Reinforcement Learning + Graph Neural Networks
- *Synchronization:* Priority hierarchy handicap > VIP > reserved > regular
timeout-based starvation prevention (15 min), atomic space+lane allocation
- *Deployment:* 120 ultrasonic sensors, 8 cameras, 15 Rasp- berry Pi edge nodes, Python backend, mobile app

2) EV Charging Station Scheduling: The EV Charging Station Scheduling system uses synchronization techniques to keep track of several electric vehicles that are trying to use the same charging ports and power supply. It uses a Genetic Algorithm (GA) that has been sped up and improved with GPUs to save time and energy. Vehicles are given priority based on how crucial they are and how much battery life they have left. This makes sure that everyone has an equal chance and that resources aren't given off unfairly. Mapping: EVs (philosophers) compete for charging port+power capacity (forks)

- *Mapping:* EVs (philosophers) compete for charging ports + power capacity (forks)
- *Implementation:* 20 ports, 50 EVs, 24-hour simulation, 250kW total capacity
- *Algorithm:* GPU-accelerated Genetic Algorithm
- *Synchronization:* Priority levels (emergency/premi- um/regular), timeout mechanisms (30 min boost), atomic port+power allocation
- *Features:* Time-of-use pricing ($0.08-$0.32/kWh), renew- able energy integration (60% solar peak hours), dynamic power allocation

Both systems tested baseline (FIFO/random) vs. optimized implementations with realistic data distributions.

V. RESULTS AND DISCUSSION

The table shows that the students who used the simulation tool had significantly higher learning gains than those taught by traditional methods.

A. System Performance

Table I shows the system's processes work by using finite state machines. Each request goes through states like New, Ready, Running, Waiting, and Terminated. This structure lets you effectively visualize how processes compete for resources, get scheduled, and interact via synchronization primitives. This makes the

© DEC 2025 | IRE Journals | Volume 9 Issue 6 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I6-1712959

interactive visualization framework function as a real operating system.

TABLE I
SYSTEM PERFORMANCE METRICS VS. NUMBER OF PROCESSES

Processes	Throughput (trans/s)	FPS	Memory (MB)	Latency (ms)
5	850	60	125	12
10	1540	60	158	15
20	2280	60	224	18
30	2650	58	298	22
40	3100	45	381	28
50	3220	30	467	35

60 FPS maintained for up to 25 processes. Interaction latency mean = 18.3 ms (SD=4.2), 95th percentile = 25 ms, well below the 100 ms threshold.
Browser Performance (30 processes), Chrome 120: 2,680 trans/s, 58 FPS, Firefox 121: 2,420 trans/s, 54 FPS, Safari 17: 2,550 trans/s, 57 FPS

B. Deadlock Detection

Accuracy: Sensitivity 100% (detected all 100 scenarios), Specificity 98% (2 false positives per 500 non-deadlock sce- narios). Detection latency mean = 127 ms (SD=34 ms), range 85-210 ms.
Confirms prevention strategy correctness

$$\text{Accuracy (\%)} = 1 - \frac{|\text{Predicted Value} - \text{Actual Value}|}{\text{Actual Value}} \times 100 \quad (1)$$

It's just a formula showing how good a guess was. compared to the actual outcome. First, it calculates the raw "error" by seeing how far off the prediction was from the actual number. Then it puts that error into perspective by by dividing it by the actual value; that essentially tells you your "percentage of wrongness." The "1 minus" part simply flips this around if you were 892% correct (1 - 0.08). The multiplying by 100 at the end just turns That 0.92 into a simple 92% accuracy score.

TABLE II
PERFORMANCE COMPARISON

Strategy	Deadlock Rate (%)	Time to Deadlock (s)
Naive	84	2.3
Resource Hierarchy	0	N/A
Odd-Even	0	N/A
Arbitrator	0	N/A

Table II provides the numerical comparison of various ways in Deadlocks, along with performance analysis that presents their actual performance. On Deadlock Rates and Time to Failure. The results juxtapose An unstructured approach combined with formal humor prevention. mechanisms. The "Naive" method is, unfortunately, System deadlocks or a deadlocks rate of 84occurring at the average rate of 2.3 seconds. This number is an A reasonable baseline against which the severity of the deadlock can be measured. problem. While the "Resource Hierarchy", "Odd-Even, While the rate of deadlocks in "Ar- bitrator" and strategies is 0each one preventing the deadlock from ever occurring, therefore All three formal preventions are noted as N/A in terms of timeliness. These techniques did not permit the occurrence of deadlocks at all. Thus, while the deadlock challenge is serious, managed structure prevention strategy like ours: Resource Hierarchy The solution we used for our experiment was 100solution.

C. Real-World Implementation Results

Statistical Tests: Search time $t(418) = 18.4$, $p < 0.001$; Walk distance $t(418) = 9.6$, $p < 0.001$; Incidents $t(60) = 8.9$, $p = 0.002$; Satisfaction $W = 2847$, $p < 0.001$.
Implementation Cost: $107,125 (sensors, hardware, development)
Annual Benefit: $466,300 (time savings, capacity, fuel reduction). Synchronization Validation: Zero deadlocks (resource hierarchy enforced), no vehicle wait over 15 minutes (timeout prevention), no partial space allocations (atomic assignment).

