Chronos OS: Phase 7 Report

Data Persistence: The Ramdisk Filesystem

Development Log

January 19, 2026

Abstract

Phase 7 of Chronos OS addresses the requirement for data persistence. To transition toward a daily-usable system, the OS must be able to retrieve data from non-volatile storage. We implemented a Read-Only Ramdisk (Initrd) by utilizing the Limine bootloader's module protocol. This allows the OS to load files into memory during the boot process and provides a programmatic interface for the Shell to list and read their contents.

Contents

1 Objective

An operating system without a filesystem is functionally ephemeral. The objective of this phase was to enable the kernel to identify and read files provided by the bootloader. This serves as the foundation for loading external drivers, user-space applications, and configuration files in future phases.

2 System Architecture

2.1 The Ramdisk Strategy

Directly implementing NVMe or SATA drivers is outside the immediate scope of a "Year 1" roadmap due to the complexity of the PCIe bus. Instead, we utilized a **Ramdisk** approach:

1. **Packaging:** Files (e.g., `welcome.txt`) are placed in the ISO root.
2. **Handover:** The bootloader (`limine.cfg`) is instructed to load these files into a specific region of RAM.
3. **Discovery:** The kernel parses the `ModuleRequest` to find the physical address and size of these memory segments.

3 Implementation Details

3.1 Memory-to-File Mapping

We implemented a `File` abstraction in `src/fs.rs`. This structure maps the raw bytes provided by the bootloader into a usable Rust slice.

```
let start = module.addr() as *const u8;
let size = module.size() as usize;

// Create a static slice representing the file's data
let data = unsafe { core::slice::from_raw_parts(start, size) };
```

Listing 1: Parsing Modules into Files

3.2 C-Style String Conversion

A technical challenge arose regarding the filename format. Limine provides paths as `CStr` (null-terminated), whereas the Rust kernel utilizes UTF-8 `String`. We implemented a conversion routine using `to_str()` to safely bridge the gap between the bootloader's C-heritage and the kernel's memory-safe strings.

4 User Interaction: Shell Integration

To verify the filesystem, we extended the Shell with two utility commands:

- `ls`: Iterates through the module response and prints all found file paths.
- `cat <filename>`: Searches the ramdisk for a filename substring and prints the ASCII content of the corresponding data slice.

5 Results and Verification

We successfully verified the ability to read a 4-line text file (`welcome.txt`) from the virtual disk. The file contents were displayed correctly in the shell, confirming that:

1. The ISO build process correctly included the external module.
2. The kernel successfully handshaked with the bootloader's module protocol.
3. The memory management system (Heap) correctly handled the allocation of strings during the `read_file` process.

6 Conclusion

Chronos OS now possesses a "Storage" layer. While currently read-only, this capability allows the kernel to be separated from its data. The next major hurdle is **Protection**: ensuring that the Shell and future user applications cannot accidentally overwrite kernel memory.