Booting & Shutting Down

Reference Material:
[1] Unified Extensible Firmware Interface Specification (Sections 2 & 3) https://uefi.org/sites/default/files/resources/UEFI%20Spec%202_6.pdf
[2] UEFI, Debian wiki: https://wiki.debian.org/UEFI
[3] UNIX and Linux System Administration Handbook, Section 2 (Booting and System Management Daemons).
[4] systemd, Archlinux wiki: https://wiki.archlinux.org/index.php/systemd

Index

- **Introduction**
- **Booting, Stage 1: Hardware**
- **Booting, Stage 2: Bootloader (GRUB)**
- **Booting, Stage 3: Kernel**
- **Booting, Stage 4: Systemd**
- **Shutting Down**

Introduction

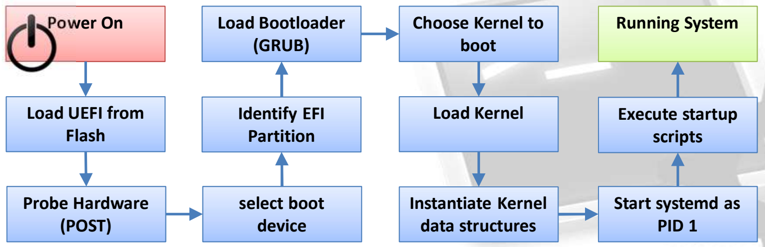
- Booting/Shutting Down are complex procedures, but system provides mechanisms to deal with them.
- … However, this is one of the potential troubles of administration.

- Goals of this Chapter:
  - Understand the basic operation of both procedures.
  - Being able to customize them.
  - Being able to solve generic problems related to Boot process.

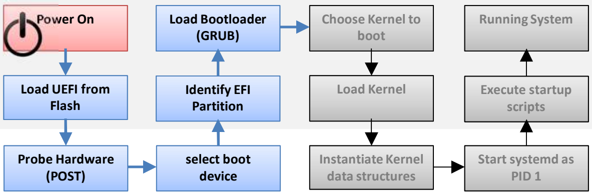
- Before: BIOS (and MBR) + SysV (see appendix), **Now: UEFI (and GPT) + Systemd**
  - Be careful with your online searchs…

Introduction

- The main target of Booting process is loading kernel (OS) in memory and starting its execution.
  - ¿Where is the kernel before booting?
  - ¿What's the content of memory before booting?
- It is a sequential process:



Index



Power On

Load UEFI from Flash

Probe Hardware (POST)

select boot device

Identify EFI Partition

Load Bootloader (GRUB)

Choose Kernel to boot

Load Kernel

Instantiate Kernel data structures

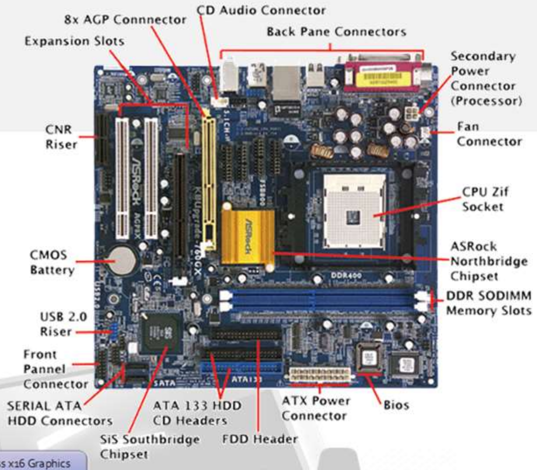
Start systemd as PID 1

Execute startup scripts

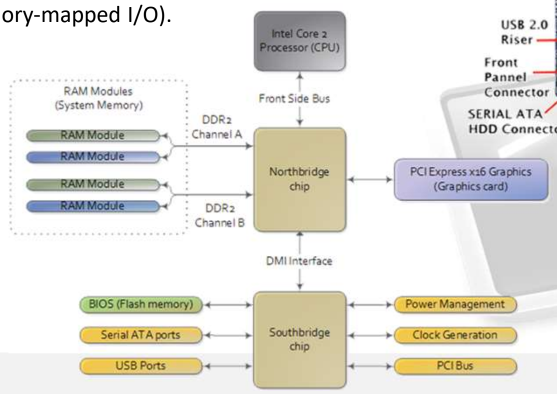
Running System

- Introduction
- **Booting, Stage 1: Hardware+UEFI**
- Booting, Stage 2: Bootloader (GRUB)
- Booting, Stage 3: Kernel
- Booting, Stage 4: Systemd
- Shutting Down

Stage 1: Hardware



- Power-On:
  - After pushing Power-On button, the **Reset Vector** indicates the CPU the direction of the first instruction to execute (FFFFFFF0h for x86). Such direction corresponds to an EPROM/Flash (motherboard) that stores the code corresponding to the Firmware (memory-mapped I/O).



Intel Core 2 Processor (CPU)

Front Side Bus

RAM Modules (System Memory)

RAM Module

RAM Module

RAM Module

RAM Module

DDR2 Channel A

DDR2 Channel B

Northbridge chip

PCI Express x16 Graphics (Graphics card)

DMI Interface

Southbridge chip

BIOS (Flash memory)

Serial ATA ports

USB Ports

Power Management

Clock Generation

PCI Bus

Initialization Firmware

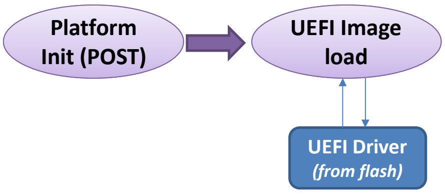
The old days…

- **BIOS (Basic Input Output System)**
  - firmware developed for the IBM PC/XT in the late 1970s
    - Adapted and expanded many times.
  - Support for MBR Partition Standard
  - 16-bit system, no disk driver available (no notion of files/directories in a disk).

Currently

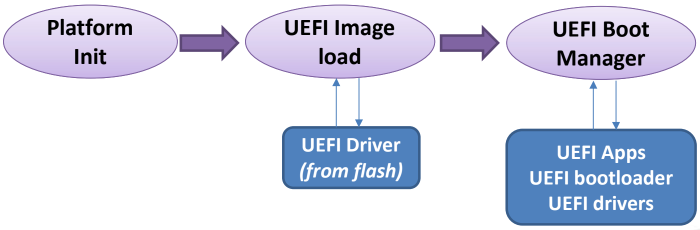
- **UEFI: Unified Extensible Firmware Interface**
  - Firmware standard derived from Intel's **EFI** (2000).
    - Intel's alternative for its new Itanium processor
  - Better disk/network support: full support for **GUID Partition Table (GPT)** (see slides 10-11) and IPv6.
  - Driver to "understand" FAT filesystem (format of the EFI System Partition or ESP).
  - No real need for a bootloader (**no GRUB required**).
  - Improved Security. UEFI **Secureboot** (run only signed apps) to prevent pre-OS malware (bootkit).
  - Closer integration between OS and pre-boot environment: Requires support from the OS (Linux, OSX, Windows10).

Stage 1: Hardware + UEFI (Boot Manager)



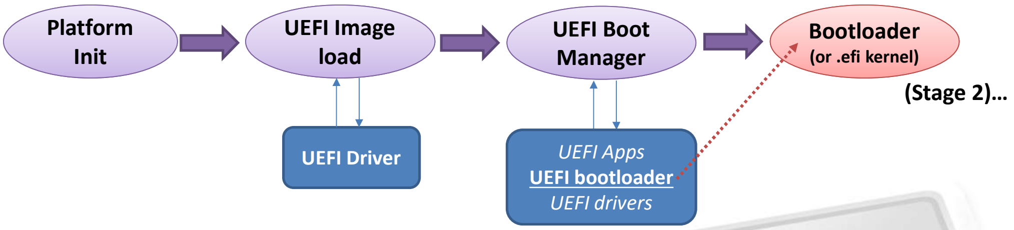
- **Power-on-self-test** (POST): examination, verification and start up of hardware devices (CPU, RAM, Controllers, etc.)
- Initialize the CPU with hardware-specific code (loaded directly from flash memory)
  - DRAM not yet avaliable, UEFI firmware uses CPU cache as RAM.
- Initialize **Main memory**, as well as hardware components required for the next phases
- Load the **device drivers** from the flash memory into main memory (now available), initialize all required hadware (disk (FAT filesystem), monitor, keyboard,…) and register/use protocols.
  - Protocol: provide text output to the console or access to a PCI device.

