Chronos OS: Phase 10 Report

System Calls: Bridging User and Kernel Space

Development Log

January 20, 2026

Abstract

With memory protection (Phase 9) fully active, user-space applications are isolated in a sandbox, unable to access hardware or kernel memory directly. Phase 10 focused on implementing a controlled communication channel: The System Call. We configured the Interrupt Descriptor Table (IDT) to allow Ring 3 applications to trigger the legacy Linux syscall vector (`0x80`). We also resolved critical Page Faults by extending the Virtual Memory Manager (VMM) to unlock not just executable code pages, but also the User Stack.

Contents

1 Overview

A secure operating system requires that User Mode applications (Ring 3) cannot execute privileged instructions (like disabling interrupts or writing to the screen buffer). However, applications still need these services. The solution is the **System Call**: a mechanism where the App asks the Kernel to perform an action on its behalf.

2 Implementation

2.1 IDT Configuration (The Gate)

By default, x86 interrupts are privileged. If a User App tries to execute `int 0x80`, the CPU throws a General Protection Fault. We modified the IDT entry for Vector 128 (0x80) to explicitly allow Ring 3 access.

```
use x86_64::PrivilegeLevel;

// Allow Ring 3 (User) to trigger this interrupt
idt[0x80]
    .set_handler_fn(syscall_handler)
    .set_privilege_level(PrivilegeLevel::Ring3);
```

Listing 1: Opening the Syscall Gate

2.2 The User Stack Issue

During Phase 9 integration, we encountered persistent `PROTECTION_VIOLATION` errors. Analysis revealed that while we had unlocked the *Code Segment* for the user app, the *Stack Segment* remained in locked Kernel memory.

The Fix: We allocated a dedicated `USER_STACK` array, calculated its physical page address via the HHDM (Higher Half Direct Map), and used the VMM to set the `USER_ACCESSIBLE` flag on that page before the jump.

```
// Unlock Code
memory::mark_as_user(&mut mapper, app_addr);

// Unlock Stack (The missing link)
let stack_addr = unsafe { &USER_STACK as *const _ as u64 };
memory::mark_as_user(&mut mapper, stack_addr);
```

Listing 2: Unlocking the Stack

3 The Execution Chain

The `exec` command now performs the following sequence:

1. **Setup:** Initializes a 4KB User Stack.
2. **Unlock:** Modifies Page Tables to allow Ring 3 access to both the function code and the stack.
3. **Transition:** Executes `iretq` to switch CPU state to Ring 3.
4. **Action:** The User App executes `asm!("int 0x80")`.

5. **Trap:** The CPU switches back to Ring 0 (Kernel Mode) and jumps to `syscall_handler`.
6. **Result:** The Kernel prints "User requested Hello World".

4 Verification results

The system successfully displayed the confirmation message:

```
[KERNEL] System Call Received (Vector 80)
[KERNEL] Origin: User Mode (Ring 3)
[KERNEL] Action: User requested 'Hello World'
```

This confirms that the isolation barrier is both secure (defaulting to deny) and permeable via the correct channels (Syscalls).

5 Conclusion: The Kernel Era Ends

Phase 10 concludes the fundamental architecture of the Chronos Kernel. We have successfully implemented:

- A Time-Aware Scheduler.
- A Read-Only Filesystem.
- Hardware Isolation (Ring 0/3).
- Virtual Memory Management.
- System Call Interface.

The system is now ready for the development of true Userspace Applications (ELF loading) in the next era of development.