Booting Linux

Roland Czerny

November 24, 2021

Overview

- Firmware
  - BIOS
  - UEFI
- Boot loader
- Kernel
- `init` program
- Booting - Architecture details
- Secure Boot

The Big Picture

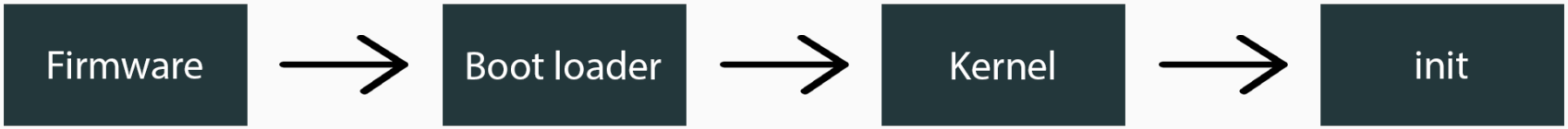


Figure 1: Boot steps

Firmware

Firmware

- Enumerate and initialize basic hardware
  - at least one CPU
  - basic memory
- Load boot code (Boot loader)
  - Firmware searches disk partitions for "stub" of boot loader
- Might even load small OS kernel (e.g. OpenBoot for SPARC)

BIOS

Basic Input/Output System

- Power-on self-test (POST)
- Search for bootable device
  - Master Boot Record (MBR): 512-byte block ending with boot signature 0x55AA
  - MBR contains partition table for 4 partitions
  - First 446 bytes hold boot loader stub: starts more capable boot loader within a partition
- Order specified by bus and controller scan order and BIOS configuration

BIOS

Limitations

- Earlier: stored on ROM, today: EEPROM
  - \+ Updates can fix bugs and add new feature
  - – Updates could brick or infect computer
- 16-bit real mode, 1MB addressable memory space
- assembly language programming

UEFI

Unified Extensible Firmware Interface

- Specification: software interface between OS and platform hardware
- Replaces BIOS: adds more functionality
- Initialize processor, memory, and peripheral hardware
- UEFI runs `.efi` applications from EFI system partition
  - Boot loader
  - UEFI shell
  - EFISTUB: use UEFI as bootloader for Linux

UEFI

Features

- Powerful pre-OS environment: GUI, multi language, network capability
- 32-bit or 64-bit, modular design, C language programming
- Legacy BIOS compatibility (CSM): boot from MBR-partitioned disks
- Secure boot
- Architecture support
  - Official: x86, x86-64, ARM (AArch32), ARM64 (AArch64)
  - Unofficial: POWERPC64, MIPS, RISC-V

Boot loader

Boot loader

BIOS-MBR

- GRUB "stub" in first 446-byte block of MBR (stage 1)
  - information on where to find /boot file system
  - stage 1.5 has information on how to read /boot file system
- Stage 2 bootloader does the complex work
  - e.g. GRUB2: /boot/grub2/i386–pc/kernel.img, configured by /boot/grub2/grub.cfg

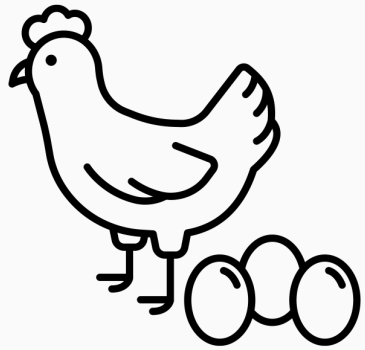
Boot loader

UEFI-GPT

- UEFI has found UEFI System Partition (FAT32 file system)
- example
  - firmware loads /EFI/BOOT/BOOTX64.EFI
  - /EFI/BOOT/BOOTX64.EFI loads /EFI/BOOT/GRUB.EFI
- typically EFI system is mounted is OS as /boot/efi

Initial RAM Disk Image

- Kernel needs to find root file system on storage device
  - might be on network storage or logical volume
  - might be encrypted
- Kernel needs modules to interact with device that stores the modules



Initial RAM Disk Image

Solution: Boot loader tells kernel how to find an initial RAM disk image

- contains kernel modules, binaries, scripts, ...
- `/init` script to find and mount real file system, so real `init` program can be run
- Boot loader passes `root=` directive to kernel
  - device name, label, UUID, ...

