

Лабораторная работа №11.

Загрузка системы и управление службами systemd

Введение

В этой лабораторной работе мы узнаем о том, что происходит внутри компьютера после нажатия кнопки питания, рассмотрим этапы, через которые проходит система во время загрузки, и узнаем, как устранять проблемы, если в процессе загрузки что-то пойдет не по плану.

Отдельного внимания удостоится подсистема инициализации и управления службами systemd. Мы посмотрим, что представляют собой юниты и как управлять службами Linux. Узнаем, почему недолюбливают systemd, но при этом ей удалось вытеснить свою предшественницу init.

Общее представление о процессе загрузки ОС Linux

Linux загружается так же, как Windows — и в том, и в другом случае компьютер тестирует оборудование, загружает ядро системы, стартует необходимые службы и запускает приложение для входа. В общем виде загрузка системы выглядит следующим образом:

- Вы нажимаете кнопку включения, и **материнская плата** подает питание на центральный процессор.
- Как только питание поступает на **процессор**, он приступает к выполнению инструкций базовой системы BIOS/UEFI, программный код которой находится в постоянном запоминающем устройстве (ПЗУ) на материнской плате. Такое поведение материнской платы и процессора предопределено их аппаратным устройством.
- Получив управление, **базовая система** выполняет проверку оборудования и приступает к поиску загрузочного диска, для чего в настройках определяется перечень допустимых устройств и порядок их использования. Обычно загрузка выполняется с жесткого диска, но это может быть и флешка, и компакт диск, и даже сетевая карта.

Загрузочный диск может быть размечен в старом формате MBR для BIOS или в новом формате GPT для UEFI, что определяет детали дальнейшей загрузки, но суть процесса от этого не меняется — базовая система передает управление загрузчику.

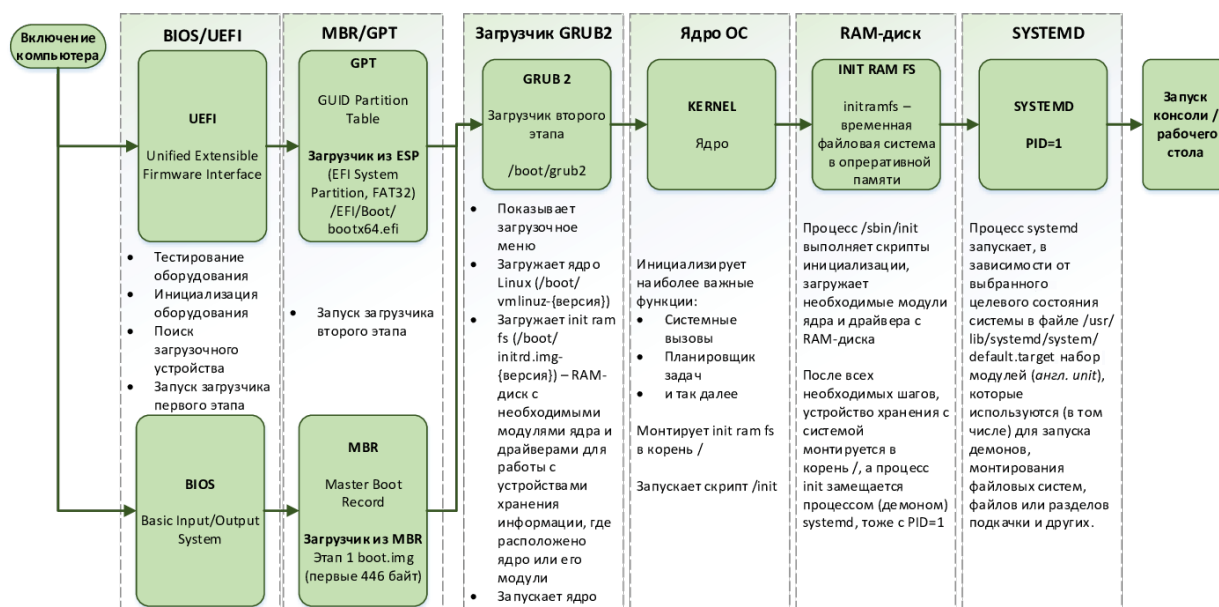
- Получив управление от BIOS/UEFI, **загрузчик** выполняет загрузку операционной системы самостоятельно или по цепочке передает управление другому загрузчику для решения поставленной задачи.

Загрузчики представляют собой вспомогательные приложения, которые находятся между прошивкой и операционной системой, чтобы избавить ОС от технических подробностей реализации загрузочных секторов.

Примечание

В старых компьютерах кнопка включения была фиксирующего типа и напоминала больше рубильник, который замыкал силовую цепь, но с переходом на ATX дежурное питание присутствует на материнской плате постоянно. Поэтому сейчас даже в выключенном состоянии компьютер все равно остается включен и может управлять блоком питания для подачи основного напряжения.

Благодаря такому устройству системы питания стала возможна реализация технологий Wake-on-LAN, iLO и IPMI. А чтобы полностью обесточить процессор, теперь кнопку включения нужно зажать на несколько секунд.



Процесс загрузки ОС Linux

На рисунке представлен весь процесс загрузки системы Linux с краткими пояснениями по каждому из этапов, которые мы будем подробно разбирать далее.

Базовые системы BIOS и UEFI

Базовая система BIOS (от англ. Basic Input-Output system — базовая система ввода-вывода) — это специальная программа, которая хранится в микросхеме на материнской

плате и автоматически запускается при включении компьютера. На техническом сленге такие программы называют прошивками.

Основное предназначение базовой системы состоит в том, чтобы проверить аппаратное обеспечение (выполнить так называемый POST от англ. Power-On Self-Test — самотестирование при включении), инициализировать оборудование, найти загрузочное устройство и запустить загрузчик.

Однако BIOS морально устарел еще лет 20 назад, поэтому в 2007 году с подачи Intel и при активном участии Microsoft был утвержден новый индустриальный стандарт UEFI (от англ. Unified Extensible Firmware Interface — унифицированный интерфейс расширяемой прошивки). Предназначение UEFI то же самое, что и у BIOS, но теперь это не простая прошивка, а целая операционная система в миниатюре, которая обладает модульной архитектурой и позволяет разработчикам железа расширять ее возможности с помощью исполняемых EFI-приложений.

Например, с помощью UEFI-приложений можно реализовать шифрование диска, низкоуровневые инструменты диагностики железа и даже снимать PNG-скриншоты прямо из консоли. А для того чтобы защитить компьютер от вредоносных программ, в системе UEFI реализована технология Secure Boot, которая позволяет автоматически проверять цифровые подписи к приложениям.

Более того, вы можете выполнить команду `file /boot/efi/EFI/Boot/bootx64.efi` и убедиться, что загрузчик GRUB для UEFI тоже является EFI-приложением, которое не напрямую управляет процессором, а пользуется API этой системы.

Прошивки для материнских плат персональных компьютеров разрабатываются несколькими компаниями, среди которых широко известны Award, AMI, Phoenix, но каждый производитель железа адаптирует их потом под себя и даже под конкретные модели плат, поэтому визуальный интерфейс и набор доступных настроек может сильно отличаться.

Разметки диска MBR и GPT

Разметка диска MBR (от англ. Master Boot Record — основная загрузочная запись) пришла на замену BPB для MS-DOS, и ее появление было связано с увеличением емкости жестких дисков. Если раньше размер носителей исчислялся десятками мегабайт, то с начала 80-х они начали расти, как на стероидах, и разработчики задумались о немыслимом... что нужно иметь возможность устанавливать две, три и даже четыре операционные системы на один диск!

Запись MBR занимает один сектор в самом начале диска, поэтому ее размер составляет 512 байт, из которых 446 байт выделяются под программный код главного загрузчика, еще 4 блока по 16 байт составляют таблицу разделов диска, а последние 2 байта содержат сигнатуру 0x55AA, указывающую на то, что этот сектор является загрузочным.

Да, вы все правильно поняли, в таблице разделов MBR всего 4 записи, но это не так страшно, так как операционные системы на программном уровне научились работать с любым количеством расширенных разделов, а устанавливать больше 2-3 систем на один диск не так, чтобы очень нужно.

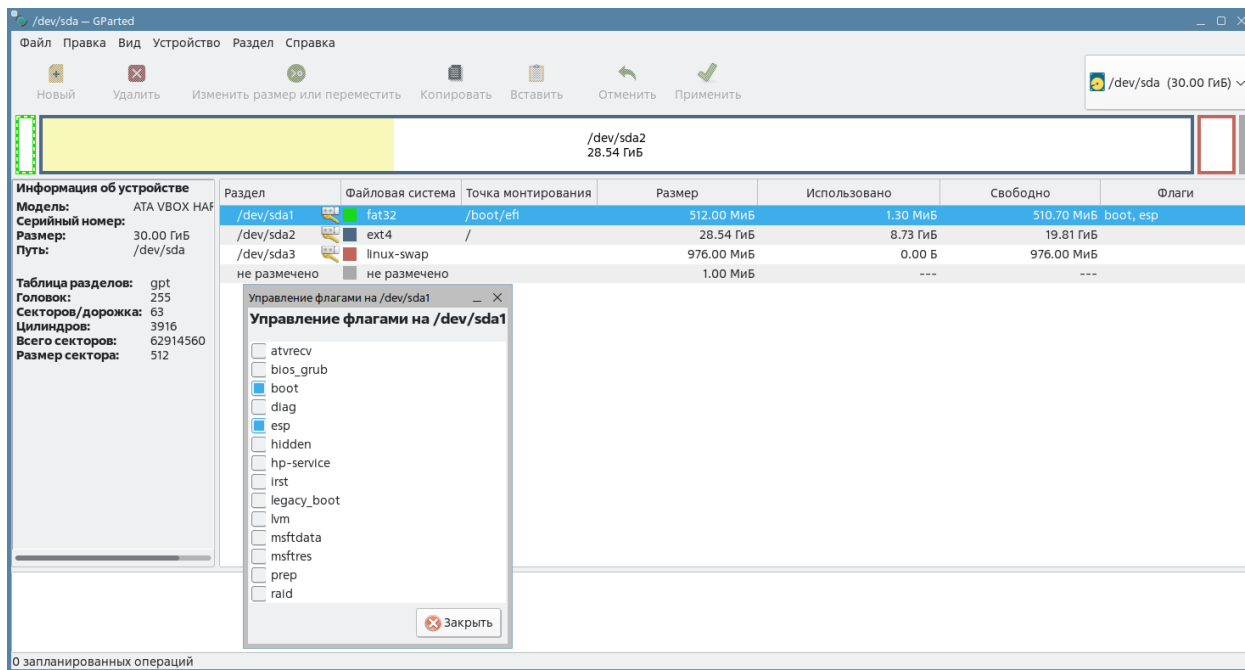
Гораздо сложнее с размерами этих записей. На описание смещения и размер диска выделяется всего по 4 байта, что при фиксированном размере сектора в 512 байт позволяет адресовать не более $2^{32} \cdot 512$ байт ≈ 2 ТБ, а это по современным меркам, прямо скажем, не много.

Чтобы решить ряд накопившихся проблем, в т.ч. обеспечить поддержку дисков размером более 2 ТБ, в рамках стандарта UEFI была разработана разметка GPT (от англ. GUID Partition Table — таблица разделов GUID). Эта разметка позволяет адресовать до 2^{64} секторов, что при стандартном размере сектора в 512 байт позволяет работать с дисками размером до 9.44 зеттабайт (1 ЗБ = 10^{21} байт), а при использовании секторов размером 4 096 байт максимальное значение увеличивается в 8 раз до 75.52 ЗБ.

Может показаться, что столько не надо, т.к. самые большие жесткие диски топчутся сейчас на отметке в несколько десятков терабайт. Но ведь можно объединить несколько таких дисков в один большой виртуальный диск, и на данный момент уже известно о создании виртуальных дисков на сотни петабайт, поэтому такие ограничения уже не видятся столь избыточными.

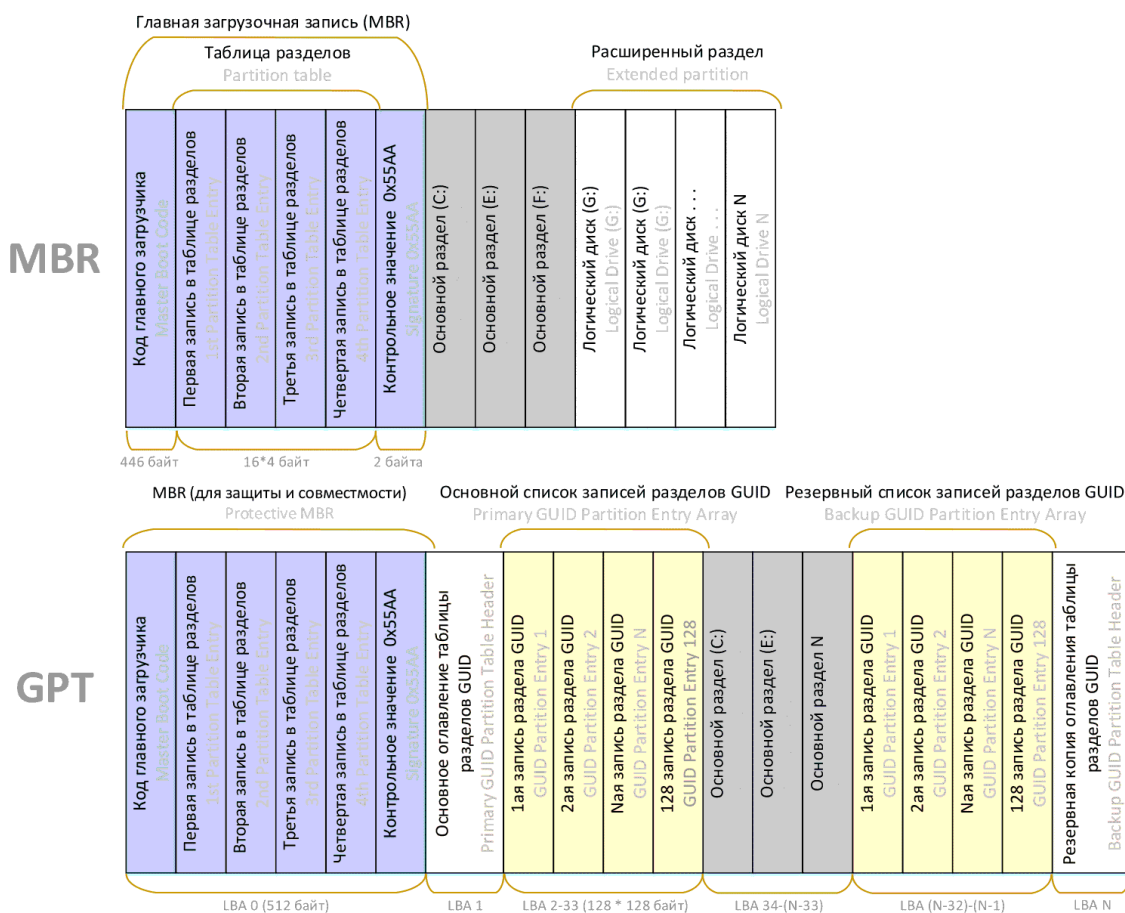
Еще одной отличительной особенностью GPT является то, что в части загрузки операционной системы эта разметка опирается на расширенные возможности UEFI, поэтому загрузочных секторов больше нет, а загрузчик представляет собой EFI-приложение, которое лежит в виде файла в специальном разделе ESP (*англ. EFI System Partition — системный раздел EFI*), который отформатирован в привычной системе FAT32. Кстати, проверьте, что у вас в системе покажет приложение GPart.

