CIA LAB BOOTING (2)

LOADING THE OS

1. What is an UEFI OS loader and where does the Ubuntu OS loader reside on the system?

UEFI OS loader is a special type of application that normally takes over control of the system from firmware conforming to this specification. When loaded, the OS loader behaves like any other UEFI application in that it must only use memory it has allocated from the firmware and can only use UEFI services and protocols to access the devices that the firmware exposes.

OS Loader as other UEFI applications stored in directories specific to software vendors. It resides on EFI defined directories in the root directory. EFI Partition Table (ESP) labelled with GUID (Globalle unique identifier). This data stored in GPT Partition entry (GUID value = C12A7328-F81F-11D2-BA4B-00A0C93EC93B) (EFI System).

The following sample directory structure for an ESP present on a hard disk

```
\EFI
   \<OS Vendor 1 Directory>
         <OS Loader Image>
   \<OS Vendor 2 Directory>
         <OS Loader Image>
   . . .
   \<OS Vendor N Directory>
         <OS Loader Image>
   \<OEM Directory>
         <OEM Application Image>
   \<BIOS Vendor Directory>
         <BIOS Vendor Application Image>
   \<Third Party Tool Vendor Directory>
         <Third Party Tool Vendor Application Image>
   \BOOT
         BOOT{machine type short name}.EFI
```

The shown structure can differ for removable media devices.

```
\EFI
   \BOOT
         BOOT{machine type short name}.EFI [1]
```

2. Describe in order all the steps required for booting the computer (util the OS starts running)

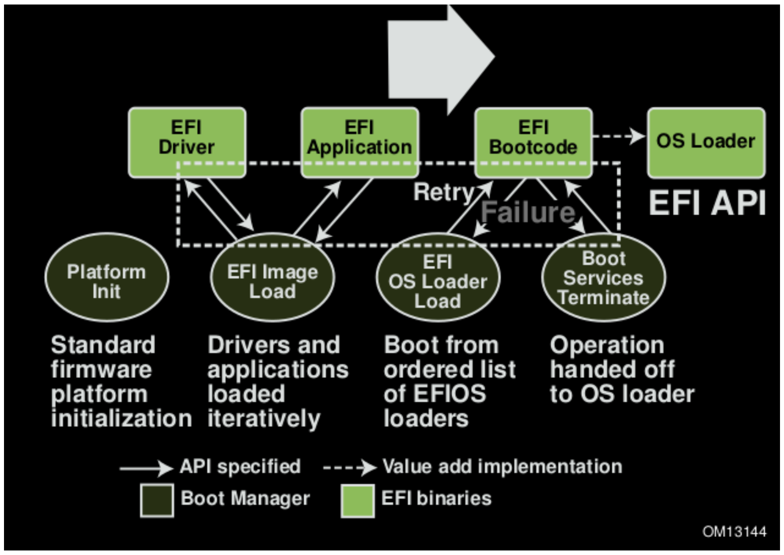


Figure 1 - Boot sequence

UEFI booting sequence begins from Platform init (Power On) phase, where Power On Self Test and Security phase which contains all the CPU initialisation code from the cold boot entry point on (south, north bridges, dram). It's job is to set the system up far enough to find, validate, install and run the PEI (Pre EFI Initialization).

At next stage the EFI Image Load enters in a loop which iteratively loads EFI drivers and applications. EFI applications are modular pieces of code like EFI drivers are. When all the drivers and applications have been loaded, the execution control is transferred to the Boot manager.

Boot manager searches boot device, starts EFI OS loader. The EFI OS loader has an ordered list of EFI OS loaders which it will use to load an OS. Instead of loading an OS it can also start a UEFI Shell.

OS launch. UEFI gives its API, transfers all initialized and loaded drivers to the OS to reduce start time.

3. What is the purpose of the GRUB bootloader in UEFI System?

For UEFI system GRUB provides a true command-based, pre-OS which gives maximum flexibility in loading operating systems with specified options or gathering information about the system.

GRUB boot loader provides menus of choices and can handle many different forms of hardware.

Grub will find the /boot/grub/menu.lst which configures its interactive menu. The location of the menu.lst, is hard-coded into grub when it is installed to the boot sector.

GRUB understands the underlying filesystems.

In some cases, the operating system is split over several partitions (like /usr), and these partitions are mounted by the boot scripts as soon as they can be.

