NAME

boot-scripts - general description of boot sequence

DESCRIPTION

The boot sequence varies in details among systems but can be roughly divided to the following steps: (i) hardware boot, (ii) operating system (OS) loader, (iii) kernel startup, (iv) init and inittab, (v) boot scripts. We will describe each of these in more detail below.

Hardware-boot

After power-on or hard reset, control is given to a program stored on read-only memory (normally PROM). In PC we usually call this program the **BIOS**.

This program normally makes a basic self-test of the machine and accesses nonvolatile memory to read further parameters. This memory in the PC is battery-backed CMOS memory, so most people refer to it as the **CMOS**, although outside of the PC world, it is usually called **nvram** (nonvolatile ram).

The parameters stored in the nvram vary between systems, but as a minimum, the hardware boot program should know what is the boot device, or which devices to probe as possible boot devices.

Then the hardware boot stage accesses the boot device, loads the OS loader, which is located on a fixed position on the boot device, and transfers control to it.

Note: We do not cover here booting from network. Those who want to investigate this subject may want to research: DHCP, TFTP, PXE, Etherboot.

OS loader

In PC, the OS loader is located in the first sector of the boot device - this is the **MBR** (Master Boot Record).

In most systems, this primary loader is very limited due to various constraints. Even on non-PC systems there are some limitations to the size and complexity of this loader, but the size limitation of the PC MBR (512 bytes including the partition table) makes it almost impossible to squeeze a full OS loader into it.

Therefore, most operating systems make the primary loader call a secondary OS loader which may be located on a specified disk partition.

In Linux the OS loader is normally **lilo(8)** or **grub(8).** Both of them may install either as secondary loaders (where the DOS installed MBR points to them), or as a two part loader where they provide special MBR containing the bootstrap code to load the second part of the loader from the root partition.

The main job of the OS loader is to locate the kernel on the disk, load it and run it. Most OS loaders allow interactive use, to enable specification of alternative kernel (maybe a backup in case the last compiled one isn't functioning) and to pass optional parameters to the kernel.

Kernel startup

When the kernel is loaded, it initializes the devices (via their drivers), starts the swapper (it is a kernel process, called kswapd in modern Linux kernels), and mounts the root filesystem (/).

Some of the parameters that may be passed to the kernel relate to these activities (e.g: You can override the default root filesystem). For further information on Linux kernel parameters read bootparam(7).

Only then the kernel creates the first (user land) process which is numbered 1. This process executes the program */sbin/init*, passing any parameters that weren't handled by the kernel already.

init and inittab

When init starts it reads */etc/inittab* for further instructions. This file defines what should be run in different *run-levels*.

This gives the system administrator an easy management scheme, where each run-level is

associated with a set of services (e.g, **S** is *single-user*, on **2** most network services start). The administrator may change the current run-level via **init(8)** and query the current run-level via runlevel(8).

However, since it is not convenient to manage individual services by editing this file, inittab only bootstraps a set of scripts that actually start/stop the individual services.

Boot scripts

Note: The following description applies to System V release 4-based systems, which currently covers most commercial UNIX systems (Solaris, HP-UX, Irix, Tru64) as well as the major Linux distributions (Red Hat, Debian, Mandriva, SUSE, Ubuntu). Some systems (Slackware Linux, FreeBSD, OpenBSD) have a somewhat different scheme of boot scripts.

For each managed service (mail, nfs server, cron, etc.) there is a single startup script located in a specific directory (*/etc/init.d* in most versions of Linux). Each of these scripts accepts as a single argument the word start -- causing it to start the service, or the word stop -- causing it to stop the service. The script may optionally accept other convenience parameters (e.g: restart, to stop and then start, status to display the service status). Running the script without parameters displays the possible arguments.

Sequencing directories

To make specific scripts start/stop at specific run-levels and in specific order, there are *sequencing directories*. These are normally in*/etc/r c[0-6S].d*. In each of these directories there are links (usually symbolic) to the scripts in the */etc/init.d* directory.

A primary script (usually */etc/rc*) is called from **inittab(5)** and calls the services scripts via the links in the sequencing directories. All links with names that begin with S are being called with the argument start (thereby starting the service). All links with names that begin with K are being called with the argument stop (thereby stopping the service).

To define the starting or stopping order within the same run-level, the names of the links contain order-numbers. Also, to make the names clearer, they usually end with the name of the service they refer to. Example: the link */etc/rc2.d/S80sendmail* starts the sendmail service on runlevel 2. This happens after */etc/rc2.d/S12syslog* is run but before */etc/rc2.d/S90xfs* is run.

To manage the boot order and run-levels, we have to manage these links. However, on many versions of Linux, there are tools to help with this task (e.g: **chkconfig(8)).**

Boot configuration

Usually the daemons started may optionally receive command-line options and parameters. To allow system administrators to change these parameters without editing the boot scripts themselves, configuration files are used. These are located in a specific directory (*/etc/sysconfig* on Red Hat systems) and are used by the boot scripts.

In older UNIX systems, these files contained the actual command line options for the daemons, but in modern Linux systems (and also in HP-UX), these files just contain shell variables. The boot scripts in */etc/init.d* **source** the configuration files, and then use the variable values.

FILES

/etc/init.d/, */etc/rc[S0-6].d/*, */etc/sysconfig/*

SEE ALSO

inittab(5), bootparam(7), **init(8),** runlevel(8), shutdown(8)

COLOPHON

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at http://www.kernel.org/doc/man-pages/.