Однако, несмотря на то, что загрузочные секторы теперь действительно больше не используются для защиты от случайного форматирования при работе пользователя со старыми приложениями, разметка GPT мимикрирует под MBR, размещая в нулевом секторе диска фейковую запись, которая указывает, что весь объем отдан под раздел GPT.



Системный раздел EFI на диске с ALSE 1.7.x

Эту запись еще называют защитной (от англ. protective), а реальная таблица разделов находится дальше по диску, см. рисунок.



Защитная MBR-запись разметки GPT

Понимая отличия MBR и GPT разметки, мы готовы перейти к рассмотрению работы загрузчиков.

Загрузчик GRUB2

Мы уже упоминали загрузчик GRUB, но он не является единственным — есть еще LILO, Syslinux и другие. Но в операционной системе ALSE выбор сделан именно в пользу GRUB2, который является эталонной реализацией загрузчика для Linux (от англ. GRand Unified Bootloader ver2 — главный универсальный загрузчик второй версии).

Этот загрузчик позволяет пользователю выбрать одну из доступных операционных систем и параметры ее запуска. Он был разработан с учетом опыта первой реализации, чтобы упростить поддержку кода и сделать его безопаснее, надежнее и чище, поэтому со временем смог полностью вытеснить старую версию. Если сейчас говорят GRUB, то подразумевают GRUB2, а предыдущую реализацию обычно называют теперь GRUB Legacy.

У загрузчика GRUB2 есть две версии: одна для BIOS (boot.img + core.img), которая управляет процессором и взаимодействует с устройствами напрямую, и вторая для UEFI (bootx64.efi), которая использует API этой системы. Но у загрузчика в любом случае есть свои собственные драйверы файловых систем, шифрования дисков, взаимодействия с TPM и др. Обладая модульной архитектурой, GRUB2 позволяет подключать компоненты по мере надобности, что значительно расширяет его функциональность.

Учитывая поддержку спецификации Multiboot, GRUB2 может загрузить любую совместимую с ней операционную систему, среди которых: Linux, FreeBSD, Solaris и многие другие. Кроме того, GRUB2 может по цепочке передать управление другому загрузчику, что позволяет загружать Windows (через загрузчик NTLDR или bootmgr), MS-DOS, OS/2 и другие системы.

Загрузчик GRUB2 выполняет следующие задачи:

- Отображает меню для выбора операционной системы.
- Загружает выбранное ядро Linux, например, из файла `boot/vmlinuz-5.15.0-70-generic`
- Монтирует диск с модулями ядра и другими файлами системы в оперативную память с помощью initramfs, например, из файла `boot/initrd.img-5.15.0-70-generic`
- Запускает ядро.

Кроме этого, загрузчик позволяет изменять параметры загрузки ядра ОС и работать с командной строкой. Устройство загрузчика и работу с ним мы подробно разберем далее.

Запуск загрузчика

По историческим причинам сложилось много разных способов загрузки ОС. Какие-то из них чаще применяются в Windows, какие-то в Linux, но достаточно понять основной принцип, чтобы все встало на свои места.

Способ загрузки BIOS > MBR > PBR

Системы BIOS работают с разметкой диска MBR, в соответствии с которой они передают управление загрузчику из основной загрузочной записи, находящейся в нулевом секторе диска. И далее загрузка операционной системы выполняется в два этапа:

- **На первом этапе** загрузчик из MBR просматривает таблицу разделов и передает управление по цепочке загрузчику из нулевого сектора активного раздела, в котором находится загрузочная запись раздела PBR (от англ. Partition Boot Record, но в ряде источников ее называют Volume Boot Record, VBR).

Загрузчик из PBR загружает драйвер файловой системы и передает управление основному загрузчику операционной системы, который представлен уже в виде файлов, а не байтов управляющих инструкций, записанных в секторах на диске.

- **На втором этапе** основной загрузчик уже располагает всем необходимым, чтобы выполнить загрузку операционной системы, поэтому он начинает загружать ядро со всеми его модулями и передает ему управление.

Однако в этой схеме между первым и вторым этапами есть еще один этап, который называют «**этап 1.5**» (или полтора). Возникает он по причине того, что запись PBR должна укладываться в один сектор размером всего 512 байт, чего явно недостаточно для размещения сколь-нибудь существенной программы.

По указанной причине программный код загрузчика делят на две части, которые нужно устанавливать обязательно вместе, так как они являются фрагментами одной программы. Первую часть загрузчика (этап 1) размещают в PBR, и ее задача заключается лишь в том, чтобы загрузить первый сектор второй части загрузчика (этап 1.5) и передать управление этому коду.

Программный код второго загрузчика прячут между нулевым сектором раздела, в котором расположена загрузочная запись, и 63-м сектором, с которого начинаются данные. Эта область свободна и не является частью какого-либо раздела из соображений совместимости с DOS, поэтому может быть использована без каких-либо опасений.

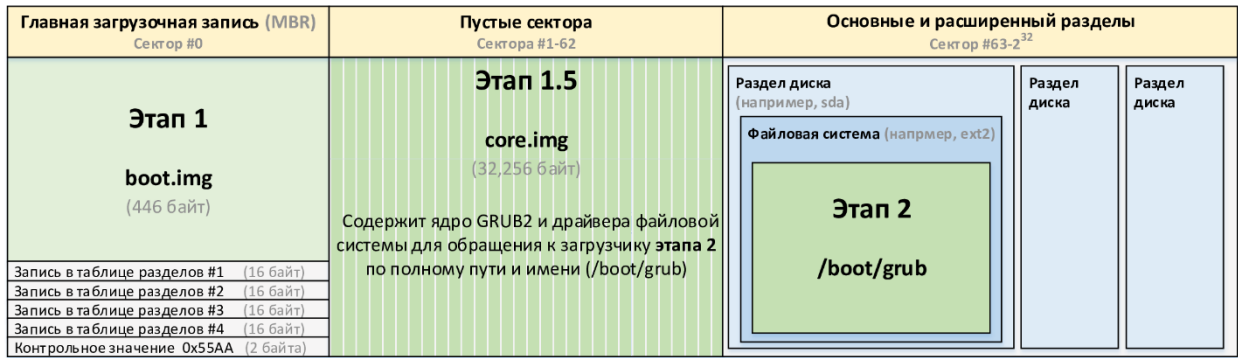
Благодаря такому трюку загрузчик второго этапа может представлять собой уже довольно сложную программу, которая способна работать даже с файловыми системами FAT32. Например, старый загрузчик NTLDR (от англ. NT Loader) использовал параметры запуска ядра из файла `boot.ini` в корне системного диска, а современный загрузчик BOOTMGR (от англ. Boot Manager) хранит эти настройки в файле реестра `/Boot/BCD` на скрытом служебном разделе System.

Ничто не мешает применить для загрузки Linux этот сценарий и установить GRUB в раздел диска, как загрузчик BOOTMGR или NTLDR, но так делают довольно редко. Обычно GRUB устанавливают напрямую в запись MBR, что будет рассмотрено далее.

Способ загрузки BIOS > MBR

Во времена, когда пользователями компьютеров были исключительно специалисты с очень высокой квалификацией, использовать флажок активности раздела для переключения между установленными операционными системами казалось вполне логичным решением. Компьютер, правда, можно было сломать, а так очень удобное решение.

Однако в дальнейшем из соображений удобства от такого подхода (внезапно) решили отказаться, и даже загрузчик NTLDR уже предлагал меню для выбора ОС. А раз уж информация об активности диска больше не требовалась, то стало возможным упростить цепочку и устанавливать основной загрузчик сразу в MBR. Именно так и поступает загрузчик GRUB, см. рисунок.



Структура MBR в разрезе размещения загрузчиков при использовании BIOS

Загрузчик GRUB первого этапа так же, как в Windows, представлен в двух частях. Первая часть `boot.img` является простым загрузчиком, который размещается в основной загрузочной записи MBR (этап 1), и его единственная задача заключается в том, чтобы загрузить вторую часть `core.img` (этап 1.5) и передать ей управление. Программный код

core.img, в свою очередь, загружает модули GRUB уже из папки /boot/grub (этап 2), с помощью которых выполняет дальнейшую загрузку операционной системы.

Это классический способ загрузки, который может применяться на устаревших системах.

Внимание

Установщик Windows конфликтует с grub-install, поскольку затирает программный код основной загрузочной записи и возвращает компьютер к использованию информации об активности разделов. Поэтому после установки Windows на тот же диск требуется восстановить код загрузчика GRUB, например, загрузившись через Live CD.

Способ загрузки BIOS > GPT и UEFI > MBR

Стандарт разметки GPT появился чуть позже UEFI, поэтому были релизы этой прошивки, которые поддерживали загрузку с MBR-дисков. И тут важно понимать, что подключить диск с разметкой MBR к современному компьютеру с UEFI не является проблемой, но вот использовать его в качестве загрузочного не стоит, даже если прошивка это позволяет.

Примечание

Допустимым считается загрузка с MBR-дисков на современных компьютерах с UEFI, если включена опция Legacy Boot режим (Compatibility Support Mode). В этом случае ваша прошивка работает как обычный BIOS.

Пользователям компьютеров с BIOS очень не нравилось, что систему можно было устанавливать только на диски объемом до 2Тб, иначе все остальное пространство становилось недоступным. Для снятия этого ограничения разработчики воспользовались обратной совместимостью GPT и MBR.

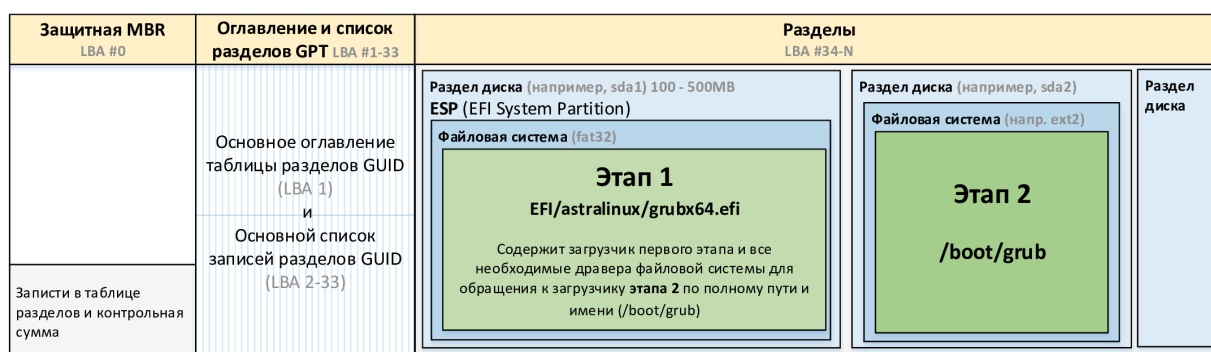
Как мы обсуждали выше, во избежание случайного удаления данных с GPT-диска в нулевом секторе располагается «защитная» MBR-запись, указывающая на то, что весь объем диска отдан под раздел GPT. В эту запись и стали записывать код загрузчика, которому BIOS передавал управление в штатном порядке. И оставалось только, чтобы код этого загрузчика и сама операционная система могли работать с диском в разметке GPT.

Еще раз отметим, что оба этих сценария не представляют практического интереса, поэтому не будут рассматриваться более подробно. Использовать разметку GPT с BIOS или MBR с UEFI нежелательно, а если вы переключите флажок «Включить EFI» для виртуальной машины с уже установленной системой, она просто не взлетит.

Способ загрузки UEFI > GPT

Как мы уже знаем, базовые системы UEFI могут работать с разделами диска, отформатированными в FAT32, поэтому позволяют полностью отказаться от каких-либо загрузочных секторов и загружать все нужные файлы напрямую из защищенного раздела ESP.

Так делает Windows, так поступает и GRUB. Например, EFI-файл, содержащий код загрузчика, можно найти по адресу [EFI/Boot/bootx64.efi](#) (копия [EFI/astralinux/grubx64.efi](#)), см. рисунок.



Структура GPT в разрезе размещения загрузчиков при использовании EFI

Сейчас такой способ загрузки считается наиболее правильным и настоятельно рекомендуется всеми разработчиками операционных систем.

Способ загрузки PXE

Можно еще вспомнить про PXE (англ. Preboot eXecution Environment, предзагрузочная среда выполнения), когда загрузка начального исполняемого кода выполняется с сетевого хранилища, что используется в том числе для установки ОС по сети и обеспечения работы тонких клиентов.

В этом случае сетевая карта под капотом просто реализует один из представленных выше сценариев для BIOS или UEFI. Загрузчик первого этапа находится в ПЗУ сетевой карты и его задача состоит в том, чтобы симитировать загрузочное устройство, используя файлы загрузчика с TFTP или HTTP-сервера. Адрес сервера и имя файла с загрузчиком сетевая карта получает от DHCP через опции 66 и 67 соответственно.

Работа загрузчика GRUB2

Рассмотрим работу загрузчика GRUB2 и этапы загрузки ОС, которые проходят под его управлением. Основные параметры, определяющие работу загрузчика это:

- Образ ядра и параметры его запуска.
- Образ инициализационной файловой системы initramfs.
- Корневая файловая система.

Работа с меню загрузки

После начала своей работы GRUB2 выводит на экран загрузочное меню. По умолчанию в нем доступно только 2 пункта - загрузка системы и загрузка системы в режиме восстановления. Перезагрузим виртуальную машину и нажмем клавишу <Вверх> при появлении загрузочного меню, чтобы остановить таймер.



Загрузочное меню GRUB2

Внизу страницы будет представлена справка по способам управления: с помощью стрелок можно переходить по пунктам меню, клавишей **e** можно отредактировать команды запуска перед загрузкой, а клавишей **c** - перейти в режим командной строки.

Просмотр и редактирование команд запуска

Перейдем в режим редактирования команд запуска. Система попросит ввести учетные данные пользователя, созданного при установке GRUB. Имя пользователя будет совпадать с именем локального администратора, в нашем случае это sa.

После ввода корректных учетных данных будут отображены команды запуска. Текст на экране доступен для редактирования, но пока в нем лучше ничего не менять (эти изменения не сохраняются и влияют только на текущий запуск ОС. Если была допущена ошибка и ОС не может загрузиться, то перезагрузка решит эту проблему).

Команды запуска ядра ОС для BIOS+MBR

Разберем, что означает каждая из этих команд:

- **load_video** – вызывает функцию, которая подгрузит видео-модули GRUB2.
- **insmod {имя_модуля}** – подгружает модуль по его имени.
- **set/unset {имя_переменной}={значение}** – как и при работе в консоли - задает или удаляет переменную.
- **search** – ищет устройство по файлу (**--file**), по имени файловой системы (**--label**) или по UUID файловой системы (**--fs-uuid**). Параметр **--set** - задает переменную, параметр **--no-floppy** - не производит поиск дисководов дискет.

