**Bootup and Initialization**

Produced and directed by the Linux operating system

All about

- SysV initialization method *aka* sysvinit (venerable)
- systemd (new)

David Morgan

---

**General PC Booting sequence**

```
+-----------------+          +-----------------+
| motherboard's firmware code | → | a boot loader |
```

© David Morgan 2004-15
What happens “downstream”?  

What happens “downstream”?  

... it depends on the operating system

... and what if the OS is Linux?

/sbin/init gets executed hardcoded in the kernel
so with linux, "downstream" == /sbin/init

and what does /sbin/init do?

Whatever it's written to do, usually run some setup stuff then call a shell.

These /sbin/init's differ!
Difference: sysVinit vs systemd init

Linux booting sequence

- motherboard’s firmware code
- a boot loader
- kernel /sbin/init
- some intervening setup stuff
- shell
- system setup and service manager
A Play in Several Acts - sysvinit

Starring roles
- kernel
- init

Supporting roles
- rc.sysinit
- rc
- mingetty
- login
- shell

Cast of Thousands
- links in /etc/rc.d/rc?.d
  - to scripts in /etc/rc.d/init.d
  - to /etc/rc.d/rc.local

Members of the Cast

- Starring roles
  - The kernel
  - The init process
- Supporting roles
  - /etc/rc.d/rc.sysinit
  - /etc/rc
  - /sbin/mingetty
- Cast of thousands
  - links in /etc/rc.d/rc?.d
Starring role: Kernel

- Kernel code loads
  - From floppy with no filesystem
    - starting from first sector
  - From floppy or hard disk with filesystem
    - from a file, e.g., /boot/vmlinuz
- Identifies/initializes hardware
- Displays/stores messages
- Invokes init process

Starring role: init Process

- Father of all processes
  - init is to process structure as root is to file structure
  - always PID number “1”
- Creates other processes
- According to /etc/inittab
  - sysinit: system boot process(es)
  - initdefault: runlevels for process selection
  - wait: processes to run
  - respawn: processes to run and rerun
Runlevels

- Runlevel – a system configuration allowing only selected processes to exist

- 0  halt
- 1  single user mode
- 2  multiuser
- 3  full multiuser
- 4  unused
- 5  X11 (GUI)
- 6  reboot

Supporting role: rc.sysinit

- Starts swapping
- Sets hostname
- Mounts filesystems
- Loads modules
- Other initialization
- (rc = run commands)
Supporting role: rc

Loops thru
   all /etc/rc.d/rc?.d/K*
calls each with parameter “stop”
      - stops it if it’s running -

   all /etc/rc.d/rc?.d/S*
calls each with parameter “start”
      - runs it if it’s stopped -

Cast of 1000s: /etc/rc.d/rc?.d/S*

Directory full of symbolic link “pointers”

S01kerneld -> /etc/rc.d/init.d/kerneld start
S10network -> /etc/rc.d/init.d/network start
S15nfsfs -> /etc/rc.d/init.d/nfsfs start
S20random -> /etc/rc.d/init.d/random start
S30syslog -> /etc/rc.d/init.d/syslog start
S40atd -> /etc/rc.d/init.d/atd start
   ...
   ...
   ...

© David Morgan 2004-15
**rc script’s effect**

Runs scripts in init.d directory

```
/etc/rc.d/init.d/kerneld start  
/etc/rc.d/init.d/network start  
/etc/rc.d/init.d/nfsfs start
/etc/rc.d/init.d/random start  
/etc/rc.d/init.d/syslog start  
/etc/rc.d/init.d/atd start
```

**init.d script template**

```
# /etc/rc.d/init.d/xyz

# see how we were called
With “start” argument
   display “Starting xyz daemon”
   start the xyz daemon

With “stop” argument
   display “Stopping xyz daemon”
   kill the xyz daemon

(how to write an init script: /usr/share/doc/initscripts-[xxx]/sysvinitfiles)
```

© David Morgan 2004-15
crond – an example init script

the /etc/rc.d/init.d/crond script
if it runs:
launches the /usr/sbin/crond executable
crond is the daemon (program) that regularly checks a list of commands that have been scheduled to run and runs them when their time comes
for a system to have scheduled commands, it is sensible to make this program launch upon boot

Customization opportunity: rc.local

- machine-specific startup code
- put changes here, not rc.sysinit or rc
Post-bootup Initialization