Stage 1: Hardware + UEFI (Boot Manager)



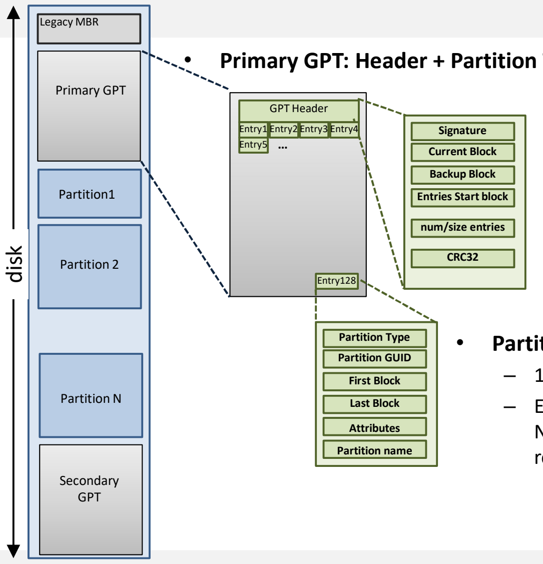
- **UEFI Boot Manager:** attempts to load UEFI applications (.efi files) in a predefined device order
  - Devices can be: storage, network.
  - Apps can be: **OS bootloader**, efi kernel, additional drivers (ext4), shell, GUI, etc.
  - **NVRAM** variables define this order
- Applications must reside on an UEFI-defined file system
  - **FAT** filesystem (format of the EFI System Partition or ESP).

Stage 1: Hardware + UEFI (Boot Manager)



- **How is Apps & Loader search process performed?**
- UEFI consults the GPT partition table to identify the ESP.
  - Remember the partition creation process in gdisk (Partition Type)
  - It then reads the target application (.efi file) from a file in the ESP and executes it.
  - Pathname to load: configuration parameter. By default (Debian): /efi/boot/bootx64.efi
- Each installed OS has its own directory in EFI partition.
- If no bootloader is used (EFI stub support enabled in kernel), all files required to load the OS (kernel, ramdisk, etc.) must be available in this partition.

Remember: GPT Disks & Partitions

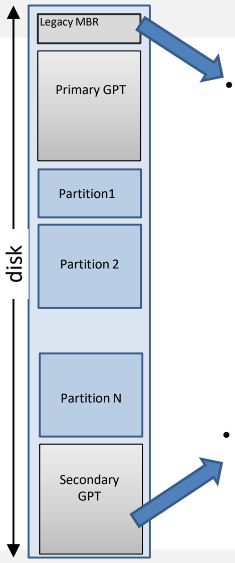


- **Primary GPT: Header + Partition Table**

- **GPT Header:**
  - Defines: usable blocks on the disk, UUID, CRC32 checksum, its own size and location, location of the secondary GPT header & table (last disk sectors).
  - Also defines the partition table (number & size of the partition entries). Minimum table: 128 entries, each 128 bytes long.

- **Partition Entries:**
  - 128 bytes for each entry block.
  - Each partition includes the following content: Name, Type, ID, First and last blocks, Attributes (e.g. read only).

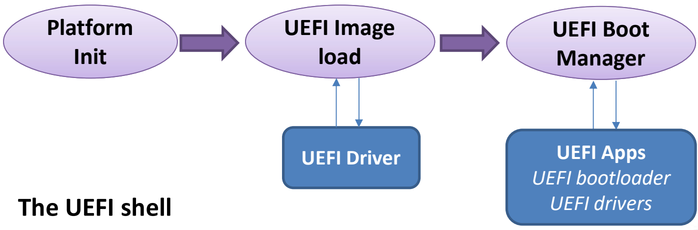
Remember: GPT Disks & Partitions



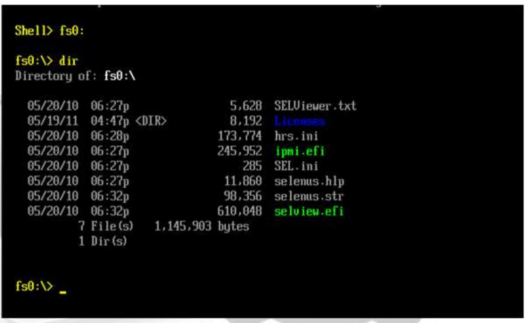
- **Protective/Legacy MBR**:
  - Compatibility MBR-GPT, 1st block reserved.
  - prevent MBR-based disk utilities from misrecognizing/overwriting GPT disks.
  - Single partition of special type (identifies a GPT disk). OS/apps which cannot read GPT recognize the disk & refuse to modify it.

- **Secondary GPT Header:**
  - Copy of the Primary GPT header, placed in last disk blocks.
  - If checksum of primary header fails, secondary is employed (not automatic).

Stage 1: Hardware + UEFI (Boot Manager)

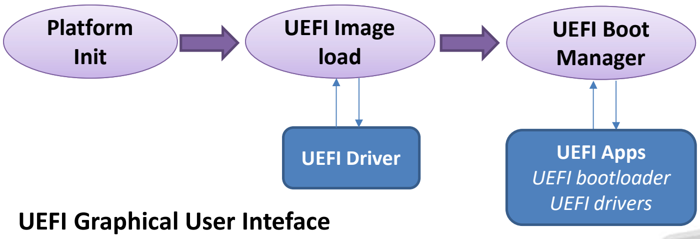


- **The UEFI shell**
  - Keep this in mind, EFI is a small OS built into the motherboard, could have its own shell. Usually, internal part of EFI, if not, can be downloaded as any other app.
  - Press “ESC” after Power On. Boot Manager -> EFI Internal Shell.
  - Many commands similar to Linux shell commands:
    - cp, ls, rm, mv, touch…
  - For a complete list of commands you can take a look to the UEFI shell specification:
    - http://www.uefi.org/sites/default/files/resources/UEFI_Shell_Spec_2_0.pdf

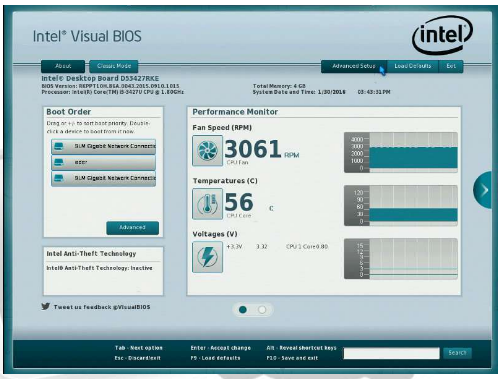


```
Shell> fs0:

fs0:\> dir
Directory of: fs0:\

  05/20/10  06:27p                  5,628  SELViewer.txt
  05/19/11  04:47p <DIR>            8,192  Licenses
  05/20/10  06:28p                173,774  hrs.ini
  05/20/10  06:27p                245,952  ipmi.efi
  05/20/10  06:27p                    285  SEL.ini
  05/20/10  06:27p                 11,860  selenus.hlp
  05/20/10  06:32p                 98,356  selenus.str
  05/20/10  06:32p                610,048  selview.efi
          7 File(s)   1,145,903 bytes
          1 Dir(s)


fs0:\> _
```

