101.2 Boot the system

Contents

Key knowledge areas

- Provide common commands to the boot loader and options to the kernel at boot time.
- Demonstrate knowledge of the boot sequence from BIOS/UEFI to boot completion.
- Understanding of SysVinit and systemd.
- Awareness of Upstart.
- Check boot events in the log files.

Partial list of the used files, terms and utilities

- dmesg
- journalctl
- BIOS
- UEFI
- bootloader
- kernel
- initramfs
- init
- SysVinit
- systemd

/etc/inittab
Configuration file read by init when it starts up to determine what process to start for various run levels.

BIOS
An acronym for Basic Input Output System, is software that controls basic computer hardware functions. BIOS is stored on a motherboard chip.

SysVinit
The traditional service management package for Linux, containing the init program (the first process that is run when the kernel has finished initializing) as well as some infrastructure to start and stop services and configure them.

UEFI
The Unified Extensible Firmware Interface (UEFI) is system firmware. UEFI uses a special partition on a drive called an EFI System Partition to store the bootloader for an OS.

bootloader
The first piece of software started by the BIOS or UEFI. It is responsible for loading the kernel with the wanted kernel parameters, and initial RAM disk before initiating the boot process.

dmesg

Displays the contents of the system message buffer

init
The parent of all processes on the system, it is executed by the kernel and is responsible for starting all other processes.

initramfs
Iinitial RAM file system, used by Linux systems to prepare the system during boot before the operating systems' init process starts.

journalctl
The log file viewer for binary log files.

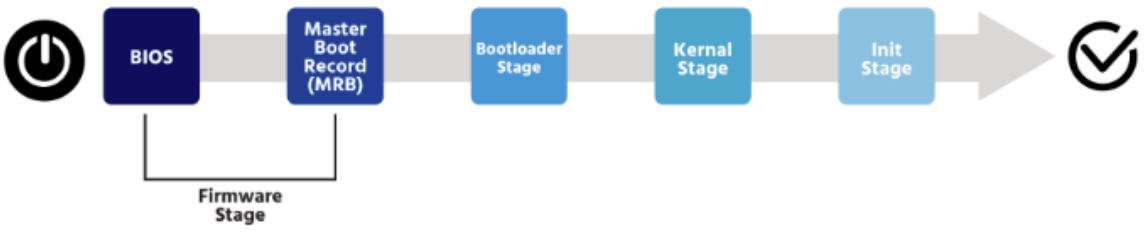
kernel
The central module of an operating system. It is the part of the operating system that loads first, and it remains in main memory.

systemd
A full replacement for init with parallel starting of services and other features, used by many distributions.

Boot Process

The boot process takes place in four main stages, some of which are modified by administrators, while for the others, it is sufficient just to be aware of the actions taking place:

1. Firmware Stage
2. Bootloader
3. Kernel Stage
4. Init Stage



In order to control the machine, the operating system's main component — the kernel — must be loaded by a program called a bootloader, which itself is loaded by a pre-installed firmware such as BIOS or UEFI. The bootloader can be customized to pass parameters to the kernel, such as which partition contains the root filesystem or in which mode the operating system should execute. Once loaded the kernel continues the boot process identifying and configuring the hardware. Lastly, the kernel calls the utility responsible for starting and managing the system's services.

Firmware Stage

The firmware stage is the first stage to take place after the computer is powered on. Most PC firmware is referred to as the *Basic Input/Output System (BIOS)*. Even on systems that have replaced the traditional BIOS with the *Unified Extensible Firmware Interface (UEFI*), the system firmware is still often referred to as BIOS.

BIOS

The BIOS has three main jobs to perform as part of the first stage of the boot process:

1. Execute a power-on self test (POST) in order to ensure the hardware of the system is functioning properly. The POST runs some basic functional checks on the CPU, memory, and peripherals so that obvious errors, such as missing memory chips or malfunctioning disk devices are found early in the boot cycle.

2. Enumerate available hardware such as memory, disks, and USB devices.

3. Find the proper boot drive from the available storage devices and load the Master Boot Record (MBR) from that device. The Master Boot Record is the first sector (or 512 bytes) of the disk. UEFI uses a special partition on a drive called an EFI System Partition to store the bootloader for an OS.

The BIOS assumes that the first 440 bytes in the first storage device — following the order defined in the BIOS configuration utility — are the first stage of the bootloader (also called bootstrap). The first 512 bytes of a storage device are named the MBR (Master Boot Record) of storage devices using the standard DOS partition schema and, in addition to the first stage of the bootloader, contain the partition table.

UEFI

The UEFI does not rely on the MBR, taking into account only the settings stored in its non-volatile memory (NVRAM) attached to the motherboard. These definitions indicate the location of the UEFI compatible programs, called EFI applications, that will be executed automatically or called from a boot menu. The standard compatible filesystems are FAT12, FAT16 and FAT32 for block devices and ISO-9660 for optical media. This approach allows for the implementation of much more sophisticated tools than those possible with BIOS.

The partition containing the EFI applications is called the EFI System Partition or just ESP. This partition must not be shared with other system filesystems, like the root filesystem or user data filesystems. The EFI directory in the ESP partition contains the applications pointed to by the entries saved in the NVRAM.

Generally speaking, the pre-operating system boot steps on a system with UEFI are:

1. The POST (power-on self-test) process is executed to identify simple hardware failures as soon as the machine is powered on.

2. The UEFI activates the basic components to load the system, like video output, keyboard and storage media.

3. UEFI's firmware reads the definitions stored in NVRAM to execute the pre-defined EFI application stored in the ESP partition's filesystem. Usually, the pre-defined EFI application is a bootloader.

