Innopolis University

SYSTEM AND NETWORKING ENGINEERING

Classical Internet Applications

LABORATORY REPORT 2 corrected

Booting(2)

Student Name	**Student ID**
Sergey Grebennikov	47611

Lecturer:

Rasheed Hussain

Submission Date : September 26, 2017

Contents

1 Introduction

This lab work is allowed to understand the process of loading and initializing the operating system. It consists of two parts:

1. Loading the OS
2. Initializing the OS

2 Main Part

2.1 Loading the OS

Now that we have Ubuntu installed, we will look at the booting process.

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

   Answer:

   A UEFI OS loader is a special type of UEFI application that normally takes over control of the system from firmware conforming to this specification [1]. The Ubuntu OS loader resides under **/boot** directory.

> Noo, not actually there. Please provide full path. I see that you did it in the second question, but the answer for the 1st isn't complete.

> The Ubuntu OS loader **grubx64.efi** resides under **/boot/efi/EFI/ubuntu** directory (Figure 3).

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

   Answer:

   When the computer is powered on, the Unified Extensible Firmware Interface (UEFI) starts initializing the firmware using the embedded hardware code. This code allows to transfer control to the next code that runs on the computer system. This part of the boot process is called the Security (SEC) phase. The UEFI software is stored on a ROM chip on the motherboard.

   The next phase of the UEFI boot process is the Pre-EFI Initialization (PEI) Phase. The PEI phase is designed to initialize low-level hardware components, such as a processor, chipset and motherboard. The purpose of the PEI Phase is to prepare the system for the Driver Execution Environment (DXE) phase.

   The firmware executed in the DXE phase is responsible for searching for and executing drivers that provide device support during the boot process.

   The final tasks are performed in the Boot Device Selection (BDS) phase. This phase initializes console devices for simple input/output operations on the system. The system control passes from the UEFI to the boot loader. The boot loader is usually stored in the EFI partition. If the UEFI OS loader successfully loads its operating system, all boot services in the system are terminated, including memory management, and the UEFI OS loader is responsible for the continued operation of the system. (Figure 1) [1][2][3]

> How and where NVRAM is involved? On what stage boot manager is used? What is boot disk order? How you can obtain it from the terminal?

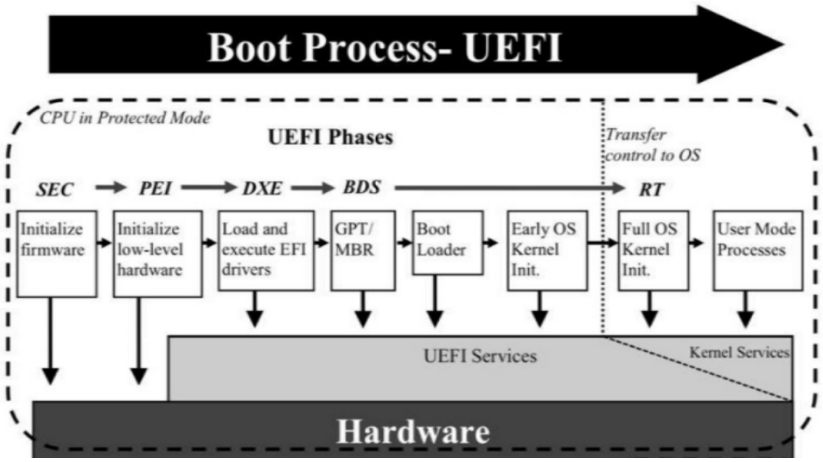

Figure 1: UEFI boot process

> **NVRAM** stores variables that define the boot configuration and is involved in the BDS phase. The boot manager is used at all stages until system control passes from UEFI to the specified OS loader or operating system kernel. Boot disk order defines which disks a boot manager should check for the operating system's boot files. Using **efibootmgr -v** utility we can obtain information about it.