Stage 1: Hardware + UEFI (Boot Manager)



- **UEFI Graphical User Inteface**
  - Real world: graphical interface (the shell might still be present)

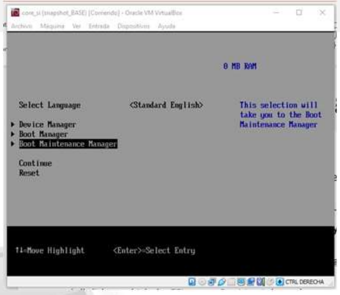


Intel® Desktop Board D53427RKE
BIOS Version: RKPPT10H.86A.0043.2015.0910.1015
Processor: Intel(R) Core(TM) i5-3427U CPU @ 1.80GHz
Total Memory: 4 GB
System Date and Time: 1/30/2016 03:43:31 PM

3061 RPM CPU Fan
56 C CPU Core
+3.3V 3.32 CPU 1 Core 0.80

Intel® Anti-Theft Technology: Inactive

UEFI Boot Manager Configuration



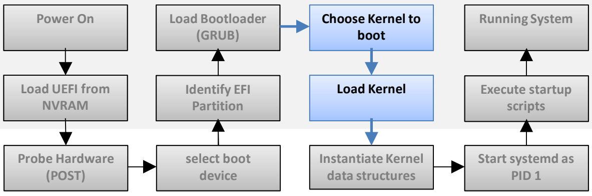
```
                                          0 MB RAM

Select Language          <Standard English>      This selection will
                                                 take you to the Boot
▸ Device Manager                                 Maintenance Manager
▸ Boot Manager
▸ Boot Maintenance Manager

  Continue
  Reset

↑↓=Move Highlight      <Enter>=Select Entry
```

- How can we modify Boot Manager configuration?
  - For example, to modify boot order.

1. Through the **UEFI graphical interface**

2. Through the **UEFI shell** app (**bcfg** command)
   - employed to modify NVRAM variables.
   - bcfg boot dump –v (list current boot options)
   - bcfg boot add/rm/mv …
3. Once the boot process if finished (kernel booted), through the **efibootmgr** command
   - # efibootmgr –v (list boot entries)
   - # efibootmgr -c -d /dev/sda -p 2 -L ”prueba" -l "\EFI\prueba\grubx64.efi”
     - (-c) to create a new entry.
     - (-d) disk on which the EFI system Partition is hosted
     - (-p) partition number on which the EFI system Partition is hosted.
     - (-L) Label to use as boot entry.
     - (-l) path of the EFI image to boot.

Index



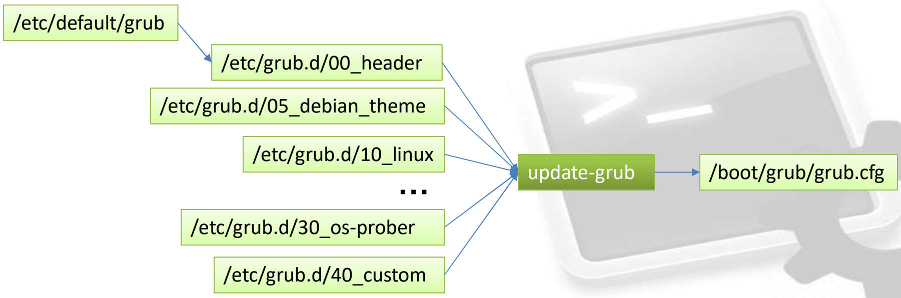
- Introduction
- Booting, Stage 1: Hardware+UEFI
- **Booting, Stage 2: Bootloader (GRUB)**
- Booting, Stage 3: Kernel
- Booting, Stage 4: Systemd
- Shutting Down

Stage 2: Bootloader (GRUB)

- ”Piece” of software between UEFI and OS Kernel (during boot process)
  - Identify and load an appropriate OS kernel.
  - Provide configuration arguments for the kernel (-s for singl—user mode).
- Task: Load Kernel into main memory to continue the boot process.
- **GRUB: GR**and **U**nified **B**ootloader
  - Developed by GNU project, default on most Linux distributions.
  - Two development branches: GRUB / GRUB2
    - Currently the most employed is GRUB2 (the one described in these slides).
  - Can work with file systems (ext2, ext3, ext4,…), directly accessing partitions.
  - Which is its current utility? multi-boot systems.
  - https://www.gnu.org/software/grub/manual/grub/grub.html

Stage 2: Bootloader (GRUB)

- **GRUB Configuration**:
  - Parameters such as: kernel to boot, boot options, boot password, etc.
  - GRUB reads its configuration params from **/boot/grub/grub.cfg**
  - grub.cfg should not be edited manually, created through **update-grub** command.
  - update-grub generates grub.cfg according to /etc/default/grub and /etc/grub.d/



Stage 2: Bootloader (GRUB)

- **GRUB Configuration**:

- The **/etc/default/grub** file:
  - GRUB_DEFAULT: default menu entry by menu position (starting at 0).
  - GRUB_CMDLINE_LINUX: params to add to the end of the “linux” command line.
  - GRUB_TIMEOUT: seconds to display menu before autoboot.

- The **/etc/grub.d/** directory (read during execution of update-grub):
  - 00_header: sets environmental variables (system file locations, video settings, etc.) and import preferences stored in /etc/default/grub.
  - 05_debian_theme: GRUB2 appearance (colours, background image, etc.)
  - 10_linux: identify kernels on the root device for the OS in use and creates entries (including associated recovery mode).
  - 30_os-prober: search for other OS
  - 40_custom: template for adding custom menu entries.

Stage 2: Bootloader (GRUB)

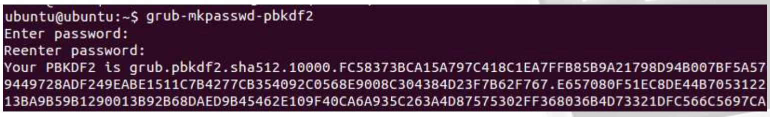
- **GRUB Command Line**:
  - GRUB supports a command-line interface for editing config file entries on the fly at boot time.
  - You can also boot non-listed OS, display system info and perform filesystem testing.
  - Some useful commands:
    - **boot**: boot the OS that was last loaded
    - **linux <path-to-kernel> [options]**: load a linux kernel
    - **reboot**: reboot the system
    - complete list: https://www.gnu.org/software/grub/manual/grub/html_node/Command_002dline-and-menu-entry-commands.html
  - Modifying kernel boot options at boot time:
    - root=<path>: root filesystem.
    - ro/rw: Mount root device read-only or read-write on boot.
    - quiet: Dissable most log messages.
    - init=<command>: Run specified binary instead of /sbin/init as init process.
    - S: boot kernel in single mode.
    - Complete list: https://www.kernel.org/doc/html/v4.14/admin-guide/kernel-parameters.html

Stage 2: Bootloader (GRUB)

- Having physical access to a system, stages 1 & 2 are a serious weakness.
  - Modifying boot options we could easily obtain superuser privileges.
- Increase protection a little bit: GRUB2 with password (still weak)
  - You can design a superuser with password to access GRUB shell.
  - Edit /etc/grub.d/00_header and at the end of the file add (remember to perform update-grub after that):

```
cat << EOF
set superusers="alumno"
password alumno 1234
export superusers
EOF
```