4. If the pre-defined EFI application is a bootloader, it will load the kernel to start the operating system.

The UEFI standard also supports a feature called Secure Boot, which only allows the execution of signed EFI applications, that is, EFI applications authorized by the hardware manufacturer. This feature increases the protection against malicious software, but can make it difficult to install operating systems not covered by the manufacturer's warranty.

Bootloader Stage

The Master Boot Record contains a partition table and a very small amount of executable code called the first stage bootloader whose purpose is to load the more feature-rich second stage bootloader. The bootloader will perform several operations, but the primary task is to load the Linux kernel into memory and execute it.

A second stage is necessary when the code needed to initialize the system properly and start the operating system does not fit in the MBR. In this case, the first stage bootloader is there to point to the second stage bootloader, which can be larger and will be the bootloader that actually starts the operating system. Once that has occurred, the kernel takes over booting the system.

UEFI systems give the firmware stage a lot more memory and capabilities so that it can handle much larger and more complicated hardware.

The most common bootloader used on machines is the *Grand Unified Bootloader (GRUB)*.

It is also possible to boot off the network through the *Preboot Execution Environment (PXE)*. In the PXE system, a compatible motherboard and network card contain enough intelligence to acquire an address from the network and use the Trivial File Transfer Protocol (TFTP) to download a special bootloader from a server.

It is possible to boot multiple operating systems at different times off of one computer in a process known as dual or *multi-booting*.

The bootloader can also pass parameters to the kernel. GRUB provides a reasonably powerful command line interface that lets an administrator make changes to the kernel before it boots without requiring that the configuration be written to disk.

Some of the most useful *kernel parameters* are:

acpi

Enables/disables ACPI support. acpi=off will disable support for ACPI.

init

Sets an alternative system initiator. For example, init=/bin/bash will set the Bash shell as the initiator. This means that a shell session will start just after the kernel boot process.

systemd.unit

Sets the systemd target to be activated. For example, systemd.unit=graphical.target. Systemd also accepts the numerical runlevels as defined for SysV. To activate the runlevel 1, for example, it is only necessary to include the number 1 or the letter S (short for “single”) as a kernel parameter.

mem

Sets the amount of available RAM for the system. This parameter is useful for virtual machines so as to limit how much RAM will be available to each guest. Using mem=512M will limit to 512 megabytes the amount of available RAM to a particular guest system.

maxcpus

Limits the number of processors (or processor cores) visible to the system in symmetric multi-processor machines. It is also useful for virtual machines. A value of 0 turns off the support for multi-processor machines and has the same effect as the kernel parameter nosmp. The parameter maxcpus=2 will limit the number of processors available to the operating system to two.

quiet

Hides most boot messages.

vga

Selects a video mode. The parameter vga=ask will show a list of the available modes to choose from.

root

Sets the root partition, distinct from the one pre-configured in the bootloader. For example, root=/dev/sda3.

rootflags

Mount options for the root filesystem.

ro

Makes the initial mount of the root filesystem read-only.

rw

Allows writing in the root filesystem during initial mount.

IMATGE DE MENÍ DEL GRUB EN EDICIÓ

```
# cat /proc/cmdline
BOOT_IMAGE=(hd0,gpt2)/vmlinuz-5.11.22-100.fc32.x86_64
root=/dev/mapper/fedora-root ro rd.lvm.lv=fedora/root rd.lvm.lv=fedora/swap rhgb
quiet
```

```
# cat /etc/default/grub
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora/root rd.lvm.lv=fedora/swap rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
```

```
# ls -l /etc/grub.d/
total 92
-rwxr-xr-x. 1 root root  9346 Feb  9  2021 00_header
-rwxr-xr-x. 1 root root   236 Feb  9  2021 01_users
-rwxr-xr-x. 1 root root   835 Feb  9  2021 08_fallback_counting
-rwxr-xr-x. 1 root root 14962 Feb  9  2021 10_linux
-rwxr-xr-x. 1 root root   833 Feb  9  2021 10_reset_boot_success
-rwxr-xr-x. 1 root root   892 Feb  9  2021 12_menu_auto_hide
-rwxr-xr-x. 1 root root 12337 Feb  9  2021 20_linux_xen
-rwxr-xr-x. 1 root root  2562 Feb  9  2021 20_ppc_terminfo
-rwxr-xr-x. 1 root root 10673 Feb  9  2021 30_os-prober
-rwxr-xr-x. 1 root root  1415 Feb  9  2021 30_uefi-firmware
-rwxr-xr-x. 1 root root   218 Feb  9  2021 40_custom
-rwxr-xr-x. 1 root root   220 Feb  9  2021 41_custom
-rw-r--r--. 1 root root   483 Feb  9  2021 README
```

```
# Exemple d'entrada de menú generada automàticament en l'instal·lació

menuentry 'Fedora (4.18.19-100.fc27.x86_64) 27 (Workstation Edition)' --class
fedora --class gnu-linux --class gnu --class os --unrestricted
$menuentry_id_option
'gnulinux-4.18.19-100.fc27.x86_64-advanced-576c8897-60aa-4158-ac30-fdead5958a1b'
{
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_msdos
        insmod ext2
        set root='hd0,msdos7'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos7
--hint-efi=hd0,msdos7 --hint-baremetal=ahci0,msdos7 --hint='hd0,msdos7'
576c8897-60aa-4158-ac30-fdead5958a1b
        else
          search --no-floppy --fs-uuid --set=root
576c8897-60aa-4158-ac30-fdead5958a1b
        fi
        linux16 /boot/vmlinuz-4.18.19-100.fc27.x86_64
root=UUID=576c8897-60aa-4158-ac30-fdead5958a1b ro rhgb quiet LANG=en_US.UTF-8
        initrd16 /boot/initramfs-4.18.19-100.fc27.x86_64.img
}
```

```
# Exemple d'entrada 'despullada', només amb es elements imprescindibles

menuentry 'Fedora PROVA'   --unrestricted {
        insmod gzio
```

```
        insmod part_msdos
        insmod ext2
        set root='hd0,msdos5'
        linux16 /boot/vmlinuz-4.18.19-100.fc27.x86_64 root=/dev/sda5 ro rhgb
quiet LANG=en_US.UTF-8
        initrd16 /boot/initramfs-4.18.19-100.fc27.x86_64.img
}
```

```
# Windows menuentry example

menuentry 'Windows 10 (on /dev/sda1)' --class windows --class os
$menuentry_id_option 'osprober-chain-8AD2A802D2A7F11D' {
        insmod part_msdos
        insmod ntfs
        set root='hd0,msdos1'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1
--hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1 --hint='hd0,msdos1'
8AD2A802D2A7F11D
        else
          search --no-floppy --fs-uuid --set=root 8AD2A802D2A7F11D
        fi
        parttool ${root} hidden-
        drivemap -s (hd0) ${root}
        chainloader +1
}
```

