AetherOS
AI-Integrated Web-Based Operating System Simulator
Version 4.0
Deployment: https://aetheros.necookie.dev
CDN: Cloudflare

Project Members
Dheyn Michael Orlanda (Lord Dheyn)
Matthew Dee (Almighty Bossman)
Jaypee Javier (Marcus / Miquella)
Victor de Mesa Jr. (Tito Vic)

Advisers
Ma'am Roxane Garbo
Ma'am Reichel Rivano

---

1. Executive Summary

AetherOS is a browser-native operating system simulator designed to model fundamental operating system mechanisms while presenting a modern desktop-style interface. The system combines deterministic kernel simulation, real-time visualization, AI-powered system introspection, persistent backend storage using a self-hosted Supabase instance, containerized infrastructure using Docker, and global content delivery via Cloudflare.

The platform demonstrates full-stack engineering across systems simulation, frontend architecture, backend integration, artificial intelligence tooling, containerization, infrastructure deployment, and DevOps workflow management.

---

2. Problem Statement

Operating Systems concepts such as CPU scheduling, virtual memory paging, and disk allocation are often difficult to internalize through static diagrams and textbook explanations. Students frequently understand theoretical definitions but struggle to visualize how scheduling policies impact fairness, how page faults degrade performance, how thrashing occurs under memory pressure, and how disk fragmentation accumulates over time.

There is currently no accessible, browser-based platform that integrates deterministic OS simulation with real-time visualization and conversational AI explanations. AetherOS addresses this gap by providing an interactive, web-native operating system laboratory.

---

3. Product Vision

AetherOS is envisioned as a deterministic OS laboratory, a visual systems simulation platform, and an AI-assisted diagnostic environment. The system bridges operating systems theory with practical visualization and conversational reasoning. It is designed not merely as a simulator but as a systems-level platform that demonstrates competence in frontend engineering, backend infrastructure, database architecture, artificial intelligence integration, and containerized deployment.

---

4. System Architecture Overview

The architecture of AetherOS is divided into five primary layers.

The Simulation Kernel runs inside a Web Worker to prevent UI blocking. It operates as a deterministic, tick-based state machine responsible for managing processes, memory, disk allocation, scheduling decisions, and event emission. It maintains authoritative system state and communicates updates to the frontend via structured messages.

The Frontend Visualization Layer is built using React and TypeScript with Vite as the build tool and Tailwind CSS for styling. Zustand is used for UI state management. The frontend renders the desktop shell, window manager, process visualizations, RAM and disk grids using the Canvas API, event logs, CLI interface, and application windows.

The Persistence Layer is implemented using a self-hosted Supabase instance backed by PostgreSQL. Supabase and PostgreSQL are deployed using Docker containers to ensure portability, reproducibility, and environment consistency across development and production systems. This layer stores simulation configurations, serialized system snapshots, AI conversation history, and scenario templates.

The AI Proxy Layer is deployed as either a Cloudflare Worker or a Node-based backend service. When using a Node backend, it is containerized using Docker to maintain consistent runtime environments and simplify deployment. This layer secures access to the OpenAI API, implements tool-based AI interaction, enforces rate limiting, and ensures that API keys remain server-side.

The AI Assistant Layer utilizes the OpenAI Responses API with tool-calling capabilities. The AI interacts with structured tools such as system_snapshot, process_inspect, memory_stats, disk_stats, trace_recent, and apply_policy, ensuring that AI explanations are grounded in actual simulation data.

---

5. Operating System Features (80/20 Implementation)

AetherOS implements a focused subset of operating system features that create a realistic desktop experience without excessive complexity.

The Desktop Shell includes a window manager with drag, resize, minimize, and maximize capabilities. It provides a taskbar or dock, an application launcher, a system tray displaying CPU, RAM, and disk usage statistics, and a clock.

The Settings Application allows modification of system-level simulation parameters such as scheduling policy, time slice duration, total memory size, disk size, theme mode, and AI operating mode.

The File Manager supports directory navigation, file creation, file deletion, renaming, and disk block visualization. All file metadata and allocation data are managed by the simulated disk subsystem.

The Command Line Interface provides commands for process listing, process termination, workload spawning, memory statistics retrieval, disk statistics retrieval, directory navigation, file creation and deletion, and AI-based explanation of events.

The Simple Browser uses an iframe-based sandbox to allow real internet browsing. While rendering real web content, it behaves as a simulated process consuming virtual CPU time and memory.

The Task Manager displays process tables, CPU usage, memory usage, and allows process termination.

The AI Assistant provides real-time system explanations, performance diagnostics, policy suggestions, scenario generation, and event interpretation.

---

6. Kernel Simulation Details

The Process Management subsystem defines each process with attributes including PID, state, priority, time slice remaining, memory allocation, workload script, and runtime metrics. Round Robin scheduling is required, while Multi-Level Feedback Queue may be implemented as an advanced feature.

The Memory Management subsystem models physical memory as fixed-size frames and implements per-process page tables. Page fault handling and Clock-based page replacement are core requirements.

The Storage Simulation subsystem models disk storage as a fixed block array with a free-space bitmap. File allocation uses a FAT-style or simplified inode-based approach. Disk I/O is simulated using latency measured in ticks, during which processes transition to a waiting state.

---

7. Infrastructure and Deployment

The system is deployed at aetheros.necookie.dev using Cloudflare DNS. Cloudflare Pages provides frontend hosting and global CDN delivery with automatic HTTPS. The backend AI proxy runs on Cloudflare Workers or as a Dockerized Node service. The database layer is powered by a self-hosted Supabase instance using PostgreSQL, containerized via Docker for portability and environment consistency.

Docker ensures reproducible builds, simplified deployment, isolated services, and scalable backend infrastructure management.

---

8. Development and Technology Stack

The frontend stack includes React, TypeScript, Vite, Tailwind CSS, Zustand, and xterm.js.

The backend stack includes Cloudflare Workers or Node.js, the OpenAI Responses API, and Supabase with PostgreSQL.

The infrastructure stack includes Docker for containerization, Supabase for backend services, and Cloudflare for CDN, DNS, and edge deployment.

Development tools include Git for version control, GitHub for repository hosting and CI/CD integration, Visual Studio Code as the primary IDE, Google Antigravity IDE for AI-assisted development, and Codex CLI for rapid code generation and refactoring.

---

9. Performance Constraints

The system targets 60 frames per second for UI responsiveness and supports at least fifty simulated processes. The kernel must operate entirely off the main thread, and snapshot serialization should complete within one hundred milliseconds.

---

10. Security Considerations

OpenAI API keys are stored server-side only. Supabase credentials are secured via environment variables and container configuration. Dockerized services isolate backend components. AI endpoints are rate-limited, and optional Row-Level Security may be enabled for multi-user expansion.

---

11. Future Expansion

Future enhancements may include multi-core CPU simulation, swap space modeling, Translation Lookaside Buffer simulation, Rust-to-WASM kernel migration, network stack simulation, collaborative multi-user sessions, and scalable container orchestration using Docker Compose or Kubernetes.

---

12. Strategic Value

AetherOS demonstrates competence in operating systems theory, deterministic simulation modeling, AI tool integration, modern frontend architecture, backend engineering, PostgreSQL database design, containerized infrastructure using Docker, CDN deployment, and DevOps practices.

It represents a full-stack systems and AI platform deployed on modern cloud infrastructure.