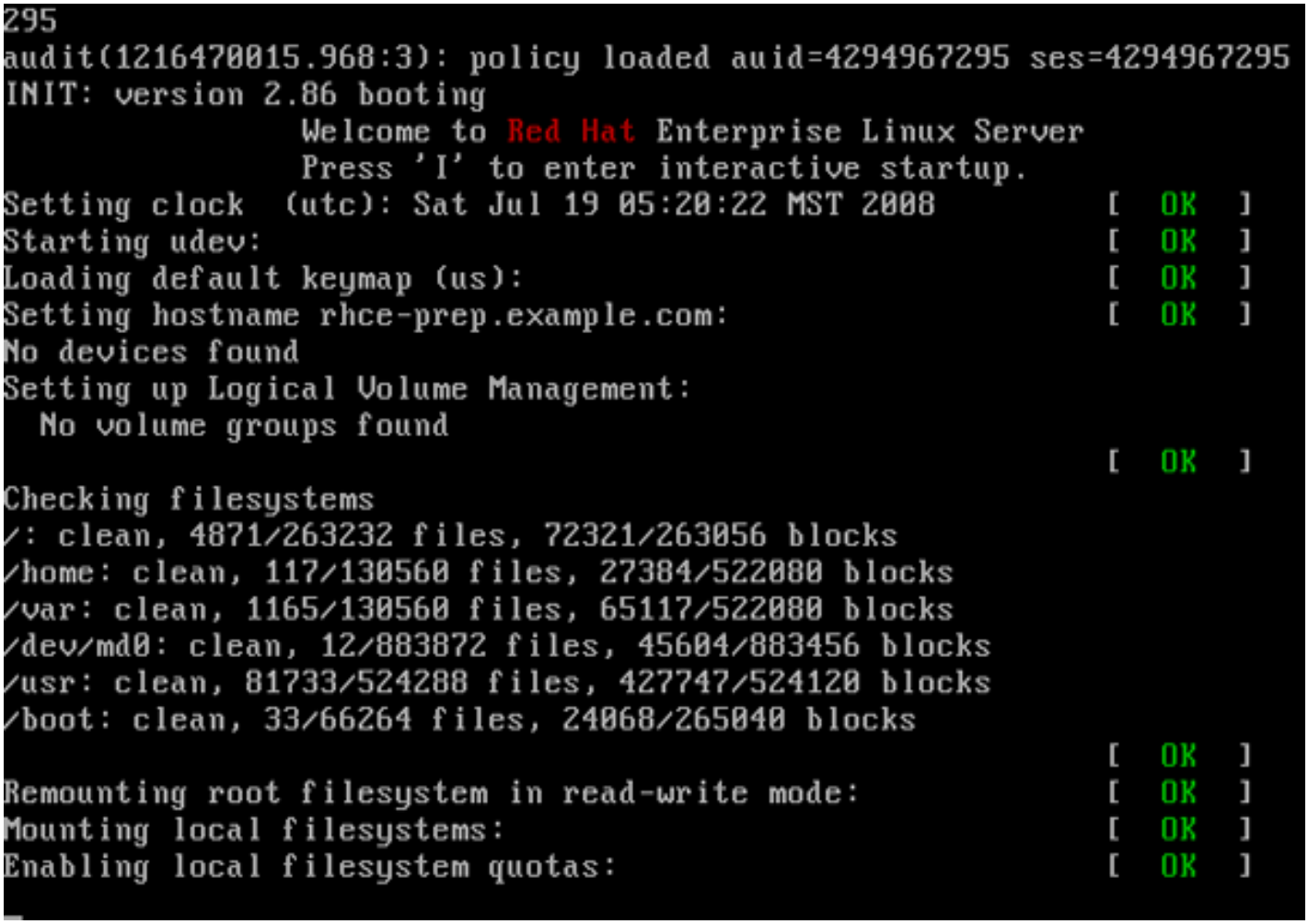
The Linux Boot Process

The "magic" that happens behind the scenes when you turn on your computer



Written by Brooke Basile for ENGL 202C, Spring 2020

Purpose

The purpose of this document is to provide a clean, easy to use resource for learning about the Linux (Intel x86) booting process. This document is aimed towards college students or hobbyists seeking a solid, conceptual knowledge of the boot sequence. This document won't include assembly or C code examples. In this document you will find visuals and definitions in the side panels to help assist you in your learning.

Understanding the booting process will assist you in understanding many other topics in Linux, and can even strengthen your programming and bug solving skills. It can also help you to understand the Windows booting process which, despite having some unique obscurities, follows the same general sequence of major events.