Changing the kernel parameters is not usually required, but it can be useful to detect and solve operating system related problems. Kernel parameters must be added to the file /etc/default/grub in the line GRUB_CMDLINE_LINUX to make them persistent across reboots. A new configuration file for the bootloader must be generated every time /etc/default/grub changes, which is accomplished by the command grub-mkconfig -o /boot/grub/grub.cfg. Once the operating system is running, the kernel parameters used for loading the current session are available for reading in the file /proc/cmdline.

Kernel stage

The bootloader then loads the kernel from disk into memory and transfers control over. The system is now running Linux and may finish booting. The word *bootstrap* used in this context, which simply signifies the method for the computer BIOS to locate and load the first part of the operating system when the computer powers up.

The kernel must initialize any hardware drivers and get the root / filesystem mounted for the next stage. The kernel must boot the system in several phases. The kernel itself is like a regular executable statically linked (can only rely on information that was already compiled in the kernel at boot time and not on information in the shared libraries).

This kernel typically lives in the /boot partition which, offen a separate partition. Some BIOS and bootloader combinations that can only access the first 1024 cylinders of the disk. The /boot directory, alternatively the /boot/efi directory, is one of the most important directories in the File Hierarchy Standard (FHS). Most modern systems use the UEFI standard for specifying the exact location where the computer looks for the files it needs to start up.

The executable comprising the kernel will decompress itself as it is loaded, leading to the name of zImage. Over time, the kernel grew even more in size, and it became a problem to

load the whole kernel into a consecutive block of memory. The bzImage kernel format came into being that allowed the kernel to be loaded into multiple memory blocks, specifically higher memory (the b stands for big).

The kernel drivers necessary to continue the boot are bundled into the filesystem that is stored beside the kernel itself in /boot. The kernel boots, mounts the initrd (he initial RAM disk initrd), loads the drivers inside, and then remounts the real root filesystem using the new drivers. This allows drivers to be added to the kernel easily and for the kernel to mount the root filesystem over almost any storage hardware—even if it's via a network.

As the kernel is booting, it is able to initialize recognized hardware and make detected devices available to the rest of the operating system.

The kernel's final job is to start the first process on the system. The first process will normally have a process id *(PID) of 1*; on a System V system, the name of this process is *init*. On systemd systems starts the *systemd* process.

- /boot
- /boot/efi
- vmlinuz

```
$ ls /boot/
config-5.11.11-100.fc32.x86_64
config-5.11.16-100.fc32.x86_64
config-5.11.22-100.fc32.x86_64
efi
extlinux
grub2
initramfs-0-rescue-f8bff953853f49b4963ed3aa06461c80.img
initramfs-5.11.11-100.fc32.x86_64.img
initramfs-5.11.16-100.fc32.x86_64.img
initramfs-5.11.22-100.fc32.x86_64.img
loader
lost+found
System.map-5.11.11-100.fc32.x86_64
System.map-5.11.16-100.fc32.x86_64
System.map-5.11.22-100.fc32.x86_64
vmlinuz-0-rescue-f8bff953853f49b4963ed3aa06461c80
vmlinuz-5.11.11-100.fc32.x86_64
vmlinuz-5.11.16-100.fc32.x86_64
vmlinuz-5.11.22-100.fc32.x86_64
```

```
$ file /boot/vmlinuz-5.11.22-100.fc32.x86_64
/boot/vmlinuz-5.11.22-100.fc32.x86_64: Linux kernel x86 boot executable bzImage,
version 5.11.22-100.fc32.x86_64 (mockbuild@bkernel01.iad2.fedoraproject.org) #1
SMP Wed May 19 18:58:25, RO-rootFS, swap_dev 0xA, Normal VGA
```

```
$ cat /proc/cmdline
BOOT_IMAGE=(hd0,gpt2)/vmlinuz-5.11.22-100.fc32.x86_64
root=/dev/mapper/fedora-root ro rd.lvm.lv=fedora/root rd.lvm.lv=fedora/swap rhgb
quiet
```

Initramfs

The initramfs is the initial root filesystem that a Linux system typically has access to. Think of it as a temporary "starter" filesystem that provides the files and drivers that are necessary to start the real root filesystem and continue the system startup.

The initramfs filesystem is a workaround to a "chicken or the egg" problem. For example, the kernel may require a disk driver to access a disk, but that driver is located on the disk it wants to access. The two workarounds to this dilemma are 1) use initramfs or 2) compile the driver into the kernel. The last solution is not applicable, the kernel will grow and must be recompilated for every new driver.

After being unpacked, the kernel will launch the init script included in the root / filesystem of the initramfs RAM disk, which will then load the necessary drivers and execute the required userspace programs needed to load the real root filesystem for the system.

After the initramfs-included init script is done loading the appropriate drivers and loading the real root filesystem, it transfers control the /sbin/init or systemd binary program on the real root filesystem to continue the initialization of the system.

