Advanced x86:
BIOS and System Management Mode Internals
Boot Process

Xeno Kovah && Corey Kallenberg

LegbaCore, LLC

All materials are licensed under a Creative Commons “Share Alike” license.

http://creativecommons.org/licenses/by-sa/3.0/

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Attribution condition: You must indicate that derivative work
"Is derived from John Butterworth & Xeno Kovah’s ’Advanced Intel x86: BIOS and SMM’ class posted at http://opensecuritytraining.info/IntroBIOS.html”

- This covers an example of a minimal BIOS boot process, just to give an idea of what the BIOS does before your operating system starts to load.
- It doesn't include everything of course because it's geared towards minimal-configuration to achieve 100% functionality.

Boot Process at a glance*

1. Power-up
    1. 16-bit Real Mode accessing address FFFF_FFF0h
2. Operating mode selection
    1. Flat protected mode
3. Preparation for memory initialization
    1. CPU microcode update
    2. Set up CPU Cache as RAM (CAR)
    3. Chipset initialization (MCHBAR, PCIEXBAR, etc.)
4. Memory initialization
    1. Configure the memory controller
5. Post memory initialization
    1. Memory test
    2. Copy firmware from flash to memory for faster execution*
        1. In the doc this is listed as firmware shadowing but I'm modifying it somewhat since the BIOS can perform the copying of flash components to memory
        2. As a general rule, it's better to copy things to memory before execution as soon as possible
    3. Memory transaction redirection (PAM registers)
    4. Setting up the stack (before memory initialization, stack will be in CPU cache)
    5. Transfer to DRAM (the BIOS flash that was copied to memory is now executed)

*This list composed from Intel Minimal Architecture Boot Loader

Boot Process at a glance*

6. Miscellaneous platform enabling
    1. Platform dependent: Initialize clocks to operate at N Hz, configure general-purpose I/O, etc.
7. Interrupt enabling - PIC (Programmable Interrupt Controller), LAPIC (Local Advanced PIC), I/O APIC, etc.
8. Interrupt tables
    1. Interrupt Vector Table (IVT)
        1. Interrupt Service Routine (ISR) – INT 10h, etc…
    2. Interrupt Descriptor Table (IDT)
        1. Exceptions
9. Setting up timers
    1. In ICH: RTC (, HPET, 8254 Programmable Interrupt Timer (PIT), TCO timer (used as watchdog)
    2. In CPU: LAPIC timer
10. Memory caching control
11. Processor discovery and initialization
    1. CPUID
    2. Startup Inter-Processor Interrupts (SIPIs) are written to a processor's LAPIC telling it to wake up and where to start executing code at (must be a 4KB aligned boundary)
        1. Intel SW Programmer's Guide has more detail on this
    3. Each processor will awaken in Real Mode

*This list composed from Intel Minimal Architecture Boot Loader

Boot Process at a glance*

12. I/O devices
    1. Super I/O can control PS/2 keyboard, serial, parallel, etc. interfaces
13. PCI device discovery
    1. Enumerate PCI devices
    2. Assign address space (port IO, Memory Mapped IO)
    3. Detect and execute Expansion ROMs (XROMs)
14. Memory map
    1. Region types
        1. Memory
        2. Reserved
        3. ACPI Reclaim
        4. ACPI NVS
        5. ROM
        6. IOAPIC
        7. LAPIC
    2. Region locations
        1. Consult your datasheet for a complete list, but we'll cover some of the more relevant ones
15. Non-Volatile (NV) storage
    1. CMOS
    2. NV SPI flash
16. Handoff to boot-loader

*This list composed from Intel Minimal Architecture Boot Loader

Only 16 steps? BIOS is simple!

- Easy to summarize, not so easy to implement
- Very complex implementations
- Even if it's only a couple MB, it's a very DENSE couple of MB
- That list taken from the Intel Minimal Architecture Boot Loader which provides a pretty good list of functional requirements
- What is left out is:
  - Initializing SMM and locking down SMRAM
  - Locking down the BIOS Flash

*This list composed from Intel Minimal Architecture Boot Loader

How do you learn about all that stuff?

- You could go review open source BIOSes' code (CoreBoot or UEFI's TianoCore reference code)
- Or you can sign an NDA with Intel and read their very large "BIOS Writer's Guide" and associated documentation for each CPU/ chipset
- At the end of the day, the open source code is more or less just doing what they're told to do by the Intel documentation