Chronos OS: Phases 25-28 Report

Persistent Storage, Filesystems, and Disk-Based Execution

System Architect

January 23, 2026

Abstract

Following the implementation of the system monitor, development shifted towards achieving data persistence. Phases 25 through 28 focused on interacting with non-volatile storage media. We implemented a hardware driver for ATA hard drives, a parser for the FAT32 filesystem, and a specialized loader capable of executing ELF binaries stored on disk. This transition allows the operating system to install and run software that is not compiled into the kernel image.

Contents

1 Phase 25: The ATA Hardware Driver

1.1 PIO Mode Communication

To interface with the hard drive, we implemented a driver for the legacy Parallel ATA (PATA) standard using Programmed I/O (PIO). Communication occurs via ports `0x1F0` through `0x1F7`.

1.2 Sector Addressing

The driver operates in LBA (Logical Block Addressing) mode (28-bit). Reading a sector involves a specific handshake:

1. **Wait:** Poll the Status Register (`0x1F7`) until the BUSY bit is clear.
2. **Select:** Write the LBA bytes to ports `0x1F3-0x1F5` and the drive select to `0x1F6`.
3. **Command:** Write `0x20` (Read Sectors) to the Command Port.
4. **Transfer:** Read 256 16-bit words from the Data Port (`0x1F0`).

2 Phase 26: FAT32 Filesystem Architecture

2.1 Struct Alignment

The FAT32 structure is strictly defined by the specification. To map raw disk bytes to Rust structures, we utilized the `#[repr(packed)]` attribute.

The Alignment Challenge: Accessing fields of a packed struct directly caused `unaligned_reference` errors (Rust Error E0793). We resolved this by copying fields to local variables before use.

2.2 The BIOS Parameter Block (BPB)

We successfully parsed Sector 0 to extract filesystem geometry:

- **Sectors Per Cluster:** Determines allocation granularity.
- **Reserved Sectors:** Offset to the File Allocation Table.
- **Root Cluster:** The starting point of the file tree (usually Cluster 2).

3 Phase 27: File Access

3.1 Directory Traversal

We implemented the `lsdisk` command to scan the Root Directory. The driver reads 32-byte `DirectoryEntry` structs, filtering out Long File Name (LFN) entries and deleted files (marked with `0xE5`).

3.2 Filename Matching

FAT32 stores 8.3 filenames in uppercase padding with spaces (e.g., "TESTAPP ELF"). We implemented a normalization routine to allow the shell to match user input ("testapp.elf") against the raw directory entries.

4 Phase 28: Execution from Storage

4.1 The Memory Mapping Problem

Running a program from Disk proved significantly harder than running from Ramdisk.

- **Ramdisk:** File is in a contiguous physical memory block managed by the Bootloader.
- **Disk:** File is read into a `Vec<u8>` on the Kernel Heap. The physical location of the Heap is not guaranteed to be contiguous or easily calculated via offsets.

4.2 The Solution: Manual Physical Loading

To execute the file securely in Ring 3, we implemented a new loader strategy in `rundisk`:

1. **Allocation:** We ask the Physical Memory Manager (PMM) for fresh, guaranteed-usable 4KB frames.
2. **Mapping:** We map these frames to the User Virtual Base Address (`0x400000`) using the VMM.
3. **Copying:** We perform a `memcpy` from the Heap buffer (the file data) into the newly mapped virtual address.

```
// Map valid physical RAM to 0x400000
let p_addr = memory::alloc_frame().as_u64();
memory::map_user_page(v_addr, p_addr);

// Copy from Vec to the new page
core::ptr::copy_nonoverlapping(src_ptr, v_addr as *mut u8, chunk_size);
```

Listing 1: The "Rundisk" Logic

5 Conclusion

The completion of Phase 28 marks a functional milestone. Chronos OS can now identify a hard drive, parse its filesystem, load a binary executable into memory, and execute it under hardware isolation. The successful trap of the System Call (`int 0x80`) confirms the integrity of the entire loading pipeline.