Ubuntu uses the UEFI OS loader to load and run the GRUB boot loader

3. What is the purpose of the GRUB boot loader in a UEFI system?

   Answer:

   The GNU GRand Unified Boot loader (GRUB) is a program which allows to choose which Operating System (OS) to boot during system boot time. It also enables the user to pass arguments to the kernel. [4]

> What do you mean by this "It also enables the user to pass arguments to the kernel"?

> User can pass arguments to the kernel when starting the kernel (usually, when invoked from a boot loader) or at runtime (through the files in **/proc** and **/sys**). You can check the parameters your system was booted up with by running **cat /proc/cmdline** and see if it includes your changes. Kernel parameters can be set either temporarily by editing the boot menu when it shows up, or by modifying the boot loader's configuration file. You can set the following arguments: root filesystem, specify the location of the initial ramdisk, run specified binary instead of **/sbin/init** (symlinked to **systemd**) as init process, boot to shell and other. [11]

4. How does the UEFI OS loader load the GRUB boot loader?

   Answer:

The UEFI-based platform reads the partition table on the system storage and mounts the EFI System Partition (ESP). The ESP contains UEFI boot loader and other utility software. Viewed from within the Ubuntu 16.04.1 file system, the ESP is **/boot/efi/** and mounted to **/dev/sda1** partition (Figure 2), and EFI software provided by Ubuntu is stored in **/boot/efi/EFI/ubuntu/** (Figure 3).

```
sergey@gsa-sne:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7,8G     0  7,8G   0% /dev
tmpfs           1,6G  9,4M  1,6G   1% /run
/dev/sda2       901G   14G  841G   2% /
tmpfs           7,8G  692K  7,8G   1% /dev/shm
tmpfs           5,0M  4,0K  5,0M   1% /run/lock
tmpfs           7,8G     0  7,8G   0% /sys/fs/cgroup
/dev/sda1       511M  3,4M  508M   1% /boot/efi
tmpfs           1,6G   92K  1,6G   1% /run/user/1000
```

Figure 2: EFI System Partition

```
sergey@gsa-sne:~$ sudo ls -l /boot/efi/EFI/ubuntu
[sudo] password for sergey:
total 3468
drwx------ 2 root root    4096 авг 14 14:12 fw
-rwx------ 1 root root   64352 авг 14 14:12 fwupx64.efi
-rwx------ 1 root root     126 авг 14 14:13 grub.cfg
-rwx------ 1 root root 1133432 авг 14 14:13 grubx64.efi
-rwx------ 1 root root 1168464 авг 14 14:13 mmx64.efi
-rwx------ 1 root root 1169992 авг 14 14:13 shimx64.efi
```

Figure 3: EFI Software

The **/boot/efi/EFI/ubuntu/** directory contains **grubx64.efi**, a version of GRUB compiled for the EFI firmware architecture as an EFI application. In the simplest case, the EFI boot manager selects **grubx64.efi** as the default boot loader and reads it into memory.

The EFI boot manager can load any of the operating system boot loaders that are present in the ESP. If the ESP contains other EFI applications, the EFI boot manager might prompt you to select an application to run, rather than load **grubx64.efi** automatically.

What about shim and secure boot?

UEFI Secure Boot is a method that allows to restrict the execution of EFI programs during system startup. UEFI firmware in this mode is performed only by those loaders that have a verified digital signature. A root CA is embedded in firmware such that it can then validate the signed bootloader (**shimx64.efi**), the signed bootloader (**shimx64.efi**) can then validate the signed 2nd stage boot loader