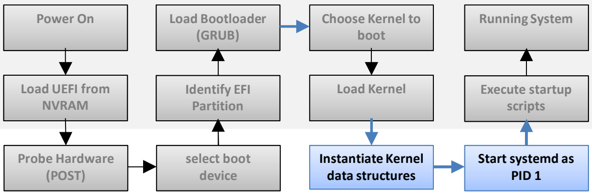
  - Notice that password is in plain text. to encrypt it: grub-mkpasswd-pbkdf2



```
ubuntu@ubuntu:~$ grub-mkpasswd-pbkdf2
Enter password:
Reenter password:
Your PBKDF2 is grub.pbkdf2.sha512.10000.FC58373BCA15A797C418C1EA7FFB85B9A21798D94B007BF5A57
9449728ADF249EABE1511C7B4277CB354092C0568E9008C304384D23F7B62F767.E657080F51EC8DE44B7053122
13BA9B59B1290013B92B68DAED9B45462E109F40CA6A935C263A4D87575302FF368036B4D73321DFC566C5697CA
```

```
cat << EOF
set superusers="alumno"
password_pbkdf2 alumno grub.pbkdf2.sha512.10000.FC58…
export superusers
EOF
```

Index



- Introduction
- Booting, Stage 1: Hardware+UEFI
- Booting, Stage 2: Bootloader (GRUB)
- **Booting, Stage 3: Kernel**
- Booting, Stage 4: Systemd
- Shutting Down

Stage 3: Loading the Kernel

- The bootloader has loaded in memory kernel & ramdisk files
  - vmlinuz-4.9.0-4-amd64
  - initrd.img-4.9-0-4-amd64
- Once finalized Stage 2, kernel execution starts:
  - The Kernel **un-compresses** itself.
  - Detects memory map, the **CPU** and its features supported.
  - Starts the **display** (console) to show information through the screen.
  - Checks the **PCI bus**, creating a table with the peripheral detected.
  - Initializes the system in charge of **virtual memory** management, including swapper.
  - Initializes the **drivers** for the peripherals detected (Monolithic or modular).
  - Mount **file system** root (“/”).
  - Calls the **systemd** process (Stage 4): PID 1, father of the rest of processes.

Stage 3: The init Ramdisk

- **RAM Disk**: fraction of main memory (RAM) formatted with a file system (tmpfs/ramfs)
  - Fast storage, but volatile!!
  - It is easy to create your own ramdisk…

```
# mkdir /tmp/ramdisk
# mount -t tmpfs -o size=1G myramdisk /tmp/ramdisk
```

  - That’s it (everything will be lost when unmounting)
  - You can test write/read speed with dd command.
- **Initial RAM Disk** (initrd): transient root FS mounted prior to real root FS (/)

Another step,
Why the hell?

Stage 3: The init Ramdisk

- **RAM Disk**: fraction of main memory (RAM) formatted with a file system (tmpfs/ramfs)
  - Fast storage, but volatile!!
  - It is easy to create your own ramdisk…

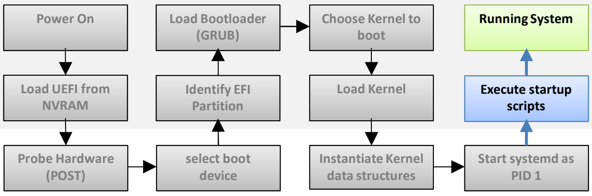
```
# mkdir /tmp/ramdisk
# mount -t tmpfs -o size=1G myramdisk /tmp/ramdisk
```

  - That’s it (everything will be lost when unmounting)
  - You can test write/read speed with dd command.
- **Initial RAM Disk** (initrd): transient root FS mounted prior to real root FS (/)
  - **Main target: load the modules (HD drive) required to make the real FS available.**
  - /sbin/init is executed
    - it mounts the “real” root file system
    - exec the /sbin/init on the “real” file system
    - after that, initrd file system is removed.The Kernel **un-compresses** itself.

```
# cp /boot/initrd.img-XXX /tmp/initrd.img.gz
# gunzip initrd.img.gz
# cpio -i -make-directories < initrd.img
```

  - Format: compressed cpio image

Index



- Introduction
- Booting, Stage 1: Hardware+UEFI
- Booting, Stage 2: Bootloader (GRUB)
- Booting, Stage 3: Kernel
- **Booting, Stage 4: Systemd**
- Shutting Down

Stage 4: Systemd

- Once the kernel is loaded, run the **system management daemon (/sbin/init)**.
  - Main goal: make sure the system runs the right **group of services (mode)** and daemons at any given time.
  - It provides a system and service manager running as PID 1.
  - In charge of startup tasks such as: setting computer name and time zone, check disk status, mount filesystems, configure network interfaces,…
  - Alternative implementations: SysV-init, BSD-init, **systemd**.
- Which are systemd main features?
  - aggressive parallelization capabilities (faster startup).
  - Socket and D-Bus activation for starting services, on-demand starting of daemons.
  - keeps track of processes using Linux control groups.
  - maintains mount and automount points.
  - logging daemon.
  - utilities to control basic system configuration.
    - hostname, date, locale, list of logged-in users, system accounts, runtime directories and settings,

SysV-init is a booting management daemon. Systemd is a **booting and run-time** management daemon.

Stage 4: Systemd

- Systemd **Unit**: encoded information/configuration for any resource/service managed by systemd.
  - This is the primary object that the systemd tools know how to deal with.
- Available Systemd unit types:
  - **.service**: A system service.
  - **.target**: A group of systemd units (for startup).
  - **.automount**: A file system automount point.
  - **.device**: A device file recognized by the kernel.
  - **.mount**: A file system mount point. (alternative to fstab entry, as seen in
  - **.socket**: An inter-process communication socket.
  - **.swap**: A swap device or a swap file.
  - **.timer**: A systemd timer.
  - …
- The behavior of each unit is defined and configured by its **unit file**.

Involved in boot process.

Stage 4: Systemd

- **Unit file syntax:** (rsync daemon).
  - Internal structure organized with sections, denoted as: [section_name].
  - At each section, behavior is defined through key-value directives (one per line).

```
[Unit]
Description=fast remote file copy program daemon
ConditionPathExists=/etc/rsyncd.conf

[Service]
ExecStart=/usr/bin/rsync --daemon --no-detach

[Install]
WantedBy=multi-user.target
```

- Location of the Unit files:
  - /lib/systemd/system/, /etc/systemd/system/

Stage 4: Systemd

***Read man pages for a complete directive list**

- **[Unit]** section directives*: define unit metadata and the relation to other units.
  - **Description/Documentation**: describe name, basic functionality and location for documentation about the unit.
  - **Requires**: lists units upon which current unit depends. If the current unit is activated, the units listed here must be activated (else this unit fails).
    - Requires directive can be replaced by creating a unit-file.requires dir in /etc/systemd/system and adding symlinks there to other unit files.
  - **Wants**: Similar to Requires but less strict (no activation required for the units listed)
    - Wants directive can be replaced by creating a unit-file.wants dir in /etc/systemd/system and adding symlinks there to other unit files.
  - **Requisite, Binds To, PartOf, Conflicts***
  - **Before**: The units listed will not be started until the current unit is marked as started.
  - **After**: The units listed will be started before starting the current one.