• **linux** {путь_до_образа_ядра_ОС и параметры запуска} – загружает ядро ОС с заданными параметрами:

- **root=UUID=...** – корневая файловая система.
- **ro** – монтирование корневого каталога только на чтение (*англ. read-only*).
- **parsec.max_ilev\ = 63** – максимальным неиерархический уровень целостности равный 63.
- **quite** - «тихая» загрузка без вывода информационных сообщений ядра (процессы активации драйверов и модулей системы, проверки файловой системы).
- **net.ifnames=0** - использование традиционного названия интерфейсов (eth0, eth1 ...).

• **initrd** {путь_до_образа_RAM-диска} - загружает RAM-диск для ядра ОС.

Ниже представлен список некоторых других параметров, которые могут быть переданы ядру:

- **splash** - разрешить отображать графическую заставку
- **rw** - монтировать корневой каталог на чтение/запись (вместо ro)
- **s** - загрузка в режиме восстановления (single mode)
- **3** - загрузка в режиме multi-user (без графики)
- **5** - загрузка в режиме graphical (графический режим)

Командная строка GRUB2

Теперь переключимся в режим командной строки, нажав **F2**.

```
GNU GRUB, версия 2.06-3~deb10u1+c1202210061917+astra3

Поддерживается несколько BASH-подобных команд редактирования строки. Есть вывод списка команд по TAB для
дополнения. Также есть везде, где возможно, вывод списка по TAB для устройств или файлов. Для выхода нажмите ESC в
любой момент.

grub> _
```

Командная строка GRUB2

Нам доступна командная строка GRUB2. Командой **help** можно вывести список всех доступных команд. Перед её вызовом можно задать постраничный вывод командой **set pager=1** (перемотки или привычной команды **less** тут нет).

```
grub> set pager=1
grub> help
.
[ ВЫРАЖЕНИЕ ]
all_functional_test
background_color ЦВЕТ
backtrace
blocklist
break [НОМЕР]
cbmemc
clear
cmosdump
cmosreset
configfile
coreboot_boottime
crc
cutmem
distrust
dump
efiemu_loadcore
efiemu_unload
exit
extract_entries_configfile
extract_legacy_entries_configfile
extract_syslinux_entries_configfile
false
freedos
gdbstub
gdbstub_stop
gptsync
hashsum
hello
hexdump
humatch
initrd
inl
inw
keystatus
kfreebsd_loadenv
kfreebsd_module_elf
knetbsd_module
kopenbsd
legacy_check_password
legacy_initrd
legacy_kernel
--ДАЛЬШЕ--
915resolution
acpi
authenticate [СПИСОК_ПОЛЬЗОВАТЕЛЕЙ]
background_image [-m (stretch|normal)] ФАЙЛ
badram
boot
cat
chainloader
cmosclean
cmosset
cmp
continue [НОМЕР]
cpuid
cryptomount
date
drivemap
echo
efiemu_prepare
eval
export ПЕРЕМЕННАЯ_ОКРУЖЕНИЯ [ПЕРЕМЕННАЯ_ОКРУЖЕНИЯ] ...
extract_entries_source
extract_legacy_entries_source
extract_syslinux_entries_source
file
functional_test
gdbstub_break
gettext СТРОКА
halt
hdparm
help [ШАБЛОН ...]
hexdump_random
inb
initrd16
insmod МОДУЛЬ
keymap
kfreebsd
kfreebsd_module
knetbsd
knetbsd_module_elf
kopenbsd_ramdisk
legacy_configfile
legacy_initrd_nounzip
legacy_password
```

Командная строка GRUB2

Командная строка GRUB2 может быть полезна при решении возникающих проблем, например, в том случае, когда конфигурационный файл был утрачен/повреждён или, как вариант, имя загрузочного устройства (hd0) было изменено в результате подключения дополнительных жёстких дисков.

Вернемся в окно просмотра и изменения команд загрузки, удалим параметр **quite** в команде `linux` и нажмем `F10` для загрузки:

```
linux /boot/vmlinuz-5.15.0-111-generic root=UUID=eba34003-adda-47f3-a50d-8b9513eb81dc ro parsec.max_ilev\
=63 det.ifnames=0
```

Как видите, в этот раз кроме статуса запуска демонов была выведена и другая информация о запуске ОС.

С помощью команды `dmesg` можно посмотреть все сообщения ядра ОС (от момента начала загрузки до текущего). После загрузки ОС откройте терминал, введите команду `sudo dmesg` и вы увидите подробный лог загрузки ОС.

```
sa@astra:~$ sudo dmesg
[sudo] пароль для sa:
[    0.000000] Linux version 5.15.0-111-generic (builder@runner-6zppmmke-project-1752-concurrent-0) (gcc (AstraLinux 8.3.0-6+b1) 8.3.0, GNU ld (GNU Binutils for AstraLinux) 2.31.1) #astra2+ci1 SMP Mon Jul 22 13:27:51 UTC 2024 (Astra 5.15.0-111.astra2+ci1-generic 5.15.152)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-111-generic root=UUID=e8a34003-adda-47f3-a50d-8b9513eb81dc ro parsec.max_ilev=63 net.ifnames=0
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000009fbfff] usable
[    0.000000] BIOS-e820: [mem 0x0000000000009fc00-0x0000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x000000000000f0000-0x000000000000ffffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000001000000-0x00000000007ffffffffff] usable
[    0.000000] BIOS-e820: [mem 0x00000000007ffff0000-0x00000000007ffffffffff] ACPI data
[    0.000000] BIOS-e820: [mem 0x00000000fec000000-0x00000000fec00ffff] reserved
```

Примечание

Инструкция по изменению параметров загрузчика доступна по ссылке: <https://wiki.astralinux.ru/pages/viewpage.action?pageId=18874503>

Инструкция по установке загрузчика GRUB2 в UEFI GPT доступна по ссылке: <https://wiki.astralinux.ru/pages/viewpage.action?pageId=130424699>

С полным руководством по GRUB и списку его команд на английском языке можно ознакомиться по ссылке: <https://www.gnu.org/software/grub/manual/grub/grub.html>

Загрузка ядра и RAM-диска

Файлы образа ядра `/boot/vmlinuz-версия_ядра` и инициализационной файловой системы `initramfs` `/boot/initrd.img-версия_ядра` — это сжатые архивные файлы.

GRUB производит распаковку и декомпрессию образа `initrd.img-версия_ядра`, а файл `vmlinuz-версия_ядра` распаковывает себя сам. После этого управление передается ядру ОС.

Настройка GRUB2

Работу кода загрузчика из образов `core.img` или `grubx64.efi` определяет файл с конфигурацией `/boot/grub/grub.cfg`. По своей сути, файл `/boot/grub/grub.cfg` — это довольно сложный sh-скрипт.

Если мы откроем этот файл, то увидим предупреждение о том, что этот файл автоматически создан командой `grub-mkconfig` на основе шаблонов из каталога `/etc/grub.d/` и файла с настройками `/etc/default/grub`.

Таким образом, внесение изменений напрямую в файл `/boot/grub/grub.cfg` — не только довольно сложная и сопряженная с риском ошибок задача, но и внесенные руками изменения будут перезаписаны при очередном запуске команды `update-grub`.

Шаблоны grub

Каталог `/etc/grub.d/` содержит шаблоны, которые могут быть использованы в качестве опций в загрузочном меню. Шаблоны — это сложные sh-скрипты. Цифра в начале шаблона определяет порядок его запуска при формировании нового конфигурационного файла `grub.cfg`. Как правило, добавление новых шаблонов и их редактирование не требуется. Для редактирования (в случае необходимости изменения загрузочного меню) предназначен файл `/etc/grub.d/40_custom`.

```
root@astra:/boot# ls -l /etc/grub.d
итого 92
-rwxr-xr-x 1 root root 10046 окт 6 2022 00_header
-rwxr-xr-x 1 root root 6260 авг 1 2022 05_debian_theme
-rwxr-xr-x 1 root root 353 мая 16 12:06 07_password
-rwxr-xr-x 1 root root 14282 окт 6 2022 10_linux
-rwxr-xr-x 1 root root 14180 окт 6 2022 20_linux_xen
-rwxr-xr-x 1 root root 12910 окт 6 2022 30_os-prober
-rwxr-xr-x 1 root root 1372 окт 6 2022 30_uefi-firmware
-rwxr-xr-x 1 root root 214 окт 6 2022 40_custom
-rwxr-xr-x 1 root root 215 окт 6 2022 41_custom
-rwxr-xr-x 1 root root 370 окт 15 2020 99_kcm_grub2
-rw-r--r-- 1 root root 483 окт 6 2022 README
```

Параметры grub

Файл `/etc/default/grub` содержит набор параметров (переменных) с комментариями, которыми можно управлять работой загрузчика. Посмотрим его содержимое:

```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

#GRUB_DEFAULT=0
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="parsec. max_ilev=63 quiet net.ifnames=0"
GRUB_CMDLINE_LINUX=""
GRUB_CMDLINE_LINUX_HARDENED="slub_debug=P page_poison=1 slab_nomerge pti=on
user.$"

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo`
GRUB_GFXMODE=1024x768
```

```
# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"

# Uncomment to get a beep at grub start
#GRUB_INIT_TUNE="480 440 1"
GRUB_DISABLE_SUBMENU=y
GRUB_DEFAULT=gnulinux-5.15.0-70-generic-advanced-eba34003-adda-47f3-a50d-8b9513e$
```

Вот некоторые параметры:

- **GRUB_TIMEOUT** - определяет продолжительность отображения загрузочного меню;
- **GRUB_DEFAULT** - пункт загрузочного меню по умолчанию;
- **GRUB_CMDLINE_LINUX** - значение этого параметра будет передано в конец строки linux (в старой версии GRUB эти параметры использовались в строке kernel) как при нормальной загрузке, так и при загрузке в режиме восстановления;
- **GRUB_DISABLE_LINUX_RECOVERY=true** - для того, чтобы избежать появления в меню режима Recovery. Если нужен режим Recovery только для конкретно взятого ядра, нужно создать необходимую запись в скрипте `/etc/grub/40_custom`.

Просмотр новой конфигурации grub и обновление файла grub.cfg

Команда `grub-mkconfig` на основании шаблонов (скриптов) из `/etc/grub.d/` и файла с параметрами `/etc/default/grub` формирует текст конфигурационного файла и выводит его на экран.

Для создания (перезаписи) конфигурационного скрипта GRUB `/boot/grub/grub.cfg` необходимо выполнить команду `update-grub` (или `grub-mkconfig -o /boot/grub/grub.cfg`). Команда `update-grub` представляет собой скрипт:

```
#!/bin/sh
set -e
exec grub-mkconfig -o /boot/grub/grub.cfg "$@"
```

Примечание

Дополнительную информацию об изменении параметров загрузчика GRUB2 можно найти по ссылке: <https://wiki.astralinux.ru/pages/viewpage.action?pageId=18874503>.

По ссылке ниже вы можете ознакомиться с инструкцией по установке GRUB2 после замены таблицы разделов на устройстве хранения с MBR на GPT <https://wiki.astralinux.ru/pages/viewpage.action?pageId=130424699>.

Запуск ядра ОС

Файл образа ядра `/boot/vmlinuz-версия_ядра` представляет собой архивный загрузочный исполнимый образ. Если мы посмотрим на тип файла командой `file`, то мы увидим следующее:

```
sa@astra:~/initrd/initramfs$ sudo file /boot/vmlinuz-5.15.0-111-generic
/boot/vmlinuz-5.15.0-111-generic: Linux kernel x86 boot executable bzImage, version 5.15.0-111-generic (builder@runner-6zppmmke-project-1752-concurrent-0) #astra2+ci1 SMP Mon Jul, RO-rootFS, swap_dev 0xA, Normal VGA
```

На самом деле это не совсем ядро, точнее не только оно. Содержимое файла `/boot/vmlinuz*` представляет собой заголовок + код настройки ядра + vmlinux (сжатое ядро), что превращает его в своего рода самораспаковывающийся архив, а название файла образа ядра состоит из **vm** - виртуальная память (наследие с зари разработки Linux), **linu** - часть linux, **z** - сокращение от «zipped» и номера версии ядра.

Ядро запускается функцией `start_kernel()`. При этом инициализируются наиболее важные функции, такие как системные вызовы, планировщик задач и другие, а также создаются все необходимые потоки ядра.

После загруженный в память образ `initrd` (за что отвечает GRUB) монтируется в корень в качестве временной файловой системы (за что отвечает ядро ОС), имеющей необходимый набор драйверов. Это позволяет работать с различным оборудованием, сохранять небольшой размер ядра и не пересобирать ядро под ту или иную конфигурацию аппаратного обеспечения. Такой механизм обеспечивается модульностью ядра Linux.

Содержимое RAM-диска

Содержимое RAM-диска представляет собой FHS-совместимую структуру каталогов, `busybox`, необходимые модули ядра и драйверы устройств. `BusyBox` - это набор UNIX-утилит командной строки, содержащихся в одном файле. Командным интерпретатором в нем выступает `ash`, совместимый с `sh`.

Посмотреть содержимое файла `initrd.img` можно командой `lsinitramfs /boot/initrd.img-$(uname -r)`, где: команда `$(uname -r)` выведет текущую версию ядра и имя ядра (`generic`).

```
sa@astra:$ lsinitramfs /boot/initrd.img-5.15.0-111-generic | head
.
bin
conf
conf/arch.conf
conf/conf.d
conf/conf.d/resume
```

```
conf/conf.d/splash
conf/initramfs.conf
conf/modules
etc
```

Для распаковки и изучения содержимого необходимо скопировать файл `initrd.img-$(uname -r)`, произвести декомпрессию и скопировать файлы из архива командами:

```
sa@astra:~$ mkdir initrd && cd initrd && cp /boot/initrd.img-$(uname -r) .
sa@astra:~/initrd$ mkdir initramfs && cd initramfs
sa@astra:~/initrd/initramfs$ gzip -cd ../initrd.img-$(uname -r) | cpio -idmv
. . .
cpio: создана ссылка usr/sbin/watchdog, указывающая на usr/sbin/uevent
cpio: создана ссылка usr/sbin/watchdog, указывающая на usr/sbin/vconfig
usr/sbin/watchdog
usr/share
usr/share/plymouth
usr/share/plymouth/plymouthd.defaults
usr/share/plymouth/themes
usr/share/plymouth/themes/details
usr/share/plymouth/themes/details/details.plymouth
usr/share/plymouth/themes/text
usr/share/plymouth/themes/text/text.plymouth
637151 блок

sa@astra:~/initrd/initramfs$ ls -l
итого 28
lrwxrwxrwx 1 sa sa 7 апр 7 16:47 bin -> usr/bin
drwxr-xr-x 3 sa sa 4096 апр 7 16:47 conf
drwxr-xr-x 7 sa sa 4096 апр 7 16:47 etc
-rwxr-xr-x 1 sa sa 6338 авг 23 2019 init
lrwxrwxrwx 1 sa sa 7 апр 7 16:47 lib -> usr/lib
lrwxrwxrwx 1 sa sa 9 апр 7 16:47 lib32 -> usr/lib32
lrwxrwxrwx 1 sa sa 9 апр 7 16:47 lib64 -> usr/lib64
lrwxrwxrwx 1 sa sa 10 апр 7 16:47 libx32 -> usr/libx32
drwxr-xr-x 2 sa sa 4096 мар 29 14:39 run
lrwxrwxrwx 1 sa sa 8 апр 7 16:47 sbin -> usr/sbin
drwxr-xr-x 8 sa sa 4096 апр 7 16:47 scripts
drwxr-xr-x 10 sa sa 4096 апр 7 16:48 usr
```

Запуск первого процесса /sbin/init с PID=0

Ядро ОС запускает скрипт `/init`, который, вызывая скрипты из каталога `/scripts`, определяет тип устройств хранения информации и файловых систем на компьютере, производит различные проверки, подгружает необходимые для них драйверы, после чего перемонтирует в качестве корня реальную файловую систему с системного диска в режиме только на чтение. Файловая система монтируется только на чтение для исключения случаев её дальнейшей деградации, если она была повреждена.

