



Startup and shut down



What is bootstrapping?

- The process by which the system initializes until it is available to users
- During bootstrapping the kernel is loaded into memory and starts execution
- Any startup script is executed during this process
- The following are some of the causes that will prevent the bootstrapping process from completing successfully:
 - Error in one of the configuration files (like /etc/fstab)
 - Damaged hardware
 - Filesystem corruption



The bootstrap process

- When you turn on the server, the following happens sequentially:
 - Code saved in the ROM (Read Only Memory) gets executed. It determines where and how to start the system's kernel
 - When the kernel loads, it checks for the system's hardware, and starts the first and most important process, the *init*. The *init* process has always PID of 1.
 - Filesystems are checked for any corruption. The system attempts to fix any recoverable errors it may find.
 - After filesystems are checked and mounted, the system's startup scripts get executed by the *init* process.
 - Finally the login prompt/GUI interface appears and the system becomes ready for users



Kernel loading

- The kernel is just a program that gets loaded into the system's memory.
- It is loaded by a program called Boot Loader that gets executed from ROM
- Kernel loading cannot be controlled by Linux because it occurs outside its scope
- While loading, the kernel probes the attached hardware, and assigns an amount of RAM for itself. This part of memory will not be available to the user side
- On Linux systems, the kernel can be found under /boot. For example, /boot/vmlinuz-3.8.13-118.el6uek.x86_64. The exact filename of the kernel is vendor specific.




Kernel processes

- Once the kernel is loaded, it creates a number of startup processes. The most important one is the `init`, having the PID of 1.
- Other processes are created after that. They can be identified – later after the system completes booting – by having square brackets around them. For example `[kworker/0:1]`. The number after the slash / represents the processor number on which the process is running. You can query for such process by using `ps -ef | grep "\[.*\]"`
- Monitoring those kinds of processes you'll notice that they have low PID's compared to other *normal* processes
- They are not and cannot be treated as user processes. Only `init` can be dealt with as such. They are portions of the kernel that are made to look like normal processes while they aren't.



Startup daemons

- After the kernel and its processes have successfully been loaded, the startup daemons start to get executed by the init process
- The startup daemons are *normal* shell scripts that load important system services like sshd, ntpd...etc.
- Starting up specific daemons depends on the system's run level.



Stage 1 – the BIOS

- ▶ On proprietary systems like AIX, HP-UX and SPARC, the boot code is not stored in BIOS, it is stored in a much more powerful firmware. Such a firmware has enough information about the attached hardware and can talk to the network on a simple level.
- ▶ On PC's the BIOS (Basic Input/Output System) is responsible for executing the boot code. It is much simpler than the firmware of a proprietary machine.
- ▶ A typical PC will have several types of BIOS: one for the machine, another for the video card and a third for SCSI if it's available.
- ▶ BIOS configuration lets you choose the media from which you want the system to start, and the order by which it'll search for alternate media (or the network), should the first one fails
- ▶ Once selected, the boot media has it's first 512 bytes checked to determine which partition contains the *boot loader*.
- ▶ The boot loader is responsible for loading the kernel

Stage 2 - GRUB

- GRUB stands for Grand Unified Boot Loader. It is the default boot loader Linux and UNIX systems running on Intel processors
- It is responsible for loading the kernel, together with some options that can be tweaked by the system administrator
- It reads configuration options from `/boot/grub/menu.lst` (on Ubuntu and SUSE), and `/boot/grub/grub.conf` on Red Hat.
- The configuration file has the following options:
 - `default=number` indicates which operating system/kernel to boot the system to. The list starts at 0. When new kernel is installed, like through system upgrades, the old ones stay available for booting in the menu, so that you can choose to boot from them if the new one breaks the system
 - `timeout=number` sets the number of seconds the system would wait for a keyboard interruption before it loads the configured options
 - `root (hd0,0)` is where to find the partition from which to load the system. The first disk and the first partition on the machine are defined as 0 and 0 respectively
 - `kernel` loads the kernel from the specified path

GRUB command line interface

- ▶ It can be invoked by typing `c` in the GRUB boot screen
- ▶ It provides a basic command line interface that has some useful grub commands
- ▶ You can have a list of the possible commands by typing `TAB` twice
- ▶ One of the most useful cases of using GRUB CLI is to boot an OS that is not configured in `grub.conf` file. Assuming that this OS is installed on the first partition of the second disk on the machine, this can be done as follows:
 - ▶ `root (hd1,0)`
 - ▶ `kernel /vmlinuz-[select the kernel image you want] root=/dev/sdb2` [root specifies where the root (/) filesystem is located. sdb2 is disk 2, partition 2]
 - ▶ `boot`
- ▶ Other useful commands include `find`. It will search the system for regular files only (no directories). You can use it on a failing system to find, for example, on which partition the kernel is installed (`vmlinuz`), or where the root partition is (`/sbin/init`)

Booting a second OS

- With GRUB, you have the option to install more than one operating system on the same machine, and select which one to use on startup.
- If the second operating system you want to use is Microsoft Windows, it's wise to install it before Linux, because Windows deletes GRUB after installation.
- Some changes have to be made in GRUB to enable the system to boot into Windows:
 - `rootnoverify(hd0,0)` this forces GRUB not to try to mount the root partition
 - `chainloader +1` makes GRUB load the boot loader from the first sector of the selected disk (disk 1 partition 1)

