Хакер - Как работает Linux: от нажатия кнопки включения до рабочего стола
nopaywall

Евгений Зобнин

Содержание статьи

Лучший способ понять, как работает операционная система, — это проследить поэтапно ее загрузку. Именно во время загрузки запускаются все те механизмы, что приводят ОС в движение. Процесс этот сложный, многоступенчатый и порой запутанный. Изучать его интересно, а открытия, которые ты сделаешь при этом, могут сильно тебя удивить.
В целом загрузку среднестатистического дистрибутива Linux можно разделить на пять стадий:

1. Загрузчик.
2. Запуск и первичная инициализация ядра.
3. Обнаружение оборудования, загрузка драйверов и подключение файловых систем.
4. Запуск системных служб (демонов).
5. Старт графической или консольной пользовательской сессии.

Мы пройдемся по всем стадиям и узнаем, что происходит во время загрузки типичного дистрибутива Linux, немного отклонившись в сторону BSD, macOS и Android по пути. Во

многих случаях это позволит понять, почему процесс загрузки Linux именно такой, какой есть.

1. Загрузчик

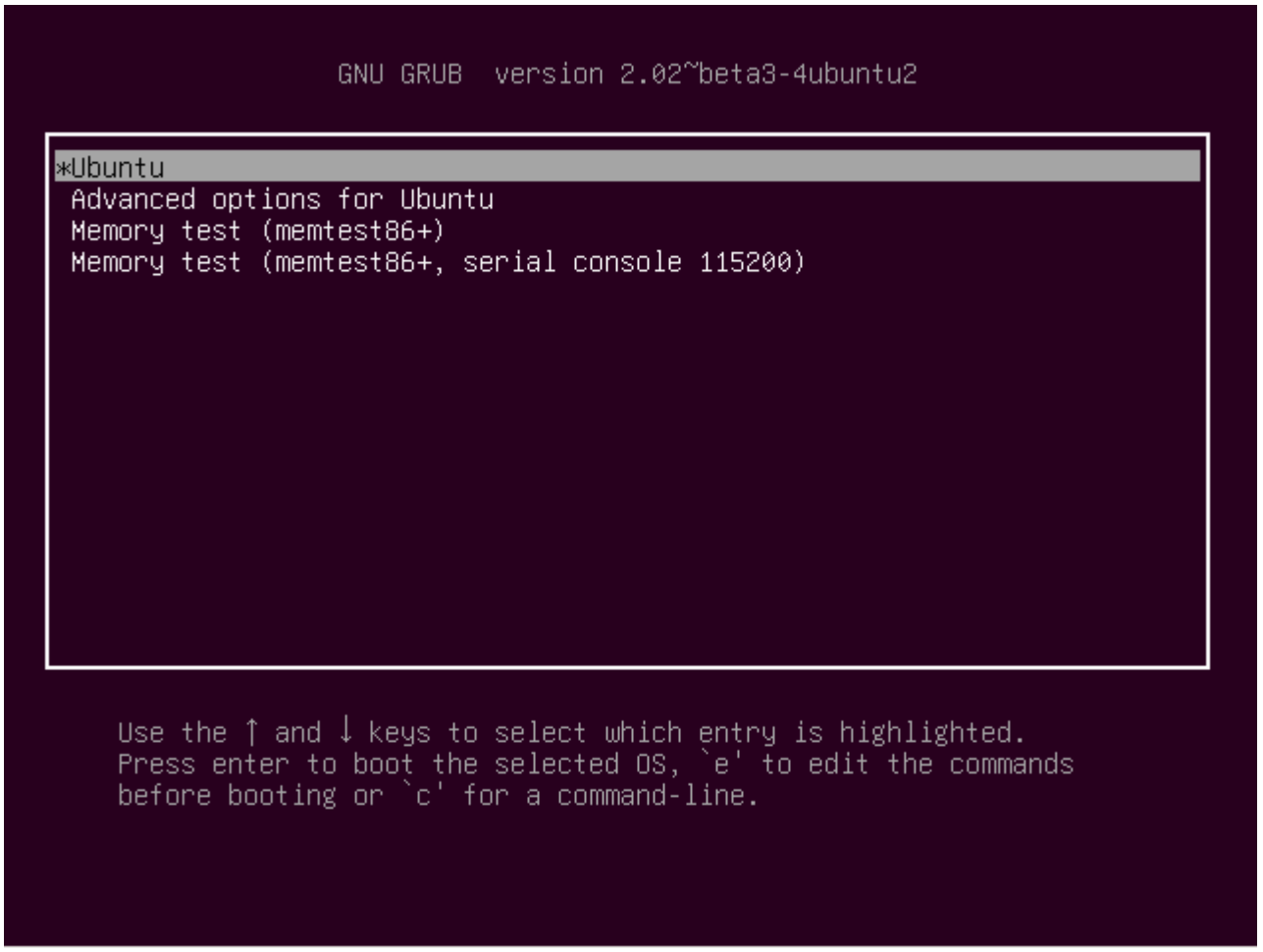
Все начинается с загрузчика, которому во время старта машины BIOS передает управление. В старые времена, когда Linux был не так популярен, в качестве загрузчика использовался LILO (Linux Loader) — простой, очень примитивный и не позволяющий менять конфигурацию загрузки на лету. Фактически конфигурационный файл был вшит в сам загрузчик, и его приходилось переустанавливать после каждой смены настроек: обновил ядро, забыл переустановить, и твой ноутбук больше не грузится.
Сегодня загрузкой Linux практически в любом дистрибутиве занимается **Grub**, изначально разработанный для операционной системы GNU/Hard. Grub гораздо сложнее LILO и фактически сам является полноценной ОС. Он не просто читает конфиг загрузки (обычно это

```
/boot/grub/grub.cfg
```