После этого скрипт запускает файл `/sbin/init`, который становится первым процессом в дереве процессов пользовательского пространства с PID=1 (как мы выяснили ранее, это символическая ссылка на `/lib/systemd/systemd`, что связано с историческим наследием и обратной совместимостью с прошлой системой инициализации systemV). С этого момента и до появления приглашения на ввод учётных данных для входа в систему процессом будет управлять подсистема инициализации systemd.

Подсистема инициализации и управления службами systemd

При запуске файла `/lib/systemd/systemd` ядро искусственно (вспомним, что рождение первого процесса происходит особым образом) создаст в своих структурах одноименный процесс с PID=1 и PPID=0. Это родитель всех остальных процессов в пользовательском пространстве. Systemd - это подсистема инициализации и управления службами, представленная комплектом базовых компонентов Linux-системы.

Подсистема systemd пришла на смену системе systemV (system-пять), в которой первый процесс именовался init. SystemV – это очень старая модель, которая имеет целый ряд недостатков, главными из которых являются медленная работа и сложность в настройке зависимостей между демонами. Systemd, созданная Леннартом Поттерингом и другими сотрудниками компании Red Hat как свободное ПО под лицензией GNU, решила эти проблемы (при этом часть концепций systemd была взята у системы инициализации Mac OS launchd).

Systemd оперирует модулями (англ. unit) - универсальными единицами, отвечающими за различные типы объектов, такие как служба (англ. service), монтирование (англ. mount), цель (англ. target), устройство (англ. device) и другие. Реализованы механизмы иерархии целевых состояний и зависимостей внутри модулей, запуск службы по запросу (англ. on-demand), сокеты и D-bus активация запуска служб. Это позволило интенсивно распараллелить запуск служб в процессе загрузки системы и контролировать очередность их запуска. Кроме этого, были введены механизмы контроля и изоляции процессов служб - механизмы пространства имен (англ. namespaces) и контрольных групп Linux (англ. cgroups).

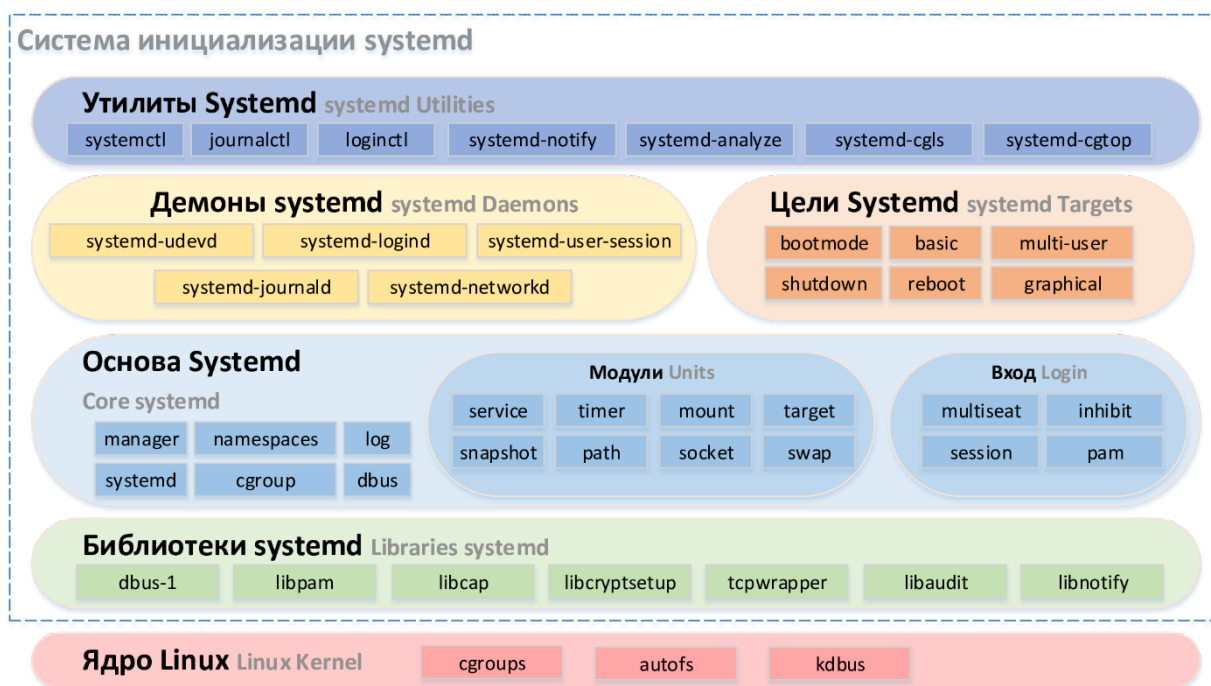
На производительность systemd наибольшее влияние оказало введение механизма модулей в целом, сокетов и автмонтирование файловых систем в частности. В системе systemV вместо модулей применялся набор скриптов, при выполнении которых могли вызываться множество команд и нерационально использоваться ресурсоёмкие утилиты, например, такие как awk, что негативно сказывалось на производительности. Введение модулей (стандартизированных конфигурационных файлов) позволило оптимизировать этот процесс. Кроме того, с распараллеливанием загрузки служб были проблемы как у systemV, так и у Upstart (система, призванная заменить systemV, которая их решала, но не в полном объеме). Созданные заранее сокеты с механизмами буферизации позволили убрать зависимости между сервисами, один из которых пишет информацию в сокет, а другой читает. Более того, этот механизм позволил перезапускать службы без потери данных - после запуска сервис сможет разобрать информацию, накопившуюся в очереди

сокета. Монтирование файловых систем по запросу (при первом обращении к файловой системе) тоже позволило сократить время загрузки ОС.

Примечание

В systemd вместо классического для Linux названия «демон» применяется название служба, так привычное уху администратора систем Windows. Концепции служб systemd в Linux и служб в Windows с первого взгляда чем-то похожи. Мы можем управлять их запуском, настраивать зависимости от других служб, запускать службы по событиям, однако, различий, обусловленных разными архитектурными решениями в системах, довольно много.

На рисунке ниже представлена вся система инициализации systemd. Рассмотрим подробнее каждый из её компонентов.



Система инициализации systemd

Модули systemd

В systemd существуют следующие типы модулей:

- **.target** - позволяет группировать модули, воплощая концепцию уровней запуска;
- **.service** - отвечает за запуск сервисов (служб), также поддерживает вызов интерпретаторов для исполнения пользовательских скриптов;
- **.mount** - отвечает за монтирование файловых систем;

- **.automount** - позволяет отложить монтирование файловых систем до фактического обращения к точке монтирования;
- **.swap** - отвечает за подключение файла или устройства подкачки;
- **.timer** - позволяет запускать модули по расписанию;
- **.socket** - предоставляет службам поддержку механизма сокет-активации;
- **.slice** - отвечает за создание контейнера `cgroupr` - группы иерархически организованных модулей, управляющих системными процессами;
- **.device** - позволяет реагировать на подключение устройств;
- **.path** - позволяет запускать модули при появлении пути в файловой системе;
- **.snapshot** - позволяет сохранять состояние менеджера `systemd`.

Каждый из таких модулей отвечает за отдельно взятый объект (службу, файловую систему, файл подкачки, сокет) или является вспомогательной частью системы `systemd`, позволяющей реализовать её преимущества (цель, устройство, таймер и другие).

Некоторые модули могут иметь в своем имени символ «@». Это означает, что этот файл является шаблоном - он не учитывается системой, а при создании нового модуля можно воспользоваться этим шаблоном, скопировав его, переименовав и изменив под свои задачи.

Модули в файловой системе представлены конфигурационными файлами. Основные места их размещения:

- Каталог `/usr/lib/systemd/system/` — модули, созданные при установке ОС;
- Каталог `/run/systemd/system/` — модули, созданные в рантайме. Этот каталог приоритетнее каталога с модулями, созданными при установке ОС;
- Каталог `/etc/systemd/system/` — модули, созданные и управляемые системным администратором. Этот каталог приоритетнее каталога модулей, созданных в рантайме.

Полный список возможных каталогов размещения модулей доступен в выводе команды `systemctl show --property=UnitPath`.

Ниже представлено содержимое каталогов `/usr/lib/systemd/system/` и `/etc/systemd/system/`

```
sa@astra:~$ ls -l /usr/lib/systemd/system
итого 1000
-rw-r--r-- 1 root root 115 ноя 22 2018 acpid.path
-rw-r--r-- 1 root root 234 ноя 22 2018 acpid.service
-rw-r--r-- 1 root root 115 ноя 22 2018 acpid.socket
-rw-r--r-- 1 root root 157 апр 19 2020 acpi-support.service
-rw-r--r-- 1 root root 502 мар 30 2019 alsa-restore.service
-rw-r--r-- 1 root root 475 мар 30 2019 alsa-state.service
lrwxrwxrwx 1 root root 9 мар 30 2019 alsa-utils.service -> /dev/null
-rw-r--r-- 1 root root 776 мая 19 2019 anacron.service
. . .
```

```

lrwxrwxrwx 1 root root 15 июн 14 2024 runlevel0.target -> poweroff.target
lrwxrwxrwx 1 root root 13 июн 14 2024 runlevel1.target -> rescue.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel1.target.wants
lrwxrwxrwx 1 root root 17 июн 14 2024 runlevel2.target -> multi-user.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel2.target.wants
lrwxrwxrwx 1 root root 17 июн 14 2024 runlevel3.target -> multi-user.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel3.target.wants
lrwxrwxrwx 1 root root 17 июн 14 2024 runlevel4.target -> multi-user.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel4.target.wants
lrwxrwxrwx 1 root root 16 июн 14 2024 runlevel5.target -> graphical.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel5.target.wants
lrwxrwxrwx 1 root root 13 июн 14 2024 runlevel6.target -> reboot.target
lrwxrwxrwx 1 root root 9 июн 14 2024 sendsigs.service -> /dev/null
-rw-r--r-- 1 root root 1486 июн 14 2024 serial-getty@.service
-rw-r--r-- 1 root root 442 фев 14 2019 shutdown.target
-rw-r--r-- 1 root root 402 фев 14 2019 sigpwr.target
. . .
-rw-r--r-- 1 root root 432 сен 19 2024 wpa_supplicant.service
-rw-r--r-- 1 root root 559 сен 19 2024 wpa_supplicant@.service
-rw-r--r-- 1 root root 591 сен 19 2024 wpa_supplicant-wired@.service
lrwxrwxrwx 1 root root 9 июн 14 2024 x11-common.service -> /dev/null

```

```
sa@astra:~$ ls -l /etc/systemd/system
```

```
итого 72
```

```

-rw-r--r-- 1 root root 207 мар 22 2023 astra-mount-lock.service
lrwxrwxrwx 1 root root 9 мар 29 14:26 astra-orientation.service -> /dev/null
-rw-r--r-- 1 root root 176 мар 22 2023 astra-pttrace-lock.service
drwxr-xr-x 2 root root 4096 мар 22 2023 astra-safepolicy.target.wants
drwxr-xr-x 2 root root 4096 мар 28 14:05 ceph_mgr@.service.d
drwxr-xr-x 2 root root 4096 мар 28 14:05 ceph-mon@.service.d
drwxr-xr-x 2 root root 4096 мар 28 14:05 ceph-osd@.service.d
lrwxrwxrwx 1 root root 42 мар 22 2023 dbus-fi.wl.wpa_supplicant1.service -> /li
b/systemd/system/wpa_supplicant.service
lrwxrwxrwx 1 root root 40 мар 22 2023 dbus-org.freedesktop.Avahi.service -> /li
b/systemd/system/avahi-daemon.service
lrwxrwxrwx 1 root root 53 мар 22 2023 dbus-org.freedesktop.nm-dispatcher.servic
e -> /lib/systemd/system/NetworkManager-dispatcher.service
lrwxrwxrwx 1 root root 47 мар 29 14:36 dbus-org.rnd2.cpuspower_gui.helper.service
-> /lib/systemd/system/cpuspower-gui-helper.service
lrwxrwxrwx 1 root root 34 мар 29 11:48 display-manager.service -> /lib/systemd/s
ystem/fly-dm.service
drwxr-xr-x 2 root root 4096 мар 22 2023 getty.target.wants
drwxr-xr-x 2 root root 4096 мар 22 2023 graphical.target.wants
drwxr-xr-x 2 root root 4096 мар 29 14:36 multi-user.target.wants
drwxr-xr-x 2 root root 4096 мар 22 2023 network-online.target.wants
drwxr-xr-x 2 root root 4096 мар 22 2023 printer.target.wants
drwxr-xr-x 2 root root 4096 мар 22 2023 sockets.target.wants
-rw-r--r-- 1 root root 1107 окт 5 2022 suspend-to-hibernate.service
-rw-r--r-- 1 root root 1096 окт 5 2022 suspend-to-poweroff.service
drwxr-xr-x 2 root root 4096 мар 28 14:20 sysinit.target.wants
drwxr-xr-x 2 root root 4096 мар 29 12:51 timers.target.wants
-rw-r--r-- 1 root root 178 мая 5 2022 usbipd.service
drwxr-xr-x 2 root root 4096 мар 28 14:20 vboxadd.service.d

```

Цели systemd

Файлы целей systemd `*.target` предназначены для группировки модулей через цепочку зависимостей. Эта группировка позволяет оперировать не просто уровнями загрузки, как это было в `init`, а набором целевых состояний системы, которые можно создавать и изменять при необходимости. Это позволяет, с одной стороны, поделить классические уровни загрузки на отдельные цели, которые можно использовать отдельно друг от друга, а с другой - создавать свои собственные цели, при которых те или иные группы сервисов запускаются или останавливаются (учитывая все их зависимости).

Каждая цель может включать в себя зависимости от различных модулей (как правило это сервисы и цели). Эти зависимости устанавливаются через параметры модуля цели и через каталоги `<имя_цели>.target.wants`.

```
sa@astra:~$ ls -l /etc/systemd/system/multi-user.target.wants/
итого 4
lrwxrwxrwx 1 root root 35 map 22 2023 anacron.service -> /lib/systemd/system/anacron.service
lrwxrwxrwx 1 root root 63 map 29 12:51 astra-event-diagnostics-healthcheck.service->
/lib/systemd/system/astra-event-diagnostics-healthcheck.service
lrwxrwxrwx 1 root root 45 map 22 2023 astra-orientation.service -> /lib/systemd/system/astra-orientation.service
. . .
lrwxrwxrwx 1 root root 42 map 22 2023 wpa_supplicant.service -> /lib/systemd/system/wpa_supplicant.service
```

Целевые состояния системы и уровни загрузки

В классическом `init` было 6 уровней загрузки (*англ. runlevel*):

- **runlevel 0** – остановка системы;
- **runlevel 1** – загрузка в однопользовательском режиме, в системе выполняется минимальный набор программ. Также существует уровень S, который служит для ввода пароля root;
- **runlevel 2** – загрузка в многопользовательском режиме без поддержки сети;
- **runlevel 3** – загрузка в многопользовательском режиме с поддержкой сети;
- **runlevel 4** – не используется;
- **runlevel 5** – загрузка в многопользовательском режиме с поддержкой сети и графическим входом в систему;
- **runlevel 6** - перезагрузка.

