Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Linux Boot Process and Run Levels/62d98b769620a53a7c187531

TUTORIAL

Linux Boot Process and Run Levels

Topics

Linux Boot Process

The Linux boot process is the procedure for initializing the system. It consists of everything that happens from when the computer power is first switched on until the user interface is fully operational. Having a good understanding of the steps in the boot process may help you with troubleshooting problems, as well as with tailoring the computer's performance to your needs. On the other hand, the boot process can be rather technical, and you can start using Linux without knowing all the details.

```
Basic Boot Sequence is:     BIOS > MBR > GRUB > KERNEL > INIT > RUNLEVEL
```

The steps of boot process are:

1. Power On the computer.
2. Basic Input/Output System (BIOS) initializes the hardware, including the screen and keyboard, and tests the main memory. This process is also called POST (Power On Self Test).
3. The boot loader is usually stored on one of the hard disks in the system, either in the boot sector (for traditional BIOS/MBR

systems) or the EFI partition (for more recent (Unified) Extensible Firmware Interface or EFI/UEFI systems). Up to this stage, the machine does not access any mass storage media. Thereafter, information on date, time, and the most important peripherals are loaded from the CMOS values (after a technology used for the battery-powered memory store which allows the system to keep track of the date and time even when it is powered off). A number of boot loaders exist for Linux; the most common ones are GRUB (for GRand Unified Boot loader), ISOLINUX (for booting from removable media), and DAS U-Boot (for booting on embedded devices/appliances). Most Linux boot loaders can present a user interface for choosing alternative options for booting Linux, and even other operating systems that might be installed. When booting Linux, the boot loader is responsible for loading the kernel image and the initial RAM disk or filesystem (which contains some critical files and device drivers needed to start the system) into memory.

4. The boot loader has two distinct stages: For systems using the BIOS/MBR method, the boot loader resides at the first sector of the hard disk, also known as the Master Boot Record (MBR). The size of the MBR is just 512 bytes. In this stage, the boot loader examines the partition table and finds a bootable partition. Once it finds a bootable partition, it then searches for the second stage boot loader, for example GRUB, and loads it into RAM (Random Access Memory). For systems using the EFI/UEFI method, UEFI firmware reads its Boot Manager data to determine which UEFI application is to be launched and from where (i.e. from which disk and partition the EFI partition can be found). The firmware then launches the UEFI application, for example GRUB, as defined in the boot entry in the firmware's boot manager. This procedure is more complicated, but more versatile than the older MBR methods. The second stage boot loader resides under /boot. A splash screen is displayed, which allows us to choose which operating system (OS) to boot. After choosing the OS, the boot loader loads the kernel of the selected operating system into RAM and passes control to it. The boot loader loads the selected kernel image and passes control to it. Kernels are almost always compressed, so its first job is to uncompress itself. After this, it will check and analyze the system hardware and initialize any hardware device drivers built into the kernel.

5. The initramfs filesystem image contains programs and binary files that perform all actions needed to mount the proper root filesystem, like providing kernel functionality for the needed filesystem and device drivers for mass storage controllers with a facility called udev (for user device), which is responsible for figuring out which devices are present, locating the device drivers they need to operate properly, and loading them. After the root filesystem has been found, it is checked for errors and mounted. The mount program instructs the operating system that a filesystem is ready for use, and associates it with a particular point in the overall hierarchy of the filesystem (the mount point). If this is successful, the initramfs is cleared from RAM and the init program on the root filesystem (/sbin/init) is executed. init handles the mounting and pivoting over to the final real root filesystem. If special hardware drivers are needed before the mass storage can be accessed, they must be in the initramfs image.
6. Near the end of the boot process, init starts a number of text-mode login prompts. These enable you to type your username, followed by your password, and to eventually get a command shell. However, if you are running a system with a graphical login interface, you will not see these at first.
7. Usually, the default command shell is bash (the GNU Bourne Again Shell), but there are a number of other advanced command shells available. The shell prints a text prompt, indicating it is ready to accept commands; after the user types the command and presses Enter, the command is executed, and another prompt is displayed after the command is done.
8. The boot loader loads both the kernel and an initial RAM–based file system (initramfs) into memory, so it can be used directly by the kernel. When the kernel is loaded in RAM, it immediately initializes and configures the computer's memory and also configures all the hardware attached to the system. This includes all processors, I/O subsystems, storage devices, etc. The kernel also loads some necessary user space applications.
9. Once the kernel has set up all its hardware and mounted the root filesystem, the kernel runs /sbin/init. This then becomes the initial process, which then starts other processes to get the system running. Most other processes on the system trace their origin ultimately to init; exceptions include the so-called kernel processes.

These are started by the kernel directly, and their job is to manage internal operating system details.