This document assumes that the reader already has an understanding of computer hardware and basic memory storage terms-- registers, primary and secondary memory, pointers, etc.

Introduction

The Linux boot process can be divided into six stages:

1. **BIOS/UEFI**: System initialization stage that enables the hardware and software of a computer to communicate with each other-- firmware
    1. ***BIOS***: Basic Input/Output System
    2. ***UEFI:*** Unified Extensible Firmware Interface
2. **MBR/GPT**: Loads and executes the bootloader
    1. ***MBR:*** Master Boot Record
    2. ***GPT:*** GUID Partition Table
3. **GRUB**: *GRand Unified Bootloader*-- loads and starts the kernel.
4. **Kernel**: The core of the operating system, upon which all other components rely.

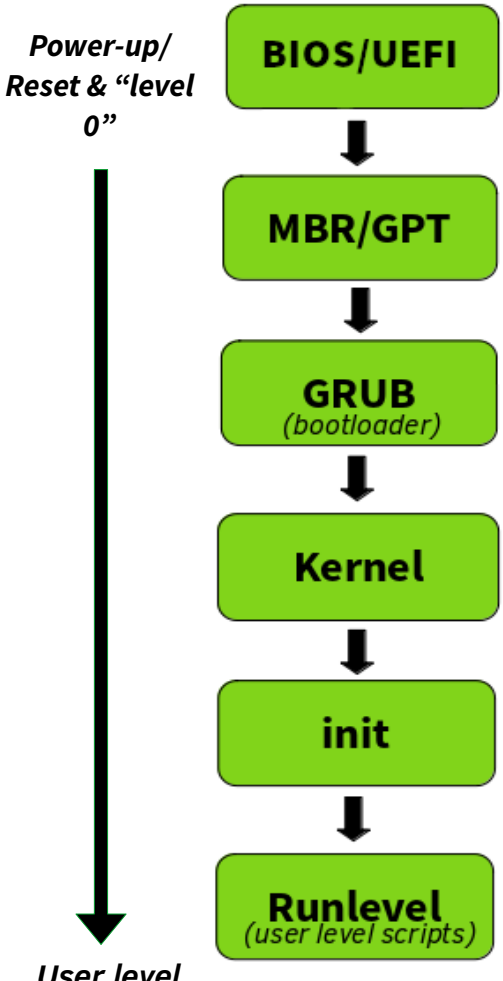


Figure 1. Linux boot process flow.

5. **Init**: The first process to run immediately after the operating system loads-- high-level system initialization.
6. **Run level**: Controls the groups of processes started by init.

These are stages that occur "before the login prompt"-- these stages set up the necessary building blocks to actually display that prompt on the screen, allow your keyboard to communicate with the computer, connect to the internet, mount filesystems, and so on.

Before getting into the BIOS/UEFI processes, which are two unique processes that each serve the same purpose, there's a few processes that occur beforehand in all computers-- think of it as the "*pre-pre-boot*". Immediately after pressing the power button on your computer, it starts working. The motherboard sends a signal to the power supply device. After receiving the signal, the power supply provides the proper amount of electricity to the computer-- the platform and the processor-- during a process known as ***power sequencing***. Once the motherboard receives the ***power good signal***-- a signal provided by the computer power supply to the motherboard that all of the voltages are within specification-- it tries to start the CPU. If this signal is not present in startup, the CPU is held in a reset state. If the signal goes down during operation, the CPU will shut down.

The CPU's initialization is triggered by a series of ***clock ticks*** generated by the ***system clock***, which functions to coordinate all CPU and memory operations by determining the ***clock speed***-- how fast instructions execute. You may already be acquainted with this concept of "clock ticks" from terms such as *under* and *overclocking*. The CPU then resets all leftover data in its registers (temporary storage areas) and sets predefined data values for each of them; a set of values shared by all CPUs since the Intel 80386.

The hardware then enters ***real mode***. The CPU's starting address is a point called the ***reset vector***, the memory location at which the CPU expects to find its first instructions to execute after reset-- it is these instructions that start the operation of the CPU, the first step in the process of booting the system. BIOS firmwares generally remains in real mode until the kernel is fully initialized, while UEFI firmwares switch to ***protected mode*** within a few instructions of exiting processor reset.

power sequencing: the process by which specific voltages are provided to the platform and processor in a specific sequence

power good signal: a signal provided by a computer power supply to indicate to the motherboard that all of the voltages are within specification and that the system may proceed to boot and operate, protects the computer from damaging itself by operating on improper voltages

system clock: synchronizes all components of the motherboard by producing clock ticks

clock ticks: electrical pulses generated by the system clock

clock speed: frequency at which the system clock can generate pulses; synchronizes all operations

real mode: an operating mode of all x86 compatible CPUs, addresses in real mode always correspond to real locations in memory

reset vector: a pointer or address at which the CPU expects to find its first instructions after reset, these instructions start the operation of the CPU, the first step in the booting process