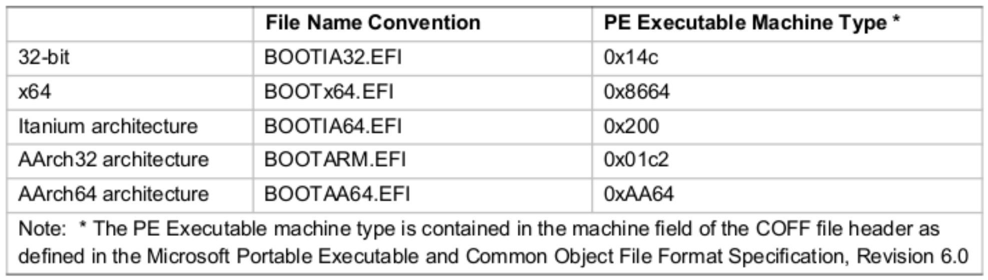
4. How does the ~~<!!!Ubuntu!!!>~~ UEFI OS loader load the GRUB boot loader?

The boot manager will attempt to load UEFI drivers and UEFI applications (including UEFI OS boot loaders) in an order defined by the global NVRAM variables.
EFI can boot from a device using the EFI_SIMPLE_FILE_SYSTEM_PROTOCOL or the EFI_LOAD_FILE_PROTOCOL.

A device that supports the EFI_SIMPLE_FILE_SYSTEM_PROTOCOL must materialize a file system protocol for that device to be bootable. If a device does not wish to support a complete file system it may produce an EFI_LOAD_FILE_PROTOCOL which allows it to materialize an image directly. The Boot Manager will attempt to boot using the EFI_SIMPLE_FILE_SYSTEM_PROTOCOL first. If that fails, then the EFI_LOAD_FILE_PROTOCOL will be used.

To generate a file name when none is present in the FilePath, the firmware must append a default file name in the form \EFI\BOOT\BOOT{machine type short-name}.EFI where machine type short-name defines a PE32+ image format architecture. Each file only contains one UEFI image type, and a system may support booting from one or more images types. Table 12 lists the UEFI image types.

Table 12. UEFI Image Types



	File Name Convention	PE Executable Machine Type *
32-bit	BOOTIA32.EFI	0x14c
x64	BOOTx64.EFI	0x8664
Itanium architecture	BOOTIA64.EFI	0x200
AArch32 architecture	BOOTARM.EFI	0x01c2
AArch64 architecture	BOOTAA64.EFI	0xAA64

Note: * The PE Executable machine type is contained in the machine field of the COFF file header as defined in the Microsoft Portable Executable and Common Object File Format Specification, Revision 6.0

EFI Boot Manager loads EFI Image (*.efi) of grub in \EFI\BOOT\BOOT{machine type short-name}.EFI

5. Explain how the GRUB boot loader, in turn, loads and run the kernel by answering these 3 questions:

(a) What type of filesystem is the kernel on?

Linux kernels could be on any filesystem: ext{2,3,4}, fat{16,32}, ntfs, vfs and so on. You can write you own filesystem and reassembly the kernel.

(b) What type(s) of filesystem does UEFI support?

Devices that support the Simple File System protocol return an EFI_FILE_PROTOCOL. The only function of this interface is to open a handle to the root directory of the file system on the volume. Once opened, all accesses to the volume are performed through the volume's file handles, using the EFI_FILE_PROTOCOL protocol. The volume is closed by closing all the open file handles. The firmware automatically creates handles for any block device that supports the following file system formats:

- FAT12
- FAT16
- FAT32 (now)

(c) What does the GRUB boot loader therefore have to do to load the kernel?

Grub passes 3 stages.

Stage 1: boot.img is stored in the master boot record (MBR) or optionally in any of the volume boot records (VBRs), and addresses the next stage by an LBA48 address (thus, the 1024-cylinder limitation of GRUB legacy is avoided); at installation time it is configured to load the first sector of core.img.

Stage 1.5: core.img is by default written to the sectors between the MBR and the first partition, when these sectors are free and available. For legacy reasons, the first partition of a hard drive does not begin at sector 1 (counting begins with 0) but at sector 63, leaving a gap of 62 sectors of empty space. That space is not part of any partition or file system, and therefore not prone to any problems related with it. Once executed, core.img will load its configuration file and any other modules needed, particularly file system drivers; at installation time, it is generated from diskboot.img and configured to load the stage 2 by its file path.