Once the kernel is fully booted and the real root filesystem is mounted, the memory allocated to the initial RAM disk can be freed.

The initramfs is built initially with the mkinitramfs utility, or updated by the update-initramfs utility, which can also create a new initramfs.

```
$ ls /boot/initramfs-*
/boot/initramfs-0-rescue-f8bff953853f49b4963ed3aa06461c80.img
/boot/initramfs-5.11.11-100.fc32.x86_64.img
/boot/initramfs-5.11.16-100.fc32.x86_64.img
/boot/initramfs-5.11.22-100.fc32.x86_64.img
```

```
$ file /boot/initramfs-5.11.22-100.fc32.x86_64.img
/boot/initramfs-5.11.22-100.fc32.x86_64.img: regular file, no read permission
```

```
lsinitrd
```

Init Stage

The init stage finishes booting the system. The first process of the operating system (also called init) is started. The init process has three important responsibilities:

1. Continue the booting process to get services running, login screens displaying, and consoles listening.
2. Start all other system processes.
3. Adopt any process that detaches from its parent.

There are distinct implementations of system initiators: init, systemd and Upstart.

SysV standard

A service manager based on the SysVinit standard controls which daemons and resources will be available by employing the concept of runlevels. Runlevels are numbered 0 to 6 and are designed by the distribution maintainers to fulfill specific purposes. The only runlevel definitions shared between all distributions are the runlevels 0, 1 and 6.

systemd

systemd is a modern system and services manager with a compatibility layer for the SysV commands and runlevels. systemd has a concurrent structure, employs sockets and D-Bus for service activation, on-demand daemon execution, process monitoring with cgroups, snapshot support, system session recovery, mount point control and a dependency-based service control. In recent years most major Linux distributions have gradually adopted systemd as their default system manager.

Upstart

Like systemd, Upstart is a substitute to init. The focus of Upstart is to speed up the boot process by parallelizing the loading process of system services. Upstart was used by Ubuntu based distributions in past releases, but today gave way to systemd.

The traditional init program uses the /etc/inittab file to determine what scripts will be executed to start the services that will be available on the system. Points to other scripts that do the work, usually stored in the /etc/init.d directory. Each service the system will run has a script that can start, stop, and restart a service, and init will cause each of these to be run in turn as needed to get the system to the desired state.

- init
- systemd
- /etc/init.d
- /etc/initttab

```
$ ps ax | head
    PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:03 /usr/lib/systemd/systemd --switched-root --system
--deserialize 30
      2 ?        S      0:00 [kthreadd]
      3 ?        I<     0:00 [rcu_gp]
      4 ?        I<     0:00 [rcu_par_gp]
      6 ?        I<     0:00 [kworker/0:0H-events_highpri]
      9 ?        I<     0:00 [mm_percpu_wq]
     10 ?        S      0:00 [rcu_tasks_kthre]
     11 ?        S      0:00 [rcu_tasks_rude_]
     12 ?        S      0:00 [rcu_tasks_trace]
```

```
$ pstree -slp | head
systemd(1)-+-ModemManager(803)-+-{ModemManager}(837)
           |                   `-{ModemManager}(867)
           |-NetworkManager(921)-+-{NetworkManager}(927)
```

```
|                     `-{NetworkManager}(928)
|-abrt-dbus(2438)-+-{abrt-dbus}(2449)
|                 `-{abrt-dbus}(2452)
|-abrt-dump-journ(881)
|-abrt-dump-journ(882)
|-abrt-dump-journ(883)
|-abrtd(845)-+-{abrtd}(871)
```

Kernel & log messages

The *dmesg* command can be executed after booting the system to see the messages generated by the kernel during boot time. The messages displayed can help an administrator troubleshoot the boot process.

Kernel messages are stored in a *kernel ring buffer* of limited size; therefore, the messages that are generated at boot time may be overwritten later as the buffer fills up.

Kernel messages and other system-related messages are typically stored in the /var/log/messages file. This file, which is considered the main log file, is alternatively named /var/log/syslog in some distributions. Although the /var/log/messages file is considered the main log file, there are other log files in the /var/log folder, which may need to be examined when trying to troubleshoot system service issues.

On a systemd-based system, the *journald* daemon is the logging mechanism, and it's configured by the /etc/systemd/journald.conf file. The format of a journald log file is binary. The main log file on a systemd-based device is the /var/log/journal file for persistent logging or the /run/log/journal file for RAM-based and non-persistent logging.