BIOS/UEFI

From the reset vector, the CPU is hardwired to run instructions from a physical component, called ***NVRAM***-- nonvolatile random access memory. These instructions constitute the system's ***firmware***-- the UEFI or BIOS. The system's firmware is the lowest level of software that interfaces with the hardware as a whole, and is the interface by means of which the bootloader and operating system kernel can communicate with and control the hardware. UEFI is a modern variant that offers more features and customizations, where BIOS is largely found on legacy devices.

NVRAM: a programmable and nonvolatile RAM; store the system's firmware and hold initial processor instructions required to bootstrap a computer system

system firmware: the lowest level of software that interfaces with the hardware as a whole, and is the interface by means of which the bootloader and operating system kernel can communicate with and control the hardware

Foremost, it's important to address a process common to both-- the ***power-on self test,*** or ***POST***. The POST is the first step of both the BIOS and UEFI. The job of the POST is to perform a check of the hardware. If there are any hardware errors, the POST will return a series of beeps called beep codes to indicate to the user what type of error has occurred. After the POST, the main memory is initialized and the system firmware is shadowed from the ROM into the RAM and initialization continues.

POST: "Power On Self Test"; part of a computer's pre-boot sequence, checks hardware for errors

As previously stated, BIOS and UEFI constitute the system's firmware. The system's firmware has two key responsibilities after the POST:

1. To enumerate and initialize various hardware components connected to the system like disk controllers and network interfaces that are required for booting.
2. Locate and load the bootloader on the primary boot device (a storage device-- i.e. hard drive, USB, floppy).

BIOS

After the POST and initialization of all necessary hardware components, the BIOS runtime searches for devices that are active and bootable in a predefined order of preference. When the device is located, the BIOS locates and loads the ***boot sector*****.** In BIOS, this boot sector is located on an MBR partition found on the first 512-byte sector of the boot disk. After the

boot sector: the sector of a persistent data storage device which contains machine code to be loaded into RAM and then executed by a computer system's built-in firmware

MBR is loaded into RAM, the BIOS yields control to it.

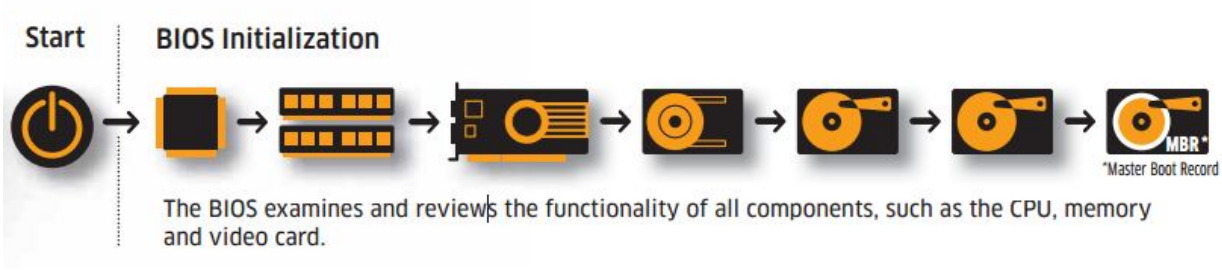


Figure 2. BIOS initialization

UEFI

Unlike BIOS, UEFI does not rely on a boot sector, defining instead a boot manager as part of the UEFI specification. UEFI booting consists of reading the boot entries in the NVRAM-- maintained by the boot manager-- to locate an ***EFI System Partition*** located on the GPT in place of MBR. UEFI will then utilize unique ***EFI applications*** located therein.

A unique feature of UEFI is the UEFI shell, which allows the user to examine memory, drive contents, verify device configuration, use the network, and output logs. This shell is found before moving into the GPT and accessing the bootloader.

