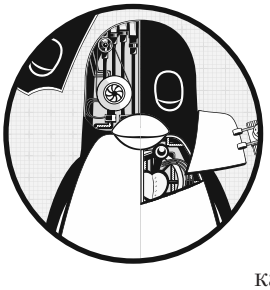
5

Загрузка ядра Linux



Ранее мы изучили физическую и логическую структуру системы Linux, говорили о том, что такое ядро и как работать с процессами. В этой главе рассмотрим, как запускается и *загружается* ядро. Другими словами, вы узнаете, как ядро перемещается в память и что оно делает до момента запуска первого пользовательского процесса.

Упрощенно процесс загрузки выглядит следующим образом.

1. BIOS или загрузочная программа компьютера загружается и запускает загрузчик.
2. Загрузчик находит образ ядра на диске, загружает его в память и запускает.
3. Ядро инициализирует устройства и их драйверы.
4. Ядро монтирует корневую файловую систему.
5. Ядро запускает программу под названием `init` с идентификатором процесса `1`. Эта точка является *началом пользовательского пространства*.
6. Программа `init` приводит в действие остальные системные процессы.
7. В какой-то момент запускается процесс, позволяющий пользователю войти в систему, обычно в конце или ближе к концу процесса загрузки.

В этой главе рассматриваются первые четыре этапа, основное внимание уделяется загрузчику и ядру. В главе 6 продолжается изучение загрузки пользовательского пространства: подробно описывается программа `systemd` — наиболее распространенная версия программы `init` в системах Linux.

Умение узнавать каждый этап процесса загрузки окажет вам неоценимую помощь при устранении проблем с загрузкой и понимании системы в целом. Однако поведение системы, заданное по умолчанию во многих дистрибутивах Linux, часто затрудняет (если не делает невозможной) идентификацию первых нескольких этапов загрузки по мере их выполнения, поэтому вы, вероятно, сможете хорошо изучить их только после того, как они завершатся и вы войдете в систему.

5.1. Сообщения при загрузке

Традиционные системы Unix при загрузке выдают множество диагностических сообщений, которые рассказывают вам о ее ходе. Сообщения поступают сначала от ядра, а затем от процессов и систем инициализации, которые постепенно запускаются. Однако они не последовательны, а в некоторых случаях даже не особо информативны. Кроме того, аппаратные улучшения привели к тому, что теперь ядро запускается намного быстрее, чем раньше, и сообщения мелькают так быстро, что бывает трудно понять, что происходит. В результате большинство современных дистрибутивов Linux делают все возможное, чтобы скрыть диагностику загрузки с помощью заставок и других форм заполнения экрана, чтобы отвлечь вас во время запуска системы.

Лучший вариант просмотреть диагностические сообщения о загрузке ядра и времени выполнения — открыть системный журнал ядра с помощью команды `journalctl`. При запуске `journalctl -k` отображаются сообщения текущей загрузки, а с помощью параметра `-b` можно просмотреть предыдущие загрузки. Мы рассмотрим этот журнал более подробно в главе 7.

Если у вас нет системы `systemd`, можете поискать файл журнала `/var/log/kern.log` или выполнить команду `dmesg` для просмотра сообщений в *кольцевом буфере ядра*.

Пример вывода команды `journalctl -k`:

```
microcode: microcode updated early to revision 0xd6, date = 2019-10-03
Linux version 4.15.0-112-generic (buildd@lcy01-amd64-027) (gcc version 7.5.0
(Ubuntu 7.5.0-3ubuntu1~18.04)) #113-Ubuntu SMP Thu Jul 9 23:41:39 UTC 2020
(Ubuntu 4.15.0-112.113-generic 4.15.18)
Command line: BOOT_IMAGE=/boot/vmlinuz-4.15.0-112-generic root=UUID=17f12d53-c3d7-
4ab3-943e-a0a72366c9fa ro quiet splash vt.handoff=1
KERNEL supported cpus:
--пропуск--
scsi 2:0:0:0: Direct-Access ATA KINGSTON SM2280S 01.R PQ: 0 ANSI: 5
sd 2:0:0:0: Attached scsi generic sg0 type 0
sd 2:0:0:0: [sda] 468862128 512-byte logical blocks: (240 GB/224 GiB)
sd 2:0:0:0: [sda] Write Protect is off
sd 2:0:0:0: [sda] Mode Sense: 00 3a 00 00
sd 2:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO
or FUA
sda: sda1 sda2 < sda5 >
sd 2:0:0:0: [sda] Attached SCSI disk
--пропуск--
```