- dmesg [-H] [--human]
- dmesg --clear
- journalctl
- Journalctl [-b] [-k |–dmesg] [--list-boots]
- /etc/systemd/journald.conf

```
$ dmesg | head
[    0.000000] Linux version 5.11.22-100.fc32.x86_64
(mockbuild@bkernel01.iad2.fedoraproject.org) (gcc (GCC) 10.3.1 20210422 (Red Hat
10.3.1-1), GNU ld version 2.34-6.fc32) #1 SMP Wed May 19 18:58:25 UTC 2021
[    0.000000] Command line:
BOOT_IMAGE=(hd0,gpt2)/vmlinuz-5.11.22-100.fc32.x86_64
root=/dev/mapper/fedora-root ro rd.lvm.lv=fedora/root rd.lvm.lv=fedora/swap rhgb
quiet
[    0.000000] x86/split lock detection: warning about user-space split_locks
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point
registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x020: 'AVX-512 opmask'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x040: 'AVX-512 Hi256'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x080: 'AVX-512 ZMM_Hi256'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x200: 'Protection Keys User
registers'
```

```
$ dmesg | grep -i error | head
[    0.255782] ACPI BIOS Error (bug): Failure creating named object
[\_SB.PCI0.TXHC.RHUB.SS01._UPC], AE_ALREADY_EXISTS (20201113/dswload2-326)
[    0.255787] ACPI Error: AE_ALREADY_EXISTS, During name lookup/catalog
(20201113/psobject-220)
[    0.255791] ACPI BIOS Error (bug): Failure creating named object
[\_SB.PCI0.TXHC.RHUB.SS01._PLD], AE_ALREADY_EXISTS (20201113/dswload2-326)
[    0.255794] ACPI Error: AE_ALREADY_EXISTS, During name lookup/catalog
(20201113/psobject-220)
```

```
$ dmesg | grep -i failed | head
[   10.941645] thermal thermal_zone7: failed to read out thermal zone (-61)
[   11.577696] vboxdrv: module verification failed: signature and/or required key
missing - tainting kernel
[   70.531743] ucsi_acpi USBC000:00: PPM init failed (-110)
```

```
NAME
       dmesg - print or control the kernel ring buffer

SYNOPSIS
       dmesg [options]

       dmesg --clear

      -C, --clear
              Clear the ring buffer.

      -c, --read-clear
              Clear the ring buffer after first printing its contents.
```

```
# journalctl -b
-- Logs begin at Wed 2021-04-07 17:02:51 CEST, end at Thu 2021-09-30 19:59:33
CEST. --
Sep 30 16:46:16 localhost.localdomain kernel: Linux version
5.11.22-100.fc32.x86_64 (mockbuild>
Sep 30 16:46:16 localhost.localdomain kernel: Command line:
BOOT_IMAGE=(hd0,gpt2)/vmlinuz-5.11>
Sep 30 16:46:16 localhost.localdomain kernel: x86/split lock detection: warning
about user-spa>
Sep 30 16:46:16 localhost.localdomain kernel: x86/fpu: Supporting XSAVE feature
0x001: 'x87 fl>
Sep 30 16:46:16 localhost.localdomain kernel: x86/fpu: Supporting XSAVE feature
0x002: 'SSE re>
Sep 30 16:46:16 localhost.localdomain kernel: x86/fpu: Supporting XSAVE feature
0x004: 'AVX re>
Sep 30 16:46:16 localhost.localdomain kernel: x86/fpu: Supporting XSAVE feature
0x020: 'AVX-51>
Sep 30 16:46:16 localhost.localdomain kernel: x86/fpu: Supporting XSAVE feature
0x040: 'AVX-51>
Sep 30 16:46:16 localhost.localdomain kernel: x86/fpu: Supporting XSAVE feature
0x080: 'AVX-51>
Sep 30 16:46:16 localhost.localdomain kernel: x86/fpu: Supporting XSAVE feature
0x200: 'Protec
```

```
# journalctl -u gpm --no-pager | head
-- Logs begin at Wed 2021-04-07 17:02:51 CEST, end at Thu 2021-09-30 19:59:59
CEST. --
Apr 07 17:29:12 localhost.localdomain systemd[1]: Starting Console Mouse
manager...
Apr 07 17:29:12 localhost.localdomain /usr/sbin/gpm[763]: *** info
[daemon/startup.c(136)]:
Apr 07 17:29:12 localhost.localdomain /usr/sbin/gpm[763]: Started gpm
successfully. Entered daemon mode.
Apr 07 17:29:12 localhost.localdomain systemd[1]: Started Console Mouse manager.
Apr 07 23:50:18 localhost.localdomain systemd[1]: Stopping Console Mouse
manager...
```

```
Apr 07 23:50:18 localhost.localdomain systemd[1]: Stopped Console Mouse manager.
-- Reboot --
Apr 07 23:55:37 localhost.localdomain systemd[1]:
/usr/lib/systemd/system/gpm.service:11: PIDFile= references a path below legacy
directory /var/run/, updating /var/run/gpm.pid → /run/gpm.pid; please update the
unit file accordingly.
Apr 07 23:55:41 localhost.localdomain systemd[1]:
/usr/lib/systemd/system/gpm.service:11: PIDFile= references a path below legacy
directory /var/run/, updating /var/run/gpm.pid → /run/gpm.pid; please update the
unit file accordingly.
```

```
# journalctl --list-boots  --no-pager | head
-107 f86ed25fb7874571a998c04d817dd09d Wed 2021-04-07 17:02:51 CEST—Wed 2021-04-07 17:09:34 CEST
-106 584991c3264e476e9c89482b47fde69c Wed 2021-04-07 17:17:22 CEST—Wed 2021-04-07 17:28:49 CEST
-105 87d66684df914986ab9b638e1fd13e8f Wed 2021-04-07 17:29:08 CEST—Wed 2021-04-07 23:51:50 CEST
-104 08ec5a1530254fcdbcc81d1d69e821dc Wed 2021-04-07 23:52:12 CEST—Thu 2021-04-08 00:09:33 CEST
-103 63928b9544fd41eeaacc3547396c3839 Thu 2021-04-08 00:09:56 CEST—Thu 2021-04-08 00:13:35 CEST
-102 73a39ea8472443ef9b8910f51f9c9295 Thu 2021-04-08 00:14:19 CEST—Thu 2021-04-08 00:18:06 CEST
-101 fa8672f0ada44d7c9e4ac42fdab47b4d Thu 2021-04-08 00:20:19 CEST—Thu 2021-04-08 00:24:30 CEST
-100 8ef836dc61de43ea91710eb0dfd95b3e Thu 2021-04-08 00:25:05 CEST—Thu 2021-04-08 00:54:10 CEST
 -99 1c647d3f1d3148c08806cdd276ce6bf0 Thu 2021-04-08 15:13:13 CEST—Thu 2021-04-08 23:49:58 CEST
 -98 bc33a270242941f4b047e15faf79d8fa Fri 2021-04-09 07:44:27 CEST—Fri 2021-04-09 21:38:24 CEST
```

```
JOURNALCTL(1)                             journalctl
NAME
       journalctl - Query the systemd journal

SYNOPSIS
       journalctl [OPTIONS...] [MATCHES...]

DESCRIPTION
       journalctl may be used to query the contents of the systemd(1) journal as
       written by systemd-journald.service(8).


       -b [[ID][±offset]|all], --boot[=[ID][±offset]|all]
           Show messages from a specific boot. This will add a match for
           "_BOOT_ID=".

       -k, --dmesg
           Show only kernel messages. This implies -b and adds the match
           "_TRANSPORT=kernel".
```