EFI System Partition: contains bootloaders or kernel images, device driver files, system utility programs, and logs

EFI applications: binary files that are loaded into memory, have their entry-point function called, and then unloaded from memory and depend on some of the elements of UEFI (accessed via the EFI System Table)

EFI system table: contains pointers to the runtime and boot services tables

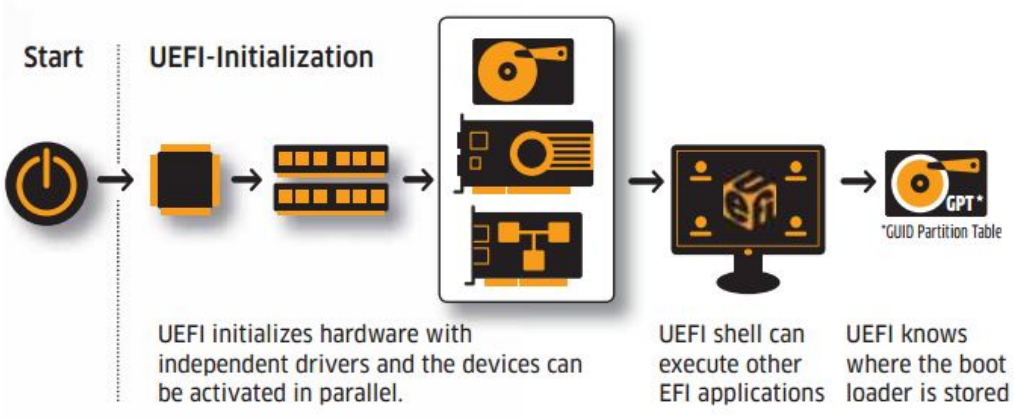


Figure 3. UEFI Initialization

MBR/GPT

MBR

The Master Boot Record is a partioning scheme for hard disks that contains the boot sector, stored in the the first 512 byte sector. These 512 bytes of code contain the ***boot image (boot.img)*** which functions to locate and read the first sector of the ***core image (core.img)*** and jump to it. Because of MBR's size restrictions, the bootloader hardcodes the location of the first sector of the core mage.

boot.img: the first part of the bootloader; written to the 512 byte MBR, its function is to read the first sector of the core image from a local disk and jump to it

core.img: the core image of GRUB (bootloader), contains enough modules to access /boot/grub/ and loads everything else from the file system at run-time

GPT

The GUID Partition Table is a partitioning scheme unique to UEFI; it uses ***universally unique identifiers (UUIDs)*** to define partitions and partition types. The GPT also contains a protective MBR for legacy reasons. The GPT can hold significantly more bytes in comparison to the MBR, supporting a maximum partition size of up to 9.4 ZB.

universally unique identifiers (UUIDs): a 128-bit number used to identify information on computer systems

On the GPT, the UEFI firmware identifies the EFI system partition, on which lies the executable second-stage boot loader; in our case this is an EFI application named "grub.efi", which is capable of mounting ***/boot*** and continuing the boot process.

/boot: a directory that holds files used in booting the operating system; for GRUB this is /boot/grub

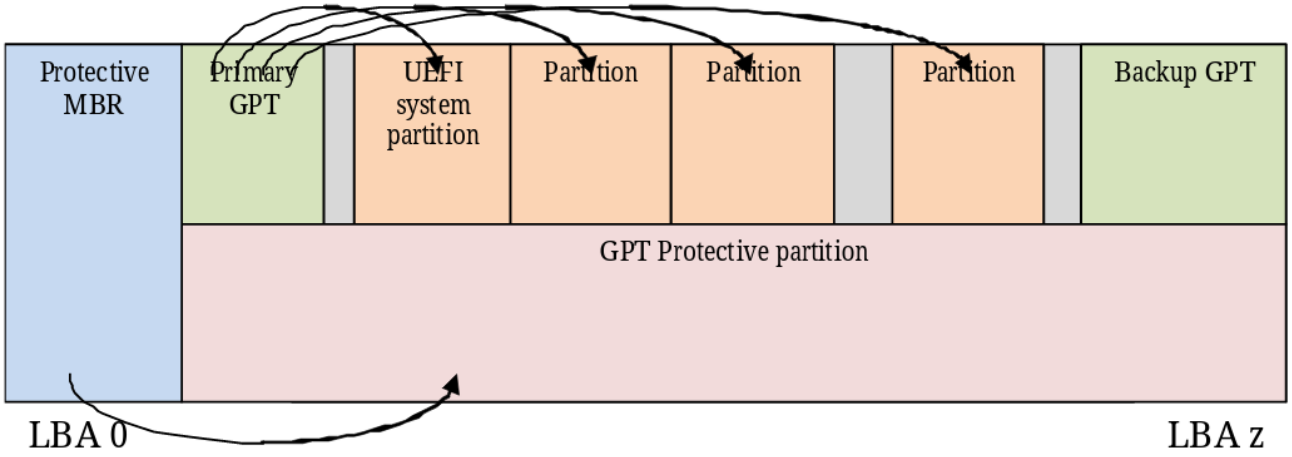


Figure 4. A sample GPT disk layout.