Для перехода между этими уровнями использовались скрипты и команды, иницирующие переход - `runlevel <уровень_загрузки>` и `telinit <уровень_загрузки>`.

Systemd реализует эти уровни (для обратной совместимости) с помощью символических ссылок `runlevel<уровень_загрузки>.target` на цели со схожим назначением. Поэтому команды `runlevel` и `telinit`, используемые в `init`, по-прежнему доступны.

Целевые состояния системы, использующиеся в `systemd`, и их соответствие уровням загрузки:

- Файл `poweroff.target` – команда `runlevel 0`;
- Файл `rescue.target` – команда `runlevel 1`, `runlevel S`;
- Файл `multi-user.target` – команда `runlevel 2`, `runlevel 3`;
- Файл `graphical.target` – команда `runlevel 5`;
- Файл `reboot.target` – команда `runlevel 6`.

```
sa@astra:~$ ls -l /usr/lib/systemd/system | grep runlevel
lrwxrwxrwx 1 root root 15 июн 14 2024 runlevel0.target -> poweroff.target
lrwxrwxrwx 1 root root 13 июн 14 2024 runlevel1.target -> rescue.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel1.target.wants
lrwxrwxrwx 1 root root 17 июн 14 2024 runlevel2.target -> multi-user.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel2.target.wants
lrwxrwxrwx 1 root root 17 июн 14 2024 runlevel3.target -> multi-user.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel3.target.wants
lrwxrwxrwx 1 root root 17 июн 14 2024 runlevel4.target -> multi-user.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel4.target.wants
lrwxrwxrwx 1 root root 16 июн 14 2024 runlevel5.target -> graphical.target
drwxr-xr-x 2 root root 4096 окт 3 2022 runlevel5.target.wants
lrwxrwxrwx 1 root root 13 июн 14 2024 runlevel6.target -> reboot.target
-rw-r--r-- 1 root root 797 июн 14 2024 systemd-update-utmp-runlevel.service
```

Для управления целевым состоянием системы в systemd применяются команды:

- Команда `sudo systemctl isolate <имя>.target` – переключит систему в выбранное целевое состояние, например:
 - Команда `sudo systemctl isolate reboot.target` – перезагрузит систему,
 - Команда `sudo systemctl isolate multi-user.target` – переведет в режим работы без графического интерфейса.
- Команда `systemctl get-default` – выведет имя цели по умолчанию.
- Команда `systemctl set-default --force <имя>.target` – изменит цель по умолчанию на выбранную.

Устройство конфигурационного файла модуля службы

Синтаксис файла модуля напоминает формат конфигурационных файлов ini в Windows. Рассмотрим содержимое одного из модулей служб. Возьмем конфигурационный файл `logind.conf` модуля `systemd-logind.service` и посмотрим его содержимое.

```
# SPDX-License-Identifier: LGPL-2.1+
#
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Login Service
Documentation=man:systemd-logind.service(8) man:logind.conf(5)
Documentation=https://www.freedesktop.org/wiki/Software/systemd/logind
Documentation=https://www.freedesktop.org/wiki/Software/systemd/multiseat
Wants=user.slice
After=nss-user-lookup.target user.slice
ConditionPathExists=/lib/systemd/system/dbus.service

# Ask for the dbus socket.
Wants=dbus.socket
After=dbus.socket

[Service]
```

```

BusName=org.freedesktop.login1
CapabilityBoundingSet=CAP_SYS_ADMIN CAP_MAC_ADMIN CAP_AUDIT_CONTROL CAP_CHOWN CAP_KILL
CAP_DAC_READ_SEARCH CAP_DAC_OV
ERRIDE CAP_FOWNER CAP_SYS_TTY_CONFIG
ExecStart=/lib/systemd/systemd-logind
FileDescriptorStoreMax=512
IPAddressDeny=any
LockPersonality=yes
MemoryDenyWriteExecute=yes
NoNewPrivileges=yes
Restart=always
RestartSec=0
RestrictAddressFamilies=AF_UNIX AF_NETLINK
RestrictNamespaces=yes
RestrictRealtime=yes
RestrictSUIDSGID=yes
SystemCallArchitectures=native
SystemCallErrorNumber=EPERM
SystemCallFilter=@system-service
WatchdogSec=3min

# Increase the default a bit in order to allow many simultaneous logins since
# we keep one fd open per session.
LimitNOFILE=524288

```

В файле в квадратных скобках обозначаются разделы. Для модуля службы существует 3 стандартных раздела:

- **[Unit]** – обычно его используют для определения метаданных и настройки отношения к другим модулям;
- **[Service]** – используется для предоставления конфигурации, которая применима только для служб;
- **[Install]** – Этот раздел является дополнительным и используется для определения условий запуска модуля.

Для модулей других типов вместо секции **[Service]** используются другие: **[Socket]**, **[Mount]**, **[Automount]**, **[Swap]**, **[Timer]**, **[Slice]**.

В каждом из этих разделов существует набор директив, с помощью которых определяется назначение и поведение модуля. Ниже представлены основные директивы.

Секция **[Unit]**

В разделе **[Unit]** можно использовать следующие директивы:

- **Description** – эта директива может использоваться для описания имени и основных функций модуля. Это значение возвращается различными инструментами systemd, поэтому полезно установить что-то короткое, конкретное и информативное.
- **Documentation** – эта директива предоставляет местоположения документации. Это могут быть внутренние доступные справочные страницы либо доступные в интернете (URL-адреса). Команда `systemctl status <имя_сервиса>` выведет эту информацию.

- **Requires** – в данной директиве перечислены все модули, от которых зависит этот модуль (служба или устройство). Если текущий модуль должен быть запущен, перечисленные здесь модули так же должны успешно запуститься. В противном случае это вызывает ошибку запуска текущего модуля. По умолчанию эти модули запускаются параллельно с текущим модулем.

- **Wants** – эта директива похожа на `Requires=`, но менее строгая. `Systemd` будет пытаться запустить любые модули, перечисленные здесь, когда запускается текущий модуль. Если эти модули не обнаружены или не запущены, текущий модуль просто продолжит запуск. Это рекомендуемый способ настройки большинства зависимостей. По умолчанию эти модули запускаются параллельно с текущим модулем.

- **Bindsto** – эта директива аналогична `Requires=`, но также приводит к остановке текущего модуля, когда соответствующий модуль останавливается.

- **Before** - модули, перечисленные в этой директиве, не будут запущены до тех пор, пока текущий модуль не будет отмечен как запущенный, если они будут запускаться одновременно.

- **After** - Модули, перечисленные в этой директиве, будут запущены до запуска текущего модуля.

- **Conflicts** - используется для задания модулей, которые нельзя запускать одновременно с текущим модулем. Запуск модуля с этой связью приведет к остановке других модулей.

- **Condition<условие>** - существует ряд директив, начинающихся с условия, которые позволяют администратору проверить некоторые состояния системы и модулей до запуска устройства, например: *ConditionFileNotEmpty* или *ConditionPathExists*. Если условие не выполнено, запуск модуля будет отменен без ошибки.

- **Assert<условие>** – подобен набору директив `Condition...`, но установленные директивы проверяют различные аспекты рабочей среды, чтобы решить, следует ли запустить модуль, например: *AssertPathExists*. Однако, в отличие от директив `Condition...`, отрицательный результат вызывает ошибку запуска текущего модуля.

Секция [Install]

В разделе [Install] можно использовать следующие директивы:

- **WantedBy** – Данная директива является наиболее распространенным способом определения того, как модуль должен быть запущен. Эта директива позволяет указать зависимость (аналогично директиве `Wants` в разделе [Unit]). Разница в том, что эта директива включена во вспомогательную секцию, позволяющую секции [Unit] оставаться относительно чистой. `Systemd` автоматически создает каталоги

вида `имя_модуля.target_модуля.wants` в каталоге с файлом модуля и добавляет эти зависимости в виде символических ссылок в каталоге `...wants`.

- **RequiredBy** – эта директива очень похожа на директиву `WantedBy`, но это строгая зависимость. И в случае невозможности запуска соответствующего модуля запуск текущего модуля будет завершён с ошибкой.

- **Alias** – эта директива позволяет запустить модуль, используя другое имя.

- **Also** – эта директива позволяет включать или отключать сразу набор модулей. Здесь можно указать сопутствующие модули, которые всегда должны быть доступны, когда этот модуль запущен.

Секция [Service]

В разделе [Service] можно использовать следующие директивы:

- **Type** – тип службы:

- **simple** – тип службы по умолчанию: `systemd` запустит эту службу незамедлительно. Процесс при этом не должен разветвляться (`fork`). Если после данной службы должны запускаться другие, то этот тип использовать не стоит (исключение — если служба использует сокетную активацию).

- **forking** – `systemd` считает службу запущенной после того, как процесс разветвляется с завершением родительского процесса. Используется для запуска классических демонов за исключением тех случаев, когда в таком поведении процесса нет необходимости. Хорошо подходит, например, для запуска `nginx`, `tomcat`.

- **oneshot** – удобен для сценариев, которые выполняют одно задание и завершаются. Если задать параметр `RemainAfterExit=yes`, то `systemd` будет считать процесс активным даже после его завершения (например, при монтировании).

- **dbus** – служба считается находящейся в состоянии готовности после появления указанного `BusName` в системной шине `DBus`.

- **notify** – идентичен параметру `Type=simple`, но с уточнением, что демон пошлет `systemd` сигнал готовности. Реализация уведомления находится в библиотеке `libsystemd-daemon.so`.

- **idle** – `systemd` отложит выполнение двоичного файла службы до окончания запуска остальных («более срочных») задач. В остальном поведение аналогично `Type=simple`.

- **User** – указание пользователя, от имени которого будет запущена служба.

- **ExecStart** – директива, где нужно указать полный путь и аргументы команды, которая должна быть выполнена для запуска процесса. Эту директиву можно указать только один раз.

- **ExecStartPre** – директива, где нужно указать полный путь и аргументы команды, которые должны быть выполнены до запуска основного процесса. Эту директиву можно использовать несколько раз.

- **ExecStartPost** – аналогична ExecStartPre за исключением того, что данная директива указывает команды, которые будут запускаться после запуска основного процесса.
- **ExecReload** – эта необязательная директива содержит команду, необходимую для перезагрузки конфигурации службы (если она доступна).
- **ExecStop** – содержит команду, необходимую для остановки службы. Если она не указана, процесс будет просто уничтожен во время остановки службы.
- **ExecStopPost** – содержит команду, выполняемую после остановки службы.
- **Slice** – содержит имя контрольной группы.
- **KillMode** – управляет способ завершения главного и дочерних процессов. Может принимать значения: **control-group** - (значение по умолчанию) завершит работу всех процессов контрольной группы, **mixed** - передаст сигнал SIGTERM главному и SIGKILL остальным процессам контрольной группы, **process** - завершит только главный процесс (не рекомендуется использовать).
- **RestartSec** – если автоматический перезапуск службы включен, задает время ожидания перед попыткой перезапуска службы.
- **Restart** – директива определяет условия, при которых systemd будет пытаться автоматически перезапустить службу. Может принимать значения: «always», «on-success», «on-failure», «on-abnormal», «on-abort» или «on-watchdog».
- **TimeoutSec** – устанавливает время, в течение которого система будет ждать старта или остановки службы, прежде чем пометить ее запуск как неудавшийся или остановку как принудительно убитую. Можете установить отдельные таймауты с помощью директив TimeoutStartSec и TimeoutStopSec.

Создание модуля службы

Создадим модуль службы для программы `/usr/bin/fly-message-monitor`, которая позволяет отправлять сообщения на рабочий стол пользователя. Для создания модуля службы запустим текстовый редактор с правами суперпользователя, чтобы открыть (создать) файл `/etc/systemd/system/fly-message-monitor.service`:

```
sa@astra:~$ sudo nano /etc/systemd/system/fly-message-monitor.service
```

Наполнить его следующим содержимым (если ваш пользователь назван не sa, а как-то иначе, то необходимо указать имя учетной записи, используемой вами):

```
[Unit]
Description=Сервис отправки уведомлений на рабочий стол
After=graphical.target

[Service]
ExecStart=/usr/local/bin/fly-message-monitor-sa
Type=simple
Restart=always
RestartSec=0
Environment="DISPLAY=:0" "XAUTHORITY=/home/sa/.Xauthority"
```

```
[Install]
WantedBy=graphical.target
```

Тут мы видим три характерные для файла службы секции Unit, Service и Install.

- В секции **[Unit]** мы используем две директивы:
 - **Description** для задания описания службы.
 - **After** для указания, что этот сервис должен запускаться только после загрузки сервисов, входящих в целевое состояние системы `graphical.target`, когда у нас будут доступны службы рабочего стола.
- В секции **[Service]**:
 - через директиву **ExecStart** мы задаем путь до исполнимого файла.
 - тип службы (Type) выберем Simple.
 - директивы перезапуска службы **Restart** и **RestartSec** выберем `always` и `0` для немедленного запуска службы в случае её падения.
 - директива **Environment** позволяет задавать или переопределять переменные для процесса сервиса. Переменные `DISPLAY` и `XAUTHORITY` необходимы, чтобы сообщения, отправляемые сервисом, доходили до рабочего стола сессии пользователя `sa`.
- В секции **[Install]** зададим только одну директиву.
 - **WantedBy** – говорим сервису, что для его работы необходимо целевое состояние системы `graphical.target`.

Создадим файл скрипта `/usr/local/bin/fly-message-monitor-sa` путем копирования файла `/usr/bin/fly-message-monitor` и последующего его изменения.

Скопируем файл `/usr/bin/fly-message-monitor`, отредактируем `/usr/local/bin/fly-message-monitor-sa`

```
sa@astra:~$ sudo cp /usr/bin/fly-message-monitor /usr/local/bin/fly-message-monitor-
```

Заменим цикл `while` в конце файла:

```
#wait changes in log file
while true ; do
    if [ -s $LOG_FILE ]; then
        MSG="empty"
        MSG=$(tail -10 $LOG_FILE)
        echo "$MSG" > ~/txt.txt
#       notify-send -u critical -t 10000 -i dialog-warning "$MESSAGE" "$MSG"
#       fly-dialog --caption "$MESSAGE" --msgbox "$MSG"
        fly-dialog --caption "$MESSAGE" --textbox ~/txt.txt
        rm ~/txt.txt
    fi

    #wait for changes
    inotifywait -qq -r -e modify $LOG_FILE &
    wait $!

    #if dir or file are deleted then wait for it again
    if [ ! -f $LOG_FILE ]; then
        wait_for_file
    fi
done
```

```

    #restart
    pkill inotifywait
fi
done

```

на:

```

#wait changes in log file
while true ; do
    inotifywait -qq -r -e modify $LOG_FILE &
    wait $!
    MSG=""; TRAY_MSG=""
    URGENCY=$(head -1 $LOG_FILE)
    TYPE=$(sed -n '2p' $LOG_FILE)
    MSG=$(tail -1 $LOG_FILE)
    notify-send -u $URGENCY -t 4950 -i $TYPE "$MESSAGE" "$MSG"
done

```

тем самым позволив управлять не только выводимым сообщением, но и его типом, а также изменив продолжительность отображения сообщения на экране.