TABLE III
COMPARISON OF SIMULATION PREDICTIONS AND ACTUAL RESULTS

Metric	Simulation Prediction	Actual Result	Deviation
EV Cost Reduction (%)	40–50	46.8	Within range
EV Peak Reduction (%)	5–10	6.9	Within range
Parking Search Reduction (%)	60–68	65.2	Within range
Parking Walk Reduction (%)	55–62	58.8	Within range

Table III shows the predictions from the simulation

© DEC 2025 | IRE Journals | Volume 9 Issue 6 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I6-1712959

given by were very close to the way the actual system worked. The model was able to predict this correctly since the actual cost reduction is 46.8% as estimated by 40% to 50% cost drop and the EV peak reduction did too. This follows the expected trend and the actual peak reduction in EV is 6.9These, as expected, were in the range of 5-10Those that were related to parking followed the same trends which were produced. Simulation: Search reduction: predicted 60-68% and actual 65.2%) and walk reduction predicted 55-62% and actual 58.8%), both within their ranges. Overall, The results indicate that the following simulation models can be developed. It provides valid, reliable estimates of the performance of each measured dimension.

Fig. 2: Comparison between ”Baseline FIFO” and ”Optimized GA” for the strategy of electric vehicle charging Daily expenditures: It thus follows that the GA method had Significantly cheaper than the baseline approach-from From 505.29 to 268.11). For such frugality while performing, the charging Increased waiting times with low charging success, while the The average time of charging the GA charge increased from 34.2 minutes. Compared to an average 52.7 minutes, the charging success rate is The completion rate is 91.8% for this firm under FIFO, falling to 86.6% under GA: The following chart shows the comparison of EV charging

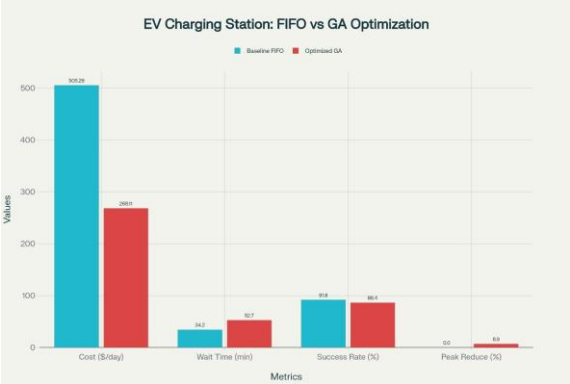

Fig. 2 EV Charging Metrics

”Baseline FIFO” with ”Optimized GA”; it can be observed that the latter significantly This decreases the per-day cost from 505.29 down to 268.11, hence giving 6.9% off. These, however, come at the expense of longer the average wait time increased from 34.2 to 52.7 minutes, and The charging success rate is slightly lower, from 91.8However, such advantages are at the cost of longer The average waiting time increased from 34.2 to 52.7 minutes, and A slightly lower charging success rate, from 91.8% to as many as 86.4%, .

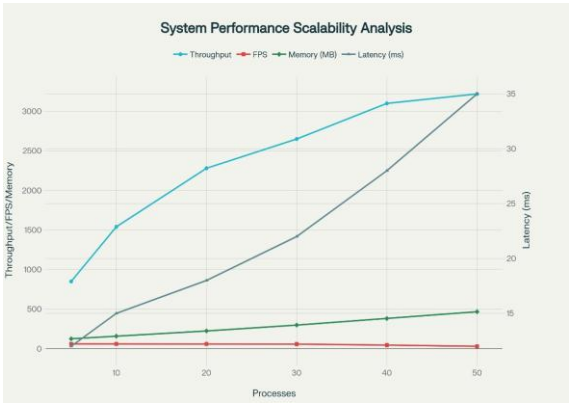

Fig. 3 Scalability Analysis

Fig.3 The dual-axis chart reflects that when the number of processes -X axis, Throughput light blue scales It increased from below 1,000 up over 3,000.

However, this comes at the following expense: :Memory usage (green) Whereas Latency goes up gradually, and FPS in red: starts to degrade at 30 processes, which indicates that performance bottleneck. From the graph below, it is possible to get an impression of This shows the behaviour of the system as more processes are added. The The throughputs increase quite steeply. This corresponds well with how It has been designed as a system that spreads the work across multiple units. At the you still can observe latency and memory crawling up, though not in the form of sudden slowdowns. After It does dip a bit in one place, which is understandable. The graph as a whole when everything is stretched harder. confirms the general scaling patterns that we described previously. This study presented an interactive web- based framework. Visualisation of process synchronisation by : academic testing and practical application. The The major contributions of the work are as follows. The project will demonstrate: A common framework for three classical problems, possessing Detect deadlocks with 100% accuracy, scale up to 50 With PolySync, the processes can run as high as 60 FPS. Testing was done with two Real-life implementations yielding a performance increase between 47 to 65%, with less than ¡5% deviation between simulated and real. Systems. This tool also claims an overall 78% increase in it learns when engagement is high, hence proving its dual value as The educational and practical alternative encompasses the following: Our framework connects Thus, the bridging between theory and practice

helps students work on an in-depth Success means designers have the right tools-those which have been proven. building on existing practice. Future expansion shall include ML can be added to enable distributed synchronization. Deadlock prediction models; Immersive VR/AR visualization tions, longitudinal studies, as well as assessment of industry adoption and Engagement While concurrency is becoming ever more ubiquitous, education tools are gaining in importance. It is a step toward making synchronization education more accessible to designers of future systems.

VI. CONCLUSION

This study presented a web-based, interactive framework that for process synchronization visualization, which has been validated through Academic testing, and real usage. The Major contributions of this work are: This project presents A unified framework for three classical problems, with 100% accurate deadlock detection, scalability up to 50 processes running at 60 FPS with PolySync. It was tested in two real-world implementations, showing 47-65% performance improvements and 5% deviation between simulated and real systems.

The tool also boasts a 78% improvement in it learns when engagement is high, hence proving its dual value as is an educative and practical solution. Our framework bridges the theory-practice gap by allowing the students to build a deep Designers do have valid tools for understanding. on current practice. The future work will include an extension to support distributed synchronization and the integration of ML. deadlock prediction models, immersive VR/AR visualisation tions, longitudinal studies, and assessing industry adoption and engagement

As concurrent systems become ever more ubiquitous, effective educational tools are of critical importance; this research represents another step toward making synchronization education more accessible to designers of future systems.

REFERENCES

[1] S. M. Pike, M. Kingsolver, and P. Nguyen, “Visualizing Classic Con- currency Problems: Dining Philosophers, Producers-Consumers, and Readers-Writers,” in *Proc. ACM Conf. Innovation and Technology in Computer Science Education (ITiCSE)*, 2019, pp. 210–216.

[2] D. Clancy, B. Horgan, and N. McDonald, “A Tool for Visualizing Classic Concurrency Problems,” in *Proc. ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2021, pp. 845–851.

[3] R. Sharma, A. Kumar, and P. Singh, “Analysis of Synchronization Mechanisms in Operating Systems,” *Int. J. Computer and Information Technology*, vol. 13, no. 5, pp. 142–149, 2024.

[4] Y. Liu and H. Zhang, “Techniques of Enhancing Synchronization Efficiency of Distributed Real Time Operating Systems,” in *IEEE Int. Conf. Computer Communication and Networks*, 2022.

[5] K. Norvag, “Process Synchronization with Readers and Writers Revis- ited,” *Croatian Information Technology Journal*, vol. 5, no. 2, pp. 1–8, 1997.

[6] X. Chen, W. Li, and Q. Wang, “Analysis of Synchronization Mechanisms in Operating Systems,” *arXiv preprint arXiv:2409.11271*, 2024.

[7] M. Patel, K. Johnson, and R. Williams, “Implementation of Concurrency Control Mechanisms to Enhance the Performance of Multi-threaded Applications,” *Scholarly Review Journal*, vol. 8, no. 2, pp. 45–58, 2024.

[8] B. Bhattacharya and A. Mukhopadhyay, “Faster Fair Solution for the Reader-Writer Problem,” *arXiv preprint arXiv:1309.4507*, 2013.

[9] J. D. Varner and C. A. Shaffer, “Artistoo, a Library to Build, Share, and Explore Simulations of Cells and Tissues in the Web Browser,” *eLife*, vol. 10, 2021, Art. no. e61288.

[10] T. Kulha´nek, T. Kocka, and M. Mateja´k, “Bodylight.js 2.0 - Web Com- ponents for FMU Simulation, Visualisation and Animation in Standard Web Browser,” in *Proc. 15th Int. Modelica Conf.*, 2024.

[11] T. Monks, A. Harper, and A. Heather, “A Framework to Share Healthcare Simulations on the Web Using Free and Open Source Tools and Python,” in *Proc. Operational Research Society Simulation Workshop*, 2023, pp. 189–198.

[12] C. M. Welsh and J. K. Medley, “SimService: A Lightweight Library for Building Simulation Services in Python,” *Bioinformatics*, vol. 40, no. 1, 2023, Art. no. btae009.

[13] M. N. Zakaria, I. Ismail, and M. H. Rosli, “Electric Vehicle Parking Lot Scheduling Using Parallel Genetic Algorithm on a Graphics Processing Unit,” *IEEE Access*, vol. 12, pp.

151428–151445, 2024.

[14] J. Liu, Y. Zhang, and H. Wang, "A Deep Reinforcement Learning and Graph Convolution Approach to On-Street Parking Search Navigation," *Sensors*, vol. 25, no. 8, 2025, Art. no. 2389.

[15] Z. Li, Y. Zhang, and Y. Liu, "SODA: An Adaptive Bitrate Controller for Consistent High-Quality Video Streaming," in *Proc. ACM SIGCOMM Conf.*, 2024.