Capstone Series: "OS-In-A-Box: From Theory to Functional Prototypes"

Series Philosophy: This series bridges the gap between theoretical knowledge and practical implementation. Each 3-week project challenges students to solve a real-world inspired problem by building a working prototype or simulation, progressing through all levels of cognitive complexity.

1. Capstone Project: Process Scheduling

Background Story: CloudServe Inc., a growing startup, offers virtual machines to developers. Users complain that during peak hours, background batch jobs (like video encoding) make their interactive applications (like IDEs and web servers) unresponsive. The current First-Come-First-Served scheduler is failing their mixed workload.

Problem Statement: Design and implement a CPU scheduling simulator to evaluate and recommend a more efficient algorithm that ensures fairness and responsiveness for both interactive (I/O-bound) and batch (CPU-bound) processes.

Project Objectives.

1. Identify process states and describe scheduling algorithms (FCFS, SJF, Priority, Round Robin).
2. Implement multiple algorithms and calculate performance metrics (Waiting Time, Turnaround Time).
3. Compare algorithm performance and design a hybrid approach for CloudServe's specific needs.

Tools & Technologies: Python, Matplotlib, Pandas, Jupyter Notebook, Git.

Platform: Cross-platform (Windows/macOS/Linux).

Project Plan:

Week 1: Foundation & Design

Steps: Research scheduling algorithms. Design the Process and Scheduler classes. Define performance metrics.

Deliverable: A design document with UML class diagrams and a literature review of scheduling algorithms.

Week 2: Implementation & Analysis

Steps: Code the FCFS and Round Robin schedulers. Implement metric calculators. Test with sample data.

Deliverable: A functional Python simulator that outputs metrics and a Gantt chart for a given process list.

Week 3: Evaluation & Reporting

Steps: Test algorithms with mixed workloads. Visualize comparative performance. Justify the best algorithm.

Deliverable: A final report with graphs and a recommendation for CloudServe, including a proposal for a potential hybrid scheduler.

2. Capstone Project: Virtual Memory Management

Background Story: The "Starlight" deep-space probe has limited physical memory (RAM) but must run large data analysis programs that exceed its capacity. The ground team needs to implement a virtual memory system that uses the probe's solid-state storage as swap space to prevent system crashes.

Problem Statement: Create a virtual memory manager simulator that implements demand paging and evaluates different page replacement algorithms to minimize total page faults, thereby conserving the probe's limited power.

Project Objectives :

1. Explain paging, page faults, and TLB concepts.
2. Implement FIFO, LRU, and Optimal page replacement algorithms.
3. Analyze algorithm performance and design a custom memory access pattern that demonstrates Belady's Anomaly.

Tools & Technologies: C or Python, Git.

Platform: Cross-platform.

Project Plan:

Week 1: Architecture Design

Steps: Design the core components: Page Table, RAM (frames), Disk (swap space). Define the address translation logic.

Deliverable: A technical specification document detailing the simulator's architecture and data structures.

Week 2: Algorithm Implementation

Steps: Code the page fault handler. Implement FIFO, LRU, and Optimal algorithms. Simulate memory reference strings.

Deliverable: A core simulator that processes a reference string and reports page fault counts for each algorithm.

Week 3: Performance Analysis

Steps: Run extensive tests with various memory access patterns. Graph the results. Document Belady's Anomaly.

Deliverable: A performance analysis report recommending the best algorithm for the rover and explaining the trade-offs.

3. Capstone Project: File System Implementation

Background Story: SecureArchive LLC is developing a new line of encrypted, off-line storage appliances. They require a simple, robust, and proprietary file system to manage data storage, avoiding the licensing complexities of ext4 or NTFS.

Problem Statement: Develop a custom file system that operates on a virtual disk (a single large file) and provides basic commands for file creation, reading, writing, and deletion, with efficient metadata and space management.

Project Objectives :

1. Identify file system components (boot block, superblock, inode, data block).
2. Implement a file system format using indexed allocation. Create tools to interact with the file system.
3. Design and build a complete file system with a custom toolset and evaluate its space efficiency.

Tools & Technologies: C, Linux.

Platform: Linux.

Project Plan:

Week 1: On-Disk Structures

Steps: Design the disk layout: superblock, inode table, data blocks. Code the mkfs tool to format a virtual disk.

Deliverable: A detailed disk layout diagram and a working mkfs tool.

Week 2: Core Operations

Steps: Implement file creation (create), writing (write), and reading (read). Build an ls tool.

Deliverable: A set of command-line tools (create, write, read, ls) for the file system.

Week 3: Advanced Operations & Robustness

Steps: Implement file deletion and block deallocation. Test for fragmentation and data integrity.

Deliverable: A fully functional file system with a delete command and a final demo showcasing all operations.

4. Capstone Project: Device & I/O Scheduling

Background Story: RenderFlow Studios' video editing servers are experiencing severe slowdowns when multiple editors access project files simultaneously. Disk I/O requests are being served in the order they arrive (FCFS), causing excessive disk head movement and high latency.

