Chronos OS: Phase 8 Report

Hardware Isolation: GDT, TSS, and Ring 3 Transition

Development Log

January 20, 2026

Abstract

Phase 8 represents the transition from a monolithic kernel model to a secure, multi-level architecture. To protect the kernel from user-space errors, x86_64 privilege level isolation was implemented. This involved defining a custom Global Descriptor Table (GDT) with user-mode segments, configuring the Task State Segment (TSS) for safe stack switching during interrupts, and writing inline assembly to manually execute the `iretq` instruction, forcing the CPU to transition from Ring 0 (kernel) to Ring 3 (user).

Contents

1 Overview

A robust operating system must isolate user applications from the kernel core. On x86 architectures, this isolation is enforced using protection rings:

- **Ring 0 (Kernel):** Full access to hardware and memory.
- **Ring 3 (User):** Restricted access. Privileged instructions such as `cli`, `hlt`, `in`, and `out` are prohibited.

The objective of this phase was to successfully execute code in Ring 3.

2 The Global Descriptor Table (GDT)

The Global Descriptor Table defines memory segmentation and privilege levels. While x86_64 systems primarily rely on paging, the GDT remains essential for privilege enforcement.

Four critical segments were defined:

1. **Kernel Code and Data** (Privilege Level 0)
2. **User Code and Data** (Privilege Level 3)

```rust
pub fn get_user_selectors() -> (u16, u16) {
    // The '| 3' sets the Requested Privilege Level (RPL) to Ring 3.
    // Without this, the CPU triggers a General Protection Fault.
    (
        GDT.1.user_code_selector.0 | 3,
        GDT.1.user_data_selector.0 | 3
    )
}
```

Listing 1: User Selector Logic

3 The Task State Segment (TSS)

The transition to Ring 3 introduced a critical failure point: **kernel stack switching**.

When code executing in Ring 3 triggers an interrupt (e.g., timer interrupt or page fault), the CPU transitions back to Ring 0 and must switch to a safe kernel stack. The stack pointer used for this transition is read from the **RSP0 field of the TSS**.

Triple Fault Root Cause

Initially, the `RSP0` field was not populated. As a result, any interrupt occurring in Ring 3 caused the CPU to attempt a stack switch to an invalid address, resulting in a triple fault and immediate system reset.

4 Ring Transition Logic (`iretq`)

The x86_64 architecture does not provide a direct instruction for transitioning from Ring 0 to Ring 3. Instead, the CPU must be tricked into performing a return from an interrupt.

This is accomplished by manually constructing an interrupt stack frame and executing the `iretq` instruction.

```
unsafe {
    asm!(
        "cli",                  // Disable interrupts during transition
        "mov ds, ax",           // Load user data selectors
        "mov es, ax",
        // ...
        "push rax",             // SS (User Data Segment)
        "push rsi",             // RSP (User Stack Pointer)
        "push 0x202",           // RFLAGS (Interrupt Flag set)
        "push rdi",             // CS (User Code Segment)
        "push rdx",             // RIP (Entry point)
        "iretq",                // Perform privilege switch
        // ...
    );
}
```

Listing 2: Inline Assembly for Ring Switch

5 Verification

To verify successful Ring 3 execution, a function named `user_mode_app`—located in kernel memory—was invoked after the transition.

Result

The CPU generated a **Page Fault (Interrupt Vector 14)** with an access violation error. This confirmed correct privilege enforcement, as Ring 3 code is prohibited from accessing kernel memory. This milestone completes Phase 8 and establishes the foundation for Phase 9 (Virtual Memory Management).