) прямо с диска, но и позволяет исправить этот конфиг на месте. Grub имеет встроенную командную строку, работает с десятком различных файловых систем и позволяет формировать сложные цепочки загрузки.
Как только пользователь выбирает нужный пункт меню (либо по истечении тайм-аута), Grub находит связанный с этим пунктом меню образ ядра Linux на диске (обычно это файл

```
/boot/vmlinuz
```

), а также закрепленный за ним образ initramfs (о нем чуть позже), загружает их в память и передает управление ядру.



Чтобы увидеть меню Grub в Ubuntu, необходимо удерживать Shift

Почему vmlinuz?

Название файла с образом ядра имеет исторические корни. Изначально образ ядра классического UNIX назывался просто unix, но, когда появилась версия ядра для процессоров с защитой памяти, чтобы избежать путаницы, название сменили на vmunix. Буква z вместо x на конце, в свою очередь, означает, что образ сжат с помощью утилиты gzip (алгоритм Deflate, тот же, что в классическом ZIP).

2. Ядро и initramfs

Получив управление, ядро начинает первичную инициализацию: запускается подсистема управления памятью, настраивается обработчик прерываний, инициализируются необходимые для дальнейшей работы ядра структуры данных. Когда эта работа будет закончена, ядро распаковывает архив **initramfs** (обычно он имеет имя вида

```
/boot/initramfs-linux.img
```

и представляет собой архив cpio, сжатый с помощью gzip) в файловую систему в оперативной памяти (tmpfs), делает ее корневой файловой системой и запускает скрипт

```
/init
```

(в различных дистрибутивах имя может отличаться).
Initramfs включает в себя базовый набор компонентов Linux-дистрибутива: стандартные системные каталоги

```
/bin
```