Problem Statement: Build a disk I/O scheduling simulator to model and analyze the performance of different algorithms (beyond FCFS) to reduce average request service time and total disk head movement.

Project Objectives :

1. Describe disk geometry and I/O scheduling algorithms (SSTF, SCAN, C-SCAN).
2. Implement FCFS, SSTF, and SCAN algorithms in a simulator.
3. Compare algorithm performance and design a workload that highlights the starvation problem in SSTF.

Tools & Technologies: Python, Matplotlib, Git.

Platform: Cross-platform.

Project Plan:

Week 1: Research & Modeling

Steps: Research disk geometry and scheduling algorithms. Model a disk with cylinders. Design the request queue.

Deliverable: A research document explaining the algorithms and a design for the simulation model.

Week 2: Simulator Development

Steps: Code the FCFS, SSTF, and SCAN algorithms. Implement a metric calculator for total head movement and average wait time.

Deliverable: A functional simulator that processes an I/O queue and outputs performance metrics.

Week 3: Analysis & Recommendation

Steps: Test with real-world I/O traces. Visualize disk head movement. Analyze trade-offs (throughput vs. fairness).

Deliverable: A comparative report with graphs, identifying the best algorithm for RenderFlow and discussing its limitations.

5. Capstone Project: Process Synchronization

Background Story: A multi-threaded financial trading application at FinTech Innovations occasionally produces incorrect portfolio valuations. Developers suspect a race condition where multiple threads are updating shared account balances without proper synchronization, leading to data corruption.

Problem Statement: Model this concurrency issue using the classic "Dining Philosophers" problem. Demonstrate the deadlock scenario and then implement a correct solution using synchronization primitives to ensure data consistency.

Project Objectives :

1. Define race conditions, deadlocks, and mutual exclusion.
2. Reproduce a deadlock in the Dining Philosophers problem. Implement a solution using semaphores/mutexes.
3. Design a test to prove the solution is deadlock-free and evaluate its performance overhead.

Tools & Technologies: C (with pthreads) or Python (with threading), Git.

Platform: Linux or macOS.

Project Plan:

Week 1: Problem Reproduction

Steps: Implement the naive Dining Philosophers problem with 5 threads and 5 chopsticks (resources).

Deliverable: A program that can demonstrably reach a deadlock state, proven by logs or a hang.

Week 2: Solution Implementation

Steps: Research and implement a solution (e.g., resource hierarchy, using a semaphore to limit concurrent access).

Deliverable: A corrected, deadlock-free implementation of the problem.

Week 3: Testing & Evaluation

Steps: Stress-test the solution. Measure any performance impact compared to the naive version.

Deliverable: A comprehensive report detailing the problem, solution, and an analysis of the concurrency-performance trade-off.

6. Capstone Project: Log Management & System Monitoring

Background Story: InnovateLab's development server frequently experiences performance issues and application crashes. The system administrators struggle to diagnose problems because log messages from different services are scattered across multiple files and there's no centralized way to view or filter important error messages in real-time.

Problem Statement: Develop a simple log management system that can aggregate, filter, and display log messages from multiple sources in a centralized dashboard, with the ability to highlight errors and warnings for quick troubleshooting.

Project Objectives:

1. Identify common log formats (syslog, application logs) and log levels (INFO, WARN, ERROR).
2. Implement a log aggregator that can tail multiple log files and parse log entries.
3. Design and build a real-time dashboard that displays filtered log messages with color-coded severity levels.

Tools & Technologies: Python, `watchdog` library (for file monitoring), `curses` library (for TUI), regular expressions, Git.

Platform: Linux

Project Plan:

Week 1: Log Parser & Aggregator
Steps: Research common log formats. Design a log entry data structure. Implement a parser that can read log files and extract timestamp, severity, and message components.
Deliverable: A Python script that can parse sample log files and output structured log data in JSON format.

Week 2: Real-time Log Monitoring
Steps: Implement file monitoring to detect new log entries. Create a mechanism to tail multiple log files simultaneously. Add basic filtering by log level (ERROR, WARN, INFO).
Deliverable: A log monitor that can watch multiple files and display new log entries in real-time with simple filtering.

Week 3: Dashboard & Advanced Features
Steps: Build a TUI dashboard with color-coded log levels. Add search functionality and time-based filtering. Implement configurable alerting for specific error patterns.
Deliverable: A complete log management dashboard with real-time monitoring, search, filtering, and basic alerting capabilities.

.

7. Capstone Project: Inter-Process Communication (IPC)

Background Story: The backend services for "GlobalChat," a messaging app, are becoming inefficient. The user authentication service, message routing service, and database service need a fast and reliable way to exchange data and signals without using slow disk-based files.

Problem Statement: Create a demonstration project using two fundamental IPC mechanisms—shared memory for high-speed data transfer and message queues for reliable signaling—to show how separate processes can communicate.

Project Objectives :

1. Describe different IPC methods (pipes, shared memory, message queues, sockets).
2. Implement a shared memory segment for data sharing and a message queue for sending commands.
3. Design a producer-consumer system using both IPC methods and analyze their performance characteristics.