Kernel command switches

- You can interrupt GRUB loading, while in the timeout period, by pressing any key. You can then select the kernel you wish to boot from, and press 'a' to start adding command switches.
- Examples of useful switches are:
 - `1` or `single` to enter single user mode
 - `n` where `n` is the run level to where you want the system to boot (covered later)
 - `init=/bin/bash` will let the kernel load `/bin/bash` instead of the `/sbin/init` process. Both of these options will allow you to access system as root without password
 - `root=/dev/sdxx` lets the system boot to a different root partition. Example:
`root=/dev/sdb1`
 - `ro` boots the system in read only mode. This is strongly recommended when you want to perform `fsck` on the disks before booting. `fsck` should never be done on a read/write system.
 - `debug` prints a more verbose output of the kernel loading status. It is useful when you want to troubleshoot a booting issue.
 - `selinux 0` or `1` disables/enables selinux module



The single user mode

- It is used to give the administrator the most basic functionality of a system, when it fails to boot
- In single user mode, multiple logins are disabled, only the administrator (root) can login
- This login is done physically on the machine and not through the network because networking is disabled in this mode.
- An administrator can choose to enter single user mode by editing the GRUB boot arguments, by using the shutdown command, or by using the telinit command
- You are allowed, in this mode, to open a shell as root without being prompted for a password. This is one way to reset the root password if lost or forgotten



Startup scripts

- They are the scripts that are run by the init process.
- They are normal shell scripts that do some “housekeeping” to the system like cleaning `/tmp` files, they also prepare the machine for use by doing tasks like setting the hostname, starting SSH daemon, and configuring the network interfaces.
- They are placed in `/etc/init.d` and symbolic links are made to them from respective, numbered directories like `/etc/rc0.d`.
- The init (or upstart) process ensures that the correct scripts are executed depending on the requested *run level* to which the system will boot.



System run levels

- A run level is a specific *state* of a UNIX/Linux system. They vary from one system to another, but the following are generally common among them:
 - 0 : halt, the system is shutdown
 - 1 : Single user mode
 - 2,3,4,5: networking is enabled
 - 6: system is rebooting
- In Linux, runlevel 3 is used for a non-GUI session, while runlevel 5 is used for a GUI-desktop session.
- The default run level of the system can be configured in `/etc/inittab` file.
- You can move from one runlevel to another, including shutdown and reboot, by using the `telinit` command followed by the desired run level.
- `Telinit -q` will force the system to re-read the `/etc/inittab` file

The init process and the /etc/rc*.d

- ▶ Startup scripts are used to both start and stop daemons. They can also be used to restart it (stop then start).
- ▶ The command is passed to the script as an argument. For example, `/etc/init.d/httpd start`
- ▶ They are placed in `/etc/init.d`. The symbolic links are located in `/etc/rc.*.d` where `*` indicates the run level at which the script should run.
- ▶ The init process looks the at rc.d directories to determine which scripts to run, depending on the configured run level.
- ▶ In an rc.d directory, the script name (symbolic link) starts with an S (start) or k (kill) followed by a number. For example `K35smb`.
- ▶ This way, init has a clear plan to which scripts to run and in what order. For example, when transitioning from run level 3 to run level 5, init runs all the scripts that start with S in `/etc/rc5.d`, in an ascending order, passing start as a command line argument to them.
- ▶ Numbers also ensure dependency between different services. For example, if httpd server daemon is started before the network service does, it will fail and you will have to manually restart it after the network is up.

Red Hat implementation

- In Red Hat systems, `init` uses `/etc/rc.d/rc` script, passing the run level as an argument
- `/etc/rc.d/rc` can run in “confirmation” mode, where it asks the user before starting each script. This is useful when troubleshooting a faulty system.
- The scripts place an empty lock file in `/var/lock/subsys`, with the same name as the daemon. Presence of this file means that the service is up and running. This file is deleted when the script is called with the stop argument.
- The `chkconfig` command is used to install, remove, and manage startup scripts. It has the following useful options:
 - `chkconfig --add script_name` will add the script to the configuration
 - `chkconfig --level script_name on | off` will enable/disable an added script in the specified run level.
 - `chkconfig --del script_name` will remove the script from the configuration
 - `chkconfig --list` will print all the configured startup scripts. Works best with `grep`.
- Red Hat also offers the `/etc/rc.d/rc.local` file, which is run after all other boot scripts have finished execution. You can use it to add custom startup scripts



LAB: creating a custom daemon with a startup script

- Write a script that prints the current date and to a log file every second
- Create a script that accepts start, stop, and restart command line arguments to start, stop, and restart the script we just created respectively
- Add the required comment `# chkconfig: 345 80 20`, where 345 is run levels 3, 4 and 5. 80 20 means that the start command will have the number 20 (S20), and 80 means that the kill command will have the number 80 (K80)
- Put this *launcher* script in `/etc/init.d` directory
- Use `chkconfig` to add this script to the respective rc directory
- Use `chkconfig` to enable the script
- Reboot the system to one of the configured run levels and ensure that the script has been launched



Shutting down and rebooting the system

- The shutdown command is the safest way to halt a system, or make it enter single user mode.
- This command has many different arguments, some of which are used to specify a grace period for the users to save their work and log out before the system is halted. You can also specify a friendly message explaining the reason for shut down and the estimated down time.
- Examples:
 - `shutdown -h time "message"` will shutdown the system at the specified time, sending the passed on message to the logged in users. The command can also be used to schedule the shutdown some minutes later, for example `shutdown +10 "message"` will shutdown the system after 10 minutes from now.
 - The `-h` forces the system to sync filesystems. That is, write any buffers in memory to the disk. The `-n` ignores this step. It is primarily used by the `fsck` command after repairing the root partition to ensure that the kernel does not overwrite the repaired superblocks with cached data.
- Rebooting the system is done by using the `-r` argument with the shutdown command. You can use all other arguments for reboot.