Начинающийся после запуска ядра процесс запуска пользовательского пространства также генерирует сообщения. Просматривать их, вероятно, будет сложнее, потому что в большинстве систем вы не найдете их ни в одном файле журнала. Скрипты запуска спроектированы для отправки на консоль сообщений, которые удаляются после завершения загрузки. Однако в системах Linux это не проблема, поскольку программа `systemd` фиксирует диагностические сообщения, которые обычно поступают на консоль при запуске и во время выполнения.

5.2. Параметры инициализации и загрузки ядра

При запуске ядро Linux инициализируется в таком порядке:

1. Проверка процессора.
2. Проверка памяти.
3. Обнаружение шины устройства.
4. Обнаружение устройств.
5. Настройка вспомогательной подсистемы ядра (сеть и т. п.).
6. Монтирование корневой файловой системы.
7. Запуск пользовательского пространства.

Первые два шага не слишком примечательны, но когда ядро добирается до устройств, возникает вопрос о зависимостях. Например, драйверы дисковых устройств могут зависеть от поддержки шины и поддержки подсистемы SCSI, как говорилось в главе 3. Позже, в процессе инициализации, ядро должно смонтировать корневую файловую систему перед инициализацией.

В целом, вам не нужно беспокоиться о зависимостях, за исключением того, что некоторые необходимые компоненты могут быть загружаемыми модулями ядра, а не частью основного ядра. Некоторым системам может потребоваться загрузить эти модули ядра до того, как будет смонтирована основная корневая файловая система. Мы рассмотрим эту проблему и ее решения для начальной файловой системы оперативной памяти (`initrd`) в разделе 6.7.

Ядро выдает определенные виды сообщений, указывающих на то, что оно готовится запустить свой первый пользовательский процесс:

```
Freeing unused kernel memory: 2408K
Write protecting the kernel read-only data: 20480k
Freeing unused kernel memory: 2008K
Freeing unused kernel memory: 1892K
```

Здесь оно не только очищает часть незадействованной памяти, но и защищает собственные данные. Затем, если вы запускаете достаточно новое ядро, то увидите, что оно запускает первый процесс в пользовательском пространстве как `init`:

```
Run /init as init process
   with arguments:
    --пропуск-
```

После этого монтируется корневая файловая система и запускается systemd, отправляя несколько собственных сообщений в журнал ядра:

```
EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts: (null)
systemd[1]: systemd 237 running in system mode. (+PAM +AUDIT +SELINUX +IMA
+APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +XZ +LZ4
+SECCOMP +BLKID +ELFUTILS +KMOD -IDN2 +IDN -PCRE2 default-hierarchy=hybrid)
systemd[1]: Detected architecture x86-64.
systemd[1]: Set hostname to <duplex>.
```

На этом этапе пользовательское пространство запущено.

5.3. Параметры ядра

Когда ядро Linux запускается, оно получает набор текстовых *параметров ядра*, содержащих дополнительные сведения о системе. Параметры определяют множество различных характеристик поведения, таких как объем диагностических выходных данных, которые должно выдавать ядро, и параметры, зависящие от драйвера устройства.