Tools & Technologies: C, Linux IPC libraries (sys/shm.h, sys/msg.h).

Platform: Linux.

Project Plan:

Week 1: Shared Memory Implementation

Steps: Create a writer process that writes data to a shared memory segment and a reader process that reads from it.

Deliverable: Two programs that successfully communicate via shared memory.

Week 2: Message Queue Implementation

Steps: Implement a message queue to send signals or commands between two processes (e.g., "new data available").

Deliverable: Two programs that can send and receive structured messages through a queue.

Week 3: Hybrid System & Analysis

Steps: Combine both IPC methods into a single demo (e.g., use message queue to notify about shared memory updates). Compare speed and complexity.

Deliverable: A final integrated demo and a report discussing the use cases for each IPC method.

8. Capstone Project: System Monitoring & Performance

Background Story: The system administrators at DataHub Corp lack a simple, centralized view of their server fleet's health. They need a lightweight tool to monitor key metrics in real-time without the overhead of enterprise monitoring suites.

Problem Statement: Develop a real-time system resource monitor that displays key performance indicators (CPU, Memory, Disk I/O, Network) in a command-line dashboard, with configurable alert thresholds.

Project Objectives :

1. Identify key system performance metrics and how to retrieve them.
2. Implement data collection for these metrics and display them in a real-time dashboard.
3. Design and add alerting functionality for when metrics exceed defined thresholds.

Tools & Technologies: Python, psutil library, curses library (for TUI), Git.

Platform: Cross-platform (Windows/macOS/Linux).

Project Plan:

Week 1: Data Collection

Steps: Learn the psutil library. Write a script to collect CPU, Memory, Disk, and Network stats.

Deliverable: A script that logs system metrics to a CSV file at regular intervals.

Week 2: Dashboard Development

Steps: Use the curses library to create a Text-Based User Interface (TUI). Display real-time metrics.

Deliverable: A real-time TUI dashboard that updates and displays system stats.

Week 3: Alerting & Enhancement

Steps: Add color-coding (e.g., red for high CPU) and configurable alert thresholds that log warnings.

Deliverable: A final, user-friendly monitoring tool with alerting features and a demonstration.

9. Capstone Project: System Bootloader

Background Story: An embedded systems company, "BareMetal Tech," wants to train its engineers on the low-level boot process of their devices. Understanding the transition from hardware power-on to a running kernel is crucial for firmware development.

Problem Statement: Create a simple bootloader that runs on an x86 emulator, switches the CPU from 16-bit Real Mode to 32-bit Protected Mode, and loads a minimal "Hello World" kernel written in C.

Project Objectives :

1. Explain the stages of the x86 boot process and the differences between Real and Protected Mode.
2. Write a bootloader in Assembly that performs the mode switch and loads a second stage.
3. Design and integrate a minimal C kernel that is loaded and executed by the bootloader.

Tools & Technologies: x86 Assembly (NASM), C, GCC, LD, QEMU (emulator).

Platform: Cross-platform (using QEMU).

Project Plan:

Week 1: Basic Bootloader

Steps: Study BIOS interrupts. Write a boot sector in ASM that prints a message and hangs.

Deliverable: A bootable disk image that displays "Stage 1 OK" when run in QEMU.

Week 2: Mode Switching & Loading

Steps: Extend the bootloader to load a second sector from the disk. Implement the switch to Protected Mode.

Deliverable: A bootloader that successfully enters 32-bit Protected Mode and jumps to a loaded second stage.

Week 3: Kernel Integration

Steps: Write a minimal C kernel that prints to video memory. Link the kernel with the bootloader.

Deliverable: A final demo showing the full chain: BIOS -> Bootloader -> Protected Mode -> C Kernel printing "Hello World!".

10. Capstone Project: System Call Interface

Background Story: A cybersecurity firm, "TraceSec," needs to analyze the behavior of untrusted software. They require a tool to log all interactions an application has with the OS kernel to detect malicious activity like unauthorized file access or network communication.

Problem Statement: Build a system call tracer utility that uses the ptrace debugging facility to intercept and log all system calls and their arguments made by a child process.

Project Objectives :

1. Describe the role of system calls and the functionality of ptrace.
2. Implement a tracer that attaches to a process and intercepts its system calls.
3. Design a tool that decodes system call numbers and arguments into a human-readable audit log.

Tools & Technologies: C, Linux, ptrace, Git.

Platform: Linux.

Project Plan:

Week 1: Basic Tracing

Steps: Research the ptrace system call. Create a program that forks a child process and traces it.

Deliverable: A simple tracer that can launch and attach to a child process.

Week 2: System Call Interception

Steps: Use ptrace to stop the child before/after each system call. Extract and log the system call number.

Deliverable: A tracer that outputs a chronological list of system call names (e.g., read, write, open).

Week 3: Argument Decoding.

Steps: Enhance the tracer to read and display arguments for specific calls (e.g., filenames for open, buffer sizes for read/write).

Deliverable: An advanced tracer that produces a detailed, human-readable audit log of an application's behavior.