Stage 2: files belonging to the stage 2 are all being held in the /boot/grub directory, which is a subdirectory of the /boot directory specified by the Filesystem Hierarchy Standard (FHS)

Once GRUB stage 2 has loaded, it presents a TUI-based operating system selection (kernel selection) menu, where the user can select which operating system to boot. GRUB can be configured to automatically load a specified kernel after a user-defined timeout; if the timeout is set to zero seconds, pressing and holding ⇧ Shift while the computer is booting makes it possible to access the boot menu

6. Do you need an OS loader and/or boot loader to load a linux kernel with UEFI? - Explain why of why not

Since version 3.3.x, and only on EFI machines, it is possible to boot the Linux kernel without using a bootloader such as GRUB. So 2 ways can be used:

In the first case you can start Linux kernel via boot loader (e.g. GRUB2). To start GRUB EFI loader goes (in case we use ubuntu loader) /boot/efi/EFI/ubuntu/grub{ia32, x64}.efi. After start GRUB will find the /boot/grub/menu.lst which configures its interactive menu to choose which OS to start.

In the second case you are able to start linux kernel via only EFI OS Loader. But you have to complete theese requirements:

- UEFI [UEFI only] mode in settings must be supported;
- OS 64-bit;
- Linux (Kernel >= 3.3).

Moreover, you have to set kernel parameters: CONFIG_EFI=y; CONFIG_EFI_STUB=y; CONFIG_EFI_VARS=y (switch on UEFI standeart support, enable booting kernel via UEFI (EFI boot stub), control UEFI using variables - to set path to kernel image).

After that to load Linux kernel with UEFI you need to paste kernel file bootx64.efi to /boot/efi/EFI/boot/bootx64.efi.

In summary, EFI OS Loader is a usual EFI Application (section 2.1.3 in UEFI specification). So it depends on a vendor of UEFI. Bootloaders like GRUB are now necessary for systems which supports booting via EFI OS Loader. Moreover, some vendors implement nice boot menu list like GRUB does, so users won't feel uncomfortable.

7. How many stages does GRUB have in MBR-based system and what is their task?

Stages already mentioned in answer 5(c).

8. Where are the different stages found on the disk?

Stage 1 resides in the MBR and because it is very small code, it basically loads the next Stage from a defined location on the disk within the first 1024 cylinders.

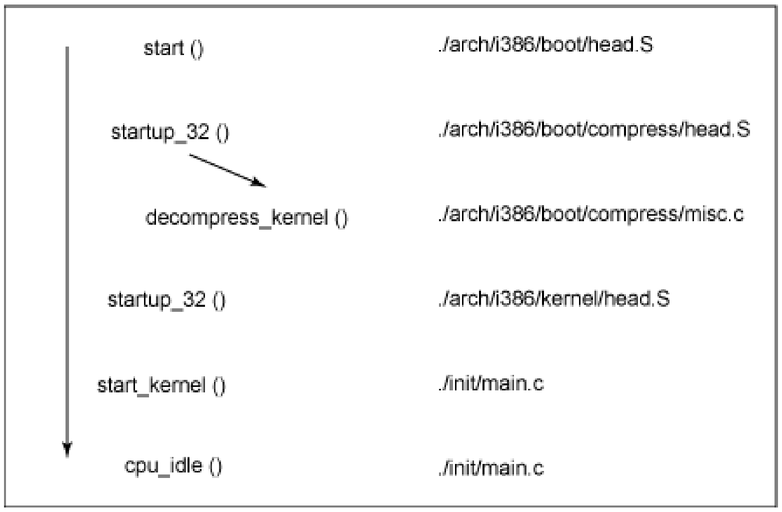
Stage 1 can load Stage 2 directly, but it’s normally set up to load Stage 1.5. Grub Stage 1.5 is located in the first 30 kilobytes of hard disk immediately following the MBR and before the first partition. Stage 1.5 can contain any drivers needed and it then loads Stage 2. Stage 2 will then load the default configuration file and any other modules needed. These files are located in the /boot/grub/ directory.