Сохраним файл службы, а затем перечитаем конфигурацию сервисов в системе командой

```
sa@astra:~$ sudo systemctl daemon-reload
```

Запустим нашу службу fly-message-monitor.service командой

```
sa@astra:~$ sudo systemctl start fly-message-monitor.service
```

Разрешим автозапуск службы при переходе между целевыми состояниями командой

```
sa@astra:~$ sudo systemctl enable fly-message-monitor.service
Created symlink /etc/systemd/system/graphical.target.wants/fly-message-monitor.service → /etc/systemd/system/fly-message-monitor.s
```

Посмотрим состояние службы командой

```
sa@astra:~$ systemctl status fly-message-monitor.service
● fly-message-monitor.service - Сервис отправки уведомлений на рабочий стол
   Loaded: loaded (/etc/systemd/system/fly-message-monitor.service; enabled; vendor preset:
   Active: active (running) since Wed 2025-04-09 17:27:56 MSK; 39s ago
 Main PID: 31190 (fly-message-mon)
    Tasks: 2 (limit: 2233)
   Memory: 768.0K
      CPU: 27ms
   CGroup: /system.slice/fly-message-monitor.service
           └─31190 /bin/bash /usr/local/bin/fly-message-monitor-sa
             └─31197 inotifywait -qq -r -e create /tmp/fly-msg
```

Проверим, работает ли служба и скрипт, как ожидалось. Отправим несколько сообщений:

```

sa@astra:~$ echo -en "critical \ndialog-error \nХьюстон, у нас проблемы!" | sudo tee /tmp/fly-
msg/msg.txt
sa@astra:~$ echo -en "critical \ndialog-warning\nШеф, все пропало!" | sudo tee /tmp/fly-
msg/msg.txt
sa@astra:~$ echo -en "normal \ndialog-information\nТеперь все в порядке!" | sudo tee /tmp/fly-
msg/msg.txt

```

```
sa@astra:~$ echo -en "critical \ndialog-error \nХьюстон, у нас проблемы!" | sudo tee /tmp/fly-msg/msg.txt
critical
dialog-error
Хьюстон, у нас проблемы!sa@astra:~$
```



Внимание, сообщение!
Хьюстон, у нас проблемы!

```
sa@astra:~$ echo -en "critical \ndialog-warning\nШеф, все пропало!" | sudo tee /tmp/fly-msg/msg.txt
critical
dialog-warning
Шеф, все пропало!sa@astra:~$
```



Внимание, сообщение!
Шеф, все пропало!

```
sa@astra:~$ echo -en "normal \ndialog-information\nТеперь все в порядке!" | sudo tee /tmp/fly-msg/msg.txt
normal
dialog-information
Теперь все в порядке!sa@astra:~$
```



Внимание, сообщение!
Теперь все в порядке!

Создадим еще одну службу `/etc/systemd/system/resource-demanding1.service`, которая будет запускать скрипт, воспроизводящий высокую нагрузку на CPU. Но перед этим подготовим скрипт, который будет запускать эта служба.

От имени суперпользователя создадим в каталоге `/usr/local/bin/` (который предназначен в том числе для хранения скриптов, запускаемых на уровне всей системы) скрипт `resource-demanding1.sh` и предоставим владельцу право на его выполнение.

```
#!/bin/bash
/usr/bin/sha256sum /dev/urandom > /dev/null
```

```
sa@astra:~$ sudo -i
[sudo] пароль для sa:
root@astra:~# echo "/usr/bin/sha256sum /dev/urandom > /dev/null" > /usr/local/bin/resource-demanding1.sh
root@astra:~# chmod u+x /usr/local/bin/resource-demanding1.sh
root@astra:~# exit
выход
sa@astra:~$ cat /usr/local/bin/resource-demanding1.sh
/usr/bin/sha256sum /dev/urandom > /dev/null
sa@astra:~$ ls -l /usr/local/bin/resource-demanding1.sh
-rwxr--r-- 1 root staff 44 anp  9 17:41 /usr/local/bin/resource-demanding1.sh
```

Создадим файл службы `/etc/systemd/system/resource-demanding1.service` со следующей конфигурацией:

```
[Unit]
Description=Сервис, воспроизводящий высокую нагрузку на CPU 0
After=graphical.target

[Service]
ExecStart=taskset 0x00000001 /usr/bin/bash /usr/local/bin/resource-demanding1.sh
Type=simple

[Install]
WantedBy=graphical.target
```

Этот сервис будет запускать скрипт `/usr/local/bin/resource-demanding1.sh` только на первом процессоре.

```
sa@astra:~$ sudo nano /etc/systemd/system/resource-demanding1.service
sa@astra:~$ sudo systemctl daemon-reload
sa@astra:~$ sudo systemctl start resource-demanding1.service
sa@astra:~$ systemctl status resource-demanding1.service
● resource-demanding1.service - Сервис, воспроизводящий высокую нагрузку на CPU 0
   Loaded: loaded (/etc/systemd/system/resource-demanding1.service; disabled; vendor prese
   Active: active (running) since Wed 2025-04-09 17:51:25 MSK; 9s ago
 Main PID: 31426 (bash)
    Tasks: 2 (limit: 2233)
   Memory: 656.0K
   CGroup: /system.slice/resource-demanding1.service
           └─2778 /usr/bin/bash /usr/local/bin/resource-demanding1.sh
             └─2779 /usr/bin/sha256sum /dev/urandom
```

PID	USER	PRI	NI	VRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2779	root	20	0	5284	740	676	R	93.7	0.0	6:28.48	/usr/bin/sha256sum /dev/urandom

Оставим службу `resource-demanding1.service` «disabled», то есть ей не разрешен автоматический запуск.

```
sa@astra:~$ sudo systemctl stop resource-demanding1.service
sa@astra:~$ sudo systemctl disabled resource-demanding1.service
```

Механизм пространства имен namespaces и контрольных групп в systemd

Пространство имен в Linux

Пространство имён (*англ. namespaces*) - это механизм ядра Linux, позволяющий изолировать и виртуализировать глобальные системные ресурсы множества процессов. Примеры ресурсов, которые можно виртуализировать: ID процессов, имена хостов, ID пользователей, доступ к сетям, межпроцессное взаимодействие и файловые системы. Одной из общих целей пространств имён является поддержка реализации контейнеров - инструмента для виртуализации на уровне операционной системы (а также других целей), обеспечивающего группу процессов иллюзией того, что они являются единственными процессами в системе. Поэтому одной из главных целей пространства имён является поддержка контейнеризации в Linux.

Linux-система при старте инициализирует один экземпляр каждого типа пространства имён, кроме пространства имён файловой системы. После инициализации можно создать или объединить дополнительные пространства имён.

Все пространства имён поддерживают вложенность, то есть между ними можно установить связь «родитель — потомок». Таким образом, некоторые пространства наследуют все свойства от своего родительского пространства имён. Однако это верно не для всех пространств.

Функциональные возможности пространства имён одинаковы для всех типов: каждый процесс связан с пространством имён и может видеть или использовать только ресурсы, связанные с этим пространством имён, и, где это применимо, с его потомками. Таким образом, каждый процесс (или его группа) может иметь уникальное представление о ресурсе. Изолированный ресурс зависит от типа пространства имён, созданного для данной группы процессов.

В Linux существует 7 типов пространств имен:

- **PID** (англ. *PID namespace*) – изоляция ID процессов - позволяет процессам из разных пространств имен одинаковые PID;
- **сети** (англ. *net namespace*) – изоляция системных ресурсов, связанных с сетями: сетевое оборудование, стеки протоколов IPv4 и IPv6, таблицы IP-маршрутизации, фильтрации, номера портов (сокеты) и другие;
- **UTS** (англ. *UTS namespace, Unix Time Sharing*) – позволяет каждому контейнеру иметь свое имя хоста и имя домена;
- **пользователя** (англ. *user namespace*) – изолируют ID пользователей и групп, корневой каталог, ключи;
- **файловой системы** (англ. *mnt namespace*) – обеспечивает независимость дерева файловой системы, ассоциированное с определённой группой процессов. Каждое пространство имён предоставляет уникальный вид файловой системы для всех процессов, принадлежащих этому пространству;
- **межпроцессорное взаимодействие** (англ. *IPC namespace*) – обеспечивает изоляцию процессов при обращении к разделяемым ресурсам, например, памяти. Процессам каждого пространства имен IPC видны только объекты, принадлежащие к этому пространству;
- **контрольные группы** (англ. *cgroup namespace*) – обеспечивают изоляцию и возможность накладывать ограничения на некоторые ресурсы компьютера: процессорные, сетевые, память, ввода-вывода. Механизм позволяет образовывать иерархические группы процессов с заданными ресурсными свойствами и обеспечивает программное управление ими.

Контрольные группы в Linux

Изначально этот механизм ядра Linux носил название контейнер процессов, но из-за путаницы с другими решениями по контейнеризации был переименован в механизм контрольных групп (англ. *cgroups*). Его предназначение - разделять и ограничивать некоторые ресурсы компьютера между процессами. Сейчас используется вторая версия этого механизма.

Сами контрольные группы находятся в псевдофайловой системе sysfs `/sys/fs/cgroup`. Одни контрольные группы можно включать в другие контрольные группы и тем самым

агрегировать их. Контрольная группа представляет собой каталог с набором файлов, где имя файла — это имя параметра, а текст внутри файла — его значение.

Контрольные группы могут управлять доступом к следующим ресурсам:

- **cpu** — ограничение выделяемой доли процессорного времени по заданному значению, если система загружена. Не ограничивает процессорные ресурсы, если система свободна;
- **cpuset** — эту cgroup можно использовать для привязки процессов в контрольной группе к указанному набору ЦП и узлов NUMA;
- **freezer** — может приостанавливать и возобновлять работу всех процессов в контрольной группе. Заморозка контрольной группы так же влияет на её потомков;
- **io** - контролирует и ограничивает доступ к заданным блочным устройствам, применяет управление вводом-выводом посредством пропусков (throttling) и ограничения сверху листовых узлов и промежуточных узлов в иерархии хранилища;
- **memory** - контроллер памяти поддерживает учёт и ограничение памяти процесса, памяти ядра и подкачки, используемой контрольной группой;
- **pids** - контроллер позволяет ограничивать количество процессов, которые могут быть созданы в контрольной группе (и её потомках);
- **rdma** (remote direct memory access) - Контроллер RDMA позволяет ограничивать использование ресурсов RDMA/IB определённой контрольной группе.

Для получения информации о том, в каких контрольных группах находятся процессы, можно воспользоваться командой `ps xawf -eo pid,user,cgroup,args`.

```
sa@astra:~$ sudo ps xawf -eo pid,user,cgroup,args
PID USER      CGROUP          COMMAND
  2 root        -               [kthreadd]
  3 root        -               _ [rcu_gp]
...
  1 root        0::/init.scope  /sbin/init
259 root        0::/system.slice/systemd-journal /lib/systemd/systemd-journald
 296 root        0::/system.slice/systemd-udevd /lib/systemd/systemd-udevd
 318 root        0::/system.slice/auditd.ser /sbin/auditd
 436 root        0::/system.slice/cron.servi /usr/sbin/cron -f
 441 avahi        0::/system.slice/avahi-daem avahi-daemon: running [astra.local]
 452 avahi        0::/system.slice/avahi-daem \_ avahi-daemon: chroot helper
 446 root        0::/system.slice/systemd-lo /lib/systemd/systemd-logind
 449 root        0::/system.slice/udisks2.se /usr/lib/udisks2/udisksd
 454 message+ 0::/system.slice/dbus.servi /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
 460 root        0::/system.slice/NetworkMan /usr/sbin/NetworkManager --no-daemon
 463 root        0::/system.slice/wpa_suppli /sbin/wpa_supplicant -u -s -O DIR=/run/wpa_supplicant GROUP=netdev
...
```

А с помощью команды `systemd-cgls` можно увидеть контрольные группы и входящие в них процессы в древовидной структуре.

```
sa@astra:~$ systemd-cgls
```

```
Control group /:
-.slice
├─user.slice
│   └─user-107.slice
│       └─user@107.service
│           └─init.scope
│               ├──1535 /lib/systemd/systemd --user
│               └──1539 (sd-pam)
│   └─session-c1.scope
│       └─1503 /usr/lib/xorg/Xorg -br -novtswitch -quiet -keeptty :0 vt7 -logfile /var/log/f
├─user-1000.slice
│   └─user@1000.service
│       └─init.scope
│           ├──9140 /lib/systemd/systemd --user
│           └──9143 (sd-pam)
│   └─session-3.scope
│       └─2029 -:0
...

```

Посмотрим содержимое каталога `/sys/fs/cgroup/system.slice`.

```
sa@astra:~$ ls /sys/fs/cgroup/system.slice/
alsa-restore.service
alsa-state.service
astra-orientation.service
auditd.service
avahi-daemon.service
avahi-daemon.socket
cgroup.controllers
cgroup.events
cgroup.freeze
cgroup.kill
cgroup.max.depth
cgroup.max.descendants
. . .

```

Посмотрим содержимое каталога, относящегося к сервису аудита.

```
sa@astra:~$ ls /sys/fs/cgroup/system.slice/auditd.service/
cgroup.controllers      cgroup.subtree_control  memory.events.local    memory.stat
cgroup.events           cgroup.threads          memory.high             memory.swap.current
cgroup.freeze           cgroup.type             memory.low              memory.swap.events
cgroup.kill             cpu.pressure            memory.max              memory.swap.high
cgroup.max.depth        cpu.stat                memory.min              memory.swap.max
cgroup.max.descendants    io.pressure             memory.numa_stat        pids.current
cgroup.procs            memory.current           memory.oom.group        pids.events
cgroup.stat             memory.events            memory.pressure         pids.max

```

В этих файлах хранится статистика (например, `cpu.stat`) и ограничения использования ресурсов.

Управление контрольными группами сервисов

Контрольную группу для сервиса можно назначить прямо через его конфигурационный файл с помощью директивы `Slice: Slice={Имя_контрольной_группы}`. По умолчанию имя контрольной группы сервиса совпадает с именем сервиса.

Контрольную группу и параметры контроля можно задать с помощью встраиваемых файлов. Для этого:

- В каталоге `/etc/systemd/system/` необходимо создать одноименный сервису каталог с суффиксом «.d», например, для сервиса `apache2` это будет `/etc/systemd/system/apache2.service.d/`.

- Внутри этого каталога создать файлы с расширением `.conf` (тут рассмотрен пример ограничения по процессорному времени):

Файл `00-slice.conf`

```
[Service]
Slice=AWESOME.slice
MemoryAccounting=yes
CPUAcounting=yes
```

Где:

- **Slice** задаёт – имя контрольной группы,
- **MemoryAccounting** и **CPUAcounting** – включает подсчет потребления ресурсов памяти и процессора соответственно.

Файл `10-CPUSettings.conf`

```
[Service]
CPUWeight=1000
```

Где:

- **CPUWeight=1000** – устанавливает вес процессов этой контрольной группы. Чем вес больше, тем больше приоритет (по умолчанию вес 100, а параметр может принимать значения от 1 до 10000).