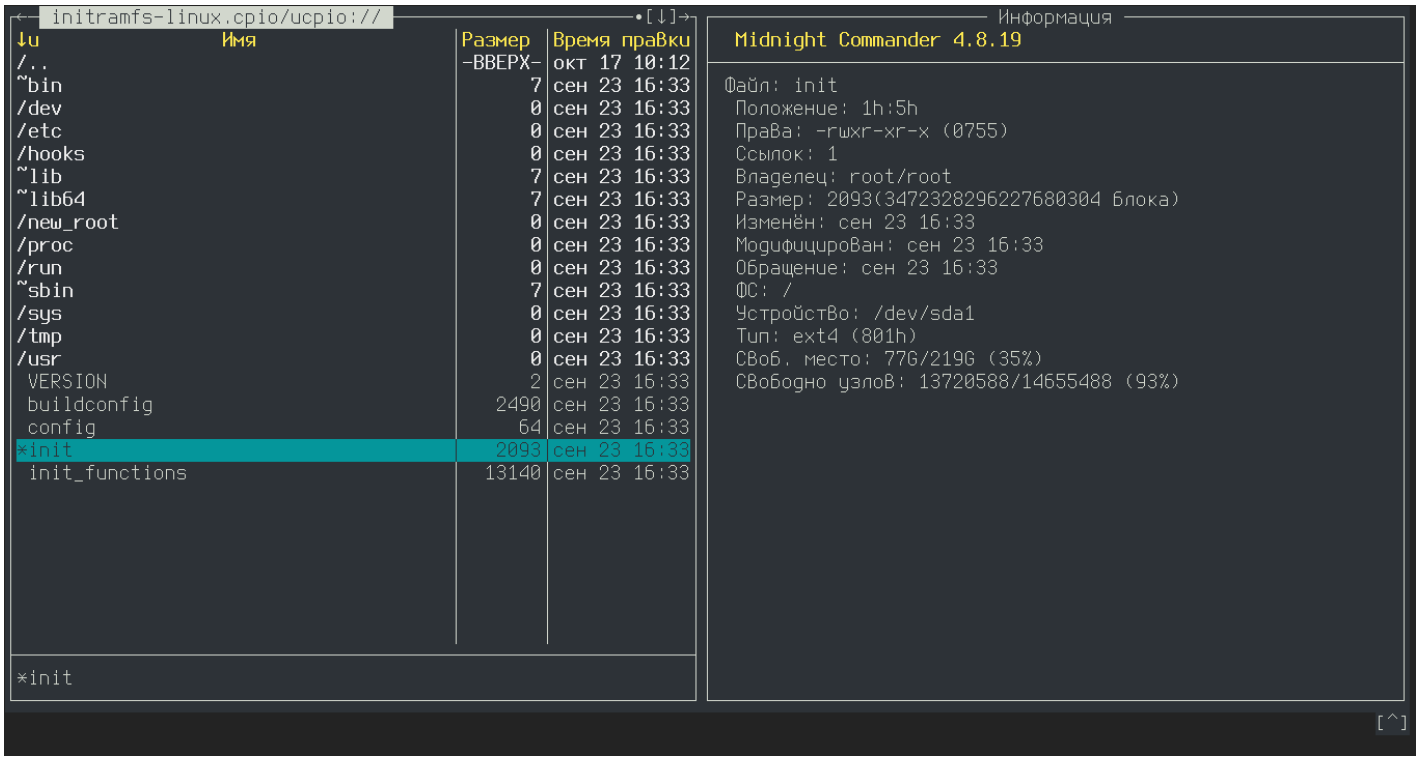
,

```
/lib
```

,

```
/etc
```

и так далее, простейший командный интерпретатор (обычно ash), набор команд BusyBox, несколько вспомогательных библиотек и набор модулей ядра (драйверов), предназначенных для работы с различными накопителями и файловыми системами.



Содержимое initramfs

Смысл существования initramfs в том, чтобы решить проблему курицы и яйца: загрузить драйверы для подключения реальной корневой файловой системы до того, как она будет подключена. Именно это и происходит, когда система запускает скрипт

```
/init
```

. Он определяет установленные в систему накопители, загружает в ядро драйверы для работы с ними, а затем подключает нужный раздел нужного накопителя (о том, какой именно, ядро узнает благодаря переданному при загрузке параметру root) к корню, перекрывая таким образом содержимое initramfs. Затем скрипт запускает

`/sbin/init`

, с которого и начинается следующий шаг загрузки ОС.

```
#!/usr/bin/ash

udevd_running=0
mount_handler=default_mount_handler
init=/sbin/init
rd_logmask=0

. /init_functions

mount_setup

# parse the kernel command line
parse_cmdline </proc/cmdline

# setup logging as early as possible
rdlogger_start

for d in ${disablehooks//,/ }; do
    [ -e "/hooks/$d" ] && chmod 644 "/hooks/$d"
done

. /config

run_hookfunctions 'run_earlyhook' 'early hook' $EARLYHOOKS

if [ -n "$earlymodules$MODULES" ]; then
    modprobe -qab ${earlymodules//,/ } $MODULES
fi
```

Скрипт init из initramfs

А что в Android?