(**grubx64.efi**), and the 2nd stage grub bootloader boots a signed **Linux kernel**. After loading the Linux kernel, the Secure Boot zone of responsibility ends. (Figure 3)

```
$ efibootmgr -v
BootCurrent: 0000
Timeout: 0 seconds
BootOrder: 0000,0001,0002,0003,0004,0005
Boot0000* ubuntu        HD(1,GPT,c355e600-4097-4791-8389-e11d473ea9a1,0x800,0x1
 ↪  00000)/File(\EFI\ubuntu\shimx64.efi)
Boot0001* DTO UEFI USB Floppy/CD        VenMedia(b6fef66f-1495-4584-a836-3492d1
 ↪  984a8d,0500000001)..BO
Boot0002* DTO UEFI USB Hard
 ↪  Drive        VenMedia(b6fef66f-1495-4584-a836-3492d1984a8d,0200000001)..BO
Boot0003* DTO Legacy USB Floppy/CD        VenMedia(b6fef66f-1495-4584-a836-3492
 ↪  d1984a8d,0500000000)..BO
Boot0004* Hard Drive        BBS(HD,,0x0)..GO..NO?.........F.a.k.e. .U.s.b.
 ↪  .O.p.t.i.o.n.............^^?.....BO..NOw.......+.S.A.T.A. . .P.M.:.
 ↪  .T.O.S.H.I.B.A.
 ↪  .D.T.0.1.A.C.A.1.0.0.............^^?.............rN.D+..,.\......^^?.....BO
Boot0005* IBA GE Slot 00C8 v1550        BBS(Network,,0x0)..BO
```

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

   (a) What type of file system is the kernel on?

   Answer:

   The Linux kernel **vmlinuz-4.10.0-32-generic** is stored under **/boot** directory using ext2/ext3 type of file system (Figure 4). But when the kernel is loaded into a memory, the boot loader also loads an initial RAM–based file system (**initramfs**), so it can be used directly by the kernel. [2]

```
sergey@gsa-sne:~$ uname -a
Linux gsa-sne 4.10.0-32-generic #36~16.04.1-Ubuntu SMP Wed Aug 9 09:19:02 UTC 20
17 x86_64 x86_64 x86_64 GNU/Linux
sergey@gsa-sne:~$ stat -f /boot/vmlinuz-4.10.0-32-generic
  File: "/boot/vmlinuz-4.10.0-32-generic"
    ID: ac5fa0c817f7546c Namelen: 255     Type: ext2/ext3
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 236092013  Free: 232445331  Available: 220446791
Inodes: Total: 59973632   Free: 59525811
```

Figure 4: Linux kernel

ok. But initramfs is loaded not by a loader but by the kernel.

Not actually. An image of this initial RAM-based file system (along with the kernel image) must be stored somewhere accessible by the Linux bootloader or the boot firmware of the computer. The bootloader will load the kernel and initial RAM-based file system image into memory and then start the kernel, passing in the memory address of the image. At the end of its boot sequence, the kernel tries to determine the format of the image from its first few blocks

of data, which can lead either to the initramfs scheme. Then the image is unpacked by the kernel into a special instance of a tmpfs that becomes the initial root file system. [12]

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

Answer:

The file system supported by the UEFI is based on the FAT file system (msdos) (Figure 5).

```
sergey@gsa-sne:~$ stat -f /boot/efi/
  File: "/boot/efi/"
    ID: 80100000000 Namelen: 1530     Type: msdos
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 130812     Free: 129942     Available: 129942
Inodes: Total: 0          Free: 0
```

Figure 5: File system supported by UEFI

What filesystems are supported by UEFI?

File systems whose specification is based on the FAT file system (*FAT12, FAT16 or FAT32*)

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

Answer:

The GRUB boot loader must choose which operating system to boot. After choosing the OS, the boot loader loads the kernel of the selected operating system into memory and passes control to it.

Ok, GRUB must choose? And how is the procedure of loading goes? Please provide include proofs in your report

The boot loader provides the user with a menu of possible boot options and has a default option that is selected after a while. Once the choice is made, the boot loader loads the kernel into memory, passes some parameters to it and gives it control. But if we only have one installed OS, then the boot loader automatically selects this OS without user intervention.

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

Answer:

To load a Linux kernel with UEFI it is necessary to have an UEFI OS loader, but it is not obligatory to have a boot loader. Since kernel version 3.3.x on EFI machines it is possible to boot the Linux kernel without using a boot loader such as GRUB.

