LINOXIDE
Linux Freedom Thoughts

Author : Raghu Sharma
Website: www.linoxide.com

Linux Boot Process

Stages of Boot Sequence

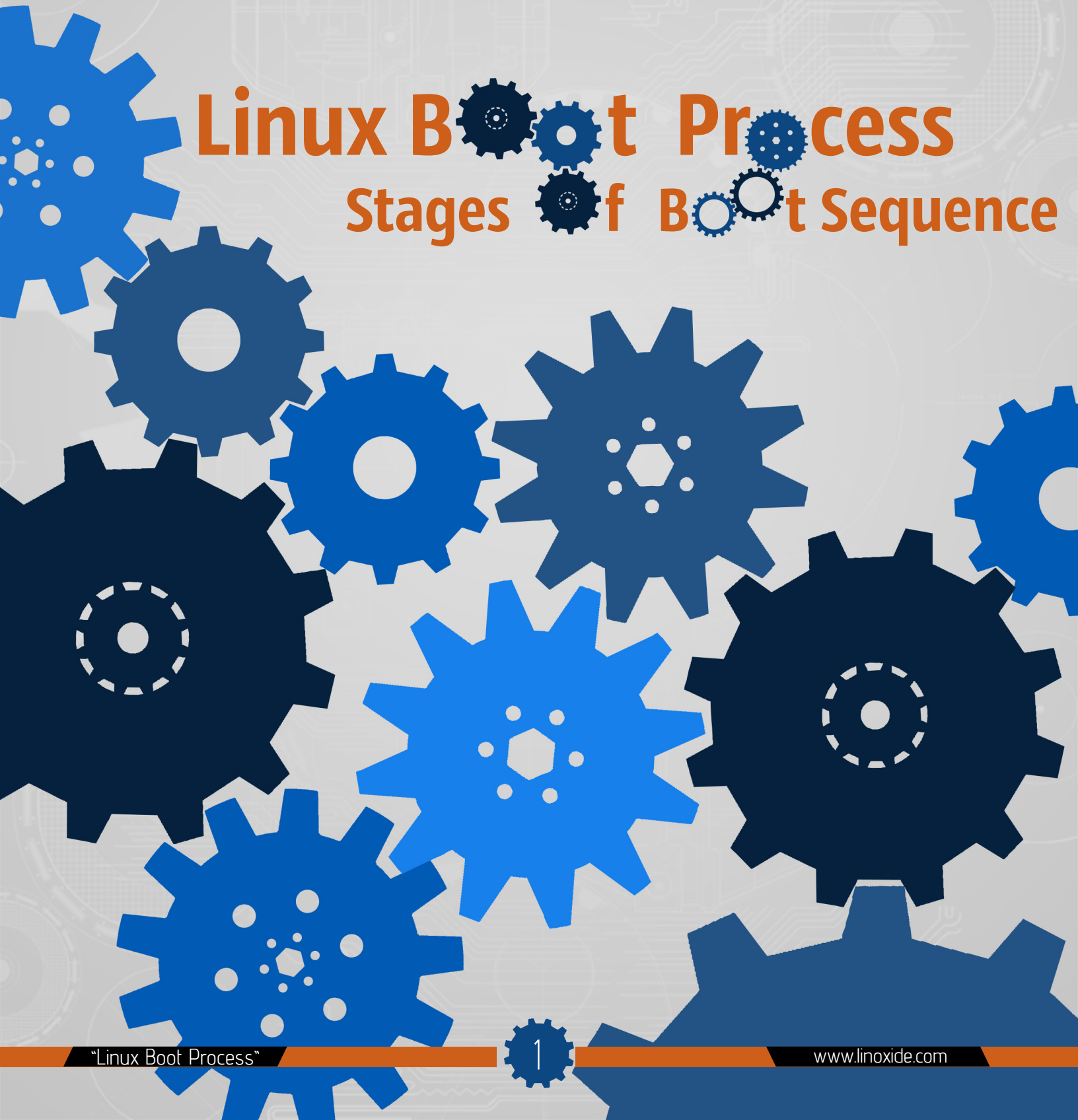

Table of Contents

The Linux Boot Process

Introduction

This article describes linux booting process in detail, what are the steps involved, which scripts run, what configuration files are read and their order, from turning on the system till getting the login prompt. Although this article projects a general view of booting a Linux system, but some configuration files and commands can be Red Hat specific.

1. BIOS Initialization

BIOS, the Basic Input Output System is a firmware program that performs a very basic level of interaction with hardware. This is the first program that takes control when the computer is powered on.The BIOS performs a test on all the hardware component and peripherals called POST, the "Power On Self Test". It initializes required hardware for booting.