Android использует очень похожий принцип загрузки. Исключение здесь состоит только в том, что initramfs не перекрывается реальной файловой системой по окончании работы

`/init`

. Вместо этого нужные разделы подключаются уже к ней: системный раздел к

`/system`

, раздел с установленными приложениями — к

`/data`

и так далее. Кроме того, ядро и initramfs здесь располагаются не в отдельных файлах, а записаны друг за другом в раздел boot.

3. Первичная инициализация

После того как скрипт

```
/init
```

из initramfs заканчивает свою работу, он запускает утилиту

```
/sbin/init
```

реальной файловой системы. С этого момента начинается инициализация самой ОС: загрузка необходимых драйверов, подключение файловых систем и разделов подкачки, настройка сетевых интерфейсов и запуск системных служб.
Исторически

```
/sbin/init
```

была очень простой утилитой, которая занималась только тем, что передавала управление определенным скриптам в зависимости от переданного ей параметра (скрипты располагались в каталогах

```
/etc/rcX.d/
```

, где X — уровень загрузки). Каждый скрипт отвечал за строго определенную операцию: один подключал перечисленные в файле

```
/etc/fstab
```

файловые системы, другой конфигурировал сетевые интерфейсы, еще один запускал демон cron (он занимается запуском периодических задач), еще один — демон syslog (он отвечает за прием журнальных сообщений и их запись на диск) и так далее. Этот стиль инициализации получил имя SystemV по имени версии UNIX, в которой он появился.
Достоинство стиля инициализации SystemV в его крайней простоте: его легко понять, его легко реализовать, с ним просто работать. Однако он абсолютно не подходит к современным реалиям.
Сегодня загрузка ОС не сводится к загрузке пары-тройки драйверов, подключению двух файловых систем, настройке одного сетевого интерфейса и запуску трех служб. Стандартная конфигурация может включать в себя десятки различных демонов, на запуск каждого из которых может уйти немало времени, а сам демон может упасть. SystemV запускает службы последовательно и не контролирует их жизненный цикл, поэтому загрузка становится медленной, а корректная работа в дальнейшем не гарантируется.
Чтобы обойти эти проблемы, разработчики macOS в свое время создали альтернативу

```
/sbin/init
```

под названием launchd. Это без преувеличения гениальная разработка — launchd не только умеет контролировать жизненный цикл служб, но и запускает их лишь тогда, когда эти службы становятся нужны. Причем делает это весьма неординарным образом. О том, как это происходит, мы еще поговорим. Сейчас в истории с launchd нас интересует другое, а именно то, что под его впечатлением был создан тот самый **systemd**. Сегодня systemd — часть большинства дистрибутивов Linux. Он гораздо сложнее

```
/sbin/init
```

и даже launchd, а в его конструкции нет и намека на уровни запуска и скрипты. Systemd оперирует понятием юнит (unit), который может олицетворять собой службу, операцию монтирования, операцию настройки сетевого интерфейса и другие.

```
[Unit]
Description=Anonymizing Overlay Network
After=network.target

[Service]
User=tor
Type=simple
ExecStart=/usr/bin/tor -f /etc/tor/torrc
ExecReload=/usr/bin/kill -HUP $MAINPID
KillSignal=SIGINT
LimitNOFILE=8192
PrivateDevices=yes

[Install]
WantedBy=multi-user.target
```

Конфиг юнита демона Tor

Для описания юнитов используется специальный декларативный язык, поэтому допустить ошибку конфигурации сложнее, чем при написании скрипта. Юниты могут иметь зависимости друг от друга и запускаются параллельно либо когда в них возникает необходимость. Например, службы (демоны), не зависящие от сетевого подключения, могут быть запущены раньше настройки сетевого интерфейса, другие — сразу после его настройки, третьи — только тогда, когда к ним обращаются приложения или другие службы.