Exercices

1. Which is the boot proccés order?
2. What does POST mind?
3. On a machine equipped with a BIOS firmware, where is the bootstrap binary located?
4. On a machine equipped with a UEFI firmware, where is the bootstrap binary located?
5. Which is the most common bootloader used on machines?
6. Enumerate three kernel options

7. Suppose the system is unable to boot due to a misinformed root filesystem location. How would the correct root filesystem, located at /dev/sda3, be given as a parameter to the kernel?

8. Which kernel option would suppress the boot hardware console messages?

9. What is an initramfs file?

10. Where are the initramfs and the kernel files? Can there be more than one?

11. Write the name of one kernel file and describe its parts.

12. Which is the first process of the operating system? Which is its PID? How can we show that information?

13. Enumerate three distinct implementations of system initiators

14. What dmesg option will automatically paginate its output, eliminating the need to use a pager command explicitly?

15. Where are the kernel messages stored?

16. How can be shown the messages of the previous (before the actual) boot?

17. How can the last boots of the system be listed?

---

18. Nosaltres treballarem amb el sistema BIOS però abans una pregunta respecte als dos sistemes BIOS/UEFI. Quin dels dos sistemes té tota la seva informació gravada a una memòria a la placa mare i quina té una part a una partició del disc dur?

19. Tenim por que ens introdueixen algun virus amb una memòria flash o cd o dvd. Volem per tant que l'ordinador vagi a buscar el Sistema Operatiu al disc dur, tot i que hi hagi una memòria flash o un cd, dvd amb sistema operatiu...

20. Ja hem configurat l'ordinador segons l'exercici anterior. Ara resulta que volem provar una live de ArchLinux que tenim gravada a un dispositiu extern. Com ho podríem fer sense entrar al setup de la BIOS? Podria ser el cas de que aquesta opció no existís? O a l'inrevés, si aquesta opció existeix, no és un forat de seguretat? I si fos així com s'hauria de resoldre aquest problema?

21. Ara volem fer una instal·lació via xarxa utilitzant el protocol PXE a un ordinador que en principi no permet aquesta opció. Que hauríem de fer?
    Se us acut alguna altra situació on pogués ser útil aquesta opció?

22. Per segons quines tasques realitzin alguns servidors ens podem trobar amb màquines molt velles. Suposem que tenim un servidor al qual no es connecta ningú directament. Només de manera remota.

23. Quin problema ens podem trobar i quina seria la solució? (si el pc no és molt vell no té sentit buscar a la BIOS)

24. Si el firmware fa servir l'estàndard UEFI en comptes de l'estàndard BIOS, pot fer servir l'opció _Secure Boot_. Quina és la funcionalitat que ens proporciona Secure Boot? Escull una opció:
    a. Si aquesta opció està activada es fa un xequeig antivirus d'inici.
    b. Si aquesta opció està activada només es poden instal·lar productes Microsoft.
    c. Si aquesta opció està activada només es poden instal·lar binaris amb signatura.
    d. Cap de les anteriors és correcta (troba quina és).

25. Volem ordenar el procés d'arrencada general d'un pc, des de que s'engega fins que obtenim la pantalla de benvinguda del corresponent sistema operatiu. Tot i que nosaltres simplificarem una mica el procés, tenim les fases desordenades. Hem d'ordenar-les:
    a. El MBR carrega el boot loader, per exemple el GRUB
    b. La BIOS carrega el MBR (master boot record), que normalment es troba al sector 0 (de 512 bytes) del disc dur (o diskette, o pen USB, CD ...) d'arrencada.
    c. El boot loader (a Linux típicament GRUB) carrega el sistema operatiu (a Linux típicament el kernel).
    d. La BIOS (basic input output service) realitza un POST (power on self test)

---

26. Volem fer un cop d'ull a l'aspecte del MBR (recordeu que són els primers 512 bytes del dispositiu). Un MBR té la següent estructura:
    Si no es veu la imatge podeu accedir a l'enllaç original de la viquipèdia:
    https://ca.wikipedia.org/wiki/Master_boot_record

Estructura d'un registre d'inici mestre (MBR)

Adreça			Descripció		Mida en bytes
Hex	Oct	Dec			
0000	0000	0	codi d'àrea		**440** (max. 446)
01B8	0670	440	signatura de disc (opcional)		**4**
01BC	0674	444	En general, nul; 0x0000		**2**
01BE	0676	446	**Taula de particions primàries** (Quatre entrades de 16 bytes, esquema de taula de particions IBM)		**64**
01FE	0776	510	55h	signatura MBR; 0xAA55	**2**
01FF	0777	511	AAh		
MBR, mida total: 446 + 64 + 2 =					**512**

a. Utilitzant l'ordre `dd` copiarem a un fitxer el MBR, és a dir els primers 512 bytes del disc dur que conté informació necessària per arrencar el nostre sistema operatiu. Li posarem de nom "mbr.bin".
(Opcions interessants per a `dd` *bs* i *count*)

```
[root@heaven ~]# dd if=/dev/sda of=mbr.bin bs=512 count=1
1+0 records in
1+0 records out
512 bytes copied, 0.0120986 s, 42.3 kB/s
```

b. Un cop hem creat aquest fitxer que conté el MBR, fem un cop d'ull amb l'ordre `file`.

```
[root@heaven ~]# file mbr.bin
mbr.bin: DOS/MBR boot sector
```

c. També podríem utilitzar ordres com `hexdump` o `od` per fer un cop d'ull al contingut del fitxer, o l'explorador de fitxers midnight commander `mc` per mirar el contingut d'aquest fitxer (opcions View/Hexadec/GoTo).
d. On es troba la informació que representa la teva taula de particions?
e. Quants bytes s'utilitzen del MBR per a representar aquesta taula?
f. Això ens dona alguna limitació de número de particions?
g. Quantes particions veus en aquest fitxer?