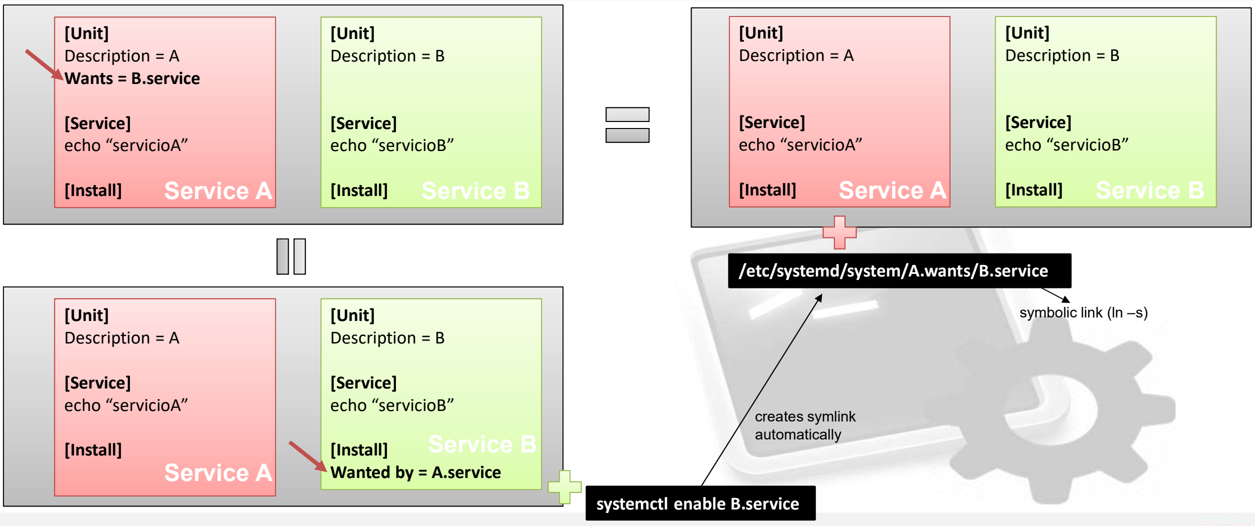
- **[Install]** section directives*: define behavior of a unit if it is enabled or disabled (systemctl enable).
  - **WantedBy**: specify a dependency in a similar way to “Wants”. When a unit with this directive is enabled, the directory /etc/systemd/system/[unit].wants is created, with a symbolic link inside to create the dependency.
  - **RequiredBy**

Stage 4: Systemd

***Read man pages for a complete directive list**

- **[Service]** section directives*: configuration of services.
  - **Type**: categorize service by their process and daemonizing behavior
    - simple: default type.
    - forking: the service forks a child process.
    - oneshot: systemd should wait for the process to exit before continuing on with other units.
    - dbus: the unit will take a name on the D-Bus bus.
    - notify: service will issue a notification when it has finished starting up.
    - idle: service will not run until all jobs are dispatched.
  - **ExecStart**: path and arguments of the command to be executed to start the service.
  - **ExecStop**: command needed to stop the service.
  - **Restart**: conditions to attempt automatic restart (always, on-success, on failure, …)
  - **TimeoutSec**: when stopping, time to wait before marking as failed (or forcing kill).
  - …

Unit File Relations (Definition)



[Unit]
Description = A
Wants = B.service

[Service]
echo “servicioA”

[Install] Service A

[Unit]
Description = B

[Service]
echo “servicioB”

[Install] Service B

[Unit]
Description = A

[Service]
echo “servicioA”

[Install] Service A

[Unit]
Description = B

[Service]
echo “servicioB”

[Install]
Wanted by = A.service
Service B

systemctl enable B.service

creates symlink automatically

[Unit]
Description = A

[Service]
echo “servicioA”

[Install] Service A

[Unit]
Description = B

[Service]
echo “servicioB”

[Install] Service B

/etc/systemd/system/A.wants/B.service

symbolic link (ln –s)

Unit File Relations (Strength)

```
[Unit]
Description = A
Wants = B.service

[Service]
echo “servicioA”

[Install]
```

Service A

```
[Unit]
Description = B


[Service]
echo “servicioB”

[Install]
```

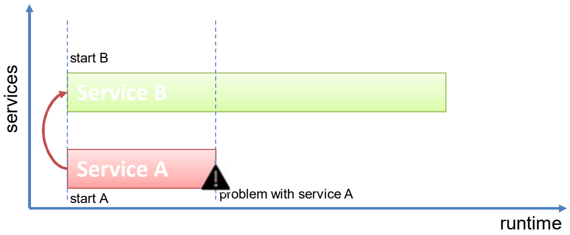
Service B

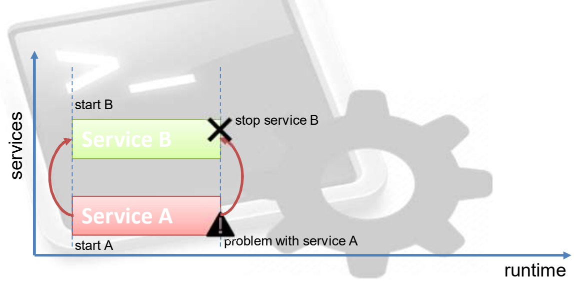
```
[Unit]
Description = A
Requires = B.service

[Service]
echo “servicioA”

[Install]
```

Service A

```
[Unit]
Description = B


[Service]
echo “servicioB”

[Install]
```

Service B





Unit File Relations (Timing)

```
[Unit]
Description = A
Wants = B.service

[Service]
echo “servicioA”

[Install]
```

Service A

```
[Unit]
Description = B


[Service]
echo “servicioB”

[Install]
```

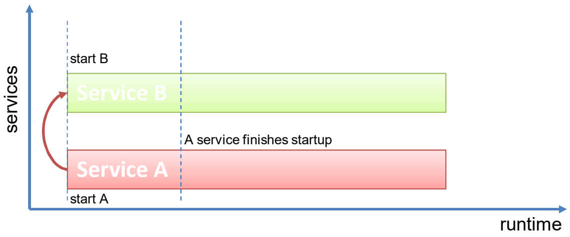
Service B

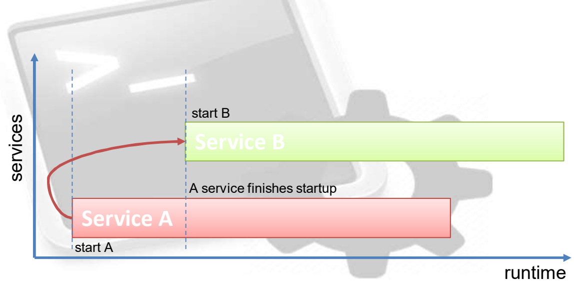
```
[Unit]
Description = A
Wants = B.service
Before = B.Service
[Service]
echo “servicioA”

[Install]
```

Service A

```
[Unit]
Description = B


[Service]
echo “servicioB”

[Install]
```

Service B





Stage 4: Systemd

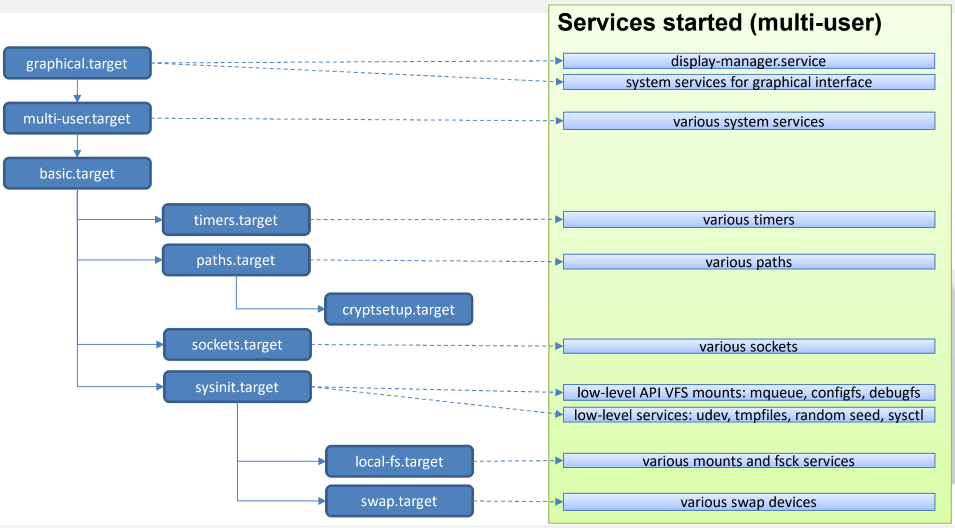
- Systemd **boot process** :
  - boot & service management handled through **Targets:** special units employed to group boot units and start up synchronization processes.
  - Targets are equivalent to SysV runlevels (see appendix for details), defining different operation modes.