4. Запуск демонов

Запуск служб (демонов) — один из ключевых моментов загрузки ОС. Особое место здесь занимает демон **udev**, без которого типичный дистрибутив Linux окажется неработоспособен.

Udev занимается управлением содержимым каталога

```
/dev
```

. Как мы все знаем, в Linux-системах этот каталог используется для хранения так называемых файлов устройств — особого типа файлов, олицетворяющих собой те или иные компоненты ПК. Именно с помощью файлов устройств в Linux происходит работа с оборудованием: читаешь файл

```
/dev/sda1
```

и получаешь содержимое первого раздела первого жесткого диска, записываешь данные в

```
/dev/fb0
```

и выводишь картинку на экран.
В раннем UNIX каталог

```
/dev
```

был статичен. Он содержал набор файлов на все случаи жизни: даже если в ПК не была установлена звуковая карта, файл

```
/dev/dsp
```

для вывода звука все равно существовал. Когда количество различного оборудования было невелико, а plug’n’play еще не родился, проблем не было: всего лишь десяток-другой файлов. Но со временем он все больше захламлялся и в итоге превратился в помойку.
Первый вариант решения этой проблемы состоял в том, чтобы подключить к

```
/dev
```

виртуальную файловую систему (devfs), содержимым которой управляло бы ядро. Оно находило все установленное оборудование и создавало файлы только для тех устройств, которые реально присутствуют в машине.
Такое решение до сих пор используется в macOS и FreeBSD, но разработчики Linux пошли другим путем. Здесь есть специальная файловая система sysfs, подключенная к каталогу

```
/sys
```

. Это нечто вроде подробной базы данных обо всех устройствах ПК, начиная от процессора и контроллера прерываний и заканчивая мышками и геймпадами.

```
j1m@x220 ~ ls /sys/class/
ata_device  block        drm             hwmon        mei       pci_bus      rfkill       spi_master  vc
ata_link    bsg          drm_dp_aux_dev  i2c-adapter  mem       phy          rtc          thermal     vtconsole
ata_port    devcoredump  firmware        ieee80211    misc      powercap     scsi_device  tpm         watchdog
backlight   devfreq      gpio            input        mmc_host  power_supply scsi_disk    tpmrm       wmi
bdi         dma          graphics        iommu        nd        pps          scsi_host    tty
binder      dmi          hidraw          leds         net       ptp          sound        usbmisc
j1m@x220 ~ ls /sys/block/sda/
alignment_offset  dev              events            ext_range  integrity  range      sda1    stat       uevent
bdi               device           events_async      holders    power      removable  size    subsystem
capability        discard_alignment  events_poll_msecs  inflight   queue      ro         slaves  trace
j1m@x220 ~
```

С помощью /sys можно не только получить информацию об устройствах, но и управлять ими

На основе информации, извлеченной из

`/sys`

, демон udev создает файлы устройств в

`/dev`

. Во время первого старта он проходит по всем устройствам в

`/sys`

, а затем засыпает и ждет, пока не будет добавлено или удалено устройство: воткнул флешку — в

`/sys`

появились новые файлы, udev проснулся и создал на их основе файл устройства в

`/dev`

, загрузив нужные драйверы.

Еще один важный для UNIX-систем демон — **syslog**. Это своего рода агрегатор логов приложений, складывающий их все в каталог

`/var/logs`

. В основанных на systemd дистрибутивах вместо него обычно используется **systemd-journald**, который хранит логи в специальном бинарном формате (syslog оперирует текстом). В него можно добавлять новые записи, но удалять нельзя. Это защита от взломщиков, которые могли бы удалить нужные строки из файлов, чтобы скрыть следы своего пребывания в системе.

В среднестатистическом дистрибутиве Linux также есть другие демоны:

- **cron** — отвечает за выполнение задач по времени. Может запускать команды через определенные промежутки или в четко заданное время. Простейший вариант использования — создание бэкапа по ночам;

- **cups** — демон печати. Следит за очередью отправленных на печать документов и отдает их принтеру;
- **systemd-logind** — управляет пользовательскими сессиями, позволяет быстро переключаться между сессиями, дает разрешение на автомонтирование устройств от лица пользователя и выполняет другие задачи;
- **dbus** — демон, обслуживающий работу шины данных, позволяющей приложениям обмениваться информацией. В основном используется в средах рабочего стола и графических приложениях;
- **NetworkManager** — конфигуратор сетевых интерфейсов. Используется только в десктопных вариантах дистрибутивов и может быть заменен на аналог. Например, wicd.

