Chronos OS: Phases 23 & 24 Report

Writable Virtual Filesystem and System Monitoring

System Architect

January 22, 2026

Abstract

Phases 23 and 24 focused on transforming Chronos from a static runtime into a dynamic environment. Phase 23 introduced a Writable Virtual Filesystem (VFS), allowing runtime creation, modification, and deletion of files in RAM. Phase 24 established a Global Scheduler architecture, enabling the creation of a "Task Manager" application that queries and visualizes kernel state in real-time within the graphical user interface.

Contents

1 Phase 23: The Writable VFS

1.1 Architecture Shift

Previous filesystem iterations relied on parsing the immutable Limine Module structure directly. To support write operations, we migrated to a **Dynamic Heap Model**.

- **Initialization:** At boot, the kernel copies immutable modules from Limine into a `Vec<File>`.
- **Storage:** This vector is wrapped in a global `Mutex`, allowing thread-safe mutation.
- **Operations:** We implemented standard POSIX-like logic: `touch` (create), `rm` (delete), and `write` (append).

1.2 Shell Integration

The shell was updated to interface with the VFS. Users can now manipulate the environment state persistently (until reboot).

Listing 1: VFS Write Logic

```
pub fn append_file(name: &str, data: &[u8]) -> bool {
    let mut fs = FILESYSTEM.lock();
    for file in fs.iter_mut() {
        if file.name == name {
            file.data.extend_from_slice(data);
            return true;
        }
    }
    false
}
```

2 Phase 24: The System Monitor

2.1 Global Scheduler Access

To visualize system load, the UI layer (Shell) needed access to kernel internals (Scheduler). We refactored the Scheduler from a local variable in `main.rs` to a global static `Mutex<Scheduler>`. This allows any subsystem to query the list of active tasks and their performance metrics.

2.2 The "Top" Command

We implemented a system monitoring tool inspired by the Unix `top` utility.

1. **Windowing:** The command spawns a dedicated, floating window.
2. **Data Acquisition:** The window logic locks the Scheduler, iterates through tasks, and retrieves the `last_cost` (CPU cycles).
3. **Visualization:** Cycle counts are normalized and rendered as ASCII bars (`[....]`), providing immediate visual feedback on process weight.

3 Concurrency Management (The Deadlock)

A critical deadlock occurred during initial testing. The main loop held the Scheduler lock while running the Shell task. The Shell task then attempted to acquire the Scheduler lock to update the monitor.

Resolution: We decoupled the rendering logic. The Shell task no longer performs the update. Instead, the main loop releases the Scheduler lock, acquires the Shell lock, and then calls a static helper `Shell::update_monitor`, ensuring locks are never held recursively.

4 Conclusion

Chronos OS now supports dynamic data manipulation and self-introspection. The user can create files and visually monitor the kernel's resource allocation in real-time. This completes the core feature set for a functional Graphical Operating System.