runlevel	target	Description
0	poweroff.target	System halt
emergency	emergency.target	Bare-bones shell for system recovery
1,s,single	rescue.target	single-user mode
2-4	multi-user.target	Multi-user modes
5	graphical.target	Multi-user + GUI
6	reboot.target	system reboot

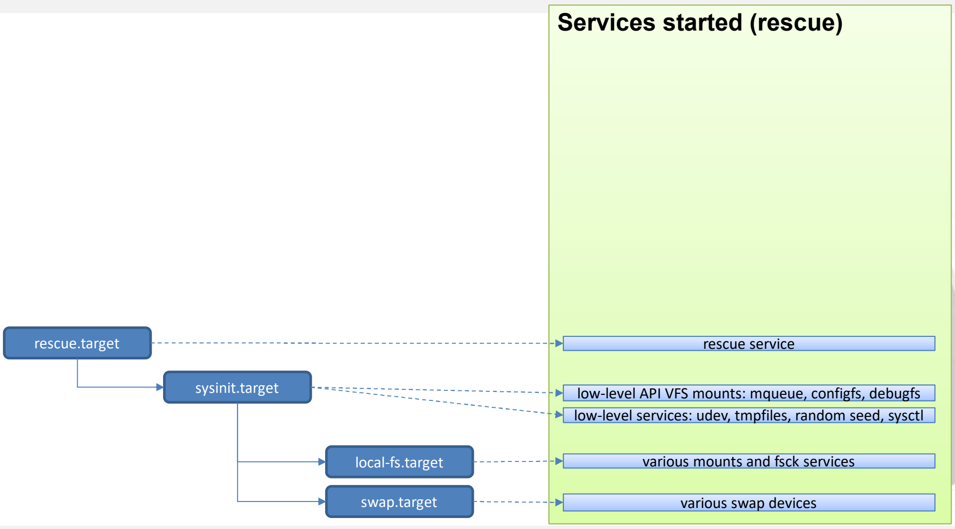
  - **Target relations to other units (targets or services) define the group of services started for each operation mode**

```
[Unit]
Description=foo boot target
Requires=multi-user.target
Wants=foobar.service
After=rescue.service
```

Stage 4: Systemd



Stage 4: Systemd



Services started (rescue)

rescue.target

sysinit.target

local-fs.target

swap.target

rescue service

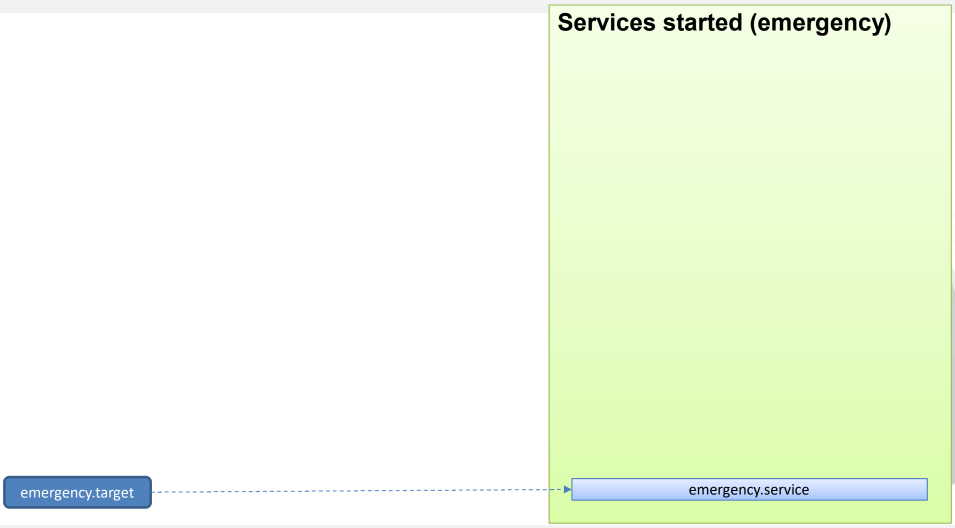
low-level API VFS mounts: mqueue, configfs, debugfs

low-level services: udev, tmpfiles, random seed, sysctl

various mounts and fsck services

various swap devices

Stage 4: Systemd



Services started (emergency)

emergency.target

emergency.service

Stage 4: Systemd

- Managing systemd through **systemctl** command
- Service administration:
  - # systemctl list-unit-files [--type=service]: list units available (of each type).
  - #systemctl list-units: list units active/waiting/failed.
  - # systemctl status –l <unit>: display service detailed information.
  - # systemctl enable/disable: activate/deactivate unit at boot.
  - # systemctl start/stop/restart: activate/deactivate/restart unit immediately.
- Operation mode administration:
  - # systemctl isolate target: change operation mode to target.
  - # systemctl get-default: see the target the system boots into.
  - # systemctl set-default target: change default target.
- System & Boot performance statistics:
  - # systemd-analyze.

Stage 4: Systemd

- Creating your own local services (do it in /etc/systemd/system)
  1. Create your own unit file (adapt one from /lib/systemd/system). Use the man pages to see a complete list of [service] and [unit] directives.
  2. Manage your dependencies,
     - Explicit: Wants, Requires directives inside unit file
     - Create the directories unit-file.wants or unit-file.requires in /etc/systemd/system and add symlinks there to other unit files.
     - Use the command systemctl: #systemctl add-wants multi-user.target custom.service
  3. Activate the dependencies: #systemctl enable custom.service

Index

- Introduction
- Booting, Stage 1: Hardware
- Booting, Stage 2: Bootloader
  - LILO
  - GRUB
- Booting, Stage 1+2 (UEFI)
- Booting, Stage 3: Kernel
- Booting, Stage 4: INIT
- **Shutting Down**

Shutting Down

- Never shut down directly (reset!).
  - If this rule is not respected, there is a high probability of loosing or corrupting system files (with a bit of bad luck, fully broken system)
  - Intermediate Buffers for disk read/write. Synchronization.
- Never shut down without warning all system users
  - Periodically programmed shut-downs.
- Steps for a correct shut down:
  - Warn all the users previously.
  - Stop all the services associated to the target
  - Send the specific signal to all the processes to end their execution.
  - Users and processes still present, killed.
  - Subsystems shut down sequentially.
  - File System unmounted (synchronizes pending changes with disk)

Shutting Down

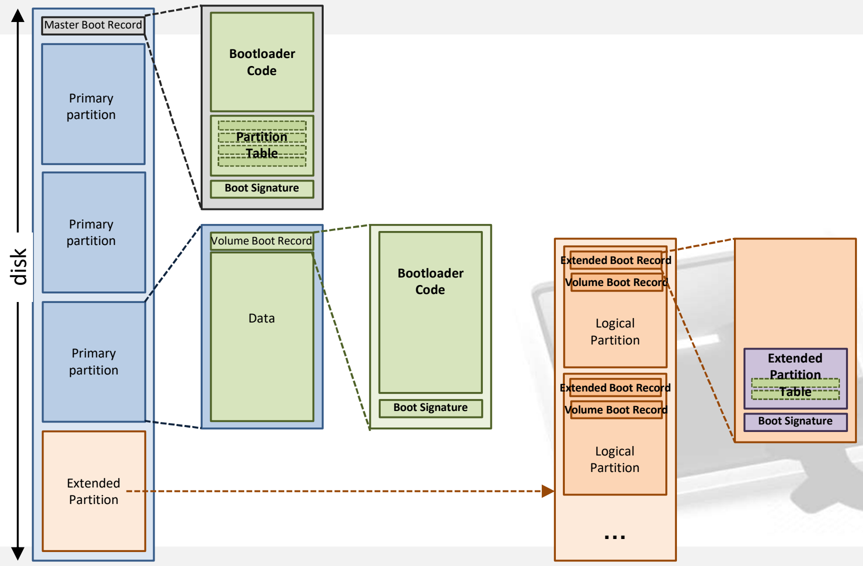
- Command **shutdown**:
  - Format: /sbin/shutdown -<options> time message
    - Option -r: reboot instead power off
    - Option -h: stop the system(with ACPI).
  - Message: message sent to all users.
  - time: delay to begin the shutdown (mandatory)
    - Format: hh:mm
    - Supports now+,minutes
- /etc/shutdown.allow or inittab
  - Avoid Ctrl+Alt+Del
- Other commands: /sbin/halt, /sbin/poweroff