INITIALIZING THE OS

9. Describe the entire startup process of Ubuntu 16.04 in the default installation. The subquestions below are leaders to help you along, they must be answered but by no means represent the entire startup process of Ubuntu…

With the kernel image in memory and control given from the stage 2 boot loader, the kernel stage begins. The kernel image isn't so much an executable kernel, but a compressed kernel image. Typically this is a zImage (compressed image, less than 512KB) or a bzImage (big compressed image, greater than 512KB), that has been previously compressed with zlib. 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. If an initial RAM disk image is present, this routine moves it into memory and notes it for later use. The routine then calls the kernel and the kernel boot begins.

Major function shown in figure 2.



With the call to `start_kernel`, a long list of initialization functions are called to set up interrupts, perform further memory configuration, and load the initial RAM disk. In the end, a call is made to `kernel_thread` to start the `init` function, which is the first user-space process. Finally, the idle task is started and the scheduler can now take control (after the call to `cpu_idle`). With interrupts enabled, the pre-emptive scheduler periodically takes control to provide multitasking.

During the boot of the kernel, the initial-RAM disk (`initrd`) that was loaded into memory by the stage 2 boot loader is copied into RAM and mounted. This `initrd` serves as a temporary root file system in RAM and allows the kernel to fully boot without having to mount any physical disks. Since the necessary modules needed to interface with peripherals can be part of the `initrd`, the kernel can be very small, but still support a large number of possible hardware configurations. After the kernel is booted, the root file system is pivoted (via `pivot_root`) where the `initrd` root file system is unmounted and the real root file system is mounted.

The `initrd` function allows you to create a small Linux kernel with drivers compiled as loadable modules. These loadable modules give the kernel the means to access disks and the file systems on those disks, as well as drivers for other hardware assets. Because the root file system is a *file system* on a disk, the `initrd` function provides a means of bootstrapping to gain access to the disk and mount the real root file system. In an embedded target without a hard disk, the `initrd` can be the final root file system, or the final root file system can be mounted via the Network File System (NFS) [2].

After kernel is booted it starts systemd (system and service manager daemon). Conceptually systemd provides a dependency system between various entities called "units" of 12 different types. Units encapsulate various objects that are relevant for system boot-up and maintenance. The majority of units are configured in unit configuration files (see 9.b subquestion answer to get configuration file names). According to kind of dependencies processes could be launched in parallel (no ***after*=, *berfore*=, *require*=, *conflict*=,** records), or in the order of previously mentioned parameters.

Last step of booting is Runlevel. Also systemd will initialize some services. They could be launched in different runlevels (0-6). They reside in /etc/rc(0-6).d. If name of service begins with S that means it is launched at startup. Some of the services could launch automatically via user configuration.
Couple of examples of start processes (daemons):

- S01killprocs (rc1.d) – provides killing of processes
- S02Dbus (rc2.d) – provides system Dbus for inter-processes communication
- S03avahi-daemon (rc2.d) – mDNS/DNS-SD daemon. Zeroconf daemon for configuring your network automatically

For user one of the most important service is systemd-logind. This is a tiny daemon that manages user logins and seats in various ways. When it is launched you can enter login/password. For desktop systems (e.g. Ubuntu Xenial Desktop) you can have desktop environment, so when desktop environment is loaded user can enter his data in nice graphical forms. In this case we have automatically set dependencies between logind and desktop environment.

(a) What is the first process started by the kernel?

First process startd by the kernel is "***init***" (in most fresh Linux systems it was replaces by ***systemd***). It is the parent of all processes on the system, it is responsible for starting all other processes; take processes which parents died under control. Processes managed by ***systemd*** are known as jobs and are defined by files in the ***/etc/systemd*** directory.

(b) Where is the configuration kept for the started process?

The default configuration is defined during compilation, so a configuration file is only needed when it is necessary to deviate from those defaults. By default, the configuration file in ***/etc/systemd/***

contains commented out entries showing the defaults as a guide to the administrator. This file can be edited to create local overrides.

The main configuration file is read before any of the configuration directories, and has the lowest precedence; entries in a file in any configuration directory override entries in the single configuration file.

Main configure files - system and session service manager configuration files:

- systemd-system.conf
- system.conf.d
- systemd-user.conf
- user.conf.d

List of other configure files (including full path):

- /etc/systemd/system.conf
- /etc/systemd/system.conf.d/*.conf
- /etc/systemd/user.conf

- /etc/systemd/user.conf.d/*.conf,
- /run/systemd/user.conf.d/*.conf
- /run/systemd/system.conf.d/*.conf
- /lib/systemd/system.conf.d/*.conf
- /usr/lib/systemd/user.conf.d/*.conf

P.S. Files in /etc/ are reserved for the local administrator, who may use this logic to override the configuration files installed by vendor packages.

