Chronos OS: Phase 17 Report

The PS/2 Mouse Driver and Non-Destructive Rendering

System Architect

January 21, 2026

Abstract

Phase 17 focuses on the implementation of a pointing device to facilitate a Graphical User Interface (GUI). We initialized the legacy 8042 PS/2 Controller to support the Auxiliary Device (Mouse) and enabled the Cascade Interrupt on the Programmable Interrupt Controller (PIC). A custom driver was written to decode 3-byte movement packets and render a cursor overlay. To prevent graphical corruption ("trails") and system deadlocks, we implemented a "Save-Restore" rendering strategy and a non-blocking mutex locking mechanism.

Contents

1 Hardware Architecture: The 8042 Controller

Despite modern USB hardware, the input subsystem often relies on PS/2 emulation for compatibility. The Mouse is managed by the same controller as the Keyboard (Port `0x60` and `0x64`), but acts as the "Second PS/2 Port."

1.1 Initialization Sequence

To enable the mouse, specific commands must be sent to the controller to "unlock" the auxiliary port.

1. **Command 0xA8:** Enable Auxiliary Device.
2. **Command 0x20/0x60:** Read/Write Controller Configuration Byte. We set Bit 1 to enable IRQ 12.
3. **Command 0xD4:** "Write to Auxiliary." Used to send commands (like `0xF4` - Enable Scanning) directly to the mouse hardware.

1.2 Interrupt Routing (The Cascade)

A critical bug halted development early in Phase 17. The mouse is connected to the **Slave PIC** (IRQ 12). The CPU interacts with the Slave PIC via **IRQ 2** on the Master PIC. Initially, IRQ 2 was masked, causing the CPU to ignore all mouse signals. We resolved this by explicitly unmasking the cascade line (`0xF8` on Master PIC).

2 The Packet Protocol

The Standard PS/2 Mouse sends data in a 3-byte packet sequence. We implemented a state machine in the interrupt handler to assemble these bytes.

- **Byte 0 (Status):** Contains Y-Overflow, X-Overflow, Y-Sign, X-Sign, and Button states. Bit 3 is always 1 (used for synchronization).
- **Byte 1 (Delta X):** Movement in the X axis.
- **Byte 2 (Delta Y):** Movement in the Y axis.

3 Graphics and Rendering Logic

3.1 The "Trail" Problem

Initial tests resulted in the cursor leaving a permanent trail of white pixels across the screen. This occurred because the driver overwrote the framebuffer data without preserving the previous state.

3.2 Solution: Save-Draw-Restore

We implemented a software cursor buffer.

1. **Restore:** Before moving, the driver writes the saved 10x10 pixel buffer back to the `prev_x, prev_y` coordinates.
2. **Save:** At the new `x, y` coordinates, the driver reads the screen pixels into the buffer.
3. **Draw:** The driver writes the white cursor pixels to the screen.

4 Concurrency and Deadlocks

4.1 The Interrupt Deadlock

A fatal crash was observed when moving the mouse while the Shell was printing text.

Analysis:

1. The Main Loop (Shell) acquires the `WRITER` Mutex to print a character.
2. A Mouse Interrupt fires, pausing the Main Loop.
3. The Mouse Handler attempts to acquire the `WRITER` Mutex to draw the cursor.
4. **Deadlock:** The Handler waits for the Shell to release the lock, but the Shell cannot run because the Handler has paused the CPU.

4.2 The Non-Blocking Fix

We utilized `WRITER.try_lock()` instead of `lock()`.

```
// If the screen is busy (Shell is printing), skip drawing this frame.
// This prevents the OS from freezing.
if let Some(mut w) = writer::WRITER.try_lock() {
    // Perform drawing operations...
}
```

Listing 1: Non-Blocking Render Logic

This ensures that the mouse interrupt never halts the system, prioritizing system stability over a single frame of cursor movement.

5 Conclusion

Chronos OS now supports 2D coordinate input. The implementation handles hardware communication, protocol decoding, and complex concurrency challenges related to shared video memory. This foundation allows for the development of Phase 19: The Window Manager and UI compositing.