APPENDIX

BIOS Firmware

- **BIOS** (Basic Input/Output System):
  - 1975: First appearance in the Operating System CP/M.
  - It runs in real address mode (16 bit): 1MB of addressable memory.
  - 1990: appears “BIOS setup utility”: allows the user to define some configuration options (boot priority).
  - ROM customized for a particular HW. Provides a small library with I/O functions to work with peripherals (keyboard, screen). Very slow (protected to real mode).
  - Emerging applications require more and more BIOS support: Security, Temperature/Power metrics (ACPI), Virtualization extensions, Turbo-Boost … (Hard to put all that in 1MB).
  - 2002: Intel develops an alternative firmware: EFI (/UEFI).

MBR Disks & Partitions



disk

Master Boot Record

Primary partition

Primary partition

Primary partition

Extended Partition

Bootloader Code

Partition Table

Boot Signature

Volume Boot Record

Data

Bootloader Code

Boot Signature

Extended Boot Record

Volume Boot Record

Logical Partition

Extended Boot Record

Volume Boot Record

Logical Partition

…

Extended Partition Table

Boot Signature

MBR Disks & Partitions

- **Master Boot Record (MBR)**:
  - First block of the Disk, 512 Bytes.
  - **Partition Table**: information about four primary partitions: begin and end blocks, size, etc. (64 bytes)
  - **Boot Signature**: Numerical value indicating the presence of valid bootloader code in the code field (0x55AA) (2 bytes).
- **Volume Boot Record (VBR)**:
  - First block of each primary partition.
  - Could contain bootloader code (indicated by Boot Signature).
- **Extended Partition**:
  - Partition that can be sub-divided into multiple **logical partitions**.
  - **Extended Boot Record (EBR)**: First block of each logical partition. It only contains a partition table with two fields. **Extended partition table** forms a linked list with all logical partitions.

MBR Disks & Partitions

- **Linux Naming Convention:**
  - Remember: I/O devices are treated as files. Under directory /dev we find all system disks.
  - generic PC: 2 IDE controllers, each can have two devices (master/slave).
    - /dev/**hda**: first device (master) of the first IDE controller.
    - /dev/**hdb**: second device (slave) of the first IDE controller.
    - /dev/**hdc**: first device of the second controller.
    - /dev/**hdd**: second device of the second controller.
  - In a disk, each **primary partition** is identified with a number from 1 to 4.
    - /dev/**hda1**: first primary partition of the hda disk.
  - **Lógical partitions** start from 5.
    - /dev/**hda5**: first logical partition of hda disk.
  - In **SCSI devices** same naming convention, changing “sd” by “hd”
    - /dev/**sda1**

Bootloader in MBR

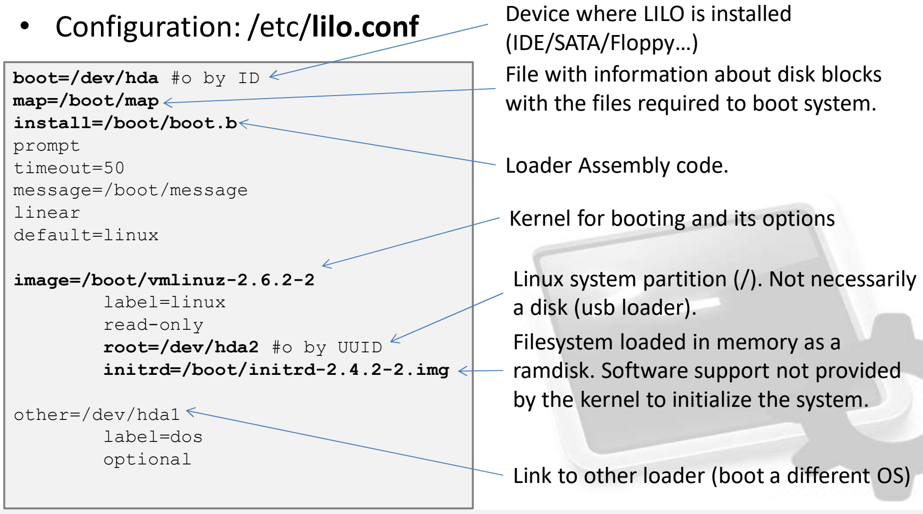
- Hardware requires an OS in charge of providing all the functionality in a computer.
- Target: loading in memory OS kernel and start running it. Loader with different locations: USB, CD, Disk …
- **Stage 2.1**:
  - Located in **MBR**: 512 first bytes (block 0) of the **active device**.
  - Loaded in memory by BIOS (Stage 1).
  - Triggers, when executed, the load and execution of Stage 2.2.
- **Stage 2.2**:
  - Located in the **active partition**, where the kernel is placed.
  - Loads the kernel in memory and transfers control to it (data initialization, drivers, check CPU, etc.)
  - After this process, the **init** process is executed (Stage 3)

LILO

- **LI**nux **Lo**ader:
  - Two stage Bootloader.
  - Does not “understand” about operating system, neither about file system. Only works with physical locations.
  - Obsolete (but easy to follow for academic purpose)
- Steps:
  - Master boot loads LILO from the first active partition and runs it.
    - LILO can be in the MBR or in the Boot Block of a primary partition. In the second case, MBR contains the necessary code to load LILO from another block.
  - LILO requests the user the kind of boot wanted (partition, kernel, mode). Through a prompt.
  - LILO loads the kernel and a ramdisk.
  - The kernel starts running once it is loaded into memory.

LILO

- Configuration: /etc/**lilo.conf**



```
boot=/dev/hda #o by ID
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=linux

image=/boot/vmlinuz-2.6.2-2
        label=linux
        read-only
        root=/dev/hda2 #o by UUID
        initrd=/boot/initrd-2.4.2-2.img

other=/dev/hda1
        label=dos
        optional
```

Device where LILO is installed (IDE/SATA/Floppy…)

File with information about disk blocks with the files required to boot system.

Loader Assembly code.

Kernel for booting and its options

Linux system partition (/). Not necessarily a disk (usb loader).

Filesystem loaded in memory as a ramdisk. Software support not provided by the kernel to initialize the system.

Link to other loader (boot a different OS)

LILO

- Configuration: /etc/**lilo.conf**
  - Any change in the files employed in boot process (boot.b, kernel, ramdisk) requires loader update:
    - map file must reflect those changes, otherwise booting process is corrupted.
    - Check if map file is updated: # lilo -q
    - Update map file: # lilo [-v]
- A booting error cannot be fixed from the shell…
- Possible error sources:
  - Installation of a new OS overwriting MBR (M$)
  - Failed kernel compilation
  - Modification in boot files without map updating.
- Rescue Systems:
  - mkbootdisk
  - Installation Live CD (option rescue) or specialized (SystemRescueCD)

INIT (SysV)

- The init process performs the following tasks:
  - Step 1: **Configuration**: read from the file **/etc/inittab** the initial configuration of the system: Operation mode, runlevels, consoles,…
  - Step 2: **Initialization**: Runs the command **/etc/init.d/rc.S** (debian), which performs a basic initialization of the system.
  - Step 3: **Services**: According to the runlevel configured, runs the scripts/services pre-established for that runlevel.
- Runlevels (Operation modes)
  - Standard: 7 levels. Each distribution its own configuration (here Debian)
  - Level **S**: only executed at boot time (replaces /etc/rc.boot)
  - Level 0: **Halt**. Employed to Shut down the system.
  - Level 1: **Single User**. Maintenance tasks (no active network)
  - Level 2-5: **Multiuser**. All the network and Graphical services activated.
  - Level 6: **Reboot**: Similar to level 0.