P.S. P.S.There are some more configuration files (such as /etc/X11 directory - determines which display manager to start; /etc/default directory - contains entries for a range of functions and services)

(c) What is the order of execution of started processes?

List of executed processes is shown in the attached file (see Appendix A: plot.svg).

It's difficult to recognize order of started processes because systemd aggressive parallelization capabilities, uses socket and D-Bus activation for starting services, offers on-demand starting of daemons, keeps track of processes using Linux control groups, supports snapshotting and restoring of the system state, maintains mount and automount points and implements an elaborate transactional dependency-based service control logic.

But firstly socket and D-Bus activation happens, after that loads network services, user environment, udev-control (user devices).

10. To understand the workings of daemons in Ubuntu, we are going to take a closer look at one aspect of the booting process: networking. Please describe the workings of Ubuntu desktop here (Ubuntu server networking is actually simpler):

(a) How is the networking started? Include the names of all configuration files and utilities.

<How networking starts>

In debian system network interfaces are initialized in runlevel S by the init script "ifupdown". Many network services are started under milti-user mode directly as daemon processes at boot time by init script. Actually, ifupdown is a packet and in Ubuntu Xenial (amd64) it contains:

- /etc/default/networking - /etc/init.d/networking is a symlink
- /etc/init/network-interface-container.conf
- /etc/init/network-interface-security.conf
- /etc/init/network-interface.conf
- /etc/init/networking.conf
- /etc/network/if-down.d/upstart
- /etc/network/if-up.d/upstart
- /lib/ifupdown/settle-dad.sh
- /lib/ifupdown/wait-for-ll6.sh
- /lib/udev/ifupdown-hotplug
- /lib/udev/rules.d/80-ifupdown.rules
- /sbin/ifdown
- /sbin/ifquery
- /sbin/ifup

other files (mostly they are manual(man) or example files) can be found here: http://packages.ubuntu.com/ru/xenial/amd64/ifupdown/filelist

<list of networking configuration files>

List of the most important configuration files (according to the ubuntu 16.04 server guide)

- /etc/network/interfaces - file containing interface configurations used on this machine (whenever up/down)
- /etc/resolv.conf
- /etc/hosts - Static hostnames are locally defined hostname-to-IP mappings
- /etc/nsswitch.conf - order in which your system selects a method of resolving hostnames to IP addresses is controlled by the Name Service Switch (NSS) configuration file
- /etc/udev/rules.d/70-persistent-net.rules - Interface logical names are configured in this file

<list of network utils>

List of the most common network utils in ubuntu (and most linux systems)

- ethtool
    - ifconfig
    - ping
    - route
    - netstat
    - traceroute
    - port scanning
    - DNS lookup
    - finger
    - whois
- lshw
- ifup/ifdown (ifupdown packet)

(b) Which process manages the networking configuration?

Main process which manages network configuration is ***network-manager***. You can find it in ***/etc/init.d/network-manager***.

The *NetworkManager* daemon attempts to make networking configuration and operation as painless and automatic as possible by managing the primary network connection and other network interfaces, like Ethernet, WiFi, and Mobile Broadband devices. NetworkManager will connect any network device when a connection for that device becomes available, unless that behavior is disabled. Information about networking is exported via a D-Bus interface to any interested application, providing a rich API with which to inspect and control network settings and operation.

~~***11. As a final installation step, make your experimentation server reachable over IPv6.***~~

REFERENCES

[1] - Unified Extensible Firmware Interface Specification version 2.6 January, 2016 ***http://www.uefi.org/sites/default/files/resources/UEFI%20Spec%202_6.pdf***