GRUB

Once control of the computer has been handed off from the BIOS/UEFI to the ***bootstrap code*** in the MBR/GPT and from the MBR/GPT to the bootstrap code in the boot sector partition, the actual logic involved in determining which operating system to load, where to load it from, which parameters/options to pass to it, and how to complete any interactions with the user that might be available-- the actual process of starting the operating system-- begins.

A ***bootloader*** is the first software program that runs when a computer starts. It is responsible for loading and transferring control to an operating system ***kernel***. Many bootloaders exist, however, GRUB (GRUB2) is the most common bootloader seen in Linux systems.

On BIOS systems, GRUB is initialized using a variety of previously discussed bootstrap images, ending with the loading of the core.img, which contains GRUB 2's kernel and drivers for handling filesystems, into memory.

On UEFI systems, GRUB is booted by the firmware directly via grub.efi in the EFI System Partition, without a boot.img.

bootstrap code: small, self starting code that loads and executes the next stage in the boot procedure (i.e. boot.img)

bootloader: loads one or more operating systems by reading file system information or through a pointer to a disk partition containing operating system files

kernel: in an operating system, the core portions of the program that reside in memory and perform the most essential operating system tasks, such as handling I/O operations and managing the internal memory

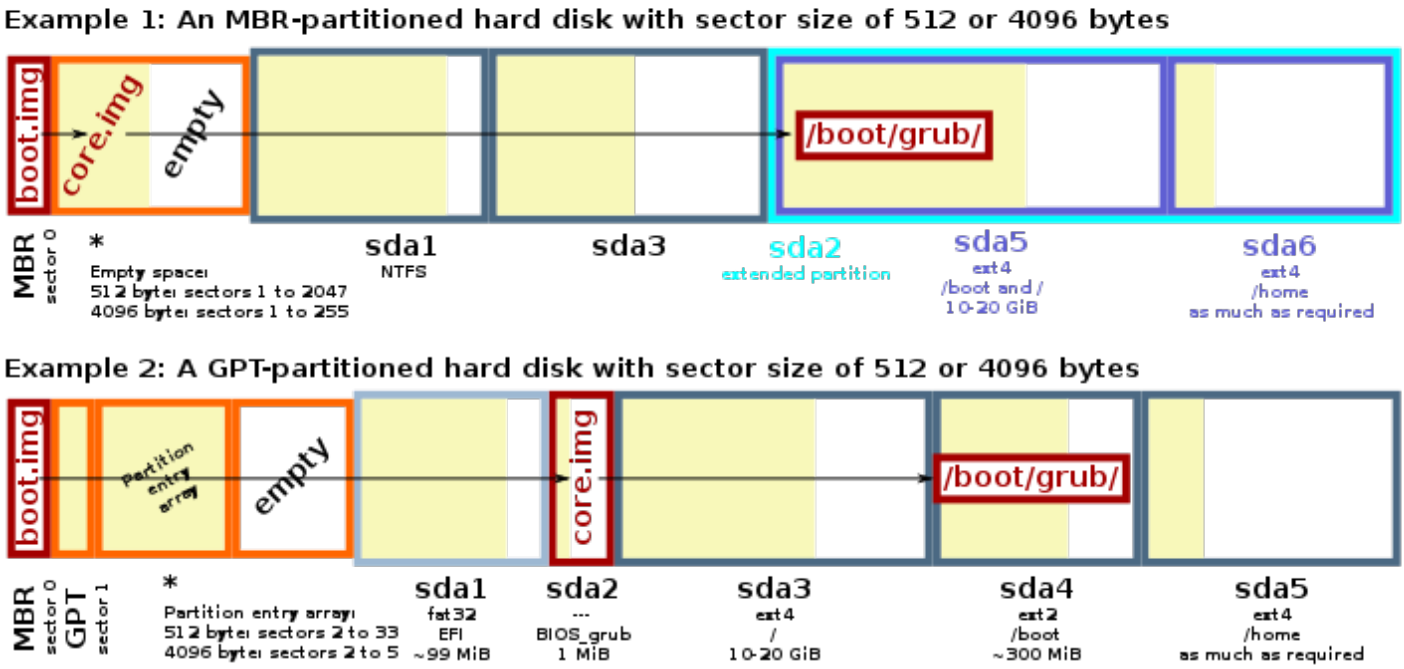


Figure 5. GNU GRUB 2 locations of boot.img, core.img, and the /boot/grub directory on both MBR and GPT partitioned disks.

After startup, GRUB presents a menu where the user can choose from operating systems found by GRUB. This is typically the first GUI presented to the user upon booting--that's right, everything we've been talking about before now happens in the background, away from user space.