Run Levels in Linux

A run level is a state of init and the whole system that defines what system services are operating. Run levels are identified by numbers. Some system administrators use run levels to define which subsystems are working, e.g., whether X is running, whether the network is operational, and so on.

Whenever a LINUX system boots, firstly the **init** process is started which is actually responsible for running other start scripts which mainly involves initialization of you hardware, bringing up the network, starting the graphical interface.

Now, the **init** first finds the default **runlevel** of the system so that it could run the start scripts corresponding to the default run level.
A **runlevel** can simply be thought of as the state your system enters like if a system is in a single-user mode it will have a **runlevel 1** while if the system is in a multi-user mode it will have a **runlevel 5**.

A **runlevel** in other words can be defined as a **pre-set single digit integer** for defining the operating state of your LINUX or UNIX-based operating system. Each runlevel designates a different system configuration and allows access to different combination of **processes**.

The important thing to note here is that there are differences in the runlevels according to the operating system. The standard **LINUX kernel** supports these seven different runlevels :

· 0 – System halt *i.e* the system can be safely powered off with no activity.

· 1 – Single user mode.

· 2 – Multiple user mode with no NFS(network file system).

· 3 – Multiple user mode under the command line interface and not under the graphical user interface.

· 4 – User-definable.

· 5 – Multiple user mode under GUI (graphical user interface) and this is the standard runlevel for most of the LINUX based systems.

· 6 – Reboot which is used to restart the system.

By default most of the LINUX based system boots to runlevel 3 or runlevel 5.

In addition to the standard runlevels, users can modify the preset runlevels or even create new ones according to the requirement. Runlevels 2 and 4 are used for user defined runlevels and runlevel 0 and 6 are used for halting and rebooting the system.

Obviously the start scripts for each run level will be different performing different tasks. These start scripts corresponding to each run level can be found in special files present under **rc sub directories**. At **/etc/rc.d** directory there will be either a set of files named **rc.0, rc.1, rc.2, rc.3, rc.4, rc.5** and **rc.6,** or a set of directories named **rc0.d, rc1.d, rc2.d, rc3.d, rc4.d, rc5.d** and **rc6.d**.

For example, run level 1 will have its start script either in file /etc/rc.d/rc.1 or any files in the directory /etc/rc.d/rc1.d.

Changing runlevel

init is the program responsible for altering the run level which can be called using **telinit** command.

For example, to change a runlevel from 3 to runlevel 5 which will actually allow the GUI to be started in multi-user mode the **telinit** command can be used as :

/*using telinit to change

runlevel from 3 to 5*/

```
telinit 5
```

The changing of runlevels is a task for the super user and not the normal user that's why it is necessary to be logged in as super user for the successful execution of the above telinit command or you can use **sudo** command as :

```
// using sudo to execute telinit
sudo telinit 5
```

The default runlevel for a system is specified in **/etc/initab** file which will have an entry **id : 5 : initdefault** if the default runlevel is set to 5 or will have an entry **id : 3 : initdefault** if the default runlevel is set to 3.

Need for changing the runlevel

·There can be a situation when you may find trouble in **logging in** in case you don't remember the password or because of the corrupted **/etc/passwd** file (have all the user names and passwords), in this case the problem can be solved by booting into a single user mode *i.e.* runlevel 1.

· You can easily halt the system by changing the runlevel to 0 by using **telinit 0**.

For permanently changing the run level just modify /etc/inittab file by following commands

```
# vi /etc/inittab
```

just change **id : 5 : initdefault** to corresponding runlevel eg 3 instead of 5 for permanently booting your Linux machine in command prompt.

Method-1: Changing run level temporarily without reboot.

We can use init command to change runlevels without rebooting the system.

Example:- if we are currently in run level 3 and want to go to run level 1, just we need to execute

```
# init 1
```

Or if you want to shutdown a machine you can take help of run level '0'. Just you need to execute

```
#init 0
```

Remember this change is not permanent and on next reboot you will get your default runlevel.

Method-2: Changing run level permanently

If you want to change your default run level then

Open the file /etc/inittab and edit entry initdefault:

```
# vi /etc/inittab
```

Let us set initdefault to 5, so that you can boot to X next time when Linux comes up:

```
id:5:initdefault:
```

Method-3:-Change run level at boot time

You can also change the run level at boot time. If your system uses LILO as the boot manager, you can append the run level to the boot command:

```
LILO: linux 3 or
LILO: linux 5
```

If your system uses GRUB, you can change the boot runlevel by pressing the `e' key to edit the boot configuration. Append the run level (in our case 5) to the end of the boot command as shown:

```
kernel /vmlinuz-2.6.18-164.el5 ro root=LABEL=/ rhgb quiet 5
```