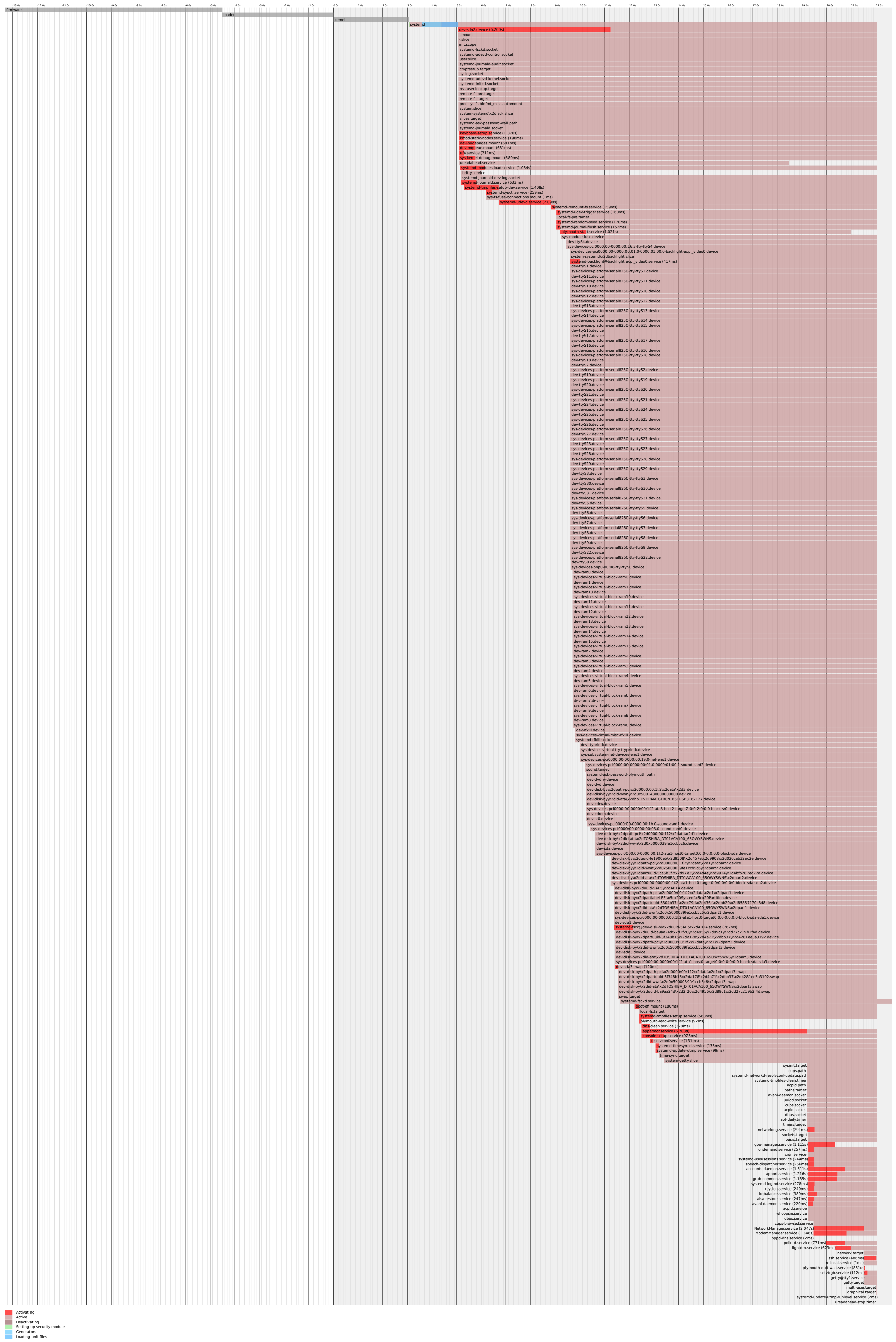
[2] – Inside the Linux boot process http://www.ibm.com/developerworks/linux/library/l-linuxboot/index.html?S_TACT=105AGX99&S_CMP=CP

APPENDIX A

***Map of booting OS Ubuntu 16.04 using* `systemd-analyze plot`**

plot.svg

Linux ( ) x86-64
Startup finished in 8.796s (firmware) + 4.507s (loader) + 3.061s (kernel) + 18.968s (userspace) = 35.333s