Boot loader

- Boot loader loads compressed kernel image
- Extract and decompress kernel into RAM and turn control over to it

Kernel

Kernel

Kernel startup

- Kernel initializes other CPU cores
- After kernel started `/init` script from Initial RAM Disk Image, the real `init` program is launched

Kernel: init /main.c

```
1 if (!run_init_process("/sbin/init") ||
2     !run_init_process("/etc/init") ||
3     !run_init_process("/bin/init") ||
4     !run_init_process("/bin/sh"))
5         return 0;
6
7 panic("No init found. Try passing init= option to
      kernel. See Linux Documentation/init.txt for
      guidance.");
```

init program

`init` program

- *Master* userspace program: control state of OS
- Ancestor of all processes: `PID 1`
- Manage all running processes: system services and user processes
- There are different `init` programs

BSD Unix init

- Uses one master boot script, which calls other scripts to start services
- One configuration script: enable/disable services

System V Unix

- Run Levels
  - Target states for running system
  - 0-6 *runlevels*: e.g. 5 is graphical login
  - /etc/inittab configuration, defines default
- /etc/rc.d/rc.sysinit does initialization tasks
- then it runs scripts in /etc/rc.d/rc5.d/ for the graphical target
- Scripts are stored in /etc/rc.d/init.d/: runlevel folders contain symlinks
- possible to switch from one level to another

Upstart

- Modification of System V method
- `inittab` just defines default
- Configuration is collection of files in `/etc/init`
- Automatically restart crashed service
- Events can trigger services

- Start only what's needed: e.g. CUPS print service or Bluetooth can be started when needed or hardware has been detected
- Aggressively parallels startup
  - Start daemons simultaneously
  - Dependencies are resolved recursively
  - IPC via sockets
- other features: auto-mounting, cgroups, ...

Units

- Booting tasks are organized into units
- Each unit contains configuration information
- Different types: *.mount, *.service , *.socket , *.path, *.target
- *.target
  - Define group of units
  - Define dependencies on other units
  - Analogous to run levels from System V and Upstart

SysV Runlevel	systemd Target	Notes
0	runlevel0.target, poweroff.target	Halt the system.
1, s, single	runlevel1.target, rescue.target	Single user mode.
2, 4	runlevel2.target, runlevel4.target, multi-user.target	User-defined/Site-specific runlevels. By default, identical to 3.
3	runlevel3.target, multi-user.target	Multi-user, non-graphical. Users can usually login via multiple consoles or via the network.
5	runlevel5.target, graphical.target	Multi-user, graphical. Usually has all the services of runlevel 3 plus a graphical login.
6	runlevel6.target, reboot.target	Reboot
emergency	emergency.target	Emergency shell

Figure 2: Comparing SysV Runlevel and systemd Targets (**aw21**)

- Default: /lib/systemd/system/default.target
- Typically symbolic link to multi-user.target or graphical.target
- Can be overwritten with parameter to kernel (e.g. for rescue.target)

```
❯ pwd
/lib/systemd/system
❯ l | grep default.target
lrwxrwxrwx    16 root 12 Nov 15:54  default.target -> graphical.target
❯ cat graphical.target
```

File: **graphical.target**

```
1   #  SPDX-License-Identifier: LGPL-2.1-or-later
2   #
3   #  This file is part of systemd.
4   #
5   #  systemd is free software; you can redistribute it and/or modify it
6   #  under the terms of the GNU Lesser General Public License as published by
7   #  the Free Software Foundation; either version 2.1 of the License, or
8   #  (at your option) any later version.
9
10  [Unit]
11  Description=Graphical Interface
12  Documentation=man:systemd.special(7)
13  Requires=multi-user.target
14  Wants=display-manager.service
15  Conflicts=rescue.service rescue.target
16  After=multi-user.target rescue.service rescue.target display-manager.service
17  AllowIsolate=yes
```

```
/lib/systemd/system
```

Figure 3: default . target —> graphical.target

```
❯ cat multi-user.target
       File: multi-user.target
   1   #  SPDX-License-Identifier: LGPL-2.1-or-later
   2   #
   3   #  This file is part of systemd.
   4   #
   5   #  systemd is free software; you can redistribute it and/or modify it
   6   #  under the terms of the GNU Lesser General Public License as published by
   7   #  the Free Software Foundation; either version 2.1 of the License, or
   8   #  (at your option) any later version.
   9
  10   [Unit]
  11   Description=Multi-User System
  12   Documentation=man:systemd.special(7)
  13   Requires=basic.target
  14   Conflicts=rescue.service rescue.target
  15   After=basic.target rescue.service rescue.target
  16   AllowIsolate=yes

❯ ls multi-user.target.wants
dbus.service  getty.target  systemd-ask-password-wall.path  systemd-logind.service  systemd-user-sessions.service
  /lib/systemd/system
```