В большинстве своем демоны обмениваются информацией с приложениями и другими демонами с помощью UNIX-сокетов. Это канал коммуникации, закрепленный за файлом. Например, демон cups создает сокет

```
/var/run/cups/cups.sock
```

(в разных дистрибутивах расположение может отличаться). Записывая в него данные, можно отправлять документы на печать.
Именно эту особенность используют launchd и systemd, чтобы запускать демоны только по мере необходимости. Трюк состоит в том, чтобы заранее создать сокеты для всех системных демонов, а сами демоны запускать только тогда, когда кто-либо запишет данные в сокет; нет смысла запускать cups при загрузке или в любой другой момент, если никто не отправляет задания на печать.

5. X Window System и PAM

Последний этап загрузки — запуск менеджера логина. В консоли функцию менеджера логина выполняет связка утилит **getty** (обычно используется ее более продвинутый вариант agetty) и **login**. Getty представляет собой рудимент, оставшийся со времен мейнфреймов и удаленных терминалов (название расшифровывается как get teletype). Он выводит в терминал текстовое сообщение и затем запускает утилиту login, которая спрашивает логин и пароль пользователя. Когда пользователь вводит корректный пароль, login запускает от его имени шелл, указанный в файле

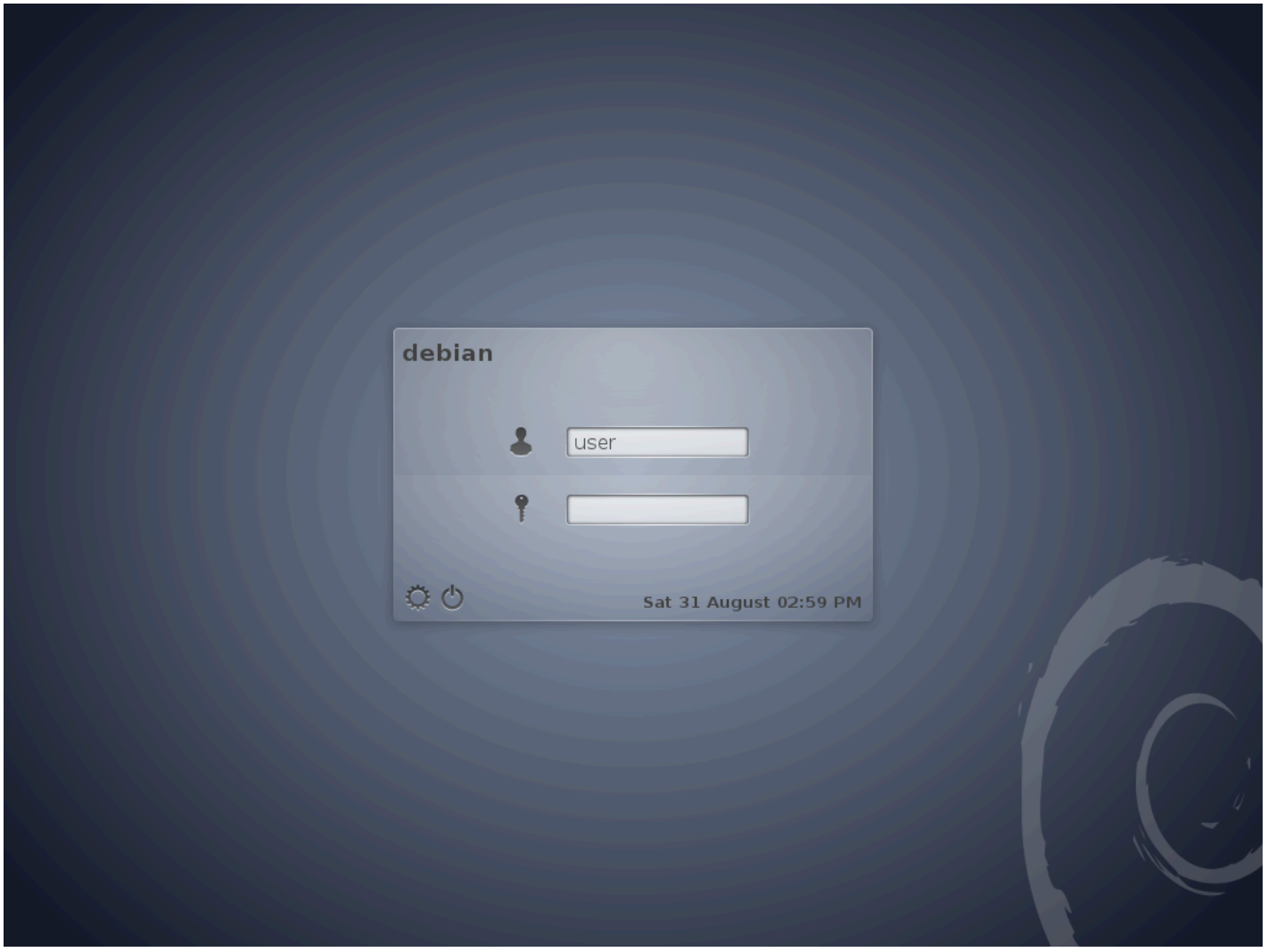
```
/etc/passwd
```

.
Графический менеджер логина называется **дисплейный менеджер** (Display Manager), и в каждой графической среде он свой. KDE использует менеджер KDM, GNOME — GDM,

также есть возможность использовать универсальный дисплейный менеджер, например Slim. В любом случае задача дисплейного менеджера — вывести на экран окно запроса имени пользователя и пароля, а после авторизации либо запустить оконный менеджер напрямую, либо выполнить ряд команд, записанных в пользовательский файл

```
~/.xinitrc
```

.



Дисплейный менеджер KDM в Debian