Solució:

- Segons l'estructura del MBR que se'm proporciona, tenim que els primers bytes estan reservats per a codi, i a partir del *byte 446* tinc la *taula de particions*.
- Cada partició disposa de 16 bytes i en total tinc 64 bytes, és a dir les 4 particions primàries que com a màxim tinc a una taula de particions de tipus DOS.

- És a dir, que la informació de les particions primàries que em mostren ordres com:

```
[root@heaven ~]# fdisk -l /dev/sda
Disk /dev/sda: 232.9 GiB, 250059350016 bytes, 488397168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x23d6168d

Device     Boot     Start       End   Sectors  Size Id Type
/dev/sda1             2048 429918207 429916160  205G  5 Extended
/dev/sda5  *          4096 209719295 209715200  100G 83 Linux
/dev/sda6        209721344 419436543 209715200  100G 83 Linux
/dev/sda7          419438592  429918207   10479616      5G  82  Linux  swap  /
Solaris
```

```
[root@heaven ~]# sfdisk -d /dev/sda
label: dos
label-id: 0x23d6168d
device: /dev/sda
unit: sectors

/dev/sda1 : start=        2048, size=   429916160, type=5
/dev/sda5 : start=        4096, size=   209715200, type=83, bootable
/dev/sda6 : start=   209721344, size=   209715200, type=83
/dev/sda7 : start=   419438592, size=    10479616, type=82
```

- estan emmagatzemades d'alguna manera en aquests 64 bytes.
- Mostrem el contingut amb l'ordre `od`:

```
od -A d -t x1 -v -j 446 mbr.bin
```

- on les opcions volen dir:
    - `-A d` que els comptadors de bytes que surten a l'esquerra es mostrin en base decimal
    - `-t x1` que representem la informació de sortida en hexadecimal(x) i grups d'1 byte
    - `-v` que no elimini duplicats (posant un '*' en comptes de mostrar la línia repetida)
    - `-j 446` que *se salti* els primers 446 bytes.
- Per interpretar els valors que es mostren utilitzarem aquest enllaç: (http://www.bydavy.com/2012/01/lets-decrypt-a-master-boot-record/) Concretamen a l'apartat **Let's decrypt a primary partition entry**
- Al nostre cas tenim:

```
[root@heaven ~]# od -A d -t x1 -v -j 446 mbr.bin
0000446 00 20 21 00 05 2b 22 89 00 08 00 00 00 00 a0 19
0000462 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000478 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000494 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000510 55 aa
0000512
```

- Per tant es veu que dels 4 grups de 16 bytes els 3 darrers són tot zero's, és a dir, només s'ha definit una partició primària (estesa):

```
27. 00 20 21 00 05 2b 22 89 00 08 00 00 00 00 a0 19
```

- Troba entre aquests números (utilitzant l'enllaç proporcionat):
  - quin és el codi i per tant el tipus de partició
  - quins són els números que fan referència a la quantitat de sectors que conté la partició (pots fer-ho al revés, amb fdisk veure quants sectors hi ha, passar-ho a hexadecimal i comparar-ho amb la cadena) 05 ---> partició estesa

```
[root@heaven ~]# fdisk -l /dev/sda
...
Device     Boot     Start       End   Sectors   Size Id Type
/dev/sda1            2048 429918207 429916160  205G  5 Extended
```

  - O sigui que hi ha 429916160 sectors, que en hexadecimal són:

```
[root@heaven ~]# echo 'obase=16;429916160' | bc
19A00000
```

  - Fixem-nos que ben col·locats (agrupats) al final de la primera línea trobem aquesta quantitat:

```
od -A d -t x4 -v -j 446 mbr.bin
00212000 89222b05 00000800 19a00000
```

Arrencada equip: GRUB

GRUB shell

1. Desem una còpia de seguretat del menú del *grub* (just in case).
Això és recomanable que ho fem sempre, al següent exercici ja veurem el perquè.

```
[root@heaven ~]# cp /boot/grub2/grub.cfg /boot/grub2/grub.cfg.bkp
```

o millor encara, canviar OLD per la data, per ex. 13122020

2. Comprovem que aquesta còpia realment funciona. Si el fitxer *grub.cfg* es
corromp una de les possibilitats és que ens enviï a una shell del grub. Anem a emular-ho.

* Reinicia el sistema
* Prem la tecla 'c' tal i com indica el grub: *... or 'c' for a command prompt*
* Ajudeu-vos del tabulador per fer l'AUTOCOMPLETE en les següents instruccions.
* Llista les particions (i disc dur) disponibles: ```ls```

```
grub> ls
(proc) (hd0) (hd0,msdos3) (hd0,msdos2)  (hd0,msdos1)...
```

* Llistem la partició Fedora:
```
grub> ls (hd0,msdos1)
```
Així com:
```
grub> ls (hd0,msdos1)/
```
* Carreguem el fitxer de configuració (tipus grub.cfg) amb l'ordre ```configfile```
gairebé ja la tenim aquí, només manca substituir els interrogants:
```
grub> configfile (hd0,msdos?)/boot/grub2/?
```
Solució:
```
grub> configfile (hd0,msdos1)/boot/grub2/grub.cfg.13122020
```
Si arrenca el grub és que el fitxer de backup és correcte.