- bootup is now complete
- getty is running
- further initialization: if someone logs in
  - getty calls login (/bin/login)
  - login calls “the shell” (/bin/sh = /bin/bash)

Supporting role: Getty

with friends init, login, and the shell
Shell Startup Files

- executed by shell when started by login
- `/etc/profile`, runs 1st
  - universal settings, all users
- `/home/username/.bash_profile`, runs 2nd
  - settings specific to user “username”

Turn services on/off manually

- starting
  - `/etc/rc.d/init.d/<script for service> start` or
  - `service <script for service> start`
- stopping
  - `/etc/rc.d/init.d/<script for service> stop` or
  - `service <script for service> stop`

- services re-read configuration files when restarted
  (restart one any time you change its config file)
Set services to boottime auto-on/off

- set it to turn on
  `chkconfig <script for service> on`
- set it to not turn on
  `chkconfig <script for service> off`

So much for sysvinit.

Now what about systemd?
**systemd**

- open source project by Lennart Pottering
- relationship to SysV init system
  - a drop-in replacement (and more)
  - coexists with sysvinit in Fedora 15 (hybrid/transitional)
  - [http://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet](http://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet)
- parallelizes the numerous boot activities
- documentation
  - author’s blog: [http://0pointer.de/blog](http://0pointer.de/blog)
  - see “14 May 2011” entry for links

---

**sysvinit replacement role**

### Process Identifier 1

On every Unix system there is one process with the special process identifier 1. It is started by the kernel before all other processes and is the parent process for all those other processes that have nobody else to be child of. Due to that it can do a lot of stuff that other processes cannot do. And it is also responsible for some things that other processes are not responsible for, such as bringing up and maintaining userspace during boot.

Historically on Linux the software acting as PID 1 was the venerable sysvinit package, though it had been showing its age for quite a while. Many replacements have been suggested, only one of them really took off: Upstart, which has by now found its way into all major distributions.

As mentioned, the central responsibility of an init system is to bring up userspace. And a good init system does that fast. Unfortunately, the traditional SysV init system was not particularly fast.

For a fast and efficient boot-up two things are crucial:
- To start less
- And to start more in parallel.

What does that mean? Starting less means starting fewer services or deferring the starting of services until they are actually needed. There are some services where we know that they will be required sooner or later (syslog, D-Bus system bus, etc.), but for many others this isn’t the case. For example, bluetoothd does not need to be running unless a Bluetooth device is actually plugged in or an application wants to talk to its D-Bus interfaces. Same for a printing system: unless the machine physically is connected to a printer, or an application wants to print something, there is no need to run a printing daemon such as CUPS. Avahi: if the machine is not connected to a network, there is no need to run Avahi, unless some application wants to use its APIs. And even SSH: as long as nobody wants to contact your machine there is no need to run it, as long as it is then started on the first connection. (And admit it, on most machines where sshd might be listening somebody connects to it only every other month or so.)

Starting more in parallel means that if we have to run something, we should not serialize its start-up (as sysvinit does), but run it all at the same time, so that the available CPU and disk IO bandwidth is maxed out, and hence the overall start-up time minimized.

*From Lennart Pottering [http://0pointer.de/blog/projects/systemd.html](http://0pointer.de/blog/projects/systemd.html)*
Default directories and files

/etc
  systemd
    system.conf
    graphical.target.wants
          default.target
    multi-user.target.wants

/lib
  systemd
    system
        xxx
        xxx
        xxx
        xxx

Many unit files

A "default target" unit file (as symlink)
and
a few "wants" subdirs containing symlinks
Centralized “systemctl” utility

- inspect and control state of systemd
- not to be confused with “sysctl” utility !!

Turn services on/off manually
(set “current” state)

- starting
  `systemctl start <service’s unit file>`
- stopping
  `systemctl stop <service’s unit file>`

- example: turn on/off the logging service
  `systemctl start rsyslog.service`
  `systemctl stop rsyslog.service`
Set services to boottime auto-on/off
(set “persistent” state)

- set it to turn on
  - `systemctl enable <service’s unit file>`
- set it to not turn on
  - `systemctl disable <service’s unit file>`

- example: turn on/off the logging service
  - `systemctl enable rsyslog.service`
  - `systemctl disable rsyslog.service`