Chronos OS: Phase 6 Report

The Interactive Shell and Command Processing

Development Log

January 19, 2026

Abstract

Phase 6 represents a pivotal shift in the user experience of Chronos OS. Previously, the system was a passive observer of hardware states. In this phase, we implemented a full-duplex Command Line Interface (CLI). This required constructing a thread-safe Input Buffer to decouple high-speed interrupt handling from the slower shell logic, implementing a command parser, and upgrading the graphics engine to support text editing features like backspacing and scrolling. The Shell now runs as a high-priority task within the Chronos Scheduler.

Contents

1 Overview

To make Chronos a "Daily Driver" candidate, the user must be able to issue commands and receive textual feedback. This required three distinct subsystems to work in unison:

1. **Input Capture:** Storing keystrokes asynchronously.
2. **Shell Logic:** Parsing strings and executing logic.
3. **Visual Feedback:** Rendering the typing in real-time.

2 The Input Architecture

2.1 The Producer-Consumer Problem

Keyboard interrupts fire asynchronously and extremely fast. The Shell, however, runs only when the Scheduler gives it CPU time. If we processed commands inside the Interrupt Handler, we would block the CPU.

Solution: We implemented a **Ring Buffer** (FIFO) using 'VecDeque'.

- **Producer:** The Interrupt Handler pushes characters into the buffer.
- **Consumer:** The Shell Task pops characters from the buffer.
- **Safety:** The buffer is wrapped in a 'spin::Mutex' to prevent data races.

```
lazy_static! {
    pub static ref KEYBOARD_BUFFER: Mutex<VecDeque<char>> = Mutex::new(VecDeque
    ::new());
}

pub fn push_key(c: char) {
    x86_64::instructions::interrupts::without_interrupts(|| {
        KEYBOARD_BUFFER.lock().push_back(c);
    });
}
```

Listing 1: The Thread-Safe Input Buffer

3 The Shell Implementation

3.1 Command Processing

The Shell is implemented as a state machine. It maintains a local 'command$_buffer$'($String$).$Itreadsfromtheglo$

Standard Char: Appended to the buffer and printed to the screen.

Newline (\n): Triggers 'execute$_command$()'.

Backspace (\x08): Triggers visual deletion logic.

3.2 Scheduler Integration

The Shell was added as a persistent task in the Scheduler. To ensure typing feels responsive, we assigned it a generous budget.

```
// Budget: 100,000 cycles (High Priority)
chronos_scheduler.add_task("Shell", 100_000, shell::shell_task);
```

Listing 2: Registering the Shell Task

4 Visual Upgrades

4.1 The Persistence Fix

In previous phases, the Main Loop called ‘writer.clear()‘ every frame to animate the background. This caused typed text to vanish instantly.

Fix: We removed the global clear command. The screen is now persistent. To prevent the Fuel Gauge from overwriting text, the visual metrics were moved to the bottom 50 pixels of the screen, leaving the top area dedicated to the CLI.

4.2 Backspace Logic

Deleting text visually is harder than deleting it from memory. We extended the ‘Writer‘ struct to support a ‘backspace()‘ method. This method moves the cursor coordinate backward by one character width and draws a background-colored rectangle to ”erase” the pixel data.

5 Results

The system now accepts user input. The following commands were implemented and verified:

- `help`: Displays available commands.
- `ver`: Displays OS version.
- `clear`: Manually clears the screen and resets the cursor.

6 Conclusion

Chronos is no longer a static kernel; it is an interactive operating system. The successful integration of the Keyboard Buffer and Shell Task proves that the custom Scheduler can handle interactive workloads alongside background processing. The next logical step is to give this Shell something to manage: a Filesystem.