Для применения этих настроек необходимо перечитать конфигурационные файлы службы командой `sudo systemctl daemon-reload` и перезапустить службу `sudo systemctl restart apache2`.

Установим ограничение на контрольную группу созданного нами сервиса `resource-demanding1.service` по использованию процессорного времени. Для этого создадим каталог `/etc/systemd/system/resource-demanding1.service.d` и создадим в нем файлы:

Файл `00-slice.conf` со следующим содержимым:

```
[Service]
MemoryAccounting=yes
CPUAcounting=yes
```

Файл `10-CPUSettings.conf` со следующим содержимым:

```
[Service]
CPUQuota=90%
```

Где:

- **CPUQuota** – устанавливает лимит потребления процессом ресурсов процессора.

```
sa@astra:~$ sudo mkdir /etc/systemd/system/resource-demanding1.service.d
sa@astra:~$ sudo nano /etc/systemd/system/resource-demanding1.service.d/00-slice.conf
sa@astra:~$ sudo nano /etc/systemd/system/resource-demanding1.service.d/10-CPUSettings.conf
sa@astra:~$ cat /etc/systemd/system/resource-demanding1.service.d/00-slice.conf
[Service]
MemoryAccounting=yes
CPUAccounting=yes
sa@astra:~$ cat /etc/systemd/system/resource-demanding1.service.d/10-CPUSettings.conf
[Service]
CPUWeight=90%
```

Перечитаем файлы конфигурации и перезапустим сервис `resource-demanding1.service`.

```
sa@astra:~$ sudo systemctl daemon-reload
sa@astra:~$ sudo systemctl restart resource-demanding1.service
sa@astra:~$ systemctl status resource-demanding1.service
● resource-demanding1.service - Сервис, воспроизводящий высокую нагрузку на CPU 0
   Loaded: loaded (/etc/systemd/system/resource-demanding1.service; disabled; vendor prese
   Drop-In: /etc/systemd/system/resource-demanding1.service.d
            └─00-slice.conf, 10-CPUSettings.conf
   Active: active (running) since Wed 2025-04-09 18:10:18 MSK; 8s ago
   Main PID: 31610 (bash)
     Tasks: 2 (limit: 2233)
    Memory: 552.0K
         CPU: 10.366s
    CGroup: /system.slice/resource-demanding1.service
            └─2838 /usr/bin/bash /usr/local/bin/resource-demanding1.sh
               └─2839 /usr/bin/sha256sum /dev/urandom
```

Как видим, введенное нами ограничение вступило в силу и теперь процессы в группе не могут потреблять более чем 90% процессорного времени, что подтверждает `htop`:

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2839	root	20	0	5284	740	680	R	88.4	0.0	3:37.43	/usr/bin/sha256sum /dev/urandom

Больше информации о контрольных группах для служб можно найти в справке `man 5 systemd.slice` и `man 5 systemd.resource-control`.

Примечание

Подробнее о groups можно почитать по ссылке: <https://manpages.debian.org/unstable/manpages-ru/cgroups.7.ru.html>

Управление модулями и другими компонентами systemd

Для управления модулями systemd существует ряд утилит:

- Утилита `systemctl` - управления службами, модулями и целевыми состояниями системы;

- Утилита `journalctl` – утилита просмотра журналов, сформированных сервисом `journald`. Позволяет гибко настраивать вывод через параметры;
 - Утилита `systemd-notify` - используется модулями `.notify` для оповещения службы `systemd` об их статусе;
 - Утилита `systemd-analyze` – позволяет производить анализ времени запуска служб;
 - Утилита `systemd-cgls` – выводит дерево контрольных групп (`cgroups`);
 - Утилита `systemd-cgtop` - аналог утилиты `top` для контрольных групп;
 - Утилита `loginctl` – инструмент управления менеджером `systemd-logind`.
- Рассмотрим применение каждой из этих утилит подробнее.

Systemctl

Утилита предназначена для управления службами и целевыми состояниями системы.

Команды для управления целевыми состояниями системы:

- Команда `sudo systemctl isolate <имя>.target` – переключит систему в выбранное целевое состояние, например, `sudo systemctl isolate reboot.target` перезагрузит систему, а `sudo systemctl isolate multi-user.target` переведет в режим работы без графического интерфейса.
- Команда `systemctl get-default` – выведет имя цели по умолчанию.
- Команда `systemctl set-default --force <имя>.target` – изменит цель по умолчанию на выбранную.

Команды для просмотра списка модулей:

- Команда `systemctl` – выводит список всех запущенных модулей;
- Команда `systemctl -a` – выводит список всех модулей;
- Команда `systemctl --type=<тип_модуля>` – выводит список всех модулей соответствующего типа.

Команды для просмотра, редактирования и создания конфигурационного файла модулей:

- Команда `systemctl cat <имя_модуля>` – выводит содержимое конфигурационного файла модуля;
- Команда `systemctl --full edit <имя_модуля>` – редактирование конфигурационного файла модуля;
- Команда `systemctl --force --full edit <имя_модуля>` – создание нового модуля.

Команды для управления сервисами:

- Команда `systemctl status <имя_сервиса>.service` – выводит информацию о статусе службы;
- Команда `systemctl enable <имя_сервиса>.service` – добавляет службу в автозагрузку;
- Команда `systemctl disable <имя_сервиса>.service` – удаляет службу из автозагрузки;
- Команда `systemctl start <имя_сервиса>.service` – запускает службу;
- Команда `systemctl stop <имя_сервиса>.service` – останавливает службу;
- Команда `systemctl restart <имя_сервиса>.service` – перезапускает службу;
- Команда `systemctl reload <имя_сервиса>.service` – перечитывает конфигурационный файл службы;
- Команда `systemctl mask <имя_сервиса>.service` – маскирует службу. Делает невозможным запуск службы как в ручном режиме, так и через зависимость от другой службы. В каталоге `/etc/systemd/system` при этом создается символической ссылкой на `/dev/null`;
- Команда `systemctl unmask <имя_сервиса>.service` – отменяет маскировку службы. Возвращает возможность запуска службы. Ссылка на `/dev/null` при этом удаляется;
- Команда `systemctl reenable <имя_сервиса>.service` – выполняет `disable`, а затем `enable` службы;
- Команда `systemctl is-active <имя_сервиса>.service` – проверяет, запущена ли служба в настоящий момент;
- Команда `systemctl list-unit-files --type=service` – выводит список всех служб и их состояние.

```
sa@astra:~$ systemctl list-unit-files --type=service
```

UNIT FILE	STATE
acpi-support.service	disabled
acpid.service	disabled
alsa-restore.service	static
alsa-state.service	static
alsa-utils.service	masked
anacron.service	enabled
apt-daily-upgrade.service	static
apt-daily.service	static
astra-event-diagnostics-healthcheck.service	enabled
astra-event-diagnostics.service	static
astra-mount-lock.service	disabled
astra-orientation.service	masked
astra-pttrace-lock.service	enabled
auditd.service	enabled
autovt@.service	enabled
avahi-daemon.service	enabled
bootlogd.service	masked
bootlogs.service	masked
bootmisc.service	masked
checkfs.service	masked
checkroot-bootclean.service	masked
checkroot.service	masked
clean-mount-point@.service	static

```
sa@astra:~$ systemctl status cups.service
```

- cups.service - CUPS Scheduler
 - Loaded: loaded (/lib/systemd/system/cups.service; enabled; vendor preset: enabl
 - Active: active (running) since Wed 2025-04-09 19:07:05 MSK; 3min 46s ago
 - Docs: man:cupsd(8)

```
Main PID: 582 (cupsd)
Status: "Scheduler is running..."
Tasks: 1 (limit: 2233)
Memory: 8.5M
CPU: 106ms
CGroup: /system.slice/cups.service
└─582 /usr/sbin/cupsd -l
```

```
sa@astra:~$ sudo systemctl stop cups.service
```

```
sa@astra:~$ systemctl status cups.service
```

```
● cups.service - CUPS Scheduler
Loaded: loaded (/lib/systemd/system/cups.service; enabled; vendor preset: enabl
Active: inactive (dead) since Wed 2025-04-09 19:11:49 MSK; 16s ago
```

```
sa@astra:~$ sudo systemctl mask cups.service
```

```
Created symlink /etc/systemd/system/cups.service → /dev/null.
```

```
sa@astra:~$ ls -l /lib/systemd/system/cups.service
```

```
-rw-r--r-- 1 root root 278 anp  5 17:04 /lib/systemd/system/cups.service
```

```
sa@astra:~$ ls -l /etc/systemd/system/cups.service
```

```
lrwxrwxrwx 1 root root 9 anp  9 19:13 /etc/systemd/system/cups.service -> /dev/null
```

```
sa@astra:~$ sudo systemctl start cups.service
```

```
Failed to start cups.service: Unit cups.service is masked.
```

```
sa@astra:~$ sudo systemctl unmask cups.service
```

```
Removed /etc/systemd/system/cups.service.
```

```
sa@astra:~$ ls -l /etc/systemd/system/cups.service
```

```
ls: невозможно получить доступ к '/etc/systemd/system/cups.service': Нет такого файла или каталога
```

```
sa@astra:~$ sudo systemctl start cups.service
```

```
sa@astra:~$ systemctl status cups.service
```

```
● cups.service - CUPS Scheduler
  Loaded: loaded (/lib/systemd/system/cups.service; enabled; vendor preset: enabl
  Active: active (running) since Wed 2025-04-09 19:15:56 MSK; 8s ago
    Docs: man:cupsd(8)
Main PID: 1947 (cupsd)
Status: "Scheduler is running..."
Tasks: 1 (limit: 2233)
Memory: 4.1M
CPU: 89ms
CGroup: /system.slice/cups.service
└─1947 /usr/sbin/cupsd -l
```

```
sa@astra:~$ sudo systemctl disable cups.service
```

```
Synchronizing state of cups.service with SysV service script with /lib/systemd/systemd-sysv-install.
```

```
Executing: /lib/systemd/systemd-sysv-install disable cups
```

```
Removed /etc/systemd/system/printer.target.wants/cups.service.
```

```
Removed /etc/systemd/system/sockets.target.wants/cups.socket.
```

```
Removed /etc/systemd/system/multi-user.target.wants/cups.path.
```

```
sa@astra:~$ sudo systemctl enable cups.service
```

```
Synchronizing state of cups.service with SysV service script with /lib/systemd/systemd-sysv-install.
```

```
Executing: /lib/systemd/systemd-sysv-install enable cups
```

```
Created symlink /etc/systemd/system/printer.target.wants/cups.service →
```

```
/lib/systemd/system/cups.service.
```

```
Created symlink /etc/systemd/system/sockets.target.wants/cups.socket →
```

```
/lib/systemd/system/cups.socket.
```

```
Created symlink /etc/systemd/system/multi-user.target.wants/cups.path →
```

```
/lib/systemd/system/cups.path.
```

Journalctl

Утилита предназначена для просмотра журналов, сформированных сервисом journald. Позволяет гибко настраивать вывод через параметры. Содержит большое количество параметров.

Примеры использования:

Команда `sudo journalctl -u cups.service -o short` – выводит информацию из журналов о сервисе печати cups в сокращённом виде.

```
sa@astra:~$ sudo journalctl -u cups.service -o short
-- Logs begin at Wed 2025-04-09 19:07:01 MSK, end at Wed 2025-04-09 19:18:48 MSK.
анр 09 19:07:04 astra systemd[1]: Starting CUPS Scheduler...
анр 09 19:07:05 astra astraevents[582]: type="server_started" desc="Планировщик за
анр 09 19:07:05 astra systemd[1]: Started CUPS Scheduler.
анр 09 19:11:49 astra astraevents[582]: type="server_stopped" desc="Планировщик yc
анр 09 19:11:49 astra systemd[1]: Stopping CUPS Scheduler...
анр 09 19:11:49 astra systemd[1]: cups.service: Succeeded.
анр 09 19:11:49 astra systemd[1]: Stopped CUPS Scheduler.
анр 09 19:11:49 astra systemd[1]: cups.service: Consumed 118ms CPU time.
анр 09 19:15:56 astra systemd[1]: Starting CUPS Scheduler...
анр 09 19:15:56 astra astraevents[1947]: type="server_started" desc="Планировщик з
анр 09 19:15:56 astra systemd[1]: Started CUPS Scheduler.
```

Команда `sudo journalctl -u cups.service -o json --since "2023-08-22 11:55:00" > cups.log.json` – записывает в файл информацию о событиях с сервисом cups, произошедших после 11:55 8 марта 2025 года в формате json.

```
sa@astra:~$ sudo journalctl -u cups.service -o json --since "2025-03-08 11:55:00" >
cups.log.json
sa@astra:~$ cat cups.log.json
{ "_SYSTEMD_UNIT": "cups.service", "_SYSTEMD_SLICE": "system.slice", "_HOSTNAME": "astra",
  "_SYSTEMD_INVOCATION_ID": "b871c800dfc244298e540889c86fe500", "SYSLOG_IDENTIFIER":
  "astraevents", "PRIORITY": "5", "_SOURCE_REALTIME_TIMESTAMP": "1744215356389025", "_GID": "0",
  "_CAP_EFFECTIVE": "1fffffffff", "_COMM": "cupsd", "_TRANSPORT": "syslog", "_SELINUX_CONTEXT": "unknown",
  "_EXE": "/usr/sbin/cupsd", "_MACHINE_ID": "08d89582f4de48ad844cd451eafb5725", "SYSLOG_TIMESTAMP": "Apr  9 19:15:56 ",
  "_BOOT_ID": "283d17686217410bafeca5804bb37860", "_SYSTEMD_CGROUP": "/system.slice/cups.service", "_UID": "0",
  "__MONOTONIC_TIMESTAMP": "540262054", "__CURSOR": "s=62781b705fd14721abefd1d32d4ba061;i=e1b;b=283d17686217410bafeca5804bb37860;m=2033bea6;t=6325ac8bab6b5;x=dd15f0ae231bcf42",
  "SYSLOG_FACILITY": "3", "MESSAGE": "type=\"server_started\" desc=\"Планировщик запущен по требованию.\"\"",
  "__REALTIME_TIMESTAMP": "1744215356389045", "_CMDLINE": "/usr/sbin/cupsd -l", "_PID": "1947" }
```

Команда `sudo journalctl -f` – выводит на экран информацию обо всех поступающих в реальном времени событиях.

Команда `sudo journalctl -f -u cups.service` – выводит на экран информацию обо всех поступающих в реальном времени событиях, связанных с сервисом cups.

