আন্তর্জাতিক ইসলামী বিশ্ববিদ্যালয় চট্টগ্রাম
الجامعة الإسلامية العالمية شيتاغونغ
International Islamic University Chittagong

Department of Computer Science and Engineering

Project Report

Course Code: CSE-3632

Course Title: Operating Systems Lab

Project Title: Smart OS Simulator

Members:

1. Name: Abdul-Al-Muheet
   ID: C223198
2. Name: MD. Shahariat Hossen
   ID: C231051
3. Name: Mohammed Sami Hasan
   ID: C231053
4. Name: Afrin Hasan Safin
   ID: C231062

Supervisor: Mohammed Arfat
Lecturer.
Department of CSE, IIUC.

Date of Submission: 07/12/2025

Remarks:

Smart OS Simulator

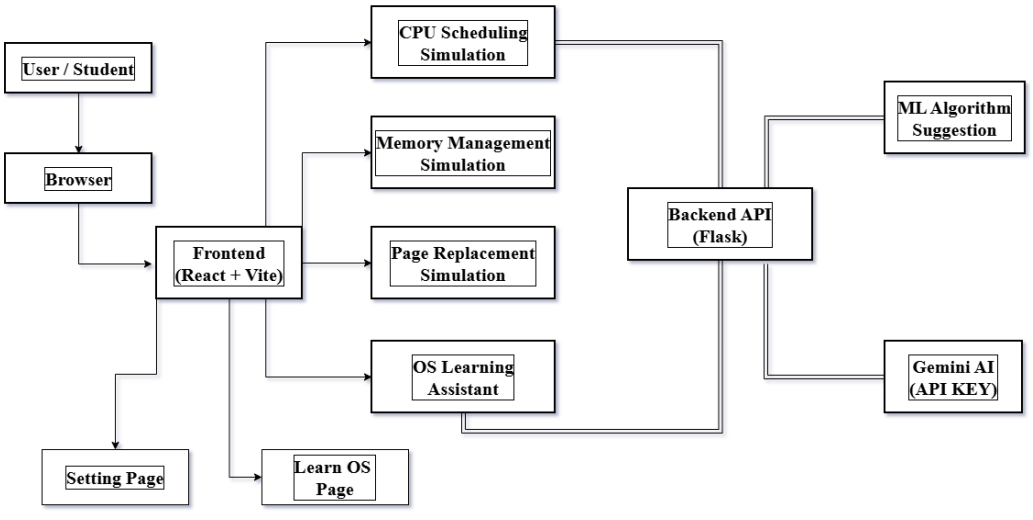
Introduction:

Smart OS Simulator is a Vite+React (TypeScript) frontend with an optional Flask backend that lets students interactively explore core Operating Systems topics: CPU scheduling, contiguous memory allocation/fragmentation, and page replacement, plus a chat-based learning assistant (Gemini when configured, otherwise a local OS knowledge base).

Objectives:

- Provide hands-on CPU scheduling simulations (FCFS, SJF, SRJF, RR, Priority, RR+Priority) with Gantt charts and computed metrics.
- Teach memory allocation strategies (first/best/worst-fit), fragmentation effects, and compaction via a visual memory layout.
- Demonstrate page replacement algorithms (FIFO, LRU, Optimal, Clock) with step-by-step state and comparisons.
- Add “learning support” features: comparative analysis, feedback/scoreboard, and an OS Q&A assistant + curated learning content.

Methodology:



1. User

The user is simply the student who opens the website to learn Operating Systems.
They come to the site to try different simulations, see how algorithms behave, and understand OS topics better instead of only reading theory.

2. Browser

The browser is the place where everything runs.
When the student opens the site in Chrome or Edge, the browser loads the website, shows the buttons, charts, and animations, and sends the student's inputs to the system.
The browser doesn't think on its own — it just shows results and passes messages back and forth.

3. Frontend (React + Vite)

The frontend is the face of the website.
This is what the student actually sees and interacts with.
It:

- Shows different pages like CPU scheduling, memory simulation, learning content, and chat
- Takes input from the student (like process details or page numbers)
- Runs simple logic and animations
- Displays charts, tables, and results clearly

If something needs more processing or AI help, the frontend asks the backend for it.

4. CPU Scheduling Simulation

Here, the student can try CPU scheduling algorithms.
They enter process details, choose an algorithm, and instantly see how the CPU would schedule those processes.
The website then shows:

- The order of execution
- Waiting time and turnaround time
- A visual timeline (Gantt chart)

This helps students *see* what usually feels confusing in theory.

5. Memory Management Simulation

This part helps students understand how memory is used. The student can see:

- How memory blocks are allocated
- What happens when memory is full
- How fragmentation occurs

Instead of imagining memory on paper, the student can actually watch memory being used and freed.

6. Page Replacement Simulation

In this section, the student learns what happens when memory pages are replaced.
They provide a page reference string, and the website:

- Shows which pages stay in memory
- Shows which pages are removed
- Counts page faults step by step

This makes it easy to compare FIFO, LRU, and other algorithms.

7. Learn OS Page

This page is like a digital notebook inside the website.
It contains:

- Simple explanations of OS topics
- Examples related to the simulations
- Extra help for understanding concepts

Students can read here when they feel confused or want revision.

8. OS Learning Assistant

This is like having a small tutor inside the website.
Students can ask questions such as:

- “Why is this algorithm better here?”
- “Why did this page fault happen?”

The assistant explains things in simple words and helps students understand results instead of just showing numbers.

9. Settings Page

The settings page lets students customize the website.
For example:

- Change theme (dark/light)
- Adjust default values for simulations
- Enable or disable AI help

This makes the website more comfortable to use.

10. Backend API (Flask)

The backend works behind the scenes. It:

- Receives requests from the frontend
- Handles heavier logic and calculations
- Talks to the AI and ML parts
- Sends clean results back to the frontend

The student never sees this part, but it makes everything work smoothly and securely.

11. ML Algorithm Suggestion

This part acts like a smart helper.
Based on the student's input, it can suggest:

- Which scheduling algorithm fits best
- Why one approach is better than another

It helps students think beyond "just run the algorithm" and understand *why* choices matter.

12. Gemini AI

Gemini AI is used when the student needs deeper explanations.
The backend sends questions to Gemini and brings back:

- Clear explanations
- Summaries
- Helpful comparisons

The API key stays hidden on the server, so it's safe.

Limitations

- Deployment/config gaps: chat calls are hard-coded to http://localhost:5000 in the frontend, while other API calls use VITE_API_BASE; backend CORS is also tailored to local dev ports.
- Security/ops: backend runs with debug=True, and a Gemini API key appears to exist in backend/.env (rotate it and keep secrets out of the repo/workspace).
- ML recommender validity: the scheduler model is trained on synthetic labeled data (simulation-based heuristic "best"), so recommendations may not generalize to all real workloads.
- No persistence/auth: simulations, settings, and chat history are in-memory/UI state only; no user accounts, storage, or rate limiting.
- Repo hygiene/size: checked-in frontend/node_modules and backend/.venv greatly bloat the project and complicate collaboration (typically excluded).
- UX text encoding: several feedback strings contain garbled characters (likely intended emojis/icons), which can reduce clarity.

Conclusion

Overall, this project is a strong educational "OS concepts" with clear algorithm implementations and visualization, and an optional AI/knowledge-base assistant; its main improvements would be production-ready configuration (API base/CORS), secret handling, and tightening repo/deployment practices.