INIT (SysV)

- Step 1, Configuration. The file /etc/inittab:

```
# /etc/inittab: init(8) configuration.

# The default runlevel.
id:2:initdefault:

# Boot-time system configuration/initialization
# script. This is run first except when booting in
# emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes S and K scripts upon change
# of runlevel.
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

```
# Normally not reached, but fallthrough in case of
# emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
…

# Note that on most Debian systems tty7 is used by
# the X Window System, so if you want to add more
# getty's go ahead but skip tty7 if you run X.
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

INIT (SysV)

- Step 1, Configuration. The file /etc/inittab:
  - Line format: **id:runlevels:action:process**
  - **id:** identifier for the entry inside inittab
  - **runlevels:** execution levels for that entry (empty means all)
  - **action:** What must init do with the process.
    - wait: wait until it finishes
    - off: ignore the entry (deactivated)
    - once: run only once
    - respawn: rerun the process if it dies
    - sysinit: ask the user what to do with that entry
    - Special: ctrlaltdel
  - **process:** sh line indicating init which process to start when this entry is reached.

INIT (SysV)

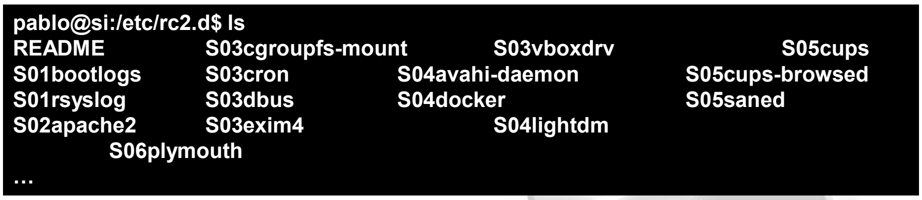
- Step 2, Initialization. The file /etc/init.d/rc:
  - Input parameters: the runlevel. Example rc 2: multiuser
  - Tasks:
    - Establish PATHs
    - Load swap space: swapon
    - Check and mount local filesystems (/ets/fstab)
    - Activate and configure the network
    - Remove not necessary files (/tmp)
    - Configure the kernel. Load modules: Drivers (managing dependencies)
    - Triggers the startup of the services associated to the runlevel.
  - Modifying the runlevel: command init, telinit
    - Allows changing from one runlevel to another
    - ¿Single User?
    - Restore original state.

INIT (SysV)

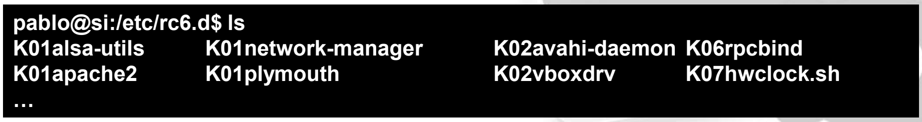
- Step 3, services. The directories /etc/init.d and /etc/rcN.d:
  - All the services available are found in /etc/init.d
    - Examples: cron, ssh, lpd, …
  - How to indicate each runlevel which ones to start?
    - With a special directory, /etc/rcN.d/ (being N the runlevel).
    - In these directories a list of links to the services is found.
  - The directory /etc/rcN.d/
    - The links begin with letters “S” or “K” plus two digits (execution order).
    - “S”: executed in ascending order when a runlevel is started (ssh start).
    - “K”: executed in descending order when shutting down (ssh stop).
    - These links are controlled with “update-rc.d”
  - S99local: script to perform local configurations
    - minor booting aspects: auxiliary kernel modules, personalized services,…
    - Employed by the administrator
    - It really runs the script /etc/rc.local

INIT (SysV)

- Step 3, services. The directories /etc/init.d and /etc/rcN.d:
  - The directory /etc/rcN.d/



```
pablo@si:/etc/rc2.d$ ls
README          S03cgroupfs-mount        S03vboxdrv                S05cups
S01bootlogs     S03cron         S04avahi-daemon        S05cups-browsed
S01rsyslog      S03dbus         S04docker              S05saned
S02apache2      S03exim4                 S04lightdm
        S06plymouth
…
```



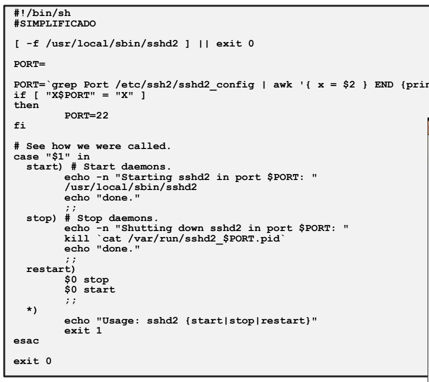
```
pablo@si:/etc/rc6.d$ ls
K01alsa-utils   K01network-manager       K02avahi-daemon K06rpcbind
K01apache2      K01plymouth              K02vboxdrv      K07hwclock.sh
…
```

INIT (SysV)

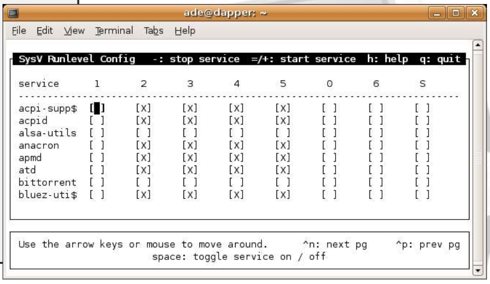
- **Manual** administration of services:
  - After booting process, services can be modified (stop running services or start new services).
  - Directly through its script (example ssh):
    - # /etc/init.d/ssh [stop/start/restart/status]
  - Or through the command service:
    - service --status-all: reads /etc/init.d/ verifying service state [+] [-] [?]
  - These changes are volatile (lost after reboot).
    - Permanent with update.rc-d
  - Checking possible errors concerning boot process
    - # tail -f /var/log/messages (Another important files: syslog, daemon.log
    - # ls -lart /var/log

INIT (SysV)

- Manual administration of services:
  - Examples of start script and services command:



```
#!/bin/sh
#SIMPLIFICADO

[ -f /usr/local/sbin/sshd2 ] || exit 0

PORT=

PORT=`grep Port /etc/ssh2/sshd2_config | awk '{ x = $2 } END {print x}' -`
if [ "X$PORT" = "X" ]
then
        PORT=22
fi

# See how we were called.
case "$1" in
  start) # Start daemons.
        echo -n "Starting sshd2 in port $PORT: "
        /usr/local/sbin/sshd2
        echo "done."
        ;;
  stop) # Stop daemons.
        echo -n "Shutting down sshd2 in port $PORT: "
        kill `cat /var/run/sshd2_$PORT.pid`
        echo "done."
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  *)
        echo "Usage: sshd2 {start|stop|restart}"
        exit 1
esac

exit 0
```



```
 SysV Runlevel Config   -: stop service  =/+: start service  h: help  q: quit

 service      1      2      3      4      5      0      6      S
 ------------------------------------------------------------------
 acpi-supp$  [█]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
 acpid       [ ]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
 alsa-utils  [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
 anacron     [ ]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
 apmd        [ ]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
 atd         [ ]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
 bittorrent  [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
 bluez-uti$  [ ]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]

 Use the arrow keys or mouse to move around.     ^n: next pg     ^p: prev pg
                        space: toggle service on / off
```