LPIC-1 101-400 – Lesson 10

101.2 Boot the system

Basic Input/Output System (BIOS)

- **BIOS** is the basic startup system on PCs (aka Firmware)
- It is saved in ROM
- Provides the Power On Self Test (POST)
- Provides the initial configuration of devices
- It detects the boot loader and starts the system

Basic Input/Output System (BIOS)

- Every vendor has a different interactive environment
- Using the interactive environment we can
  - configure devices
  - set the time
  - enable/disable system devices
  - enable/disable peripherals
  - set the boot devices
  - set security passwords for BIOS and startup
- Most vendors have today replaced the legacy BIOS with the more powerful UEFI (Unified Extensible Firmware Interface)

Show the kernel startup parameters

- **`$ cat /proc/cmdline`** `# show the kernel startup parameters`
- These parameters can be set by the boot loader configuration file
- **/boot/grub/menu.lst** # Legacy **GRUB** configuration file
- **/boot/grub/grub.cfg** # **GRUB2** configuration file on Debian and most systems
- **/boot/grub2/grub.cfg** # **GRUB2** configuration file on Red Hat, CentOS and Fedora

The */etc/default/grub* file

- For GRUB2 in most cases do not edit the configuration files directly. Usually there is a helper configuration file under **/etc/default/grub** and we can edit startup parameters in there.
- **$ cat /etc/default/grub**

```
GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=3
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null ||
echo Debian`
GRUB_TERMINAL="console serial"
GRUB_CMDLINE_LINUX_DEFAULT="console=tty1
console=ttyS0"
GRUB_CMDLINE_LINUX="
```

- If we want to add kernel parameters we can add these in the GRUB_CMDLINE_LINUX variable and then run:
  # update-grub

Some important kernel parameters

- **quiet** # hide messages during startup
- **rw** # mount root device for read and write
- **ro** # mount root device for read-only
- **root=/dev/sda1** # set root filesystem
- **S** # enter single user mode (runlevel 1)
- **init=/sbin/init** # set alternative startup program instead of **/sbin/init** (e.g. /bin/bash)
- **acpi=off** # disable power saving subsystem
- **console=ttyS0 #** set serial port as the default terminal
- **debug** # enable debug mode
- **initrd=/boot/initrd.gz** # set initial ramdisk
- **mem=1024M** # alternative configuration of available memory
- **noapic** # disable APIC subsystem
- **nousb** # disable USB subsytem

Start-up procedure of a Linux system

1. The BIOS (or UEFI) starts first and when POST is run, and the hardware is initialized, it detects the boot device
2. The BIOS finds the boot sector and boots the bootloader:
    1. S**tage 1**: Load the basic boot loader
    2. **Stage 1.5**: (optional) load file system drivers
    3. **Stage 2**: loads the full blown boot loader from **/boot/grub**
3. The bootloader loads the operating system chosen in the menu or the default entry
4. If the chosen system is Linux the kernel is loaded from **/boot/vmlinuz-<kernel-version>**.
5. The kernel load the **init** program which enters the default runlevel.
6. The system enters the default runlevel and all services configured under it will start

http://xkcd.org/340/

Kernel and Initial Ramdisk (initrd)

- **$ ls -la /boot/vmlinuz-*** # show kernel files

```
-rw------- 1 root root 8249080 Apr 24 07:42 /boot/vmlinuz-4.15.0-20-generic
-rw------- 1 root root 8257272 May 23 20:49 /boot/vmlinuz-4.15.0-23-generic
-rw------- 1 root root 8257272 Jun 13 11:33 /boot/vmlinuz-4.15.0-24-generic
```

- **$ ls -lah /boot/initrd*** # show initial ramdisk files

```
-rw-r--r-- 1 root root 56M Jun 30 08:10 /boot/initrd.img-4.15.0-20-generic
-rw-r--r-- 1 root root 57M Jun 30 08:11 /boot/initrd.img-4.15.0-23-generic
-rw-r--r-- 1 root root 57M Jul  3 10:05 /boot/initrd.img-4.15.0-24-generic
```

- The **vmlinuz-*** files are the available kernels. Usually the most recent is loaded. We can load older kernels from the GRUB menu if the most recent kernel is broken for some reason. We can identify the version of the running kernel with **`uname -r`**
- The **initrd-*** files are the available initial ramdisk that contain all the device drivers necessary for startup. Their versions match those of the kernel files. Each respective **initrd** for each kernel is temporarily loaded in memory, the necessary drivers are loaded and when startup is complete it is unloaded from memory. A missing **initrd** can cause a **kernel panic** which is a failed startup!