Figure 4: multi—user.target

```
❯ cat basic.target
```

File: **basic.target**

```
1   #  SPDX-License-Identifier: LGPL-2.1-or-later
2   #
3   #  This file is part of systemd.
4   #
5   #  systemd is free software; you can redistribute it and/or modify it
6   #  under the terms of the GNU Lesser General Public License as published by
7   #  the Free Software Foundation; either version 2.1 of the License, or
8   #  (at your option) any later version.
9
10  [Unit]
11  Description=Basic System
12  Documentation=man:systemd.special(7)
13  Requires=sysinit.target
14  Wants=sockets.target timers.target paths.target slices.target
15  After=sysinit.target sockets.target paths.target slices.target tmp.mount
16
17  # We support /var, /tmp, /var/tmp, being on NFS, but we don't pull in
18  # remote-fs.target by default, hence pull them in explicitly here. Note that we
19  # require /var and /var/tmp, but only add a Wants= type dependency on /tmp, as
20  # we support that unit being masked, and this should not be considered an error.
21  RequiresMountsFor=/var /var/tmp
22  Wants=tmp.mount
```

Figure 5: basic . target

```
❯ cat sysinit.target
      File: sysinit.target
   1  #  SPDX-License-Identifier: LGPL-2.1-or-later
   2  #
   3  #  This file is part of systemd.
   4  #
   5  #  systemd is free software; you can redistribute it and/or modify it
   6  #  under the terms of the GNU Lesser General Public License as published by
   7  #  the Free Software Foundation; either version 2.1 of the License, or
   8  #  (at your option) any later version.
   9
  10  [Unit]
  11  Description=System Initialization
  12  Documentation=man:systemd.special(7)
  13  Conflicts=emergency.service emergency.target
  14  Wants=local-fs.target swap.target
  15  After=local-fs.target swap.target emergency.service emergency.target

❯ ls sysinit.target.wants
cryptsetup.target          lvm2-lvmpolld.socket              sys-kernel-debug.mount            systemd-firstboot.service             systemd-machine-id-commit.service  systemd-sysusers.service           systemd-update-done.service
dev-hugepages.mount        lvm2-monitor.service              sys-kernel-tracing.mount          systemd-hwdb-update.service           systemd-modules-load.service       systemd-tmpfiles-setup-dev.service  systemd-update-utmp.service
dev-mqueue.mount           proc-sys-fs-binfmt_misc.automount systemd-ask-password-console.path systemd-journal-catalog-update.service systemd-random-seed.service       systemd-tmpfiles-setup.service      veritysetup.target
kmod-static-nodes.service  sys-fs-fuse-connections.mount     systemd-binfmt.service            systemd-journal-flush.service         systemd-repart.service             systemd-udev-trigger.service
ldconfig.service           sys-kernel-config.mount           systemd-boot-system-token.service systemd-journald.service              systemd-sysctl.service             systemd-udevd.service
```

Figure 6: sysinit . target

OpenRC

- systemd is sometimes criticized of doing *too much*
  - journald , ...
- OpenRC is a lightweight alternative that focuses on being an init system
- Simple configuration, dependency based
- Native init system for Gentoo

Booting - Architecture details

Architecture details

- x86
  - FSBL: BIOS/UEFI
- ARM
  - FSBL: U-Boot, Trusted Firmware-A
  - Boot loader provides ARM tags (ATAG) to kernel
    - size and location of system memory, root file system location

Architecture details

RISC-V

- Privilege model
  - User Mode (U-mode), Supervisor Mode (S-mode), Hypervisor Mode (H-mode), Machine Mode (M-mode)
- Abstractions
  - Device tree
  - Supervisor Binary Interface (SBI): Interface between M-mode and S-mode
- FSBL
  - Provides OpenSBI (implementation of SBI)
  - e.g. Berkeley Bootloader (BBL) or EDK2 UEFI
- SSBL (e.g. U-Boot) is payload to OpenSBI

Secure Boot

UEFI Secure Boot

- Protect against malicious code before OS kernel is loaded
- Verify code loaded by UEFI
- Checksums and signatures
- Most x86 hardware pre-loaded with Microsoft keys
- Option to add extra signing keys
- Secure Boot is enabled by default on most modern systems

Thank you for your attention!

References i

[1] Bob Cromwell
How Linux Boots, Run Levels, and Service Control.
https://cromwell-intl.com/open-source/linux-boot.html.
Online; accessed 23.11.2021

[2] ArchWiki
systemd.
https://wiki.archlinux.org/title/Systemd.
Online; accessed 23.11.2021