firmware
loader
kernel
systemd
dev-sda2.device (6.200s)
-.mount
-.slice
init.scope
systemd-fsckd.socket
systemd-udevd-control.socket
user.slice
cryptsetup.target
syslog.socket
systemd-journald-audit.socket
systemd-udevd-kernel.socket
systemd-initctl.socket
nss-user-lookup.target
remote-fs-pre.target
remote-fs.target
proc-sys-fs-binfmt_misc.automount
system.slice
system-systemd\x2dfsck.slice
slices.target
systemd-ask-password-wall.path
systemd-journald.socket
keyboard-setup.service (1.370s)
kmod-static-nodes.service (198ms)
dev-hugepages.mount (681ms)
dev-mqueue.mount (681ms)
ufw.service (211ms)
sys-kernel-debug.mount (680ms)
ureadahead.service
systemd-modules-load.service (1.034s)
brltty.service
systemd-journald-dev-log.socket
systemd-journald.service (633ms)
systemd-tmpfiles-setup-dev.service (1.408s)
systemd-sysctl.service (259ms)
sys-fs-fuse-connections.mount (1ms)
systemd-udevd.service (2.098s)
systemd-remount-fs.service (159ms)
systemd-udev-trigger.service (160ms)
local-fs-pre.target
systemd-random-seed.service (170ms)
systemd-journal-flush.service (152ms)
plymouth-start.service (1.021s)
sys-module-fuse.device
dev-ttyS4.device
sys-devices-pci0000:00-0000:00:16.3-tty-ttyS4.device
sys-devices-pci0000:00-0000:00:01.0-0000:01:00.0-backlight-acpi_video0.device
system-systemd\x2dbacklight.slice
systemd-backlight@backlight:acpi_video0.service (417ms)
dev-ttyS1.device
sys-devices-platform-serial8250-tty-ttyS1.device
dev-ttyS11.device
sys-devices-platform-serial8250-tty-ttyS11.device
dev-ttyS10.device
sys-devices-platform-serial8250-tty-ttyS10.device
dev-ttyS12.device
sys-devices-platform-serial8250-tty-ttyS12.device
dev-ttyS13.device
sys-devices-platform-serial8250-tty-ttyS13.device
dev-ttyS14.device
sys-devices-platform-serial8250-tty-ttyS14.device
sys-devices-platform-serial8250-tty-ttyS15.device
dev-ttyS15.device
dev-ttyS17.device
sys-devices-platform-serial8250-tty-ttyS17.device
dev-ttyS16.device
sys-devices-platform-serial8250-tty-ttyS16.device
sys-devices-platform-serial8250-tty-ttyS18.device
dev-ttyS18.device
dev-ttyS2.device
sys-devices-platform-serial8250-tty-ttyS2.device
dev-ttyS19.device
sys-devices-platform-serial8250-tty-ttyS19.device
dev-ttyS20.device
sys-devices-platform-serial8250-tty-ttyS20.device
dev-ttyS21.device
sys-devices-platform-serial8250-tty-ttyS21.device
dev-ttyS24.device
sys-devices-platform-serial8250-tty-ttyS24.device
dev-ttyS25.device
sys-devices-platform-serial8250-tty-ttyS25.device
dev-ttyS26.device
sys-devices-platform-serial8250-tty-ttyS26.device
dev-ttyS27.device
sys-devices-platform-serial8250-tty-ttyS27.device
dev-ttyS23.device
sys-devices-platform-serial8250-tty-ttyS23.device
dev-ttyS28.device
sys-devices-platform-serial8250-tty-ttyS28.device
dev-ttyS29.device
sys-devices-platform-serial8250-tty-ttyS29.device
dev-ttyS3.device
sys-devices-platform-serial8250-tty-ttyS3.device
dev-ttyS30.device
sys-devices-platform-serial8250-tty-ttyS30.device
dev-ttyS31.device
sys-devices-platform-serial8250-tty-ttyS31.device
dev-ttyS5.device
sys-devices-platform-serial8250-tty-ttyS5.device
dev-ttyS6.device
sys-devices-platform-serial8250-tty-ttyS6.device
dev-ttyS7.device
sys-devices-platform-serial8250-tty-ttyS7.device
dev-ttyS8.device
sys-devices-platform-serial8250-tty-ttyS8.device
dev-ttyS9.device
sys-devices-platform-serial8250-tty-ttyS9.device
dev-ttyS22.device
sys-devices-platform-serial8250-tty-ttyS22.device
dev-ttyS0.device
sys-devices-pnp0-00:08-tty-ttyS0.device
dev-ram0.device
sys-devices-virtual-block-ram0.device