The Linux Kernel

- Started as a hobby project from Linus Torvalds
- It is now developed by the Linux Foundation
- Monolithic Architecture
- It can be extended using modules
- It is a collaborative project with the participation of more then 200 companies
- Supports almost all legacy and modern CPU architectures

The Linux Kernel

- When there is a kernel upgrade the old kernel is usually available
- You need a restart to launch a new kernel
- There is a **livepatch** module that allows the patching of a loaded kernel!
- Every specific kernel has its own modules under **/lib/modules** in a subdirectory named after the kernel version

http://xkcd.org/456/

Kernel Messages

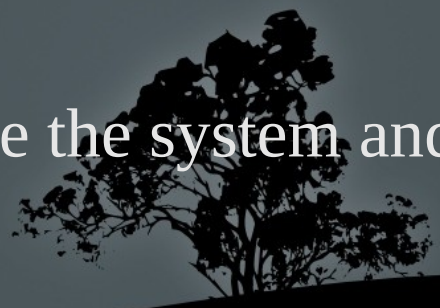
- `$ dmesg | less # show all kernel messages`
- `$ dmesg | tail -n20 # show the 20 most recent kernel messages`
- Kernel messages provide valuable information about the running kernel, as well as information about the CPU, the memory, system devices, modules, partitions etc

Log Files for system messages

- **/var/log/messages** # this is the main system log file that records kernel ans general system messages
- **/var/log/syslog** # has replaced **/var/log/messages** in some distributions
- `$ tail -f -n30 /var/log/messages` # watch live events as they come in the main log file

Init systems

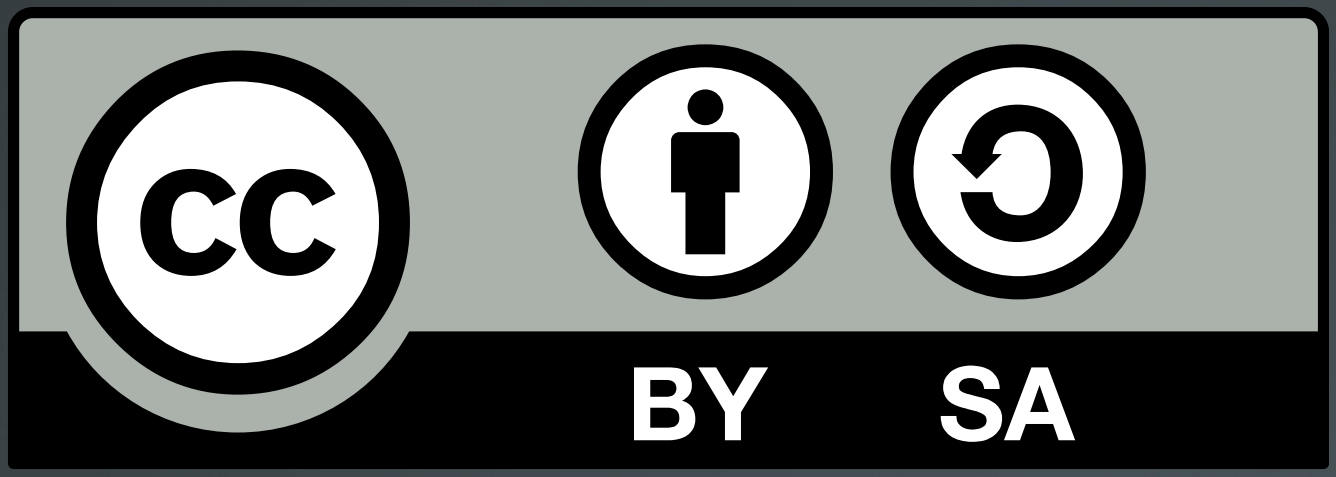
- Init systems are responsible for the initialization of a Linux system and service startup
- In modern Linux systems you can find mainly two init systems:
  - **System V init (sysvinit)**: legacy init system for Linux systems
  - **systemd**: a more powerful init system with concurrent services startup and service management
- In old Ubuntu systems (version 14.04 and back) you can find the **Upstart** init system which is also a prominent **sysvinit** alternative
- After the kernel is loaded the **init** executable is called (/sbin/init)
- After **init** is run the init system will initialize the system and start the services



sysvinit vs systemd

- **sysvinit:**
  - Sequential services startup
  - Based on scripts under **/etc/init.d**
  - Separates the services into **runlevels**
- **systemd:**
  - Concurrent services startup
  - You can define service dependencies
  - Services are monitored and conditionally restarted if they crash
  - Separates the services into **targets**

License



The work titled ”LPIC-1 101-400 – Lesson 10” by Theodotos Andreou is distributed with the Creative Commons Attribution ShareAlike 4.0 International License.