Chapter 39 init: SystemV, Upstart, systemd - Notes

39.3 Learning Objectives:

- Understand the importance of the **init** process
- Understand how **systemd** (and **Upstart**) arose and how they w ork.
- Use **systemctl** to configure and control **systemd**.
- Explain how the traditional **SysVinit** method w orks and how it incorporates **runlevels** and w hat happens in each one.
- Use **chkconfig** and **service** (and alternative utilities) to start and stop services or make them persistent across reboots w hen using **SysVinit**.

39.4 The init Process

`/sbin/init` (usually called **init**): first user-level process (or task) run on system, continues to run until system is shutdow n. Traditionally, has been considered **parent** of all user processes, although technically that is not true, as some processes are started directly by kernel.

init coordinates later stages of boot process, configures all aspects of the environment, and start processes needed for logging into the system. **init** also w orks closely w ith kernel in cleaning up after processes w hen they terminate.

Traditionally, nearly all distributions based **init** process on UNIX's venerable **SysVinit**. How ever, this scheme w as developed decades ago under rather different circumstances:

- Target w as multi-user mainframe systems (and not personal computers, laptops, and other devices)
- Target w as single processor system
- Startup (and shutdow n) time w as not an important matter; it w as far less important than getting things right.

Startup view ed as serial process, divided into series of sequential stages. Each stage required completion before the next could proceed. Thus, startup did not easily take advantage of parallel processing that could be done on multiple processors or cores.

Secondly, shutdow n/reboot w as seen as a relatively rare event, and exactly how long it took w as not considered important.

Modern systems have required new er methods w ith enhanced capabilities.

39.5 Startup Alternatives

To deal w ith intrinsic limitations in SysVinit, new methods of controlling system startup developed. While there are others, tw o main schemes adopted by Enterprise distributors:

1. **Upstart**: developed by Ubuntu, first included in 6.10 release in 2006, made default in 9.10 release in 2009. Also adopted in Fedora 9 (in 2008), in RHEL 6 and its clones (CentOS, Scienctific Linux, Oracle Linux), and in openSUSE w as offered as an option since version 11.3. Has also been used in various embedded and mobile devices.
2. **systemd**: more recent, Fedora w as fist major distribution to adopt in 2011. Standard since RHEL 7 and Ubuntu 16.04.

RHEL 7 based on systemd and every other major Linux distribution has adopted it, made it the default or announced plans to do so. Main systemd developers closely tied to Linux kernel community. Even Ubuntu phased out Upstart in its favor.

Migrations to new er schemes complicated, bugs and missing features can be very disabling, so there have been essential required compatibility layers. Thus, SysVinit utilities and methods w ill persist for long time, even if under the hood things are quite different.

History of all this and controversies quite complicated. Colorful personalities have ensured not all discussion is of technical nature In our context, w ill not look through this lens.

Next, going to concentrate on systemd and SysVinit w ith briefer section on Upstart. (even though RHEL 6 and some other distributions had used Upstart, has been thoroughly hidden behind compatibility layer using normal SysVinit utilities.)

39.6 systemd

systemd system and session manager for Linux rapidly taking root in all major distributions. Features include follow ing:

- Is compatible w ith SysVinit scripts
- Boots faster than previous systems
- Provides aggressive parallelization capabilities
- Uses socket and D-Bus activation for starting services
- Replaces shell scripts w ith programs
- Offers on-demand starting of daemons
- Keeps track of processes using cgroups
- Supports creating snapshots and restoring of the system state
- Maintains mount and automount points
- Implements an elaborate transactional dependency-based service control logic
- Can w ork as a drop-in replacement for SysVinit.

Instead of bash scripts, systemd uses `.service` files. In addition, systemd sorts all daemons into their ow n Linux cgroups (control groups).

systemd backw ard compatible w ith SysVinit and the concept of runlevels supported via runlevel **targets**. **telinit** program emulated to w ork w ith runlevels.

39.7 systemd Configuration Files

Although systemd prefers to use set of new standardized configuration files, can also use distribution-dependent legacy configuration files as fall-back.

