LINUX FOUNDATION

TRAINING

Managing the Startup Process and Related Services

Overview

Booting a Linux operating system involves a sequence of events to complete startup:

1. BIOS / POST
2. Bootloader
3. Kernel
4. Operating System

At the start of the sequence the system BIOS performs Power On Self Test (POST) to ensure bare metal system functionality eg: RAM is available. After passing POST, the BIOS works through its disk boot order seeking a MBR or EFI system partition to execute before aborting if none are found.

The boot loader is installed in the first 512 bytes (when using MBR) or the EFI system partition. The most widely used bootloader is the GRUB2 bootloader. GRUB2 provides a basic interface to select which Linux kernel to boot along with other configurations like rescue modes.

The bootloader starts the Linux kernel and system image with default parameters like boot disk and graphics options. You can change kernel parameters set by the bootloader to change how the system initialises, for example interrupting the process to enter single user mode (also known as rescue or runlevel 1), which is used for recovery tasks such as resetting passwords.

Once the Linux kernel has provided sufficient hardware access the operating systems first process (process id 1) is started with the responsibility of bringing online the services and processes that make the rest of the system

Key Ideas

Bootloader: The GRUB2 bootloader is loaded by the UEFI / BIOS executing the EFI System Partition on the system's hard disk. The BIOS executes each disk in its configured priority before aborting if no bootloader is found. The boot loader uses parameters to start the linux kernel.

System initialization: After the bootloader starts the kernel, the kernel starts the first process which initializes the rest of the system, and runs until the system is shut down. Linux system initialization for a long time was handled by the Unix-inspired SystemV init process, which ran scripts to start services in a defined and configurable order to reach a series of states, called runlevels.

The Upstart initialization system was developed as a replacement to SystemV that would trigger actions based on events, rather than running scripts in a particular order. Upstart is backwards compatible with SystemV runlevels, and can run SystemV init scripts. Upstart was used as the default system initialization in Ubuntu up until version 15.04,

The Systemd initialization system was developed as a more modular, performant, integrated alternative to Upstart. Systemd targets replace, and in most cases mimic the legacy sysvinit runlevel concept. Systemd can run SystemV init scripts. Systemd is used by CentOS 7 and OpenSuse 13.

Runlevels: Initialization brings the Linux system to specific target states.

SYSTEMV RUN LEVEL	SYSTEMD EQUIVA-LENT	DESCRIPTION
0 (HALT)	poweroff.target	Shuts down the system.
1 (SINGLE-USER MODE)	rescue.target	Mode for administrative and system rescue tasks. Only the root user can log in.
2 (MULTI-USER MODE)		All users can log in, but network interfaces aren't configured and networks services are not exported. Display manager is not started.
3 (MULTI-USER MODE WITH NETWORKING)	multi-user.target	Starts the system normally. Display manager is not started.
5 (START THE SYSTEM NOR-MALLY WITH APPROPRIATE DISPLAY MANAGER (WITH GUI))	graphical.target	Same as runlevel 3, but with a display manager.
6 (REBOOT)	reboot.target	Reboots the system.

Service Management: Once the Linux kernel has booted, it starts the first process, an initialization system (either SystemD or Upstart). Systemd and Upstart not only manage the startup process, but are also used to manage the services running on your system. That includes starting and stopping services, configuring them to persistently start at boot time, and shutting the system down.

Example Scenario

As a system administrator, sometimes a broken system may be need to be recovered. This example uses the rescue target to login as root.

Once the system has been recovered, check to see if the server is running an ssh daemon to allow access over the network. If your system uses Upstart, install the sysv-rc-conf package.

Now Do It

Rescue Mode:

1. Interrupt the automatic selection of the default boot options in the GRUB menu.
2. Edit the kernel arguments to make the system boot into rescue or single user mode.
3. Continue booting the system with your custom parameters.
4. Reboot the system

Service Management:

1. List the running services on your system.
2. Check to see if the ssh service (daemon) is running on your system.
3. Start, stop, and restart the ssh service.
4. Configure the ssh service to start automatically at boot time.
5. Shutdown the system.

If you remember nothing else...

Before debugging a faulty network service check to make sure the system is actually configured to start at boot using systemctl is-enabled <service name>. Single user systemctl list-unit-files can be used for a complete list.

Answer Key

Rescue Mode:

1. During system boot at the GRUB prompt press E
2. Find the kernel arguments line and append systemd.unit=rescue.target (systemd) or single (upstart).
3. Press Ctrl X to boot with custom parameters
4. The boot process completes.

Service Management:

1. List the running services on your system.

Systemd:

```
#systemctl
# systemctl | grep running
session-1.scope  loaded active running   Session 1 of user root
session-2.scope  loaded active running   Session 2 of user root
auditd.service   loaded active running   Security Auditing Service
...
```

Upstart:

```
$ initctl list | grep running
mountnfs-bootclean.sh start/running
rsyslog start/running, process 754
...
```

2. Check to see if the ssh service (daemon) is running on your system.

Systemd:

```
# # systemctl | grep ssh
sshd.service  loaded active running   OpenSSH server daemon
```

Upstart:

```
$ initctl list | grep ssh
ssh start/running, process 910
```

3. Start, stop, and restart the ssh service.

Systemd:

```
# systemctl start sshd
# systemctl stop sshd
# systemctl restart sshd
```

Upstart:

```
$ sudo start ssh
$ sudo stop ssh
$ sudo restart ssh
```

4. Configure the ssh service to start automatically at boot time.

Systemd:

```
# systemctl enable sshd
ln -s '/usr/lib/systemd/system/sshd.service' '/etc/systemd/system/multi-user.target.wants/sshd.
service'
# systemctl is-enabled sshd
enabled
```

Upstart:

```
$ sudo sysv-rc-conf sshd on
```

5. Shutdown the system.

Systemd:

```
# systemctl poweroff
```

Upstart:

```
# shutdown -h now
```

LINUX FOUNDATION

Get Certified!

Get more information at http://training.linuxfoundation.org/certification/lfcs