Вы можете изучить параметры, переданные в активное ядро вашей системы, просмотрев файл `/proc/cmdline`:

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.15.0-43-generic root=UUID=17f12d53-c3d7-4ab3-943e
-a0a72366c9fa ro quiet splash vt.handoff=1
```

Параметры представляют собой либо простые однословные флаги, такие как `ro` и `quiet`, либо пары *`key=value`*, например `vt.handoff=1`. Многие параметры не особо значимы для работы системы (например, флаг `splash` используется для отображения экрана загрузки), однако `root` — один из важнейших. Он отвечает за расположение корневой файловой системы, без него ядро не сможет правильно запустить пользовательское пространство. Корневую файловую систему можно указать как файл устройства, например:

```
root=/dev/sda1
```

В большинстве современных систем существуют две более распространенные альтернативы вывода. Первый способ позволяет отобразить логический том подобно этому:

```
root=/dev/mapper/my-system-root
```

Второй — отображает UUID (см. подраздел 4.2.4):

```
root=UUID=17f12d53-c3d7-4ab3-943e-a0a72366c9fa
```

Оба варианта эффективны, поскольку не зависят от конкретного сопоставления устройств ядра.

Параметр `ro` указывает ядру смонтировать корневую файловую систему в режиме «только для чтения» при запуске пользовательского пространства. Так и должно быть: режим «только для чтения» гарантирует, что утилита `fsck` сможет безопасно проверить корневую файловую систему, прежде чем пытаться выполнить более серьезные задачи. После проверки процесс загрузки повторно подключает корневую файловую систему в режиме чтения/записи.

Обнаружив незнакомый параметр, ядро Linux сохраняет его. Позже оно передает параметр программе `init` при запуске пользовательского пространства. Например, если вы добавляете параметр `-s` к параметрам ядра, оно передает `init` указание, что та должна запускаться в однопользовательском режиме.

Если вас интересуют основные параметры загрузки, на странице руководства `bootparam(7)` приведен их обзор. Если вам нужен конкретный параметр, проверьте файл `kernel-params.txt`, который поставляется с ядром Linux.

Изучив основы параметров, вы можете смело перейти к главе 6, чтобы узнать особенности запуска пользовательского пространства, начального диска оперативной памяти и программы init, которую ядро запускает в качестве своего первого процесса. В оставшейся части данной главы подробно описано, как ядро загружается в память, запускается и получает свои параметры.

5.4. Загрузчики

В начале загрузки, до загрузки ядра и команды init, программа *загрузчика* запускает ядро. Задача загрузчика проста: ему нужно загрузить ядро в память с диска, а затем запустить его с набором определенных параметров. Однако выполнить ее сложнее, чем кажется. Чтобы понять, почему так, рассмотрим вопросы, на которые должен ответить загрузчик:

- Где находится ядро?
- Какие параметры ядра должны быть переданы ему при запуске?

Ответы, как правило, заключаются в том, что ядро и его параметры обычно находятся в корневой файловой системе. Кажется, что параметры ядра легко найти, но помните, что само ядро еще не запущено, а чаще всего именно оно просматривает файловую систему, чтобы найти необходимые файлы. Хуже того, драйверы устройств ядра, обычно применяемые для доступа к диску, также недоступны. Это похоже на извечный вопрос: что было раньше, курица или яйцо? В данном случае ситуация еще сложнее, но сейчас давайте рассмотрим, как загрузчик преодолевает препятствия, связанные с драйверами и файловой системой.

Загрузчику действительно нужен драйвер для доступа к диску, но не тот, который использует ядро. В системе Windows для доступа к дискам загрузчики применяют традиционную базовую систему ввода-вывода (BIOS, Basic Input-Output System) или более новый интерфейс *Unified Extensible Firmware Interface (UEFI)*. (Интерфейсы *Extensible Firmware Interface (EFI)* и *UEFI* подробнее рассмотрим в подразделе 5.8.2.) Современное дисковое оборудование включает встроенное ПО, позволяющее BIOS или UEFI получать доступ к подключенному оборудованию хранения данных через *стандартизованный механизм адресации (Logical Block Addressing, LBA)*. LBA — это универсальный простой способ доступа к данным с любого диска, но его производительность довольно низка. Это не проблема, поскольку загрузчики часто являются единственными программами, которые должны использовать этот режим для доступа к диску: после запуска ядро получает доступ к собственным высокопроизводительным драйверам.