Example of new configuration file: `/etc/hostname`, w hich w ould replace `/etc/sysconfig/network` in Red Hat, `/etc/hostname` in SUSE, and `/etc/hostname` (adopted as the standard) in Debian.

Other files might include:

- `/etcvconsole.conf`: default keyboard mapping and console font
- `/etc/sysctl.d/*.conf`: drop-in directory for kernel **sysctl** parameters
- `/etc/os-release` distribution ID file

systemd backw ard compatible w ith SysVinit, so using old commands w ill generally w ork. Supports the concept of runlevels, supported via runlevel *targets*, and **telimit** is emulated to w ork w ith runlevels.

39.8 systemctl

systemctl: main utility for managing services. Basic syntax:

```
$ systemctl [options] command [name]
```

Below, can see some examples of how you can use **systemctl**:

- To show the status of everything that systemd controls:

```
$ systemctl
```

- To show all available services:

```
$ systemctl list-units -t service --all
```

- To show only active services:

```
$ systemctl list-units -t service
```

- To start (activate) one or more units:

```
$ sudo systemctl start foo
$ sudo systemctl start foo.service
$ sudo systemctl start /path/to/foo.service
```

where a unit can be a service or a socket.

- To stop (deactivate) a service:

```
$ sudo systemctl stop foo.service
```

- To enable/disable a service:

```
$ sudo systemctl enable sshd.service
$ sudo systemctl disable sshd.service
```

Equivalent of `chkconfig on/off` and doesn't actually start the service.

Note: Some **systemctl** commands in the above examples can be run as non-root user, others require running as root or with **sudo**.

For an excellent summary fo how to go from **SysVinit** to **systemd**, see the SysVinit to Systemd Cheatsheet.

39.10 A Word About SysVinit

SysVinit was standard method for starting and shutting down systems, and for managing services for many years.

However, has been replaced on all major Linux distributions by systemd. Still value in covering how it works, as it is being replaced, as compatibility layers have been put into place to ensure the older methods can still be used. Furthermore, sometimes third party software has not been updated to use **systemctl** methods of systemd (a prominent example: VMWare, as of this writing).

Besides discussion of runlevels and other ingredients, **chkconfig** and **service** commands will also be explained. Eventually, will probably move to drop discussion of SysVinit, as w ell as Upstart.

36.11 SysVinit Runlevels

As a SysVinit system starts, passes through sequence of **runlevels** w hich define different system states; numbered from 0 to 6.

Runlevel 0 reserved for **system halt** state, runlevel 1 for **single-user mode**, and runlevel 6 for **system reboot**. Other runlevels used to define w hat services running for a normal system, and different distributions define them somew hat differently. Eg., on Red Hat-based systems, runlevel 2 defined as running system w ithout netw orking or **X**, runlevel 3 includes netw orking, and runlevel 5 includes netw orking and **X**. Table below summarizes SysVinit rundow n levels.

System Runlevels

Runlevel	Meaning
S, s	Same as 1
0	Shutdow n system and turn pow er off
1	Single User Mode
2	Multiple user, no NFS, only text login
3	Multiple user, w ith NFS and netw ork, only text login
4	Not used
5	Multiple user, w ith NFS and netw ork, graphical login w ith X
6	Reboot

Current runlevel can be simply displayed w ith **runlevel** command:

```
$ runlevel
N 5
```

w here first character is the previous level, `N` means unknow n.

telinit can be used to change runlevel of system. Eg., to go from runlevel 3 to runlevel 5, type:

```
$ sudo /sbin/telinit 5
```

39.12 SysVinit and /etc/inittab

When **init** process started, first thing it does is to read `/etc/inittab`. Historically, this file told **init** w hich scripts to run to bring system up each runlevel, and w as done w ith series of lines, one for each runlevel:

```
id:runlevel(s):action:process
```

w here:

- `id`: a unique 1-4 character identification for the entry
- `runlevel(s)`: zero or more single character or digit identifiers indicating w hich runlevel the action w ill be taken for
- `action`: describes the action to be taken
- `process`: specifies the process to be executed.

