PhonexOS

“Ash”

Architected System Hypercore

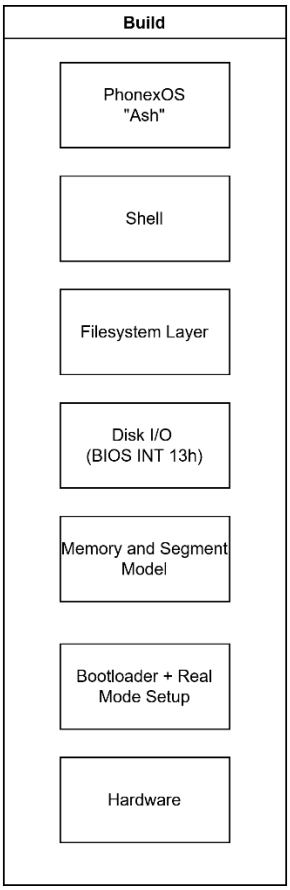
Aditya Mishra

github.com/aam-007/ash

Abstract. PhonexOS “Ash” is a custom-built operating system designed from first principles. It is not derived from Unix, DOS, or any traditional lineage. Instead, it is constructed to demonstrate how a complete computing environment can be built on bare hardware using simple, transparent, and deterministic ideas. Ash focuses on clarity over complexity. It emphasizes a readable boot process, a predictable filesystem, and a straightforward command line interface that handles essential file and utility operations. The system is intended as both an educational tool and a foundation for future architectural experiments.

1. System Overview

“Ash” is built for x86 systems running in real mode. It uses BIOS interrupts to communicate with hardware. There is no multitasking, no complex memory model, and no external dependencies. The OS consists of a bootloader, a kernel layer, a custom filesystem, and a command driven shell. Everything runs within a one-megabyte memory space.



2. Boot Process

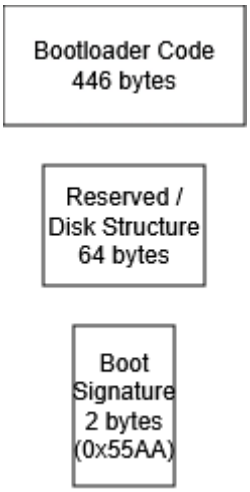
The boot process follows four stages. Each stage is minimal and predictable.

2.1 BIOS Stage

On startup, the BIOS performs hardware initialization and searches for bootable media. Once found, it loads the first 512 bytes of the boot device into memory at address 0x7C00. It then transfers control to that memory region.

2.2 Boot Sector Layout

The boot sector is arranged as follows:



The bootloader must fit entirely within the first 446 bytes. Its job is to load the “Ash” kernel from predetermined disk sectors and pass execution to it.

2.3 Transition to the Kernel

When the bootloader has loaded the kernel into memory, it sets up segment registers, clears the screen, loads the required interrupts, and jumps into the kernel entry point. The entire process is deterministic and does not rely on any dynamic memory.

2.4 Initialization Sequence

Once the kernel takes control, it completes these tasks:

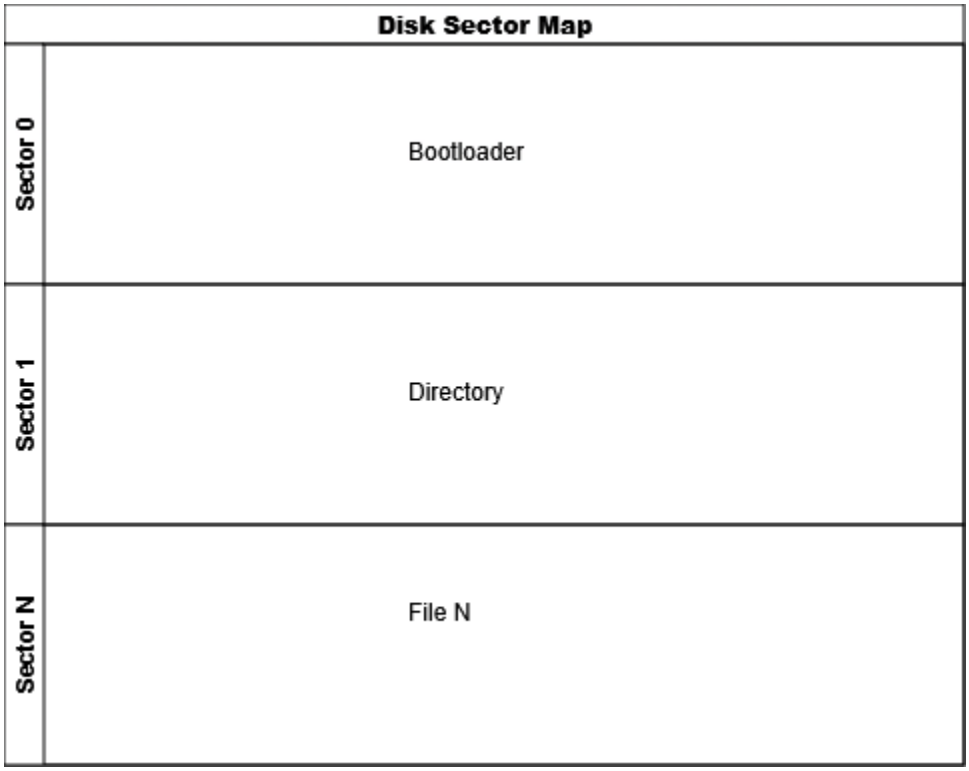
1. Initialize text output routines
2. Initialize keyboard input routines
3. Mount the filesystem by reading the directory sector
4. Prepare buffers and IO structures
5. Start the shell

The shell then becomes the primary interface for the user.

3. Filesystem Architecture

The filesystem is the core of “Ash”. It follows a fixed sector per file model. Each file occupies exactly one 512-byte sector.

3.1 Disk Layout Diagram



This structure avoids fragmentation and eliminates the need for allocation tables.

3.2 Directory Table Structure

The directory sector contains a list of entries with:

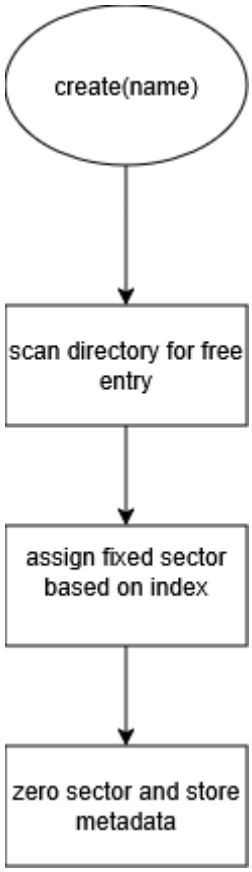
- filename
- an exists flag
- the associated sector index
- static metadata fields

Each entry has a fixed size. The directory has a maximum number of entries defined at compile time.

3.3 File Operation Algorithms

File operations are driven by very simple, deterministic algorithms.

File Creation



File Reading

- Look up filename
- Determine sector
- Read that sector with INT 13h
- Return its contents

File Deletion

- Clear entry in directory
- Zero the file sector
- Update the directory sector

Listing Files

- Iterate over directory entries
- Print names where exists flag is set

3.4 Sector Mapping Philosophy

Each file directly maps to an index. This makes file operations predictable and traceable. It also reflects the educational nature of “Ash”.

3.5 Limitations

The system allows only a small number of files, each limited to 512 bytes. This is intentional. “Ash” values transparency more than scalability.

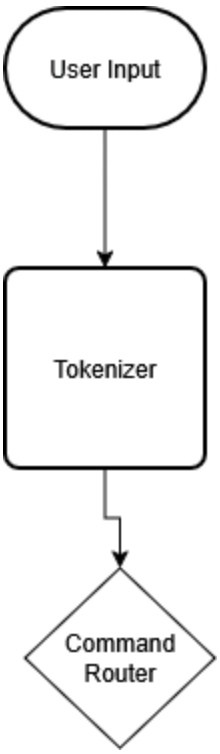
4. Shell Architecture

The shell is a simple but powerful interface.