ПРИМЕЧАНИЕ

Чтобы определить, задействует ли ваша система BIOS или UEFI, запустите утилиту efibootmgr. Если вы получите список целей загрузки, значит, в системе применяется UEFI. Если же придет сообщение, что переменные EFI не поддерживаются, — применяется BIOS. Кроме того, можете проверить, есть ли в вашей системе файл /sys/firmware/efi. Если это так, она использует интерфейс UEFI.

Как только доступ к необработанным данным диска разрешен, загрузчик должен выполнить работу по поиску нужных данных в файловой системе. Большинство распространенных загрузчиков могут считывать таблицы разделов и имеют встроенную поддержку доступа к файловым системам только для чтения. Таким образом, они могут находить и считывать файлы, необходимые для загрузки ядра в память. Эта возможность значительно упрощает динамическую настройку и усовершенствование загрузчика. Загрузчики Linux не всегда имели такую возможность, а без нее настроить загрузчик сложнее.

Для ядра наблюдается тенденция добавления новых функций (особенно в том, что касается технологии хранения данных), за которыми следуют загрузчики, добавляющие отдельные упрощенные версии этих функций для компенсации.

5.4.1. Задачи загрузчика

Основная функция загрузчика Linux включает в себя возможность выполнять следующие задачи:

- выбирать одно из нескольких ядер;
- переключаться между наборами параметров ядра;
- разрешить пользователю вручную переопределять и редактировать имена и параметры образов ядра (например, для входа в однопользовательский режим);
- обеспечить поддержку загрузки других операционных систем.

За время, прошедшее с момента создания ядра Linux, загрузчики стали более эффективными, появились новые функции, например история командной строки и системы меню, но основной задачей всегда была гибкость в выборе образа ядра и параметров. (Удивительно, но необходимость в некоторых задачах действительно снизилась. Например, поскольку вы можете выполнить аварийную или восстановительную загрузку с USB-накопителя, не нужно беспокоиться о ручном вводе параметров ядра или переходе в однопользовательский режим.) Современные загрузчики обеспечивают бóльшую мощность, чем когда-либо, что может быть особенно удобно, если вы создаете собственные ядра или хотите настроить параметры ядра.

5.4.2. Обзор загрузчиков

Вот основные загрузчики, которые вы можете встретить в системах Unix:

- **GRUB.** Почти универсальный загрузчик для систем Linux с версиями BIOS/MBR и UEFI.
- **LILO.** Один из первых загрузчиков Linux. ELILO — версия UEFI.
- **SYSLINUX.** Может быть настроен для запуска из множества файловых систем.
- **LOADLIN.** Загружает ядро из MS-DOS.
- **systemd-boot.** Простой менеджер загрузки UEFI.
- **coreboot (ранее LinuxBIOS).** Высокопроизводительная замена BIOS для персональных компьютеров, которая может содержать ядро.
- **Linux Kernel EFISTUB.** Плагин ядра для загрузки ядра непосредственно из системного раздела EFI/UEFI (ESP).
- **efilinux.** Загрузчик UEFI, предназначенный для применения в качестве модели и эталона для других загрузчиков UEFI.

Эта книга посвящена в основном загрузчику GRUB. Причины использования других загрузчиков заключаются в том, что их проще настроить, чем GRUB, они быстрее или предоставляют другие функции специального назначения.

Чтобы изучить информацию о загрузчике, перейдите в приглашение загрузки и введите имя ядра и параметры. Однако для этого нужно знать, как попасть в приглашение загрузки или меню. К сожалению, иногда это сложно найти, потому что дистрибутивы Linux сильно видоизменяют поведение и внешний вид загрузчика. Бывает, что, просто наблюдая за процессом загрузки, невозможно определить, какой именно загрузчик используется дистрибутивом.

В следующих разделах рассказывается, как получить приглашение загрузки, чтобы ввести имя ядра и параметры. После этого вы сможете узнать, как настроить и установить загрузчик.