How ever, In more recent systems such as RHEL 6 w hich hide upstart behind compatibility layer, the only uncommented line and the only thing being set in this file is the default runlevel w ith the line:

```
id:5:initdefault
```

This is the level to stop at w hen booting the system. How ever, if another value specified on kernel command line, init ignores default. (This is done by simply appending right integer to kernel command line.) Default level is usually 5 for a full multi-user, netw orked graphical system, or 3 for a server w ithout a graphical interface.

Some recent systemd-based distributions (including RHEL 7) do not use this file at all; all lines in it are comments, but it is kept around to avoid breaking old scripts.

39.13 SysVinit Startup Scripts

Traditional method is to first run the `rc.sysinit` script, w hich performs numerous functions, such as starting LVM, mounting filesystems, etc. This script resides in `/etc` directory, but more likely you w ill find it in `/etc/rc.d` w ith symbolic link to `/etc`.

Next, `rc` script (in same directory) is run w ith desired runlevel as argument. This causes system to to to `rc.d/rc[0-6].d` directory and run all the scripts in there as in:

```
$ ls -lF /etc/rc.d/rc5.d
total 0
lrwxrwxrwx. 1 root root 14 Sep 3 10.05 K05pcmd -> ../init.d/pmcd*
lrwxrwxrwx. 1 root root 14 Sep 3 10.05 K05pmie -> ../init.d/pmie*
....
lrwxrwxrwx. 1 root root 14 Sep 3 10.05 S10network -> ../init.d/network*
lrwxrwxrwx. 1 root root 14 Sep 3 10.05 S19vmware -> ../init.d/vmware*
```

The `rc.local` script may be used to start system-specific applications.

Note:

- All the actual scripts are in `/etc/init.d`. Each runlevel directory just links back to them.
- **Start** scripts start w ith **S** in the name
- **Kill** scripts start w ith **K** in the name.

The existence or non-existence of a script's symbolic link in a runlevel directory determines w hether or not the script is executed at that runlevel.

The number follow ing the **K** or **S** in each script's name determines the order in w hich the scripts are invoked. the script name is also the name of the service.

Controlling w hich initialization scripts are run on entry to each runlevel involves managing the symbolic links. While it is possible to manage these links manually, there are utilities such as **chconfig** w hicha re designed to do this consistently and more easily.

39.14 chkconfig

chkconfig: used to query and configure w hat runlevels the various sytem services are to run in. Can see some **chkconfig** examples below :

- Check particular service to see if it is set up to run in the current runlevel:

```
$ chkconfig some_service
```

This w ill return true if the service is configured to be running, false otherw ise. Should now that even if it is configured to be running, might currently be stopped.

- See w hat services are configured to run in each of the runlevels:

```
$ chkconfig --list [service name]
```

- Turn on a certain service next time the system boots:

```
$ sudo chkconfig some_service on
```

- Do not turn a certain service on next time the system boots:

```
$ chkconfig some_service off
```

- You should not that the on and off do not affect the current state by starting or stopping a service. You w ould have to do that w ith:

```
$ sudo service some_service [stop | start]
```

Not difficult to add your ow n services and w rite your ow n startup scripts. One only has to place a script in `/etc/init.d` w hich has to have certain features in it (just some lines at the top!) and then use `chkconfig --add` to enable or `chkconfig --del` to disable use of the on and off instructions etc.

How does **chkconfig** actually determine w hich number should appear after the `S` or `K` in a symbolic link, and how does it know w hich runlevels to set on or off and w hat state to set the symbolic links in? The information is in the scripts themselves, w hich contain a line near the top like:

```
# chkconfig: 2345 10 90
```

The first argument after the **chkconfig:** is there to define w hich runlevels to have the service on by default. In the above example that means levels 2, 3, 4, and 5. The second and third numbers are the numerical prefixes in the start and stop scripts, so in the above they start w ith **S10** and **K90**.

39.15 service

Every operating system has **services** w hich are usually started on system initialization and often remain running until shutdow n. Such services may be started, stopped, or restarted at any time, generally requiring root privilege. On a Linux system using or emulating SysVinit, the services are those in the `.etc.init.d` directory.