Systemd-analyze

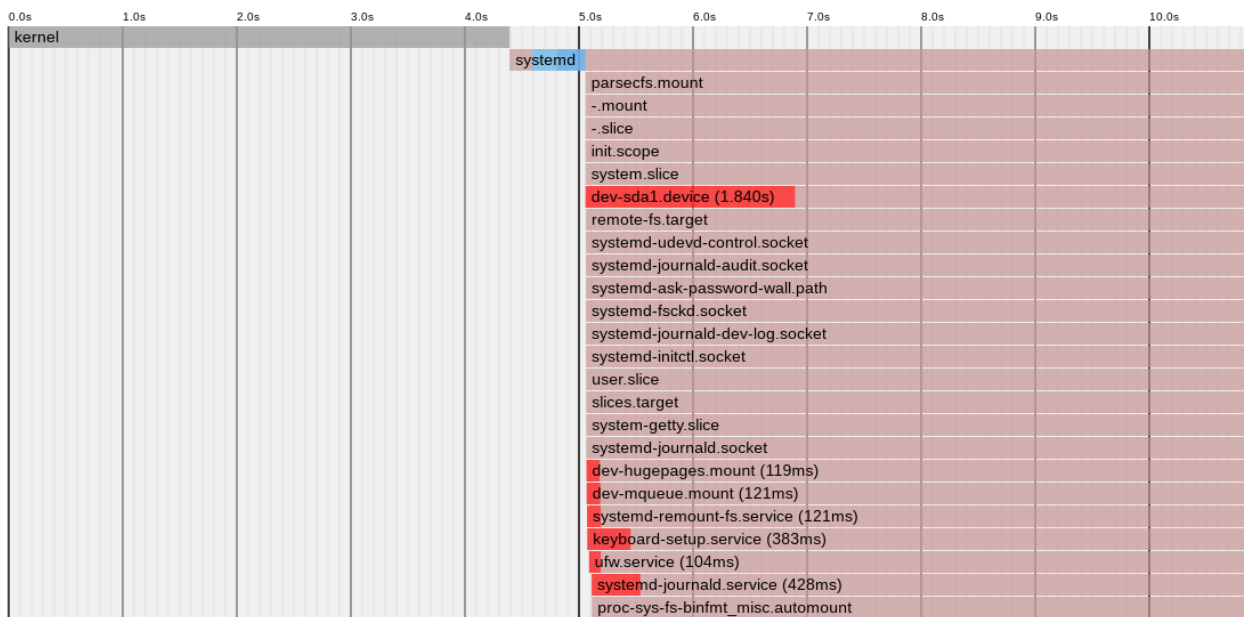
Утилита `systemd-analyze` позволяет производить анализ времени запуска служб.

```
sa@astra:~$ systemd-analyze
Startup finished in 4.919s (kernel) + 8.295s (userspace) = 13.214s
graphical.target reached after 8.245s in userspace
```

Вывод утилиты можно сохранить в файл с расширением. svg, просматривать, например, в браузере chromium:

```
sa@astra:~$ systemd-analyze plot > plot.svg
sa@astra:~$ chromium plot.svg
```

Astra Linux astra (Linux 5.15.0-111-generic #astra2+ci1 SMP Mon Jul 22 13:27:51 UTC 2024) x86-64 oracle
Startup finished in 4.391s (kernel) + 8.542s (userspace) = 12.934s graphical.target reached after 8.098s in userspace



Анализ времени запуска служб

Systemd-cgls

Утилита `systemd-cgls` выводит дерево контрольных групп (cgroups).

```
sa@astra:~$ systemd-cgls
Control group /:
-.slice
├─user.slice
│   └─user-107.slice
│       └─user@107.service
│           └─init.scope
│               ├── 990 /lib/systemd/systemd --user
│               └─992 (sd-pam)
│   └─session-c2.slice
│       └─1467 /usr/lib/xorg/Xorg -br -novtswitch -quiet -keeptty :0 vt7 -logfile /v
├─user-1000.slice
│   └─user@1000.service
│       └─init.scope
│           ├── 1527 /lib/systemd/systemd --user
│           └─1528 (sd-pam)
```

```

└─session-5.scope
  └─1471 -:0
  └─1545 fly-wm
  └─1617 /usr/bin/dbus-launch --exit-with-session --sh-syntax
  └─1618 /usr/bin/dbus-daemon --syslog --fork --print-pid 5 --print-address 7
  └─1641 /usr/bin/kglobalaccel5
. . .

```

Systemd-cgtop

Утилита `systemd-cgtop` — аналог утилиты `top` для контрольных групп.

```

sa@astra:~$ systemd-cgls
Control group /:
-.slice
Control Group                               Tasks    %CPU    Memory  Input/s  Output/s
/                                             271      1,0     1.1G    -        -
/user.slice                                139      8,2     724.7M  -        -
/user.slice/user-1000.slice                 135      8,0     438.1M  -        -
/user.slice/u...1000.slice/session-5.scope  133      7,8     435.4M  -        -
/system.slice                               58       0,2     294.5M  -        -
/user.slice/user-107.slice                   4       0,2     136.7M  -        -
/user.slice/u...107.slice/session-c2.scope   2       0,2     61.7M   -        -
/system.slice/vboxadd-service.service        14      0,1      2.5M   -        -
/system.slice/libflygetexe-bin.service        1       0,1     352.0K  -        -
/system.slice/syslog-ng.service               11      0,0     45.9M   -        -
/init.scope                                  1       -        8.0M   -        -
/system.slice/NetworkManager.service         3       -     17.0M   -        -
/system.slice...ostics-healthcheck.service   1       -     26.6M   -        -
/system.slice/auditd.service                  2       -      4.0M   -        -
/system.slice/avahi-daemon.service            2       -      1.3M   -        -
/system.slice/cron.service                    1       -     10.9M   -        -

```

loginctl

Утилита управления менеджером systemd-logind.

- Команда `loginctl` — выводит список сессий.

```

sa@astra:~$ loginctl
SESSION UID USER  SEAT  TTY
  5 1000 sa      seat0 tty7
 c2 107 fly-dm seat0 tty7
2 sessions listed.

```

- Команда `loginctl session-status <номер_сессии>` — выводит статус сессии по её номеру.

```

sa@astra:~$ loginctl session-status 5
5 - sa (1000)
  Since: Wed 2025-04-09 19:08:18 MSK; 25min ago
  Leader: 1471 (fly-dm)
  Seat: seat0; vc7
  TTY: tty7
  Service: fly-dm; type x11; class user
  State: active
  Unit: session-5.scope
    └─1471 -:0

```

```
—1545 fly-wm
—1617 /usr/bin/dbus-launch --exit-with-session --sh-syntax
—1618 /usr/bin/dbus-daemon --syslog --fork --print-pid 5 --prin
—1641 /usr/bin/kglobalaccel5
—1648 /usr/bin/VBoxClient --clipboard
```

• Команда `loginctl show-session` — выводит свойства самого менеджера (без аргументов) или указанной сессии.

```
• sa@astra:~$ loginctl show-session
• EnableWallMessages=no
  NAutoVTs=6
  KillUserProcesses=yes
  RebootToFirmwareSetup=no
  IdleHint=no
  IdleSinceHint=0
  IdleSinceHintMonotonic=0
  BlockInhibited=handle-power-key:handle-suspend-key:handle-hibernate-key:handle-lid
  DelayInhibited=shutdown:sleep
  InhibitDelayMaxUSec=5s
  UserStopDelayUSec=10s
  HandlePowerKey=ignore
  HandleSuspendKey=suspend
  HandleHibernateKey=hibernate
  HandleLidSwitch=suspend
  HandleLidSwitchDocked=ignore
  HoldoffTimeoutUSec=30s
  IdleAction=ignore
  IdleActionUSec=30min
  PreparingForShutdown=no
  PreparingForSleep=no
  Docked=no
  LidClosed=no
  OnExternalPower=yes
```

• Команда `loginctl lock-session/unlock-session имя_сессии` — активирует/деактивирует блокировку экрана в указанной сессии.

• Команда `loginctl kill-session имя_сессии` — отправляет сигнал SIGKILL одному или нескольким процессам сессии.

```
sa@astra:~$ loginctl list-sessions
SESSION UID USER SEAT TTY
  5 1000 sa seat0 tty7
  c2 107 fly-dm seat0 tty7
2 sessions listed.
```

Практические задания

Задание 1.

1. Перезагрузите тестовую виртуальную машину и в момент её запуска измените команды запуска так, чтобы отключить «тихую» загрузку ОС.
2. Перенастройте загрузчик, чтобы при последующих запусках ОС «тихий» режим загрузки ОС был отключен, а тайм-аут отображения меню загрузки GRUB был 7 секунд. Убедитесь, что настройки применились.

3. Выведите на экран целевое состояние системы по умолчанию. Перезагрузите машину, используя целевое состояние systemd.
4. Выведите на экран сообщения ядра ОС от начала загрузки и ознакомьтесь с ними.

Задание 2.

Для успешного прохождения этого задания у виртуальной машины должны быть два CPU.

1. Если вы не создали модули службы fly-message-monitor.service и resource-demanding1.service (раздел 17.6) в процессе изучения материала модуля, то создайте их, в том числе настройте контрольную группу для службы resource-demanding1.service.
2. Создайте службу **resource-demanding2.service** по аналогии со службой resource-demanding1.service, но так, чтобы скрипт, который она запускает, назывался **resource-demanding2.sh** и запускался на **втором процессоре**.
3. Создайте скрипт **/usr/local/bin/cpu_usage.sh** со следующим содержанием:

```
#!/bin/bash
CPU_THRESHOLD_WARNING=85 # %
CPU_THRESHOLD_CRITICAL=95 # %
CPU_THRESHOLD_PERIOD=5 # секунд
CPU_THRESHOLD_ITERATION=3 # раз
MSG_FILE="/tmp/fly-msg/msg.txt"

while ;; do
# Get the first line with aggregate of all CPUs
cpu_now=$(head -n1 /proc/stat)
# Get all columns but skip the first (which is the "cpu" string)
cpu_sum="${cpu_now[@]:1}"
# Replace the column separator (space) with +
cpu_sum=$(( ${cpu_sum// /+} ))
# Get the delta between two reads
cpu_delta=$((cpu_sum - cpu_last_sum))
# Get the idle time Delta
cpu_idle=$((cpu_now[4] - cpu_last[4]))
# Calc time spent working
cpu_used=$((cpu_delta - cpu_idle))
# Calc percentage
cpu_usage=$((100 \* cpu_used / cpu_delta))

# Keep this as last for our next read
cpu_last=("${cpu_now[@]}")
cpu_last_sum=$cpu_sum

if [[ "$cpu_usage" -ge "$CPU_THRESHOLD_WARNING" ]]; then
((cpu_warning_count++))
if [[ "$cpu_usage" -ge "$CPU_THRESHOLD_CRITICAL" ]]; then
((cpu_critical_count++))
else
cpu_critical_count=0
fi
else
cpu_warning_count=0
cpu_critical_count=0
fi

msg_critical="critical\\ndialog-warning\\n<H1>Критически высокая нагрузка на
```

```

CPU!</H1><p>Использование CPU $cpu_usage%\t$(date)</p>"
msg_warning="critical\\ndialog-warning\\n<H1>Высокая нагрузка на CPU!</H1><p>Использование CPU
$cpu_usage%\t$(date)</p>"
msg_ok="normal\\ndialog-information\\n<H1>Нагрузка на CPU в пределах
нормы</H1><p>Использование CPU
$cpu_usage%\t$(date)</p>"

if [[ cpu_critical_count -ge CPU_THRESHOLD_ITERATION ]]; then
    echo -en $msg_critical > $MSG_FILE
    send_ok=1
elif [[ cpu_warning_count -ge CPU_THRESHOLD_ITERATION ]]; then
    echo -en $msg_warning > $MSG_FILE
    send_ok=1
elif [[ send_ok -eq 1 ]]; then
    echo -en $msg_ok > $MSG_FILE
    send_ok=0
fi

# Wait a second before the next read
sleep $CPU_THRESHOLD_PERIOD

echo "CPU usage at $cpu_usage%"
echo "Warning counter $cpu_warning_count"
echo "Critical counter $cpu_critical_count"
done

```

Не забудьте сделать файл исполнимым.

4. Создайте модуль службы и назовите его **cpu-monitor.service**, чтобы он запускал скрипт **/usr/local/bin/cpu_usage.sh** и стартовал автоматически при загрузке ОС только при цели **graphical.target**.
5. Выведите на экран содержимое конфигурационного файла модуля **cpu-monitor.service**, используя **systemctl**.
6. Отредактируйте файл модуля так, чтобы он запускался только после службы **fly-message-monitor.service**, используя утилиту **systemctl**.
7. Запустите службу **cpu-monitor.service** и убедитесь, что она работает.
8. Переведите систему в целевое состояние **multi-user.target** и убедитесь, что службы **cpu-monitor.service** и **fly-message-monitor.service** не запущены.
9. Переведите систему в целевое состояние **graphical.target** и убедитесь, что службы **cpu-monitor.service** и **fly-message-monitor.service** запустились.
10. Запустите службу **resource-demanding1.service**.
11. Запустите службу **resource-demanding2.service** и проверьте, что при высокой (более 85%) нагрузке на CPU вы получаете сообщения от скрипта мониторинга.

Задание 3.

1. Измените настройки контрольной группы службы **resource-demanding1.service** так, чтобы суммарная нагрузка на два CPU была в пределах **85-94%**. Убедитесь, что сообщение от скрипта мониторинга изменилось.

2. Измените настройки контрольной группы службы **resource-demanding1.service** так, чтобы **суммарная нагрузка** на два CPU была ниже **85%**. Убедитесь, что сообщение от скрипта мониторинга изменилось и больше не появляется.

3. Выведите на экран дерево контрольных групп и найдите в нем контрольную группу для сервиса **resource-demanding1.service**.

Задание 4.

1. Выведите на экран все запущенные модули подсистемы **systemd**.
2. Выведите на экран все модули служб подсистемы **systemd**.
3. Выведите на экран в отдельном терминале информацию обо всех поступающих в реальном времени событиях подсистемы **systemd**.
4. Перезапустите службу **resource-demanding1.service**. Выведите на экран её статус.
5. Остановите службы **resource-demanding1.service**, **resource-demanding2.service** и **cpu-monitoring.service**.
6. Выведите на экран информацию из журналов о сервисе **resource-demanding1.service** в сокращённом виде.

Вопросы

1. Какое сочетание технологий рекомендуется использовать в современных системах?
2. Расположите пункты запуска компьютера с ОС Linux от момента включения до момента запуска ядра в правильном порядке:
 - a. Загрузка **init ram fs**
 - b. Тестирование и инициализация оборудования
 - c. Поиск загрузочного устройства
 - d. Загрузка ядра
 - e. Запуск загрузчика первого этапа
 - f. Вывод загрузочного меню
 - g. Запуск загрузчика второго этапа
 - h. Запуск ядра
3. Расположите пункты запуска компьютера с ОС Linux от момента запуска ядра до момента загрузки системы в выбранном целевом состоянии:
 - a. Загрузка модулей ядра и драйверов с RAM-диска
 - b. Монтирование **init ram fs** в корень
 - c. Инициализация наиболее важных функций ядра
 - d. Запуск процесса **systemd**
 - e. Монтирование устройства с системным диском в корень

f. Запуск скрипта /init

g. Запуск модулей systemd

4. В каком файле следует выполнять настройки GRUB?

5. Какая команда позволит применить новую конфигурацию GRUB?

6. Какого модуля systemd не существует?

7. Какие целевые состояния ОС соответствуют следующим уровням загрузки?

A. runlevel 0

B. runlevel 1

C. runlevel 2

D. runlevel 3

E. runlevel 4

F. runlevel 5

G. runlevel 6

8. Какие секции могут присутствовать в конфигурационном файле модуля службы systemd:

9. Какой командой можно отслеживать события со службами systemd в реальном времени?

10. Какая команда выведет список всех служб и их состояния: