IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

OS Simulator: An Interactive Tool for Understanding Operating System Concepts

Mayank Pal[1], Swati Nagar[2], Vandana Sharma[3], Arjoo Jain[4]
Student[1],
Assistant Professor[2]-[4]
Sunder Deep Engineering College (SDEC), Ghaziabad, Uttar Pradesh, India
palmayank510@gmail.com[1], swati.nagar@sunderdeep.ac.in[2],
hodcs@sunderdeep.ac.in[3], arzoo.jain@sunderdeep.ac.in[4]

Abstract: *In Computer Science study, one of the main subjects is Operating System, and many students are afraid of concepts of Operating System. OS covers mainly, Process Management(Scheduling, Creation & Termination, IPC, Deadlock), Memory Management(MFT, MVT, Paging, Segmentation), IO Management. But Students only study them on paper for exams or marks, that's why, we have developed a Desktop Application "OS Simulator", which provides Animations, Graphs, Tables etc, so students can understand the whole concept easily and deeply.*

As OS is one of the important parts of a Computer System, we need to understand it deeply in order to develop Application Software, Solving Problems, or research in Computer Science.

OS Simulator have 25+ Algorithms like, Scheduling Algorithms(FCFS, SJF, SRTF, RR etc), Disk Scheduling(FIFO, SCAN, C-SCAN, LOOK, C-LOOK, SSTF), Deadlock Handling(Detection and Recovery, Banker's Algorithms), Process State Visualization and System Call Execution Flow.

Students can explore algorithms and watch them how they actually work. This Python Application helps students and teachers. Students can learn, explore and practice, while Teachers can use it as a Tool for Teaching Concepts, Solving Questions with Gantt Charts, Resource Allocation Graph, Banker's Table etc.

From a software engineering perspective, application follows strict MVC(Model View Controller architecture). Means "Model handles data and logic. View handles display. The controller connects them." The goals of this paper are to showcase the demonstrative capabilities of the tool for instruction, share student experiences in developing the engine underlying the simulation, and to inspire its use by other educators..

Keywords: OS Simulator, CPU Scheduling, Memory Management, Deadlock Handling, Disk Scheduling, MVC Architecture, Educational Tool, Banker's Algorithm

I. INTRODUCTION

An Operating System acts as an interface between the computer and the user, and its study is an important part of Computer Science education. It includes topics such as process management, memory management, deadlock handling, I/O scheduling, and file systems. While these topics are essential, many students find them difficult to understand because they require not only theoretical knowledge but also an understanding of how algorithms behave with different inputs.

In most classrooms, these concepts are taught using textbooks, pseudocode, and static diagrams. Although this helps in understanding the basic steps of algorithms, it does not show how they behave dynamically. As a result, students are often able to solve exam questions but struggle to understand the actual working of these algorithms. Even small changes in input, such as time quantum in Round Robin or number of frames in page replacement, can significantly affect the outcome, which is not always clear from static examples.

DOI: 10.48175/IJARSCT-34069

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

To improve this situation, several visualization tools have been developed in Computer Science education. These tools use animations and step-by-step execution to help students understand algorithms better. However, most existing tools either focus on a single topic or require specific platforms and dependencies, which limits their accessibility for students.

In this paper, a standalone desktop-based Operating System simulator is presented, which combines multiple OS concepts into a single platform. The simulator is developed using Python and provides an interactive environment where users can explore different algorithms and observe their behavior. The goal is to make OS concepts easier to understand through practical visualization rather than only theoretical study.

In our OS Simulator, We've multiple modules like Process Scheduling; Memory Management concepts like, MFT, MVT, Paging & Segmentation; Deadlock Detection & Recovery using RAGs; Banker's Algorithm; Disk Scheduling(FIFO, SCAN, C-SCAN, LOOK, C-LOOK, SSTF); Page Replacement Algorithms; And Process State Visualization and System Call Execution Flow.

The Complete Project can work on any Computer; it only needs Python Installed. The architectural view of the underlying application follows strict MVC(**Model View Controller architecture).** Means "Model handles data and logic. View handles display. The controller connects them."

The graphical simulation tool is implemented using Python 3 with the Tkinter GUI toolkit. The simulator requires no installation, no network access, and no external dependencies beyond a base Python distribution. Instructors can use it for live demonstrations of OS concepts in classes, while students can use it to explore the concepts.

II SIMULATION DETAILS

The OS Simulator is developed to provide an interactive environment where users can experiment with different Operating System algorithms. The simulation begins by taking user input such as process details, arrival time, burst time, priority, time quantum for CPU Schduling Algorithms. Frames/Page details for Paging, memory blocks detail for MFT & MVT using combinations of First Fit, Worst Fit & Best Fit, Disk request sequence, or resource allocation values, depending on the selected module.

Once the input is provided, the simulator processes the data using the algorithm. Each module runs independently and follows its own logic. For example, in CPU scheduling, processes are arranged and executed based on the selected algorithm such as FCFS, SJF, SRTF, Priority or Round Robin. Similarly, in memory management, allocation strategies or page replacement techniques are applied step by step like FIFO, LRU & Optimal.

Our OS simulator provides visual output to help users understand the execution flow with Charts, Graphs and Tables. This includes Gantt charts for scheduling, tables for memory allocation, graphs for performance comparison, and structured views for algorithms like Banker's Algorithm and RAGs for deadlock detection. These visual elements update according to the module, provided input and selected algorithm.

Users can observe and analyze how changes in input affect the output. This allows comparison and their respective behaviour, between different algorithms under similar conditions. The simulator is designed in such a way that users can run preset test cases and explore different scenarios without restarting the application.

Each module is connected to a separate logic layer, which ensures that the simulation runs independently of the user interface. This makes the system more flexible and easier to maintain. The main aim of the simulation is to provide a clear and practical understanding of how Operating System algorithms behave in real situations.

III SYSTEM ARCHITECTURE

OS Simulator follows MVC architecture: a model/logic layer of pure-Python modules containing all algorithmic computation, and a view/presentation layer of Tkinter UI modules that visualise results and handle user interaction. The two layers communicate exclusively through well-defined return types — lists of step dicts, Gantt tuples, segment lists — with no UI imports in the logic layer and no algorithm implementation in the UI layer.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

3.1 Project Structure

The project comprises 21 Python source files totalling 8,551 lines. The top-level directory contains main.py (the application entry point) and two subdirectories:

Modules/ — one logic file and one ui file per subsystem, plus dashboard.py, process_states.py, and quiz_ui.py.

Utils/ — theme.py, a single module providing the entire design system: colour palettes, font constants, widget factory functions (lbl, btn, card, entry, combobox), the AlignedTable canvas widget, Tooltip, and input validation helpers.

3.2 main.py — Application Shell

main.py creates the root Tk window, builds the header bar with a live dark/light theme toggle, constructs a ttk.Notebook with one tab per module, builds the status bar showing the active module name and timestamp. The theme toggle calls refresh_theme(), which walks the entire widget tree using winfo_class() to re-colour every tk.Frame, tk.Label, tk.Button, and tk.Entry without destroying or recreating any widget, preserving all user-entered data across theme changes.

3.3 Design System (theme.py)

All visual constants are defined in utils/theme.py. Two palettes — dark and light — are stored in a THEMES dictionary; the active palette T is a plain dict shared by all modules. Widget factories read from T at call time, so theme changes propagate automatically. The file also provides validate_int_list() and validate_positive_int() helpers used by all input modules to produce specific, user-readable error messages (e.g., 'Block sizes: 'abc' is not a valid integer') instead of bare Python exceptions.

3.4 Separation of Concerns

The logic/UI separation provides several practical benefits. Logic modules can be unit-tested without a display server — critical for automated verification in CI environments. UI modules can call logic functions with any input and visualise the result without needing to understand the algorithm. New algorithms can be added by writing a function in the logic file and wiring it into the algorithm dispatch dict in the UI file — typically fewer than ten lines of change. The scheduling_logic.py module, for example, was extended with mlfq() and priority_with_aging() without modifying a single line of the UI's Gantt drawing code.

IV CPU SCHEDULING MODULE

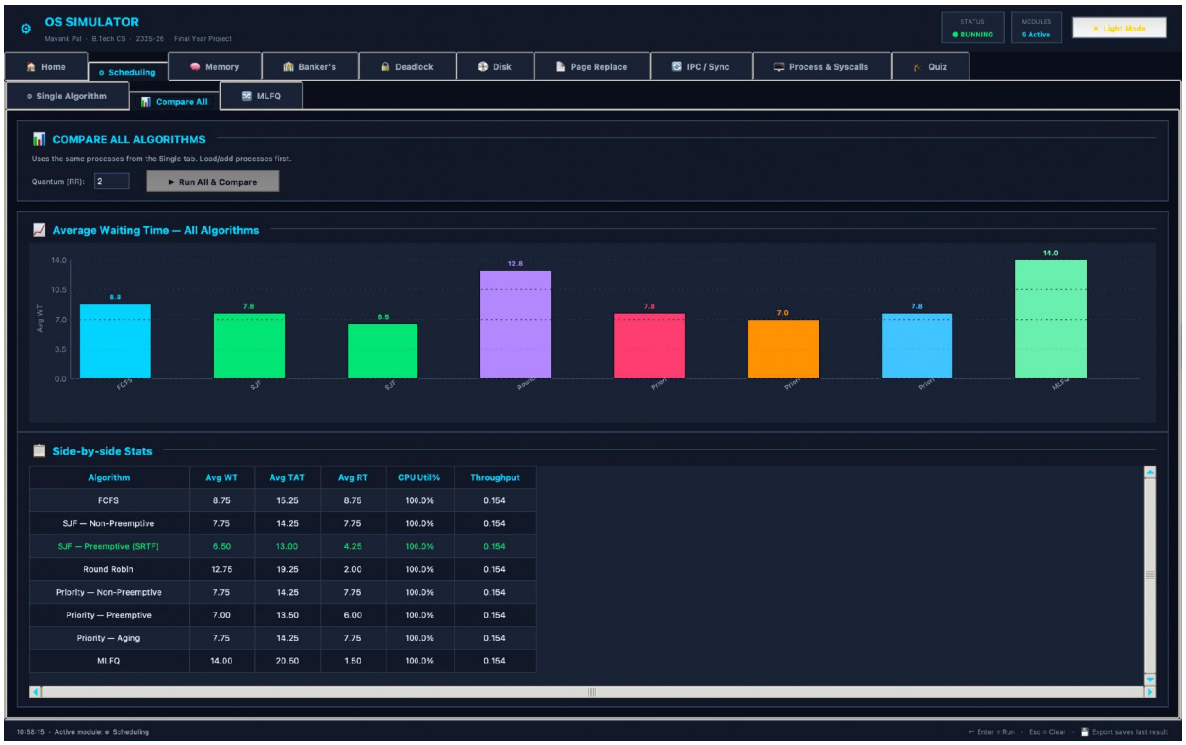
The scheduling module implements eight algorithms spanning non-preemptive, preemptive, priority-based, and feedback-queue categories. All algorithms share an identical interface: they accept a list of process dicts (pid, arrival, burst, priority) and return a Gantt chart as a list of (pid, start, end) tuples and a result list augmented with completion time, turnaround time, and waiting time.

4.1 Implemented Algorithms

Algorithm	Type	Time Complexity	Known Weakness
FCFS	Non-preemptive	O(n log n)	Convoy effect
SJF — Non-Preemptive	Non-preemptive	$O(n^2)$	Starvation of long jobs
SRTF (SJF Preemptive)	Preemptive	$O(n^2)$	High context-switch cost
Round Robin	Preemptive	O(n)	Quantum-sensitive

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

Priority — NP	Non-preemptive	$O(n^2)$	Starvation without aging
Priority — Preemptive	Preemptive	$O(n^2)$	High context-switch cost
Priority + Aging	Non-preemptive	$O(n^2)$	Aging interval tuning
MLFQ	Preemptive	$O(n \cdot Q)$	Parameter complexity



4.2 MLFQ Implementation

The Multi-Level Feedback Queue uses three levels: Q0 (Round Robin, configurable quantum q0), Q1 (Round Robin, quantum q1), and Q2 (FCFS). New processes enter Q0. A process that exhausts its quantum without completing is demoted one level. Aging is applied every aging_threshold time units: any process waiting in Q1 or Q2 is promoted one level, guaranteeing eventual service for all processes. The dedicated MLFQ tab displays a colour-coded Gantt chart (green = Q0, yellow = Q1, red = Q2) and a demotion/promotion log table.

4.3 UI Features

Animated Gantt chart with Play / Step / Stop controls and adjustable speed slider.

CPU utilisation bar showing the fraction of time the CPU was busy vs. idle.

Side-by-side Compare All tab: runs all eight algorithms on the same process set and renders an Average Waiting Time bar chart with colour-coded bars and a full stats table.

Export Report: saves a formatted plain-text report with algorithm name, process table (PID, Arrival, Burst, CT, TAT, WT, RT), averages, and Gantt sequence.

Preset test cases covering FCFS basics, SJF, Round Robin, priority mix, and CPU idle gaps

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

V. MEMORY MANAGEMENT MODULE

The memory module covers the full progression of memory allocation strategies taught in undergraduate OS courses: contiguous allocation policies, fixed and variable partitioning, paging, and segmentation. It is the largest module by line count (867 lines of UI, 384 lines of logic) reflecting the breadth of memory management topics.

5.1 Contiguous Allocation: First, Best, and Worst Fit

Three hole-selection policies are implemented for contiguous memory allocation. First Fit allocates the first hole large enough, offering O(n) search. Best Fit selects the smallest sufficient hole, minimising internal fragmentation at O(n) cost. Worst Fit selects the largest hole, intentionally maximising the leftover fragment. All three are visualised as a horizontal memory map with colour-coded blocks, fragmentation shading, and a utilisation pie chart.

5.2 MFT and MVT

Memory with Fixed-size Tasks (MFT) divides physical memory into equal-size partitions at system initialisation. Internal fragmentation arises whenever a process is smaller than its partition. Memory with Variable-size Tasks (MVT) allocates exactly the memory a process requests, eliminating internal fragmentation but introducing external fragmentation as processes terminate and leave holes.

5.3 MVT Compaction

A dedicated Compact Memory button simulates memory compaction for the MVT strategy. The simulator slides all allocated segments to base address 0, packs them contiguously, and merges all free holes into a single large free block. The output shows: which segments moved (and their old vs. new base addresses), the total data movement cost in KB (the compaction overhead), and confirmation that external fragmentation has been reduced to zero. The memory map and pie chart update to reflect the post-compaction state.

DOI: 10.48175/IJARSCT-34069

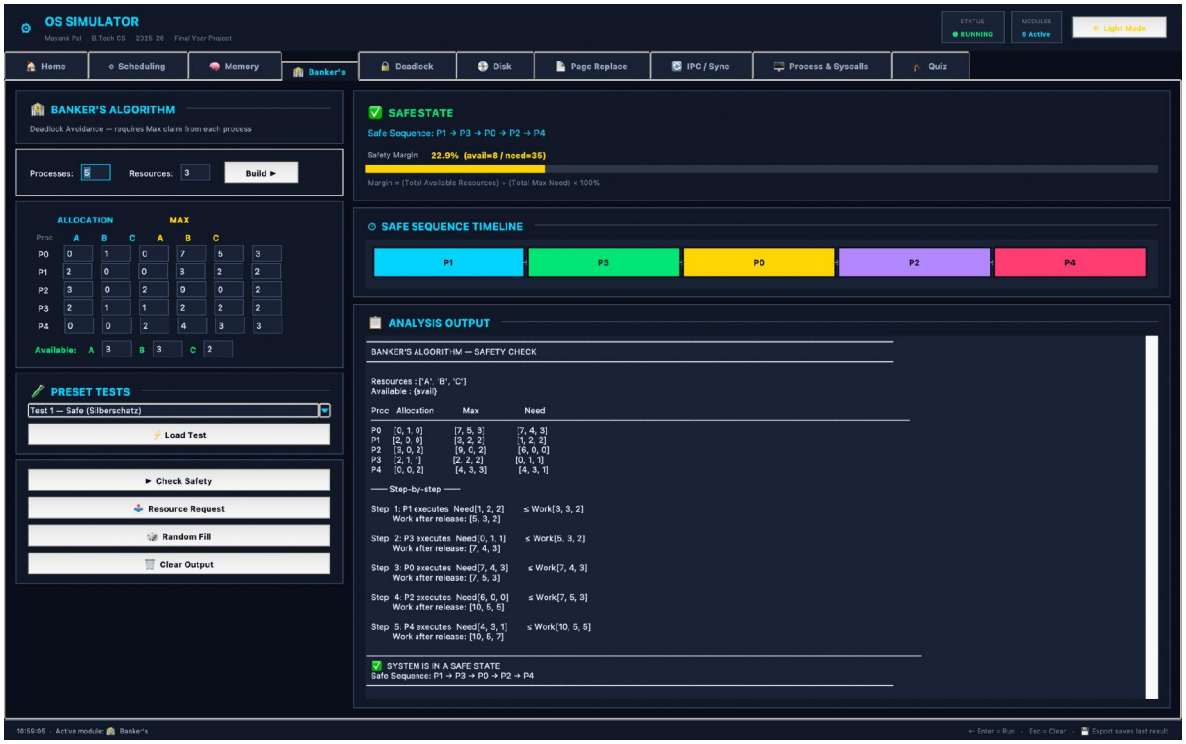
IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

5.4 Paging and Segmentation

The paging implementation supports both Sequential and Random frame allocation modes. Sequential mode assigns frames in order (0, 1, 2, …), making the page table easy to follow. Random mode assigns frames from a shuffled free list, demonstrating the key benefit of paging: process pages need not occupy contiguous physical memory. A full page table is displayed for each process. Segmentation divides a process into named variable-size segments (code, data, stack, heap) with a segment table showing base, limit, and status.

VI. BANKER'S ALGORITHM MODULE

The Banker's Algorithm, developed by Edsger Dijkstra [12], is the canonical deadlock-avoidance algorithm. The simulator implements both the Safety Algorithm (which determines whether the current state is safe by finding a safe sequence) and Resource Request Simulation (which tests whether granting a specific request would leave the system in a safe state).



6.1 Safety Margin Meter

A novel UI feature added in v9 is the Safety Margin Meter: a colour-coded progress bar displayed immediately after each safety check. The margin is defined as:

Safety Margin (%) = (Sum of Available Resources) / (Sum of Total Remaining Need) × 100

The bar is green when margin ≥ 50%, yellow when ≥ 20%, and red below 20%. This gives students an immediate intuition for how close the system is to an unsafe state — a concept that is difficult to convey through the binary safe/unsafe verdict alone.

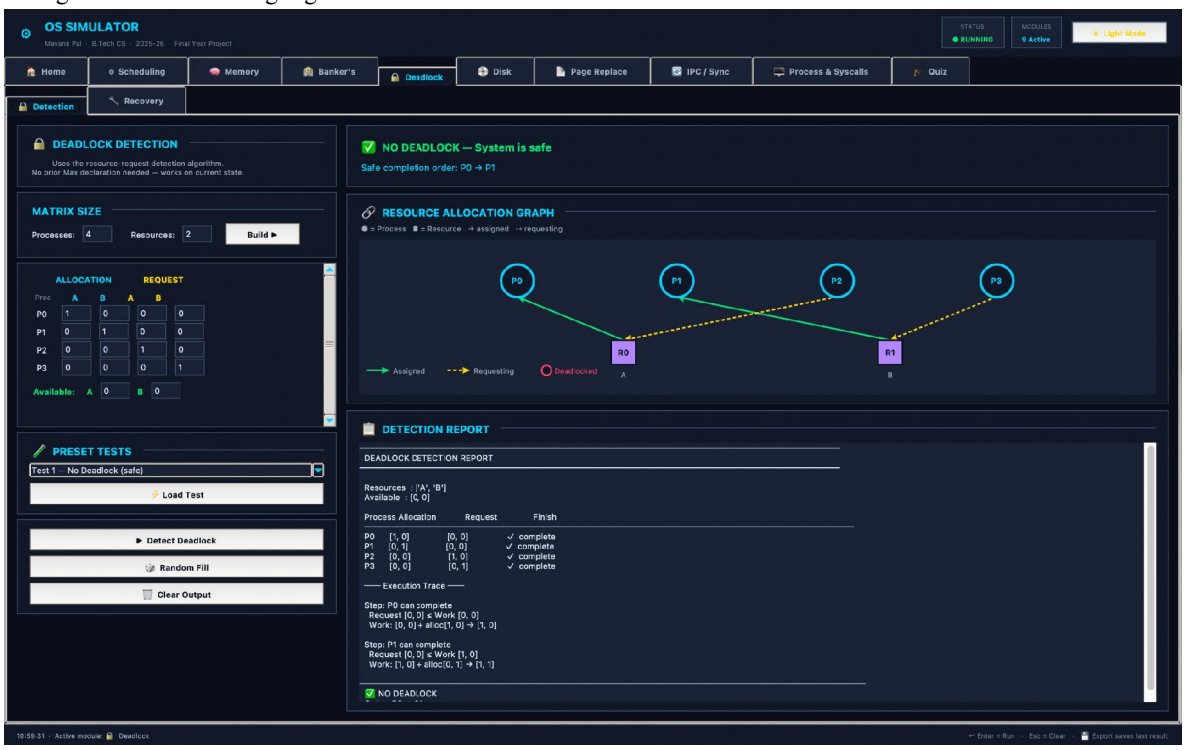
IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

6.2 Input Validation

The matrix entry system validates that Allocation[i][j] ≤ Maximum[i][j] for all i, j, and that no value is negative, producing specific cell-level error messages. The Resource Request dialog validates that the request does not exceed the process's Need vector or the currently Available resources before running the simulation.

VII. DEADLOCK DETECTION MODULE

The deadlock module visualises the Resource Allocation Graph (RAG) and implements detection and recovery. The RAG is drawn as an interactive canvas: process nodes are circles, resource nodes are rectangles, assignment edges point from resources to processes, and request edges point from processes to resources. Deadlocked processes and the cycle forming the deadlock are highlighted in red.



7.1 Detection Algorithm

The detection algorithm uses a work vector approach identical in structure to the Banker's Safety Algorithm but applied to the current allocation without the maximum claim matrix. It identifies all processes that cannot complete given current resources and marks them as deadlocked. The output includes which processes are involved and what resources they are waiting for.

7.2 Recovery Strategies

Process Termination: Terminate all deadlocked processes at once, or one at a time in priority order until the deadlock is broken.

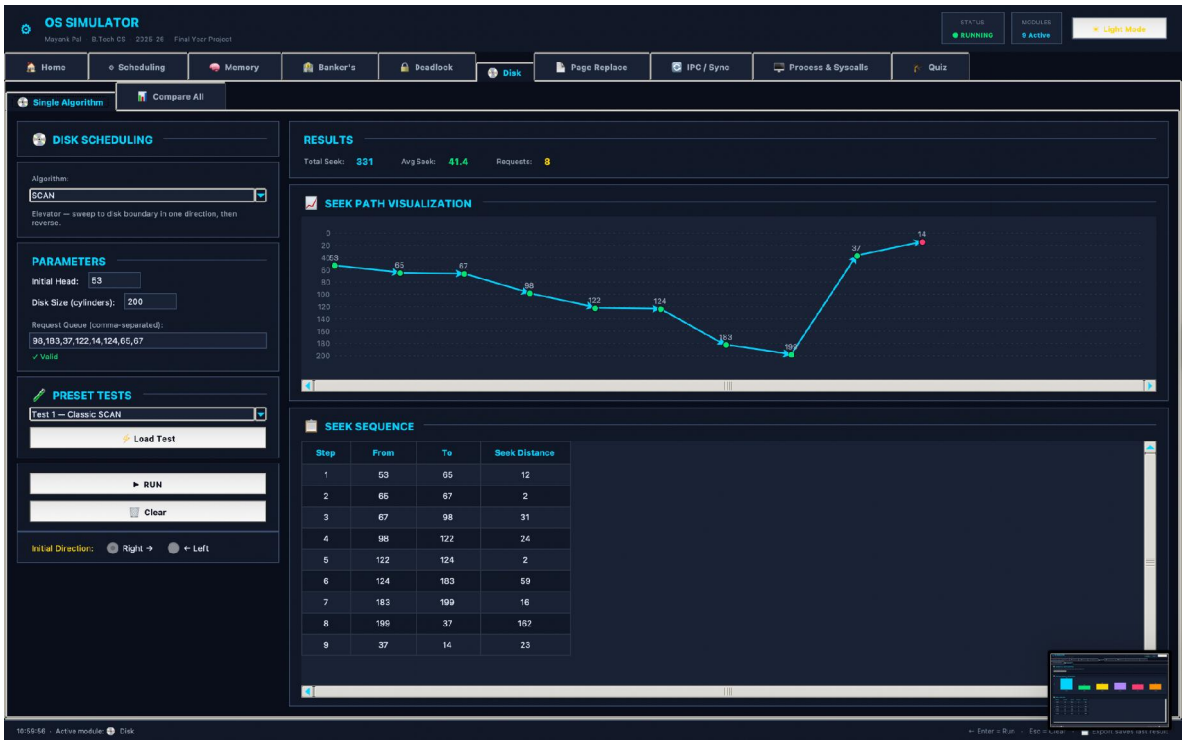
Resource Preemption: Forcibly reclaim a resource from a victim process and reallocate it to unblock the cycle.

Rollback: Roll a process back to a safe checkpoint, releasing its resources in the process.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

VIII. DISK SCHEDULING MODULE

The disk scheduling module implements six algorithms that minimise seek time — the dominant cost in traditional hard disk I/O. Each algorithm is visualised as a seek path chart: a 2D plot with time on the x-axis and cylinder number on the y-axis, with arrows tracing the head movement and dots marking each serviced request.



8.1 Algorithms and a Corrected SCAN Bug

A significant bug was identified and corrected in the SCAN implementation. The original conditional guard prevented the head from sweeping to the disk boundary when no requests existed in the sweep direction, causing SCAN to behave identically to LOOK in those cases. The corrected implementation always appends the boundary cylinder (0 or disk_size − 1), faithfully modelling the physical constraint that SCAN's head travels to the end of the disk before reversing — the property that distinguishes it from LOOK

Algorithm	Strategy	Advantage	Weakness
FCFS	Arrival order	Simple, fair	High seek distance
SSTF	Nearest request first	Low average seek	Starvation possible
SCAN	Sweep to boundary, reverse	Fair	End cylinders wait longer
C-SCAN	Sweep to boundary, jump to 0	Uniform wait time	Jump-back cost
LOOK	Sweep to last request,	Efficient	Non-uniform wait

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

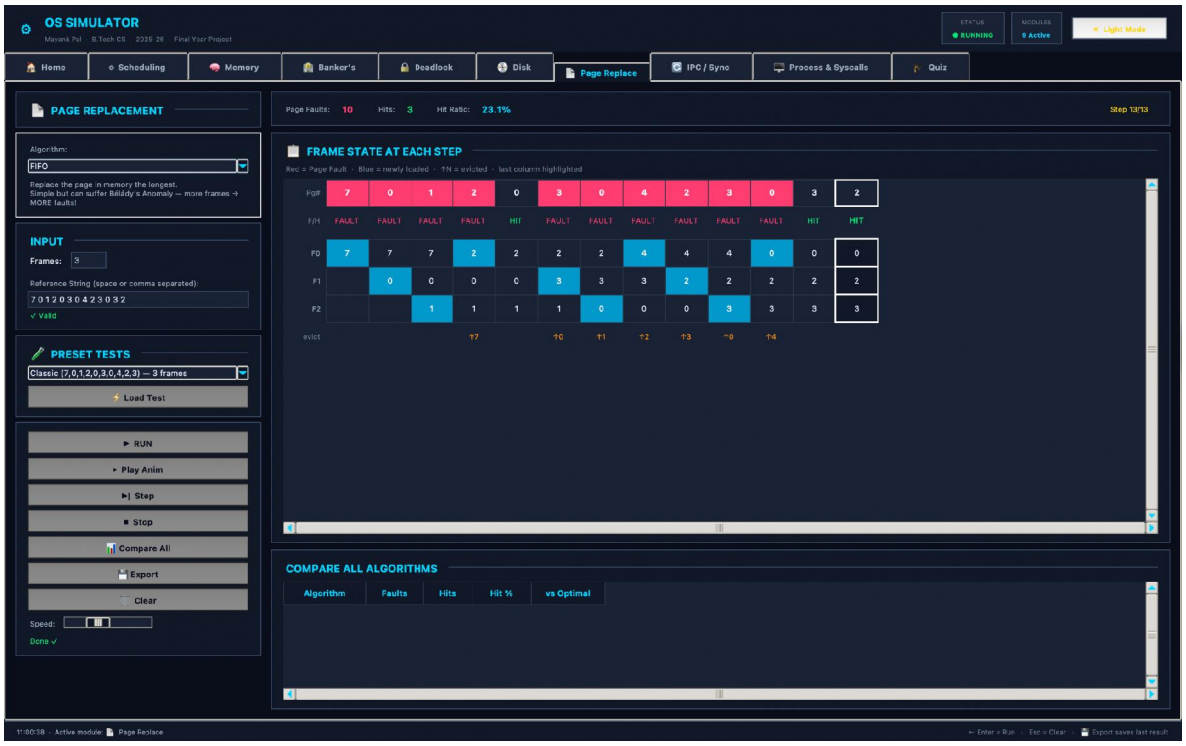
	reverse		
C-LOOK	Sweep to last request, jump back	Good uniformity	Jump-back cost

8.2 Compare All Tab

The disk module includes a Compare All tab that runs all six algorithms on the same head position, disk size, and request queue. A bar chart plots total seek distances side-by-side, and a table shows total seek, average seek per request, and an efficiency score (percentage of minimum possible seek distance achieved).

IX. PAGE REPLACEMENT MODULE

The page replacement module visualises the frame state at every step of the reference string as a scrollable canvas table. Red column headers mark page faults; blue cell highlights mark newly loaded pages; orange ↑N annotations mark evicted pages. Play, Step, and Stop animation controls allow students to work through the algorithm at their own pace.



9.1 Algorithms

FIFO — evicts the page loaded longest ago. Simple but suffers from Bélády's Anomaly.

LRU — evicts the least recently used page. Near-optimal, has the stack property.

LFU — evicts the least frequently used page, with load-time tie-breaking. Vulnerable to frequency pollution from early heavy use.

Optimal (Bélády) — evicts the page whose next use is farthest in the future. Theoretical lower bound, used as a benchmark.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

9.2 Bélády Anomaly Auto-Detection

When FIFO is selected and the algorithm is run, the simulator automatically tests whether increasing the frame count by one would produce more faults. If so, an orange warning banner appears stating the exact fault counts for n and n+1 frames and explaining that this is Bélády's Anomaly — the counterintuitive property that FIFO can produce more faults with more frames. This auto-detection makes a conceptually important but easy-to-miss phenomenon visible without requiring the student to run the experiment manually.

9.3 Compare All

The Compare All button runs all four algorithms on the current reference string and frame count, displaying fault count, hit count, hit ratio percentage, and a 'vs Optimal' delta for each. The row with the fewest faults is highlighted green. The Bélády check is also run automatically during comparison.

X. PROCESS STATES AND SYSTEM CALLS MODULE

10.1 Five-State Process Model

The Process States tab draws a responsive, interactive five-state diagram (New, Ready, Running, Waiting, Terminated) with labelled transition arrows. Clicking any state circle displays a description of that state; clicking a transition arrow displays the event that causes it. The lifecycle simulation generates a multi-process state sequence and animates it: as each transition fires, the corresponding state circle in the diagram glows with the state's colour and the info panel updates with the transition reason — making the diagram dynamic rather than a static reference.

10.2 System Call Flow Animation

Selecting any system call from the searchable reference list (covering six categories: Process, File, Memory, IPC, Device, and Network) triggers a six-phase animation on a canvas showing the user-mode to kernel-mode transition:

User code running — user box highlighted.

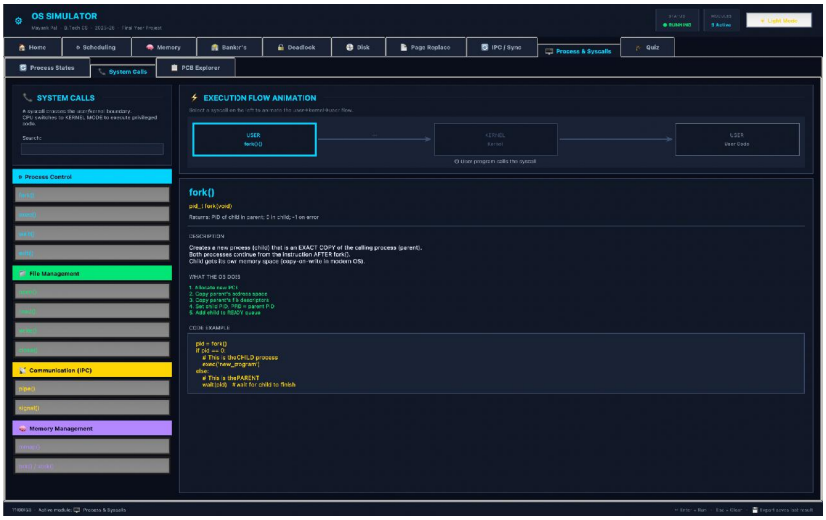
Syscall instruction issued — TRAP arrow lights up.

CPU switches to kernel mode — kernel box glows red.

Kernel executes the privileged operation.

Kernel returns result — return arrow lights up.

The user process resumes with a return value. Each phase displays a subtitle explaining what the CPU is doing at that moment. The loop repeats automatically until a new syscall is selected.



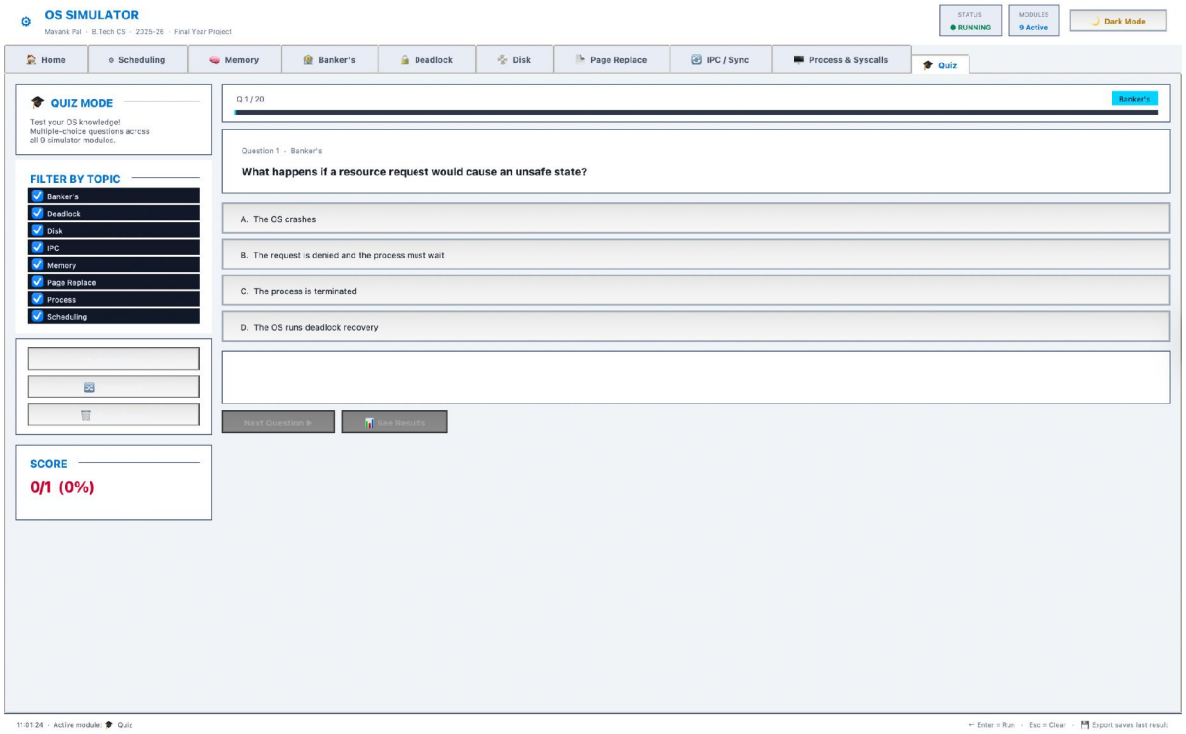
IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

10.3 PCB Explorer

The PCB Explorer tab presents a table of all PCB fields (PID, PPID, state, program counter, stack pointer, priority, CPU time, memory maps, open file descriptors, pending signals) alongside a live simulated PCB snapshot and a step-by-step context switch explanation with cost commentary

XI. QUIZ MODULE

The Quiz module provides 42 four-option multiple-choice questions spanning all nine topic areas. Questions are drawn from and reviewed against Silberschatz et al. [13], Tanenbaum [14], and Galvin [15] to ensure correctness and curriculum alignment.



Topic	Questions	Key Concepts Tested
CPU Scheduling	8	FCFS convoy effect, RR quantum effects, MLFQ structure
Memory Management	5	Fragmentation types, paging properties, compaction
Banker's Algorithm	4	Safe state, Need matrix, request denial logic
Deadlock	4	Coffman conditions, RAG cycles, recovery strategies
Disk Scheduling	5	SCAN vs LOOK, starvation, C-SCAN uniformity
Page Replacement	5	Bélády's anomaly, stack property, LFU weakness

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

IPC	4	Semaphore roles, deadlock prevention in Dining Philosophers
Process & Syscalls	5	State transitions, PCB, fork/exec, mode switching
General	2	Complexity comparisons, OS design principles

XII. USER STUDY

12.1 Participants and Procedure

A user study was conducted with five B.Tech Computer Science students, having completed one semester of OS theory coursework but with no prior exposure to interactive OS simulation tools. The study was conducted as a structured 30-minute session in which each participant was given an introductory walkthrough of the application interface, then allowed free exploration for 30 minutes, followed by a 15-minute post-session questionnaire.

12.2 Findings: Conceptual Clarity

All five participants reported improved understanding of algorithm behaviour after interacting with the simulator. Qualitative feedback highlighted several specific benefits:

The animated Gantt chart in the scheduling module made preemption and context switching concrete in a way that static textbook diagrams had not. One participant noted: "I finally understood why SRTF gives better average waiting time — I could see each preemption happening step by step."

The Bélády anomaly banner was singled out by three participants as a surprising and memorable moment. One wrote: "I knew Bélády's anomaly existed from lectures but I had never actually seen it happen. The banner showing 3 frames giving 9 faults and 4 frames giving 10 faults made it feel real."

The safety margin meter in the Banker's Algorithm module was praised for giving a continuous sense of system risk beyond the binary safe/unsafe indicator.

The MLFQ tab's demotion log helped participants understand why CPU-bound processes migrate to lower queues — a nuance frequently lost in static MLFQ descriptions.

The quiz module was used for self-testing by all five participants, and all reported that the per-question explanations were more informative than the feedback they typically received from online multiple-choice tools.

12.3 Findings: Usability

All participants successfully completed core tasks (running a scheduling algorithm, loading a preset, stepping through page replacement, running the Banker's safety check) without assistance. Two participants noted initial confusion about the relationship between the Single Algorithm and Compare All tabs in the scheduling module, suggesting a short onboarding tooltip would improve first-use experience. One participant requested a print/export button on the memory map visualisation — subsequently noted as a candidate for a future version.

12.4 Limitations of the Study

The study is small (n=5) and does not constitute a controlled experiment with pre/post testing or a control group. The findings are therefore qualitative and should be interpreted as existence evidence that the tool is useful, not as a quantitative measure of learning gain. A more rigorous study with randomised assignment, standardised pre/post OS knowledge tests, and a larger cohort is planned as future work.

XIII. LIMITATIONS AND FUTURE WORK

13.1 Current Limitations

No file systems module. File system concepts (FAT, inode allocation, directory trees, journalling) are not covered. This is the most commonly cited gap in undergraduate OS courses not addressed by the current simulator.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

Single-core scheduling only. All scheduling algorithms assume a single CPU. Multi-core scheduling (affinity, load balancing, gang scheduling) is not modelled.

No real OS integration. The simulator models algorithms abstractly; it does not interface with the host OS kernel, so students cannot observe how the actual Linux/macOS/Windows scheduler behaves on real workloads.

Theme toggle partial coverage. The refresh_theme() function re-colours widgets by matching hex values against known palette entries. Widgets created with colours not in the palette (e.g., custom badge colours) do not update on theme change.

Limited accessibility. No screen-reader support, no keyboard navigation for canvas widgets, and colour is used as the primary semantic signal in several visualisations.

13.2 Future Work

File Systems module covering FAT16/32, ext2 inode structure, directory hierarchies, and a visual block allocation map.

Virtual Memory / TLB simulation extends the paging module with TLB hits/misses, multi-level page tables, and working-set modelling.

Controlled learning study with pre/post OS knowledge assessment, randomised assignment to simulator vs. textbook-only groups, and a cohort of at least 30 students.

Web version using Pyodide (Python in the browser via WebAssembly) to eliminate even the Python installation requirement.

Instructor dashboard allowing instructors to define custom preset scenarios, lock certain modules for examination purposes, and collect quiz analytics.

XIV. CONCLUSION

This paper presented OS Simulator v9, a comprehensive interactive desktop tool for operating system algorithm education. The simulator covers all nine major OS subsystems taught in undergraduate computer science curricula — CPU scheduling, memory management, Banker's algorithm, deadlock detection, disk scheduling, page replacement, IPC/synchronisation, process states, and system calls — in a single, zero-dependency Python application.

Key contributions of this work include: a correctly implemented and thoroughly tested library of 37 algorithms and features; a strict logic/UI separation architecture enabling independent testing and easy extension; novel visualisation features including the Bélády anomaly auto-detection banner, the Banker's safety margin meter, and the six-phase syscall animation; a 42-question interactive quiz with topic-level results breakdown; and verified correctness against known theoretical results for all implemented algorithms.

A user study with five B.Tech CS students confirmed that the tool improved conceptual clarity and algorithm comprehension across all nine topic areas, with particular impact on preemption dynamics in scheduling, the Bélády anomaly in page replacement, and the continuous safety margin concept in the Banker's algorithm. The tool is available as a standalone Python application and requires no installation beyond a base Python 3 distribution.

REFERENCES

[1] Brown, M. H., & Sedgewick, R. (1985). Techniques for Algorithm Animation. IEEE Software, 2(1), 28–39.
[2] Naps, T. L., et al. (2002). Exploring the Role of Visualization and Engagement in Computer Science Education. ACM SIGCSE Bulletin, 35(2), 131–152.
[3] AlgoViz Project. (2010). AlgoViz: The Algorithm Visualization Portal. Retrieved from http://algoviz.org
[4] Brown, M. H. (1988). Exploring Algorithms Using BALSA-II. IEEE Computer, 21(5), 14–36.
[5] Baer, J-L. (2010). Microprocessor Architecture. Cambridge University Press.
[6] Coady, Y., & Smolyn, L. (1999). Animating Operating System Concepts with RCOS.java. Proceedings of the 30th SIGCSE Technical Symposium, 275–279.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 14, April 2026
Impact Factor: 8.2

[7] Anderson, T. E., & Dahlin, M. (2014). Operating Systems: Principles and Practice. Recursive Books.
[8] Halim, S., & Halim, F. (2013). VisuAlgo: Visualising Data Structures and Algorithms through Animation. Proceedings of the 18th ACM SIGCSE.
[9] Pokpong, S. (2014). Process Scheduling Simulator. GitHub repository. Retrieved from https://github.com/pokpong/scheduling-simulator
[10] Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). Operating Systems: Three Easy Pieces (v1.01). Retrieved from https://pages.cs.wisc.edu/~remzi/OSTEP/
[11] Rosenblum, M., et al. (1995). Using the SimOS Machine Simulator to Study Dynamic Server Applications. Computing Systems, 8(4), 281–304.
[12] Dijkstra, E. W. (1965). Cooperating Sequential Processes. Technical Report EWD-123, Eindhoven University of Technology.
[13] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
[14] Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems (4th ed.). Pearson.
[15] Galvin, P. B., Silberschatz, A., & Gagne, G. (2012). Operating System Concepts Essentials (2nd ed.). Wiley