dev-ram1.device
sys-devices-virtual-block-ram1.device
dev-ram10.device
sys-devices-virtual-block-ram10.device
dev-ram11.device
sys-devices-virtual-block-ram11.device
dev-ram12.device
sys-devices-virtual-block-ram12.device
dev-ram13.device
sys-devices-virtual-block-ram13.device
dev-ram14.device
sys-devices-virtual-block-ram14.device
dev-ram15.device
sys-devices-virtual-block-ram15.device
dev-ram2.device
sys-devices-virtual-block-ram2.device
dev-ram3.device
sys-devices-virtual-block-ram3.device
dev-ram4.device
sys-devices-virtual-block-ram4.device
dev-ram5.device
sys-devices-virtual-block-ram5.device
dev-ram6.device
sys-devices-virtual-block-ram6.device
dev-ram7.device
sys-devices-virtual-block-ram7.device
dev-ram9.device
sys-devices-virtual-block-ram9.device
dev-ram8.device
sys-devices-virtual-block-ram8.device
dev-rfkill.device
sys-devices-virtual-misc-rfkill.device
systemd-rfkill.socket
dev-ttyprintk.device
sys-devices-virtual-tty-ttyprintk.device
sys-subsystem-net-devices-eno1.device
sys-devices-pci0000:00-0000:00:19.0-net-eno1.device
sys-devices-pci0000:00-0000:00:01.0-0000:01:00.1-sound-card2.device
sound.target
systemd-ask-password-plymouth.path
dev-dvdrw.device
dev-dvd.device
dev-disk-by\x2dpath-pci\x2d0000:00:1f.2\x2data\x2d3.device
dev-disk-by\x2did-wwn\x2d0x5001480000000000.device
dev-disk-by\x2did-ata\x2dhp_DVDRAM_GT80N_BSCRSP3162127.device
dev-cdrw.device
sys-devices-pci0000:00-0000:00:1f.2-ata3-host2-target2:0:0-2:0:0:0-block-sr0.device
dev-cdrom.device
dev-sr0.device
sys-devices-pci0000:00-0000:00:1b.0-sound-card1.device
sys-devices-pci0000:00-0000:00:03.0-sound-card0.device
dev-disk-by\x2dpath-pci\x2d0000:00:1f.2\x2data\x2d1.device
dev-disk-by\x2did-ata\x2dTOSHIBA_DT01ACA100_65OWYSWNS.device
dev-disk-by\x2did-wwn\x2d0x5000039fe1ccb5c6.device
dev-sda.device
sys-devices-pci0000:00-0000:00:1f.2-ata1-host0-target0:0:0-0:0:0:0-block-sda.device
systemd-fsck@dev-disk-by\x2duuid-5AE5\x2dAB1A.service (767ms)
dev-sda3.swap (120ms)
swap.target
systemd-fsckd.service
boot-efi.mount (180ms)
local-fs.target
systemd-tmpfiles-setup.service (568ms)
plymouth-read-write.service (92ms)
dns-clean.service (328ms)
apparmor.service (6.703s)
console-setup.service (923ms)
resolvconf.service (131ms)
systemd-timesyncd.service (133ms)
systemd-update-utmp.service (99ms)
time-sync.target
system-getty.slice
sysinit.target
cups.path
systemd-networkd-resolvconf-update.path
systemd-tmpfiles-clean.timer
acpid.path
paths.target
avahi-daemon.socket
uuidd.socket
cups.socket
acpid.socket
dbus.socket
apt-daily.timer
timers.target
networking.service (291ms)
sockets.target
basic.target
gpu-manager.service (1.115s)
ondemand.service (257ms)
cron.service
systemd-user-sessions.service (244ms)
speech-dispatcher.service (256ms)
accounts-daemon.service (1.551s)
apport.service (1.216s)
grub-common.service (1.185s)
systemd-logind.service (278ms)
rsyslog.service (240ms)
irqbalance.service (389ms)
alsa-restore.service (247ms)
avahi-daemon.service (220ms)
acpid.service
whoopsie.service
dbus.service
cups-browsed.service
NetworkManager.service (2.047s)
ModemManager.service (1.346s)
pppd-dns.service (2ms)
polkitd.service (771ms)
lightdm.service (625ms)
network.target
ssh.service (886ms)
rc-local.service (1ms)
plymouth-quit-wait.service (851us)
setvtrgb.service (112ms)
getty@tty1.service
getty.target
multi-user.target
graphical.target
systemd-update-utmp-runlevel.service (2ms)
ureadahead-stop.timer
🟥 Activating
Active
Deactivating
Setting up security module
Generators
Loading unit files