4.1 Input Buffer Model

User input is collected into a fixed length buffer. Once the user presses Enter, the command is parsed and routed to the correct handler.

4.2 Command Execution Pipeline



4.3 Built in Commands

The shell supports several functions:

- write
- read
- delete
- list

- calculator
- clear
- help

4.4 Interaction Philosophy

The shell is quiet until spoken to. It prints only what the user requests. It avoids noise and always aims for clarity.

5. System Components

5.1 Memory Model

Real mode limits “Ash” to 1 MB of addressable memory. This limit simplifies the model and reinforces determinism.

0x00000 to 0x003FF Interrupt Vector Table
0x00400 to 0x00500 BIOS Data Area
0x07C00 Bootloader
0x10000 Kernel and Shell

5.2 BIOS Interrupts

“Ash” does not include hardware drivers. It uses BIOS interrupts for communication.

INT 10h Screen output

INT 16h Keyboard input

INT 13h Disk access

5.3 System Utilities

The OS includes simple utilities:

- A calculator that performs basic operations
- Color coded status messages for clarity
- Input validation for filesystem operations

These utilities help demonstrate the completeness of the environment.

6. Design Philosophy

“Ash” follows several guiding principles:

1. **Simplicity above all**
   Every system is easier to reason about when its mechanisms are visible.
2. **Determinism is a feature**
   No hidden state. No unpredictable memory allocation. No background tasks.
3. **A filesystem that refuses to hide its structure**
   One file equals one sector. Everything can be traced.
4. **The entire stack is yours**
   No dependencies. No legacy constraints.
5. **Minimalism as clarity**
   By removing unnecessary features, the remaining architecture becomes more approachable.

“Ash” is closer to an educational microkernel than to Unix or DOS. It gives the developer complete ownership of the system.

7. Security and Reliability

“Ash” is not a secure operating system by modern standards. It runs in real mode with no privilege separation. There is no process model and no user permissions. However, lack of complexity also eliminates many classes of bugs. No multitasking means no race conditions. No dynamic memory means no heap corruption. The system embraces a trust model where the user has complete control.

8. Future Extensions

“Ash” can be extended in many directions:

- Multi sector file storage
- FAT style allocation tables
- Protected mode
- Paging
- Pre-emptive multitasking
- ELF executable loading
- Basic GUI layer
- Network stack
- A scripting language for the shell

These extensions would move “Ash” closer to a general-purpose system, although they would reduce its simplicity.

9. Conclusion

“Ash” is a minimal, transparent, and deterministic operating system built entirely from scratch. It demonstrates how much functionality is possible within a very small architecture. By controlling the entire stack from the boot sector to the shell, the developer gains a deep understanding of how computers work at the lowest levels. The system stands as both a technical foundation and an educational project.

10. References

1. **Intel Corporation.**
   Intel 64 and IA-32 Architectures Software Developer’s Manual.
   Volume 1–3, 2023.
   The definitive reference for real mode memory layout, interrupt handling, segmentation, and CPU initialization.

2. **Phoenix Technologies.**
   BIOS Interface Technical Reference.
   1996.
   Core documentation for BIOS interrupts such as INT 10h, INT 13h and INT 16h, which “Ash” uses for video, disk, and keyboard operations.

3. **Tanenbaum, Andrew S. and Bos, Herbert.**
   Modern Operating Systems.
   Pearson, 2015.
   Explains foundational OS principles including bootstrapping, kernel structure, simple filesystems, and shell architecture.

4. **OSDev Wiki.**
   Bootloader Design, VGA Text Mode, ATA PIO, Simple Filesystems.
   OSDev.org, Accessed 2025.
   A practical, community-reviewed resource for low-level OS construction relevant to boot sectors, directory tables, and sector-based IO.

5. **ATA/ATAPI Command Set – ACS-4 Specification.**
   INCITS T13 Technical Committee, 2016.
   Provides authoritative details on sector-based disk operations that inform PhonexOS’s read and write routines.