Одновременно с запуском дисплейного менеджера запускается графическая система X Window System, а в современных дистрибутивах — **Xorg**. Это клиент-серверная система вывода графики на экран, где сервер отвечает за компоновку общей картинки, сформированной различными приложениями-клиентами. X Window System — не графическая среда, а лишь прослойка, позволяющая приложениям отправлять картинку на экран и получать события ввода от пользователя. Чтобы построить на ее основе графический интерфейс, также нужен **менеджер окон** (window manager), который позволит пользователю управлять окнами приложений.

Менеджер окон может работать как обособленно (например, fluxbox, window maker, i3), так и в составе комплексной среды рабочего стола, (KDE, GNOME, XFCE). Кроме менеджера окон, они также включают в себя набор средств для формирования

полноценного десктопа: панель задач в нижней или верхней части экрана, док, систему расположения иконок на рабочем столе и прочее. Обычно каждый из этих элементов управляется одним или несколькими специальными приложениями.
Независимо от того, какой способ входа использует юзер, за контроль доступа всегда отвечает PAM. Это модульная система аутентификации пользователя, которая может проверять его личность самыми разными способами, попутно выполняя ряд проверок. По умолчанию PAM использует аутентификацию исключительно с помощью пароля, но, поменяв конфигурационные файлы

```
/etc/pam.d
```

, порядок аутентификации можно изменить, добавив к нему, например, необходимость приложить палец к сканеру отпечатков, вставить специальную флешку-ключ и даже использовать подтверждение с помощью смартфона. О том, как это сделать, мы уже писали.

Конфигурационный файл PAM утилиты login
На этом процесс загрузки можно считать законченным. Ты либо выполняешь логин в консоль и видишь приглашение командного интерпретатора или вводишь данные учетной записи в окно дисплейного менеджера, и в результате на экране появляется рабочий стол.

Вместо заключения

Так выглядит загрузка современного дистрибутива Linux. Некоторые не особо важные и не особо интересные детали я опустил, но попытался рассказать о самом главном. Если вдуматься, это не такой уж и сложный процесс, фактически операционная система готова принимать команды пользователя уже на этапе подключения initramfs, все остальное — запуск окружения, которое необходимо для полноценной работы пользователя.