3. Anem a simular una situació pitjor. No trobem cap fitxer de configuració grub que funcioni bé. Volem arrencar Fedora de manera manual. Fem abans un cop d'ull al fitxer *grub.cfg* fixant-nos concretament en 3 instruccions molt importants: ``` ... # l'ordre insmod carrega mòduls (drivers) al kernel set root ... linux /boot ... initrd /boot .... ``` Farem el següent, també des de la shell del GRUB, ajudant-nos amb el tabulador per autocompletar:

* Activem la partició a on es troba el sistema de fitxers /:
```
grub> set root=(hd0,1) # el 1er disc dur és hd0, el 2on disc dur hd1, però la 1a partició és 1 ...
```

Carreguem el kernel:
```
grub> linux /boot/vmlinuz-4..... root=/dev/vda1
```
* Carreguem el disc ram inicial *initrd*:
```
grub> initrd /boot/initramfs-4....
```
* Finalment arrenquem:
```
grub> boot
```

* Alerta: encara que sembli redundant si ens oblidem *root=/dev/vda1*
al carregar el kernel no es carregarà el sistema

PASSWORDS

En principi GRUB està configurat de manera que qualsevol usuari pot fer de tot amb grub:
seleccionar qualsevol entrada del menú, editar-la o accedir a una shell del GRUB.

En certs casos (per exemple la creació d'un quiosc) ens pot convenir que l'accès a aquestes funcionalitats
sigui autenticat mitjançant passwords.

Abans de passar a la part pràctica respondrem a les següents preguntes ajudant-nos de l'enllaç
[GRUB2 Security](https://www.gnu.org/software/grub/manual/grub.html#Security)

4. Per poder tenir autenticació, hem d'activar una variable d'entorn que contindrà l'usuari administrador
(o els usuaris administradors), com es diu aquesta variable?

superusers

5. En el moment que activem la variable d'entorn anterior, els usuaris que no siguin administradors
poden editar les entrades del grub o accedir a la shell del grub?

No

6. I els no administradors poden escollir les diferents entrades del grub?

Sí

7. Quina o quines seran les paraules clau que faran que qualsevol pugui arrencar una certa entrada del menú
o que només la poguin escollir uns quants usuaris autenticats?

* Si volem que qualsevol pugui arrencar una certa entrada del menú hem d'afegir a l'entrada adient:

```
--unrestricted
```

* Si volem que només uns quants usuaris autenticats puguin arrencar una certa entrada del menú hem d'afegir a l'entrada adient:

```
--users user1,user2
```

Fem ja la part pràctica. **IMPORTANT**: després d'editar el `grub.cfg.bkp` utilitzem `grub2-script-check`.

8. Suposem que la variable d'abans només conté un usuari de nom "root" i contrasenya "jupiter",

tot i que l'ideal és escriure als scripts que es troben a */etc/grub.d/* no ho farem i escriurem directament a /boot/grub2/grub.cfg (*)
concretament a la secció */etc/grub.d/01_users* just després de l'etiqueta BEGIN posarem

```
set superusers="root"
password root jupiter
password hisx1 hisx1
```

Fes els canvis adients perquè tothom pugui accedir al Debian però només el superusuari i hisx1 puguin accedir al Fedora

Pot ser d'ajuda l'enllaç: [GRUB2/Password](https://help.ubuntu.com/community/Grub2/Passwords)

El superusuari pot fer el que vulgui, per tant ja no hem d'afegir res més.
En canvi, si volem que l'usuari *hisx1* pugui escollir l'opció del matí, al *menuentry* que representa aquesta opció al *fitxer grub.cfg*
haurem d'afegir l'opció --users hisx1 tal i com s'explica l'[enllaç abans esmentat](https://www.gnu.org/software/grub/manual/grub.html#Security)

9. Fins ara hem treballat amb passwords no encriptats, per tant podrien ser fàcilment *crackejats*.
Afortunadament GRUB2 ens proporciona encriptació mitjançant l'ordre ```grub2-mkpasswd-pbkdf2```
Fes el mateix d'abans, amb els mateixos passwords (jupiter i hisx respectivament)
però canviant el que sigui necessari per utilitzar aquests passwords encriptats.

El password el possarem com:

```
password_pbkdf2 root grub.pbkdf2.sha512.10000.167A9F696972AEDF69A22FA9976F648230EB5184ACDE49D1D36793E7BD5B6FD753A1BAE1A3265F71C9E99DD413BF26941BB4A4B8607E1C17526486341E1FA769.BAE163E324BDCA2B68FDC66EFEF451C07E57A45753D068BF8198A47C0F72FFEB7F659C2A04BBEB17C5EC756BD10DA533124C8044218BF4231DEC10EA6C1EC351
```

sempre i quan aquest sigui el password xifrat generat a l'executar:

```
[angel@dheaven ~]# grub2-mkpasswd-pbkdf2
Enter password:
Reenter password:
PBKDF2 hash of your password is grub.pbkdf2.sha512.10000.167A9F696972AEDF69A22FA9976F648230EB5184ACDE49D1D36793E7BD5B6FD753A1BAE1A3265F71C9E99DD413BF26941BB4A4B8607E1C17526486341E1FA769.BAE163E324BDCA2B68FDC66EFEF451C07E57A45753D068BF8198A47
```

C0F72FFEB7F659C2A04BBEB17C5EC756BD10DA533124C8044218BF4231DEC10EA6C1EC351
```

#### Altres opcions

10. Com ho faries perquè arranquès l'entrada de FEDORA  en 5 segons?

    Al fitxer */boot/grub2/grub.cfg* escriuria
    ```
    set timeout=5
    ```
    ```