After POST executes successfully, BIOS looks for a boot device from a list of devices. Modern BIOSs allow you to configure this order of devices (sometimes called boot preference) that BIOS checks for booting. These boot devices can be any one of floppy drive, CDROM, hard drive, a network interface or other removable media (such as USB flash drive).

The BIOS checks for the boot sector on the bootable device. Boot sector is the first physical sector on the storage device, and contains the code required for booting the machine.

2. The Master Boot Record

In case of hard disks and many other mass storage media, the boot sector is MBR. MBR consists of 512 bytes at the first sector of the hard disk. It is important to note that MBR is not located inside any partition. MBR precedes the first partition. The layout of MBR is as follows:

- First 446 bytes contain bootable code.
- Next 64 bytes contain partition information for 4 partitions (16×4). That is why the hard disks can have only 4 primary partitions, as the MBR can store information for 4 partitions only. So if you need more than 4 partitions on the hard disk, one of the primary partition has to be made extended, and out of this extended partition, logical partitions are created.

• Last 2 bytes are for MBR signature, also called magic number. (Thus total of 446 + 64 + 2 = 512 bytes).

================	======	======	======	======	===
446	16	16	16	16	2
================	======	======	======	======	===

The first 446 bytes of MBR contain the code that locates the partition to boot from. The rest of booting process takes place from that partition. This partition contains a software program for booting the system called the 'bootloader'.

3. About GRUB

According to gnu.org, "A bootloader is the first software program that runs when a computer starts". GRUB or GRand Unified Bootloader is the bootloader program for Linux like operating systems. There are mainly two versions of Grub is available (Grub version 1 and 2). Now most linux ditros started using grub version 2. One main feature of grub is that it can be installed using linux image and no need for running operating system.

Grub is a multi-stage bootloader (Stage1 ,Stage 1.5 and Stage 2). 3 stages for grub version 1 and grub version 2 is explained below.

Grub Version 1 stages

Stage 1 can load Stage 2 directly, but it is 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. If this space is not available (Unusual partition table, special disk drivers, GPT or LVM disk) the install of Stage 1.5 will fail. The stage 1.5 image contains file system drivers. This enables stage 1.5 to directly load stage 2 from any known location in the filesystem, for example from /boot/grub. Stage 2 will then load the default configuration file and any other modules needed. - See more at: http://linoxide.com/booting/boot-process-of-linux-in-detail/#sthash.0N56z4pF.dpuf

Grub Version 2 stages

Stage 1 -> boot.img is stored in the MBR (or optionally in any of the volume boot records) and addresses the next Stage by an LBA48 address (the 1024 cylinder boundary of GRUB legacy is omitted); 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 parti-

tion, when this sectors are free and available. For legacy reasons, the first partition of a hard-disc does not begin at sector 1 (counting begins with 0) but at sector 63 leaving a gap of 63 sectors of empty space, that is not part of any partition of file system and therefor 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 stage 2.

Refer at bottom of this page for boot process using grub version 2

Stage 2 -> files belonging to stage 2 are all being held in the /boot/grub-directory, which is a sub-directory of /boot/ directory.

Grub Boot Stage Errors

Grub Stage 1 Errors

Hard Disk Error
Floppy Error
Read Error
Geom Error

Grub Stage 1.5 Errors

The Stage 1.5 handles errors is to print an error number in the form "Error: "

Grub Stage 2 Errors

Selected item won't fit into memory
Selected disk doesn't exist
Disk read error
Disk write error"
Disk geometry error
Attempt to access block outside pa
No such partition
Bad filename (must be absolute pathname or blocklist)
Bad file or directory type
File not found
Inconsistent filesystem structure
Filesystem compatibility error, can\'t read whole file

```
Error while parsing number
Device string unrecognizable
Invalid device requested
Loading below 1MB is not supported
Unsupported Multiboot features requested
Unknown boot failure
Must load Multiboot kernel before modules
Must load Linux kernel before initrd
Cannot boot without kernel loaded
Unrecognized command
Bad or incompatible header on compressed file
Bad or corrupt data while decompressing file
Bad or corrupt version of stage1/stage2
```

Here is a sample grub.conf:

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE: You have a /boot partition. This means that
# all kernel and initrd paths are relative to /boot/, eg.
# root (hd0,0)
# kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00
# initrd /initrd-version.img #boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu title Red Hat Enterprise Linux Server (2.6.18-238.el5)
root (hd0,0)
kernel /vmlinuz-2.6.18-238.el5 ro root=/dev/VolGroup00/LogVol00
initrd /initrd-2.6.18-238.el5.img
title Windows XP Pro
rootnoverify (hd0,1)
chainload +1
```

GRUB allows you to choose from a list of available operating systems. The different OS's have different title in grub.conf. Corresponding to this configuration file, the user at GRUB menu will be asked to choose from two options: Red Hat Enterprise Linux Server (2.6.18-238.el5), and Windows XP Pro. The commands following the title are the commands that run in the background when an entry is selected from the GRUB menu. For example, when the user selects first option of Red Hat Enterprise Linux, following three commands execute:

```
root (hd0,0)
kernel /vmlinuz-2.6.18-238.el5 ro root=/dev/VolGroup00/LogVol00
initrd /initrd-2.6.18-238.el5.img
```

Corresponding to this menu entry, the first command, i.e. "root (hd0,0)" specifies the partition on which a compressed kernel image and initrd file are located.

The second command i.e. "kernel /vmlinuz-2.6.18-238.el5 ro root=/dev/VolGroup00/LogVol00" tells which kernel image to use (in this case vmlinuz-2.6.18-238.el5). The arguments to this command are 'ro' and 'root'. 'root' specifies the device on which root directory of the filesystem (i.e. / directory) is located; 'ro' means that this partition is to be mounted in read only mode (i.e. the kernel mounts the root partition in read only mode). Note that the partition for root filesystem and the partition on which this kernel image resides (i.e. boot partition) are different.

The third command is the location of initrd. Before going into the details of what initrd is, let's look at a problem caused at the boot time.

The Chicken / Egg Module Problem and initrd

The kernel needs to mount the root filesystem (as specified in the second command above). This filesystem may be on some partition with one of the following capabilities:

- Logical Volume Management
- Software RAID
- NFS (Network File System)
- Some other encrypted partition

The Linux kernel does not have these features compiled into it. But they are present as modules. So the kernel needs to load these modules in order to mount the root filesystem. These modules are present under /lib/modules/ directory, which is present on the root filesystem itself. But this root filesystem is not mounted yet (that is what we have been trying to do so far). So how can kernel access the modules for mounting a filesystem which are present on the filesystem itself (without mounting it)?

The kernel and GRUB provide a solution through initrd, the initial RAM disk. It contains the modules needed to mount the filesystem, and only modules needed for that filesystem are included.

GRUB also supports chainloading, the method used to pass control to other bootoader. Chainloading is used by GRUB to boot operating systems like windows. This can be checked in the above configuration file, under the title Windows XP Pro.

4. Kernel Initialization

After GRUB stage 2, the location of kernel and necessary modules through initrd are known. Now the kernel is loaded into memory and initialized. The initrd image is compiled and mounted into the memory. It serves as a temporary root file system and helps kernel to boot properly without mounting any root file system. Now that all the drivers are loaded into memory, and kernel has booted, kernel mounts the root filesystem in read only mode, and starts the first process.

5. The init Process

'init' is the first process started by kernel (initialization process). It is parent of all processes. The PID (Process ID) of init process is always 1. This process persists till the computer halts. It is responsible for the whole state of system. The settings for this process are stored in its configuration file, /etc/inittab (system initialization table).Before diving deeper into the details of this file and proceeding any further with the boot process, let's discuss about runlevels:

6. Runlevels

Runlevel is the state in which a system boots. It can boot in a single user mode, multiuser mode, with networking, and with graphics etc. Following are the default runlevels defined by Linux:

0: Halt or shutdown the system
1: Single user mode
2: Multi-user mode, without networking
3: Full multi user mode, with NFS (typical for servers)
4: Officially not defined; Unused
5: Full multi user with NFS and graphics (typical for desktops)
6: Reboot

There are some additional runlevels, which are rarely used:

s,S or single: Alternate single user mode
emergency: Bypass rc.sysinit (discussed later in this article)

Summarizing runlevels: Runlevel 1 is used for maintenance purpose, because this is very limited runlevels. Only the minimum scripts run in this runlevel. Only root user can log in. No other user can log into this runlevel. Runlevel 2 is more relaxed than runlevel 1. Here all the users can login, but networking service is not running. Runlevel 3 provides a full working environment. All users can log in, networking is enabled. Runlevel 4 is for experiment purpose only. In runlevel 5, graphical console is available. Runlevel 0 is the halt state of system, and switching to runlevel 6 will reboot the system.

Now that we are clear with runlevels, let's continue our discussion about init process and its configuration file, inittab. 'inittab' file for RHEL is:

```
#
# inittab This file describes how the INIT process should set up
# the system in a certain run-level.
#
# Author: Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
# Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
# 0 – halt (Do NOT set initdefault to this)
# 1 – Single user mode
# 2 – Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 – Full multiuser mode
# 4 – unused
# 5 – X11
# 6 – reboot (Do NOT set initdefault to this)
#
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

```
# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# When our UPS tells us power has failed, assume we have a few minutes
# of power left. Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"

# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

The general format of entries in this file is:

id:runlevel(s):action:process

"id" is the unique sequence of characters for identifying an entry in this file.
"runlevels" are the runlevels for which that entry is applicable.
"action" is the action that is to be taken for the runlevel.
"process" specifies the process that is to be executed.

Now let's look at some entries in this file. The first uncommented line, i.e. "id:3:initdefault:" specifies the default runlevel in which the system will boot. According to this file, the system will boot into runlevel 3 by default. One more important line is system initialization line (the one with id: si)

```
si::sysinit:/etc/rc.d/rc.sysinit
```

This line tells init process to execute "/etc/rc.d/rc.sysinit" script. This is the first script executed by init during booting process.

7. Script rc.sysinit

When this script executes, it asks the user for interactive setup (Press 'I' to enter interactive startup). This script performs a lot of functions for the system that include:

- Setting hostname
- It checks and mounts filesystems (/proc, /sys, and others in /etc/fstab).
- Enables SWAP
- Sets SELinux
- Configures kernel parameters from sysctl.conf
- Sets system clock
- Loads required modules
- Setting a minimal path
- Initializes required serial and other ports
- Enables RAID and LVM
- And the root filesystem, which was mounted read-only earlier (as specified in GRUB configuration file), is checked and remounted read-write.

After this script has executed, the runlevel specific files are run, according to "/etc/inittab". In our file, the default runlevel was 3. So the line corresponding to this runlevel is:

```
l3:3:wait:/etc/rc.d/rc 3
```

There are directories corresponding to each runlevel under "/etc/rc.d" directory. According to this line, the scripts in /etc/rc.d/rc3.d directory are run. Let's list the files in this directory. (Other directories like rc1.d, rc2.d ... have similar files in them).

```
[root@redhat-server ~]# ls -l /etc/rc.d/rc3.d/
total 304
lrwxrwxrwx 1 root root 17 Jul 3 03:51 K01dnsmasq -> ../init.d/dnsmasq
lrwxrwxrwx 1 root root 24 Jul 3 03:51 K02avahi-dnsconfd -> ../init.d/avahi-dnsconfd
lrwxrwxrwx 1 root root 24 Jul 3 03:54 K02NetworkManager -> ../init.d/NetworkManager
lrwxrwxrwx 1 root root 16 Jul 3 03:50 K05conman -> ../init.d/conman
lrwxrwxrwx 1 root root 19 Jul 3 03:51 K05saslauthd -> ../init.d/saslauthd
```

```
lrwxrwxrwx 1 root root 17 Jul 3 03:52 K05wdaemon -> ../init.d/wdaemon
lrwxrwxrwx 1 root root 16 Jul 3 03:50 K10psacct -> ../init.d/psacct
lrwxrwxrwx 1 root root 13 Jul 3 03:53 K20nfs -> ../init.d/nfs
lrwxrwxrwx 1 root root 14 Jul 3 03:53 K24irda -> ../init.d/irda
lrwxrwxrwx 1 root root 19 Jul 3 03:53 K35vncserver -> ../init.d/vncserver
lrwxrwxrwx 1 root root 17 Jul 3 03:56 K35winbind -> ../init.d/winbind
lrwxrwxrwx 1 root root 20 Jul 3 03:51 K50netconsole -> ../init.d/netconsole
lrwxrwxrwx 1 root root 16 Jul 5 10:10 K50vsftpd -> ../init.d/vsftpd
lrwxrwxrwx 1 root root 20 Jul 3 03:53 K69rpcsvcgssd -> ../init.d/rpcsvcgssd
lrwxrwxrwx 1 root root 16 Jul 3 03:56 K73ypbind -> ../init.d/ypbind
lrwxrwxrwx 1 root root 14 Jul 3 03:52 K74ipmi -> ../init.d/ipmi
lrwxrwxrwx 1 root root 14 Jul 3 03:49 K74nscd -> ../init.d/nscd
lrwxrwxrwx 1 root root 14 Jul 3 04:00 K74ntpd -> ../init.d/ntpd
lrwxrwxrwx 1 root root 15 Jul 3 03:49 K80kdump -> ../init.d/kdump
lrwxrwxrwx 1 root root 15 Jul 3 03:52 K85mdmpd -> ../init.d/mdmpd
lrwxrwxrwx 1 root root 20 Jul 3 03:49 K87multipathd -> ../init.d/multipathd
lrwxrwxrwx 1 root root 24 Jul 3 03:51 K88wpa_supplicant -> ../init.d/wpa_supplicant
lrwxrwxrwx 1 root root 14 Jul 3 03:52 K89dund -> ../init.d/dund
lrwxrwxrwx 1 root root 18 Jul 3 03:49 K89netplugd -> ../init.d/netplugd
lrwxrwxrwx 1 root root 14 Jul 3 03:52 K89pand -> ../init.d/pand
lrwxrwxrwx 1 root root 15 Jul 3 03:49 K89rdisc -> ../init.d/rdisc
lrwxrwxrwx 1 root root 14 Jul 3 03:53 K91capi -> ../init.d/capi
lrwxrwxrwx 1 root root 25 Jul 3 03:52 K99readahead_later -> ../init.d/readahead_later
lrwxrwxrwx 1 root root 23 Jul 3 03:52 S00microcode_ctl -> ../init.d/microcode_ctl
lrwxrwxrwx 1 root root 25 Jul 3 03:52 S04readahead_early -> ../init.d/readahead_early
lrwxrwxrwx 1 root root 15 Jul 3 03:54 S05kudzu -> ../init.d/kudzu
lrwxrwxrwx 1 root root 16 Jul 3 03:51 S07iscsid -> ../init.d/iscsid
lrwxrwxrwx 1 root root 19 Jul 3 03:49 S08ip6tables -> ../init.d/ip6tables
lrwxrwxrwx 1 root root 18 Jul 3 03:49 S08iptables -> ../init.d/iptables
lrwxrwxrwx 1 root root 18 Jul 3 03:51 S08mcstrans -> ../init.d/mcstrans
lrwxrwxrwx 1 root root 14 Jul 3 03:53 S09isdn -> ../init.d/isdn
lrwxrwxrwx 1 root root 17 Jul 3 03:51 S10network -> ../init.d/network
lrwxrwxrwx 1 root root 16 Jul 3 03:49 S11auditd -> ../init.d/auditd
lrwxrwxrwx 1 root root 21 Jul 3 03:51 S12restorecond -> ../init.d/restorecond
lrwxrwxrwx 1 root root 16 Jul 3 03:51 S12syslog -> ../init.d/syslog
lrwxrwxrwx 1 root root 18 Jul 3 03:49 S13cpuspeed -> ../init.d/cpuspeed
```

Some files in this directory start with S and others with K. The files starting with S correspond to the scripts that are to be 'started' in that particular runlevel, and the ones with K correspond to the ones that are to be 'killed'. These files are just soft links to scripts under "/

etc/rc.d/init/d" directory (One soft link points to "/etc/rc.local" which itself is a soft link to "/etc/rc.d/rc/local"). The scripts in "/etc/rc.d/init.d/" are daemons. Daemons are the processes that run in background and provide some kind of service, like http daemon (httpd) provides web service.

After all these scripts have been executed, then finally "/etc/rc.local" script runs. If there is some command or script that you want to be executed at system startup, you can put it in this script.

```
[root@redhat-server ~]# cat /etc/rc.local
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.

touch /var/lock/subsys/local
```

Now that all these scripts have been executed successfully, login prompt appears.

```
Red Hat Enterprise Linux Server release 5.6 (Tikanga)
Kernel 2.6.18-238.el5 on an i686

redhat-server login:
```

Bootloader Grub Version 2 Break Down Linux Boot Process Steps

1.Hardware Power Up , CPU getinto real mode and jumps to fixed location 0xFFFF0 ie hardwired in the CPU circuit.

2.BIOS program (ROM)is stored in that location will be executed

3.BIOS code will select the boot device (hard disk , CD ROM , Floppy , usb etc) that we have configured.

4.Once BIOS undestand the boot device , BIOS code will load the MBR to RAM. grub-install is the program that write data into MBR.

5.BIOS code jumps to the start address of the loaded MBR

6.The core.img from /boot/grub/core.img is loaded in memory. Once core.img has been loaded it can use search.mod to find all further /boot/grub files

7.The executed core.img code now initialises all the modules that are built into it (linked into core.img); one of these modules will be a filesystem driver capable of reading the filesystem on which directory /boot/grub lives.

8.You can link config file into core.img if you need booting thru pxe , remote nfs etc. Normal case its not required.

9.Core.img now loads file /boot/grub/normal.mod dynamically from disk , this module provides an interactive command-line.

10.Grub will load a file /boot/grub/grub.cfg which is a shell script and it will then configures the standard Grub boot menus.

11.In grub.cfg "linux /vmlinuz-linux root=/dev/sda3 ro" , "linux" module loads a kernel into memory