Once the boot options have been selected, GRUB loads the specified kernel image and the ***initramfs*** into memory and passes control to the kernel.

initramfs: an initial ram file system that is offered to the kernel prior to when the real root file system is available; provides the necessary executables and system files to load the modules to make the real file systems available and access the real root file system

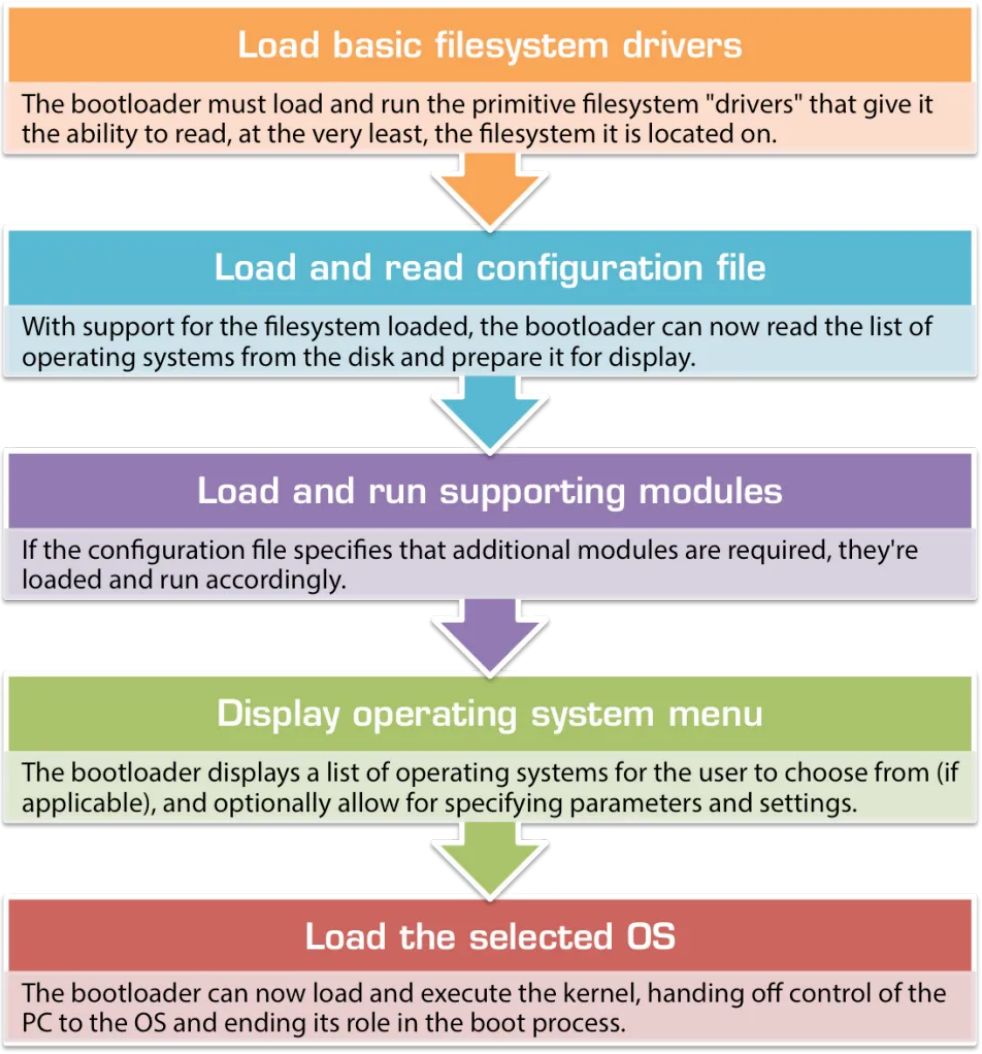


Figure 6. Summary of the GRUB2 bootloader flow.

Kernel

Understanding the entirety of the Linux kernel can be likened to taming an elder dragon, so this section will be kept brief. To get an idea of just how complex the Linux kernel is, and just how *much* the Linux kernel does, refer to this map of the Linux kernel:

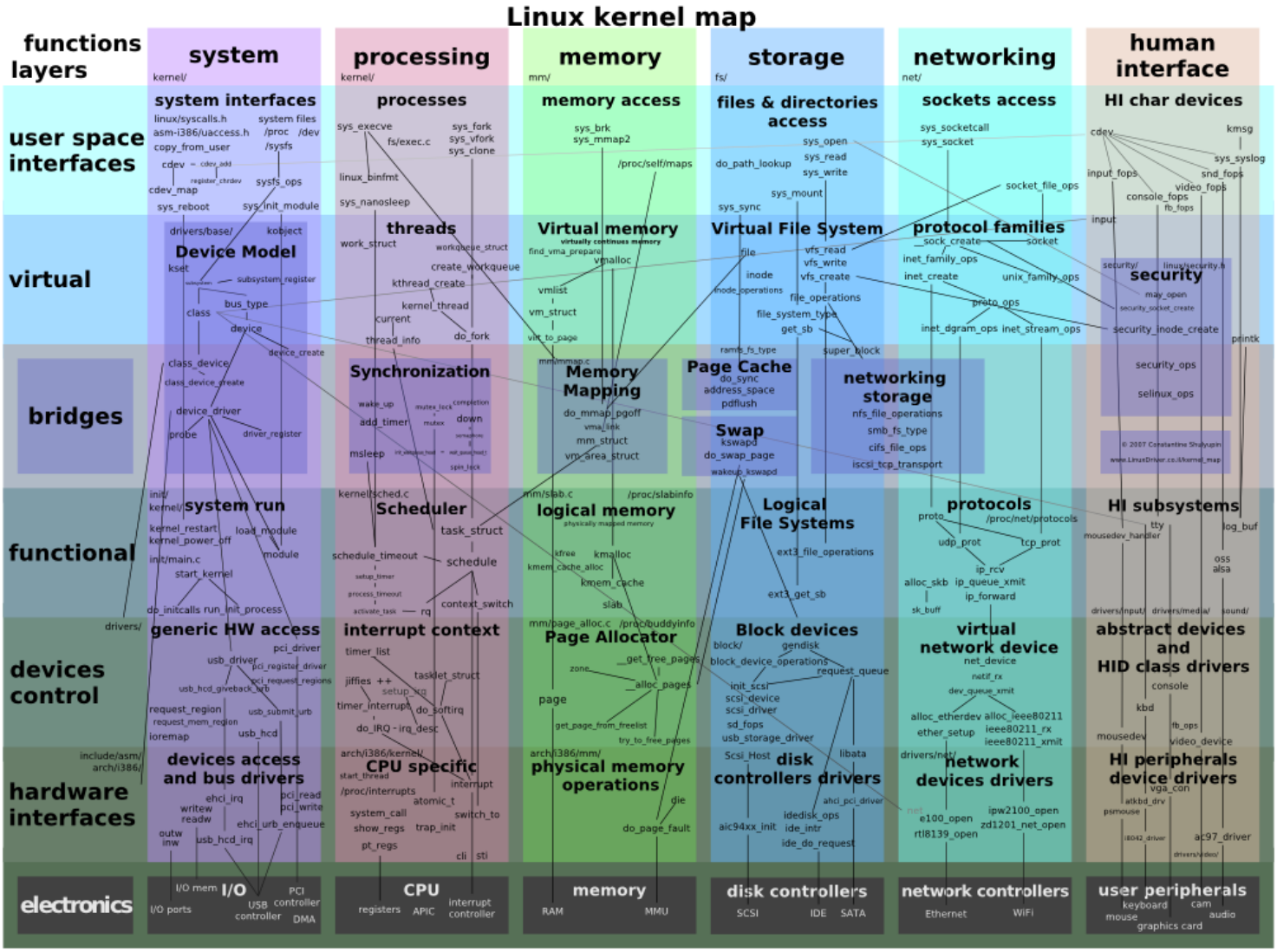


Figure 7. Map of the Linux kernel.

The kernel has complete control over everything in the system-- it facilitates interactions between hardware and software. The kernel handles input and output requests from software, translating them into instructions for the CPU. The kernel handels memory and peripherals like keyboards, monitors, and speakers. Think of the kernel as a layer between what you see on the screen and the bare metal beneath; without it, you wouldn't be able to communicate with that bare metal at all, whether via keyboard, mouse, or screen.

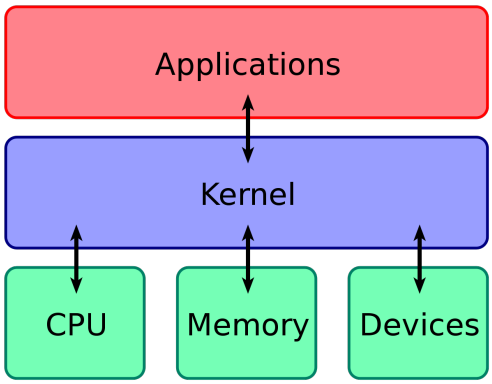


Figure 8. The relationship between user space, the kernel, and bare metal processes.