This release introduces an **EFI Boot Stub** that allows an OS image to be loaded and executed directly by EFI firmware. [5]

ok. Do you need to recompile the kernel?

No. We can use **UEFI Shell** or **efibootmgr** utility to embed the kernel parameters within a UEFI boot entry to boot OS directly.

In an MBR-based system, the GRUB boot loader is distributed over the disk in multiple parts.

7. How many parts (or stages) does GRUB have in such a system, and what is their task?

   Answer:

   To fit within the MBR boot scheme the GNU GRUB boot loader itself is divided into 3 stages.

   Stage 1: In this stage, image **boot.img** load the next stage of boot loader and at installation time it is configured to load the first sector of image **core.img**.

   Stage 1.5: In this stage, image **core.img** contains file system drivers, enable it to directly load stage 2 from any known location in the file system, for example from **/boot/grub**.

   Stage 2: This stage loads the default configuration file and any other modules needed to load a kernel or select kernel if there are multiple operating systems. (Figure 6)

Is it possible to skip stage 1.5? How many stages in GRUB2?

Yes, the 1.5 stage is optional and loaded by the first-stage loader in case the second-stage loader is not contiguous or if the file-system or hardware requires special handling in order to access the second-stage loader. GRUB2 has two (optionally three) stages.

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

   Answer:

   Stage 1: The image **boot.img** resides at the first sector of the hard disk in the master boot record (MBR).

   Stage 1.5: The image **core.img** is located in the sectors between the MBR and the first partition, if they are available and free.

   Stage 2: The files belonging to this stage are stored in the **/boot/grub** directory. (Figure 6)

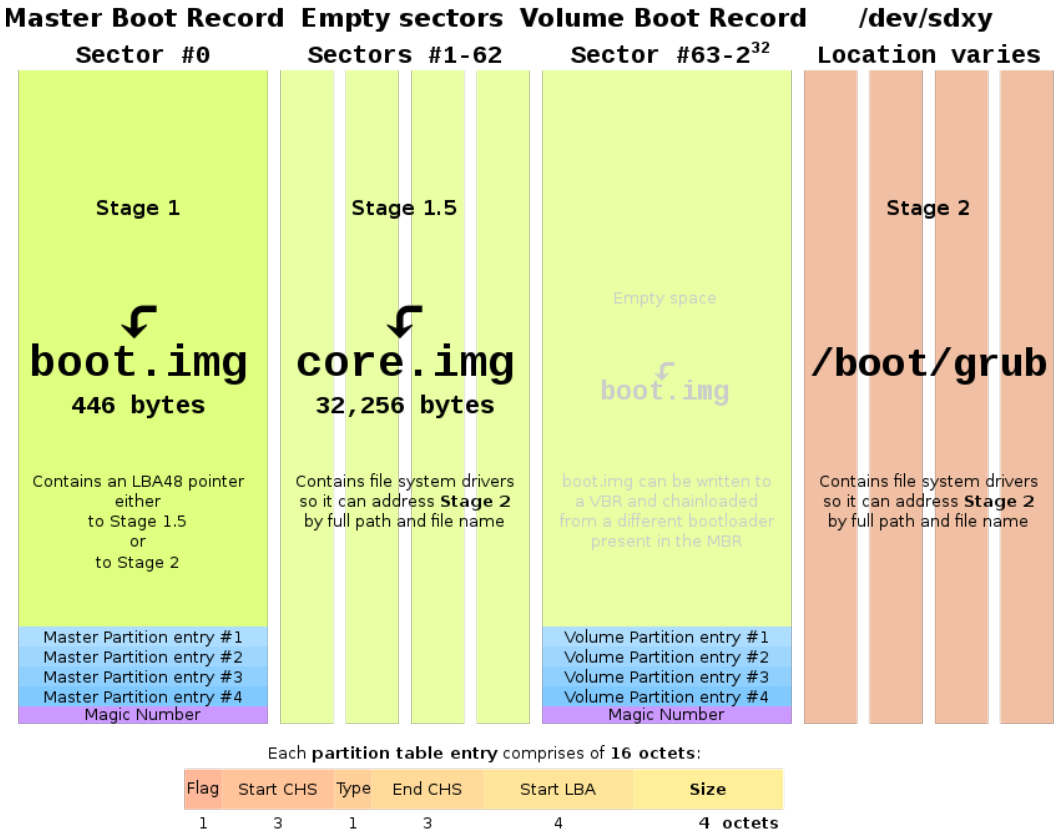

Figure 6: GNU GRUB on MBR partitioned hard disk drives

2.2 Initializing the OS

After the machine has booted up, the chosen OS is started and goes through a (large) number of steps before you end up at the login prompt. We will be looking at the startup procedure of Ubuntu.

9. Describe the entire startup process of Ubuntu 16.04 in the default installation.

   Answer:

   (a) The boot loader loads the kernel and an initial RAM–based file system (initramfs) into memory, so it can be used directly by the kernel

   (b) The kernel uncompresses itself

Where is the configuration kept? And what is the difference in it?

Systemd brings the concept of the **systemd units**. **Units** are represented by configuration files located in one of the directories:

- ***/lib/systemd/system*** - location of the standard configuration units.
- ***/usr/lib/systemd/system/*** - units from the installed Deb packages.
- ***/run/systemd/system/*** - units created in runtime. This directory is more important than the directory with the installed units from the packages.

- ***/etc/systemd/system/*** - units created and managed by the system administrator. This directory is more important than the catalog of units created in runtime.

(c) The kernel check and analyze the system hardware attached to the system

ok, but how is the boot order configured?

Systemd handles boot and services management process using "**targets**" that group different boot units and start up synchronization processes. The very first *target* executed by *systemd* is **default.target**. But **default.target** is actually a symlink to **graphical.target** that locates at **/usr/lib/systemd/system/** directory. At this stage, **multi-user.target** has been invoked and this target keeps its further sub-units inside **/etc/systemd/system/multi-user.target.wants/** directory. "multi-user.target" passes control to another layer "**basic.target**". "basic.target" unit is the one that starts usual services specially graphical manager service. It uses **/etc/systemd/system/basic.target.wants/** directory to decide which services need to be started, basic.target passes on control to **sysinit.target**. "sysinit.target" starts important system services like file System mounting, swap spaces and devices, kernel additional options etc. sysinit.target passes on startup process to **local-fs.target**. local-fs.target , no user related services are started by this target unit, it handles core low level services only. (Figure 9)

(d) The kernel initialize any hardware device drivers built into it

(e) The initramfs file system image contains programs and binary files that perform all actions needed to mount the proper root file system

(f) The kernel finds root file system

(g) After the root file system has been found, it is checked for errors and mounted

(h) The mount program instructs the kernel that a file system is ready for use

(i) The kernel associates file system with a specific point in the overall hierarchy of the file system (the mount point)

(j) The initramfs is cleared from memory

(k) The kernel runs the **init** process on the root file system under (**/sbin**) directory

(l) But actually the process **init** is just symbolic link (Figure 7) to the shared object (daemon) **/lib/systemd/systemd** [1]

```
sergey@gsa-sne:~$ ls -l /sbin/init
lrwxrwxrwx 1 root root 20 авг 14 14:10 /sbin/init -> /lib/systemd/systemd
```

Figure 7: Symbolic link to systemd

[1]Systemd is designed to be backwards compatible with SysV init scripts, and provides a number of features such as parallel startup of system services at boot time, on-demand activation of daemons, support for system state snapshots, or dependency-based service control logic.[9]

(m) **systemd** provides a dependency system between various entities called "**units**" of 11 different types[2] which encapsulate various objects that are relevant for system boot-up and maintenance (Figure 8). [10]

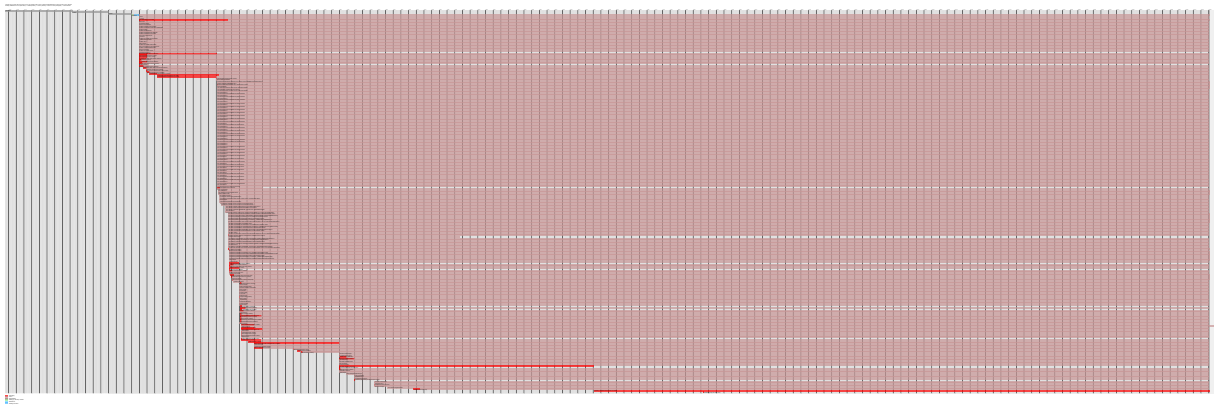

Figure 8: Entire startup process

(n) On boot **systemd** activates the target unit **default.target** whose job is to activate on-boot services and other on-boot units by pulling them in via dependencies(Figure 9). [10]

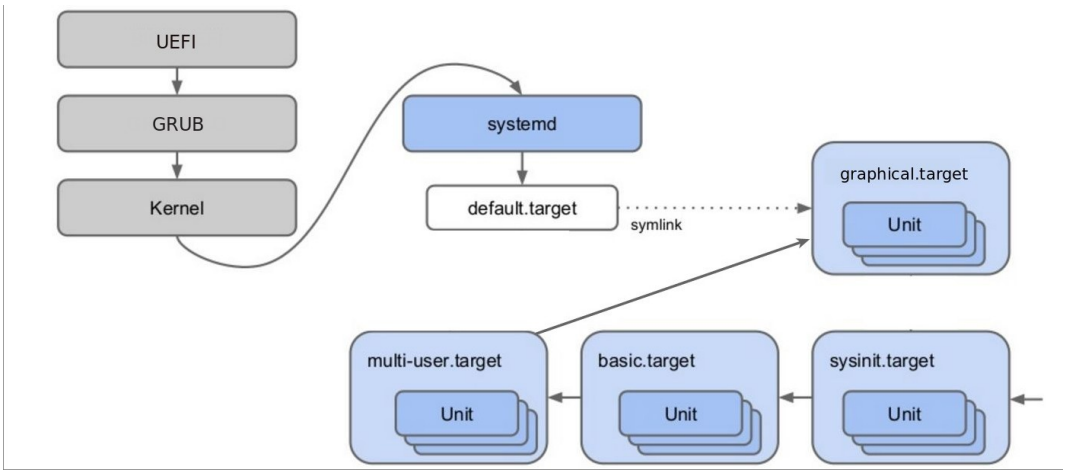

Figure 9: Ubuntu boot process with systemd

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.

    Answer:

    When **default.target** is started it refers to **multi-user.target** which points to the service unit configuration files **networking.service** and **network-manager.service**.

[2]The following unit types are available: 1. Service units, 2. Socket units, 3. Target units, 4. Device units, 5. Mount units, 6. Automount units, 7. Timer units, 8. Swap units, 9. Path units, 10. Slice units, 11. Scope units

The unit file **networking.service** run the network services and install necessary targets. The unit file **network-manager.service** is a symbolic link to the **NetworkManager.service** which starts network management daemon - NetworkManager. NetworkManager will execute scripts in the **/etc/NetworkManager/dispatcher.d** directory or subdirectories in alphabetical order in response to network events. (Figure 10)

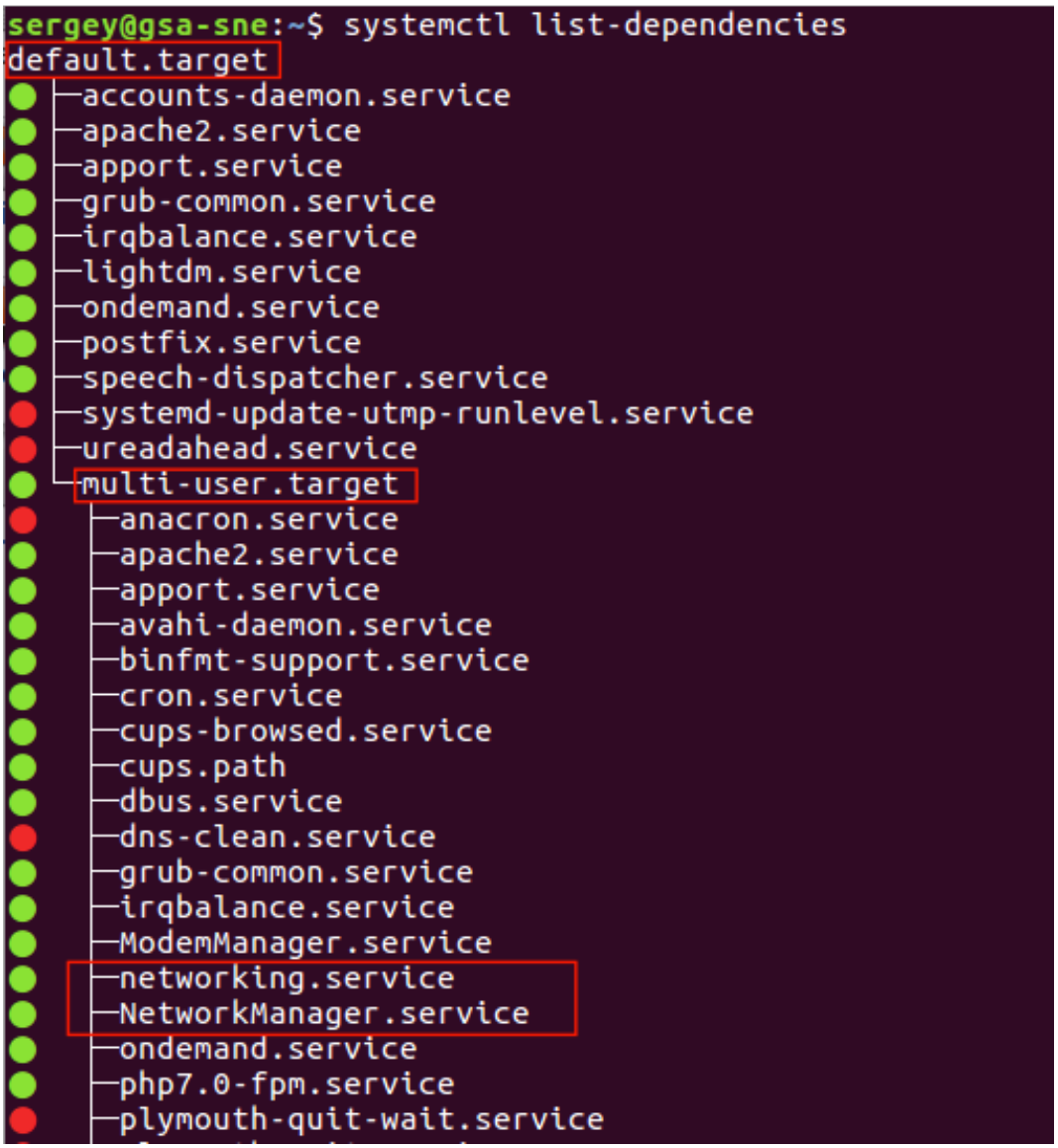

Figure 10: Control the systemd system and service manager

Where is the configuration kept? What about network interfaces and devices?

networking.service raises or downs the network interfaces:

```
# cat /lib/systemd/system/networking.service
...
[Service]
Type=oneshot
EnvironmentFile=-/etc/default/networking
```

```
...
ExecStart=/sbin/ifup -a --read-environment
ExecStop=/sbin/ifdown -a --read-environment --exclude=lo
...
```

The network configuration files are located in the **/etc/NetworkManager** directory:

```
$ sudo cat /etc/NetworkManager/system-connections/Wired\ connection\ 1

[connection]
id=Wired connection 1
uuid=2b49fa71-9f88-3e12-a21e-f1be45ea8e22
type=ethernet
autoconnect-priority=-999
permissions=
secondaries=
timestamp=1505449323

[ethernet]
duplex=full
mac-address=EC:B1:D7:3D:30:6A
mac-address-blacklist=

[ipv4]
address1=188.130.155.46/27,188.130.155.33
dns=8.8.8.8;
dns-search=
method=manual

[ipv6]
addr-gen-mode=stable-privacy
dns-search=
ip6-privacy=0
method=auto
```

(b) Which process manages the networking configuration?

Answer:

Network Manager

Can you provide full process/daemon name from the terminal? What other daemons/services are launched that network manager delegated some control?

It process is located in **/usr/sbin/NetworkManager** path and is started with **/lib/systemd/system/NetworkManager.service** unit:

```
# cat /lib/systemd/system/NetworkManager.service
...
ExecStart=/usr/sbin/NetworkManager --no-daemon
...
```

3 Conclusion

The process of loading the operating system may differ depending on the platform. There are also different types of operating system boot on the Linux kernel. In general, the process of running any operating system consists of several stages. But if simplify the bootstrapping scheme, it can imagine this as follows: firstly the hardware is tested and the necessary set of drivers is installed, then using the firmware utilities is selected the boot loader that loads the kernel of the operating system. Next, the kernel starts the initial process and services for installing the complete operating system. But, of course, this process is much more complicated and has its pitfalls.

4 References

[1] Unified Extensible Firmware Interface Specification [http://www.uefi.org/sites/default/files/resources/UEFI_Spec_2_7.pdf]

[2] LinuxFoundationX: LFS101x.2 Introduction to Linux. Course Chapter 03: Linux Structure and Installation. Section 2: The Boot Process [https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS101x.2+1T2015/]

[3] BIOS vs UEFI [http://blog.csdn.net/ymzhou117/article/details/7085621]

[4] Red Hat Enterprise Linux 6. Installation Guide [https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Installation_Guide/index.html]

[5] Linux Kernel Newbies [https://kernelnewbies.org/Linux_3.3#head-b8360946e9fb561ee77f807ced5396e64aabeca4]

[6] Wikipedia: GNU GRub https://en.wikipedia.org/wiki/GNU_GRUB

[7] Wikipedia: systemd https://en.wikipedia.org/wiki/Systemd

[8] Linux boot up process - systemd http://weng-blog.com/2016/11/linux-boot-systemd/

[9] Red Hat Enterprise Linux, 7.4, System Administrator's Guide, Chapter 9. Managing Services with systemd https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/chap-Managing_Services_with_systemd.html

[10] Man page: systemd

[11] Kernel parameters https://wiki.archlinux.org/index.php/kernel_parameters#GRUB

[12] Initial RAM-disk https://en.wikipedia.org/wiki/Initial_ramdisk

[13] Systemd Boot Process a Close Look in Linux https://linoxide.com/linux-how-to/systemd-boot-process/