You can see the current status of a particular service by doing:

```
$ sudo service network status
Configured devices:
lo eth0 eth1 eth2 wlan0
Currently active devices:
lo eth0

$ sudo service vsftpd status
vsftpd (pid 5284) is running...
```

service takes a number of options, w hich vary according to the particular service; for example:

```
$ sudo service network
Usage: /etc/init.d/network {start|stop|restart|reload|status}

$ sudo service iptables
Usage: /etc/init.d/iptables {start|stop|restart|condrestart|status|panic|save}
```

All **service** really does is change directory to `/etc/init.d` and run the appropriate script in that directory w ith the supplied options.

Can see the status of all the services on the system w ith:

```
$ sudo service --status-all
acpid (pid 4170) is running...
anacron (pid 4540) is running...
atd (pid 4553) is running...
....
smartd (pid 4614) is running...
smbd is stopped
......
```

Starting and stopping services w ith **service** is only effective during the current operation of the system; all changes are lost upon reboot. To cause a particular service to be turned on or not during system initialization, on Red Hat-based systems one uses **chkconfig** as described earlier.

39.16 chkconfig and service on Debian-based systems

On Debian-based systems, including Ubuntu, the utilities w ill w ork only if you have installed the **sysvinit-utils** and **chkconfig** packages, as in:

```
$ sudo apt install sysvinit-utils chkconfig
```

How ever, revent versions of Ubuntu no longer package **chkconfig**; w ill have to use **update-rc.d** utility about to be described.

As alternative, can use the more native commands on these systems. Eg., equivalent of **service** utility:

```
$ sudo invoke-rc.d cups [ status | start | stop ]
$ sudo status cups
```

to check or change the **cups** situation. **status** command is more rence and preferred over **invoke-rc.d**. Likew ise, equivalent of **chkconfig** w ould be:

```
$ sudo update-rc.d cups [ defaults | purge ]
$ sudo sysv-rc-conf cups [ on | off ]
```

Will have to consult **man** pages for full documentation.

39.17 Upstart

Upstart: **event-driven**, rather than being a set of serial precedures. Event notifications are sent to init process to tell it to execute certain commands at the right time after prerequisites have been fulfilled. Because Upstart is being superseded by systemd, won't spend as much time on it or do exercises with it. Upstart configuration files include:

```
/etc/init/rcS.conf
/etc/rc.sysinit
/etc/inittab
/etc/init/rc.conf
/etc/rc[0-5].d
/etc/init/start-ttys.conf
```

When kernel starts init process, this causes **`rcS.conf`** script to be executed. This, in turn, causes **`rc-sysinit.conf`** to be run.

`rc-sysinit.conf` will do a number of tasks, including starting the LVM, mounting filesystems, and then executing all of the runlevel scripts for the default runlevel specified in `/etc/inittab`.

This is accomplished by executing **`rc.conf`** and parring it the runlevel. These runlevel scripts bring up the services on the system. Finally, additional scripts such as **`prefdm.conf`** (for runlevel 5 only) are executed.

As mentioned previously, `/etc/inittab` is deprecated, and is not used only for setting up the default runlevel via the **`initdefault`** line. Other configuration is done via Upstart jobs in the `/etc/init` directory. Generally, Upstart events will be found in the `/etc/event.d` directory.

Eg., number of active **`tty`** consoles is now set by the **`ACTIVE_CONSOLES`** variable in `/etc/sysconfig/init` which is read by the `/etc/init/start-ttys.conf` job. Default value is **`ACTIVE_CONSOLES=/dev/tty[1-6]`**, which starts a getty process on **`tty1`** through **`tty6`**.

While Upstart was included in RHEL 6, RHEL 7 uses systemd. Ubuntu is currently the only major Linux distribution using Upstart, but has announced plans to move to systemd.

39.18 Upstart utilities

Using **initctl** you can view, start, stop jobs in much the same way that **service** does. Syntax is:

```
$ initctl options command
```

where **`options`** can have the following values:

- **start**: Starts a job
- **stop**: Stops a job
- **restart**: Restarts a job
- **reload**: Sends **HUP** signal to a job
- **status**: Queries status of a job
- **list**: Lists known jobs
- **emit**: Emits an event

Good summary of how to use **initctl** and many other features of Upstart can be found online: Upstart Intro, Cookbook & Best Practices

##

Back to top

Previous Chapter - Table of Contents - Next Chapter