The Linux kernel is most often stored as binary in the form of a compressed image in the /boot directory. The ***Linux kernel image*** is a "snapshot" image of binary when the kernel was in a running state, an image of the Linux kernel that is able to run by itself (autonomously) after giving control to it.

kernel image: a "snapshot" of the kernel binary when it was in a running state, requiring no compilation/installation, it is able to run by itself after giving control to it

The first step of the kernel phase is the ***kernel loading stage***. At the head of this kernel image is a routine that does some minimal amount of hardware setup and then decompresses the kernel contained within the kernel image and places it into high memory. With the initramfs present, this routine moves initramfs into memory and notes it for later use. The routine then calls the kernel and the ***kernel startup stage*** begins.

kernel loading stage: hardware setup, decompression of the kernel image and transfer into memory

kernel startup stage: where the kernel is loaded

When the kernel is loaded, it initializes and configures the computer's memory and the various hardware attached to the system. It then locates the compressed initramfs image in in a predetermined location in memory, decompresses it, mounts it, and launches the init script located within. This init script is typically a shell script whose purpose is to mount the root file system.

init

Init is the first user-space process started during the booting of the computer system, and is a ***daemon*** process that continues running until the system shuts down.

daemon: a computer program that runs a background process, rather than beign under the direct control of an interactive user

The first thing init does is read its initialization file, which gives init an initial configuration script for the environment to take care of everything your system needs to have done at

its initialization-- setting the path, checking and mounting file systems, setting the clock, and so on. Init continues to read this configuration file to gauge how the system should be set up in each run level, and sets the default ***run level*.**

After determining the default run level for the system, init starts all of the background processes that are necessary for the system to run, progressing through the run levels to reach the default run level.

Run level processes

At this point in the boot process, the init executes all the user level processes-- the SSH daemon, timeserver daemon, windowing system, display manager, and so on. Now, the computer has completely finished booting and (depending on the configuration) will display a user login interface to bring the user into the graphical desktop environment.

Literature Cited

1. 6 stages of Linux boot process (Startup sequence). (n.d.). The Geek Stuff. https://www.thegeekstuff.com/2011/02/linux-boot- process/
2. About initramfs. (n.d.). Welcome to Linux From Scratch!. https://www.linuxfromscratch.org/blfs/view/svn/postlfs/initramfs.html
3. Alex. (n.d.). From bootloader to kernel · Linux inside. GitBook. https://0xax.gitbooks.io/linux-insides/content/Booting/linux-bootstrap-1.html
4. Arch boot process. (n.d.). ArchWiki. Retrieved April 15, 2020, from https://wiki.archlinux.org/index.php/Arch_boot_process
5. The BIOS/MBR boot process. (2015, February 28). NeoSmart Knowledgebase. https://neosmart.net/wiki/mbr-boot-process/
6. Booting an operating system. (2015, April 12). Rutgers University Computer Science. https://www.cs.rutgers.edu/~pxk/416/notes/02-boot.html
7. Dice, P. (2017). Quick boot.
8. EFI system table. (n.d.). Google Sites. https://sites.google.com/site/uefiforth/bios/uefi/uefi/4-efi-system-table/4-3-efi-system-table
9. GRUB. (n.d.). The GNU Operating System and the Free Software Movement. https://www.gnu.org/software/grub/manual/grub/grub.pdf
10. Inside the Linux boot process. (n.d.). IBM Developer. https://developer.ibm.com/technologies/linux/articles/l-linuxboot/
11. Introduction to Linux. (n.d.). The Linux Documentation Project. https://tldp.org/LDP/intro-linux/html/sect_04_02.html

12. Introduction to the booting process. (n.d.). doc.opensuse.org - Documentation Guides & Manuals. https://doc.opensuse.org/documentation/leap/archive/42.3/reference/html/book.opensuse.reference/cha.boot.html

13. (n.d.). The Linux Documentation Project. https://www.tldp.org/LDP/Linux-Dictionary/Linux-Dictionary.pdf

14. Linux for Researchers. (n.d.). The University of Virginia. https://galileo.phys.virginia.edu/compfac/courses/sysadmin1/13-kernel/presentation-notes.pdf

15. Overview of booting process. (n.d.). books.gigatux.nl. https://books.gigatux.nl/mirror/applicationdevelopment/7277/DDU0009.html

16. Some basics of MBR V/S GPT and BIOS V/S UEFI. (n.d.). Manjaro. https://wiki.manjaro.org/index.php?title=Some_basics_of_MBR_v/s_GPT_and_BIOS_v/s_UEFI

17. UEFI Specification. (n.d.). Unified Extensible Firmware Interface Forum. https://uefi.org/sites/default/files/resources/UEFI_Spec_2_8_A_Feb14.pdf

18. Understanding the boot process — BIOS vs UEFI. (n.d.). Linux Hint – Exploring and Master Linux Ecosystem. https://linuxhint.com/understanding_boot_process_bios_uefi/