NAME

boot − System bootup process based on UNIX System V Release 4

DESCRIPTION

The **bootup process** (or "**boot sequence**") varies in details among systems, but can be roughly divided into phases controlled by the following components:

(1) hardware

(2) operating system (OS) loader

(3) kernel

(4) root user-space process (*init* and *inittab*)

(5) boot scripts

Each of these is described below in more detail.

Hardware

After power-on or hard reset, control is given to a program stored in read-only memory (normally PROM); for historical reasons involving the personal computer, this program is often called "the **BIOS**".

This program normally performs 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**"; outside of the PC world, it is usually called "the **NVRAM**" (nonvolatile RAM).

The parameters stored in the NVRAM vary among systems, but as a minimum, they should specify which device can supply an OS loader, or at least which devices may be probed for one; such a device is known as "the **boot device**". The hardware boot stage loads the OS loader from a fixed position on the boot device, and then transfers control to it.

Note: The device from which the OS loader is read may be attached via a network, in which case the details of booting are further specified by protocols such as DHCP, TFTP, PXE, Etherboot, etc.

OS loader

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

In a traditional PC, the OS loader is located in the initial 512-byte block of the boot device; this block is known as "the **MBR**" (Master Boot Record).

In most systems, the OS loader is very limited due to various constraints. Even on non-PC systems, there are some limitations on 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 much functionality into it.

Therefore, most systems split the role of loading the OS between a primary OS loader and a secondary OS loader; this secondary OS loader may be located within a larger portion of persistent storage, such as a disk partition.

In Linux, the OS loader is often either **lilo**(8) or **grub**(8).

Kernel

When the kernel is loaded, it initializes various components of the computer and operating system; each portion of software responsible for such a task is usually consider "a **driver**" for the applicable component. The kernel starts the virtual memory swapper (it is a kernel process, called "kswapd" in a modern Linux kernel), and mounts some filesystem at the root path, /.

Some of the parameters that may be passed to the kernel relate to these activities (for example, the default root filesystem can be overridden); for further information on Linux kernel parameters, read **bootparam**(7).

Only then does the kernel create the initial userland process, which is given the number 1 as its **PID** (process ID). Traditionally, this process executes the program */sbin/init*, to which are passed the parameters that haven't already been handled by the kernel.

Root user-space process

Note: The following description applies to an OS based on UNIX System V Release 4. However, a number of widely used systems have adopted a related but fundamentally different approach known as **systemd**(1), for which the bootup process is detailed in its associated **bootup**(7).

When */sbin/init* starts, it reads */etc/inittab* for further instructions. This file defines what should be run when the */sbin/init* program is instructed to enter a particular *run-level*, giving the administrator an easy way to establish an environment for some usage; each run-level is associated with a set of services (for example, run-level **S** is *single-user* mode, and run-level **2** entails running most network services).

The administrator may change the current run-level via **init**(1), and query the current run-level via **runlevel**(8).

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

Boot scripts

Note: The following description applies to an OS based on UNIX System V Release 4. However, a number of widely used systems (Slackware Linux, FreeBSD, OpenBSD) have a somewhat different scheme for 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, etc.). Running the script without parameters displays the possible arguments.

Sequencing directories

To make specific scripts start/stop at specific run-levels and in a specific order, there are *sequencing directories*, normally of the form */etc/rc[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); this primary script calls each service's script via a link in the relevant sequencing directory. Each link whose name begins with 'S' is called with the argument "start" (thereby starting the service). Each link whose name begins with 'K' is called with the argument "stop" (thereby stopping the service).

To define the starting or stopping order within the same run-level, the name of a link contains an **order-number**. Also, for clarity, the name of a link usually ends with the name of the service to which it refers. For 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 these links is to manage the boot order and run-levels; under many systems, there are tools to help with this task (e.g., **chkconfig**(8)).

Boot configuration

A program that provides a service is often called a "**daemon**". Usually, a daemon may receive various command-line options and parameters. To allow a system administrator to change these inputs without editing an entire boot script, some separate configuration file is used, and is located in a specific directory where an associated boot script may find it (*/etc/sysconfig* on older Red Hat systems).

In older UNIX systems, such a file contained the actual command line options for a daemon, but in modern Linux systems (and also in HP-UX), it just contains shell variables. A boot script in*/etc/init.d* reads and includes its configuration file (that is, it "**sources**" its configuration file) and then uses the variable values.

FILES

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

SEE ALSO

init(1), **systemd**(1), **inittab**(5), **bootparam**(7), **bootup**(7), **runlevel**(8), **shutdown**(8)