Тема 1.4 Ядро операционной системы, загрузка ОС система инициализации

План занятия

1. Архитектура ядра операционной системы.
2. Виды ядра
2. Загрузка ОС
3. Инициализация системы
4. Итоги
5. Домашнее задание

По итогу занятия вы получите представление о внутреннем устройстве ядра Linux и научитесь работать с модулями ядра. Как происходит загрузка ОС и ее инициализация.

Виды архитектуры ядер ОС

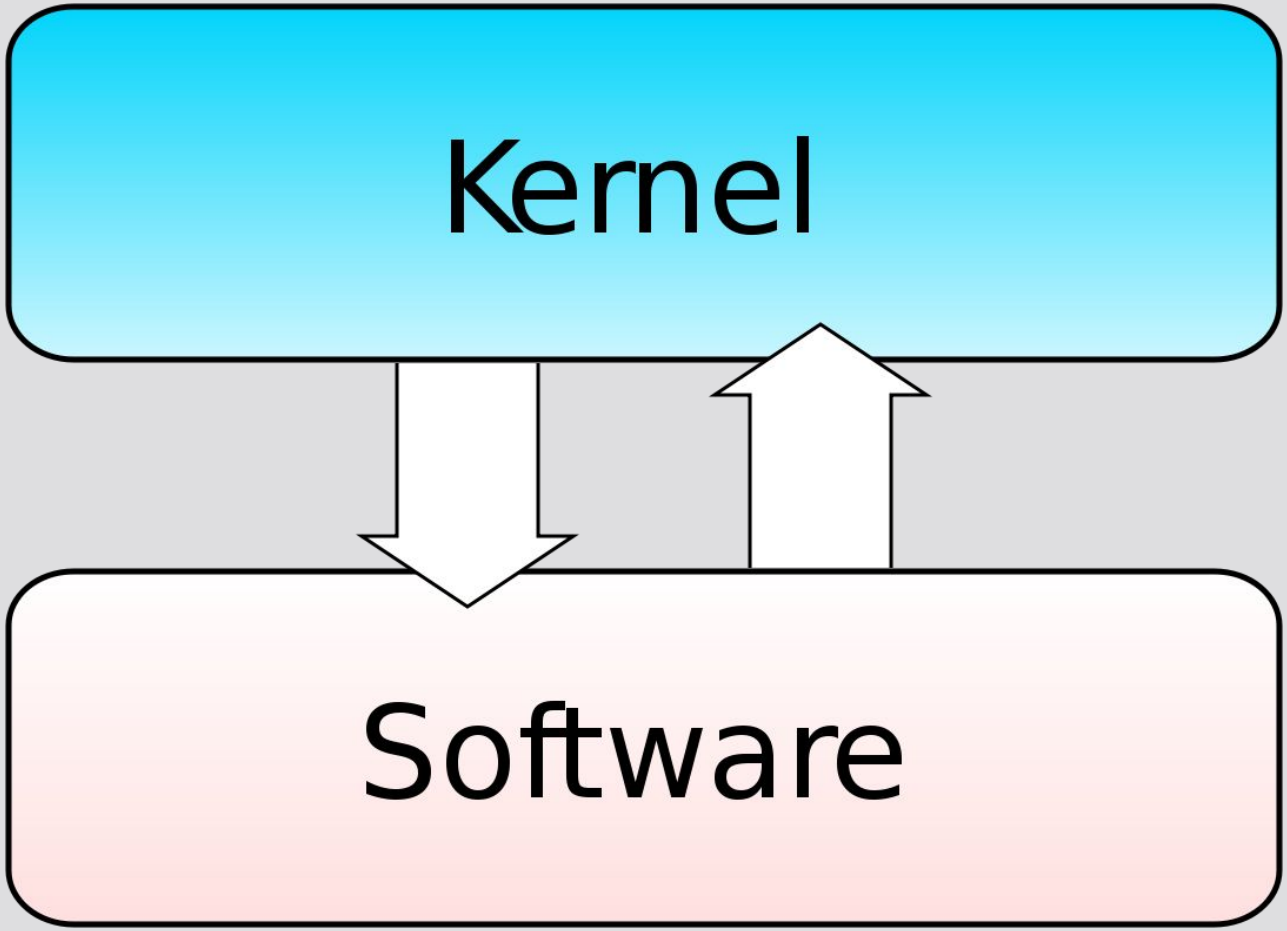
1. Монолитное ядро предоставляет богатый набор абстракций оборудования. Все части монолитного ядра работают в одном адресном пространстве. Это такая схема операционной системы, при которой все компоненты её ядра являются составными частями одной программы, используют общие структуры данных и взаимодействуют друг с другом путём непосредственного вызова процедур. Монолитное ядро — старейший способ организации операционных систем. Примером систем с монолитным ядром является большинство UNIX-систем.

Достоинства: Скорость работы, упрощённая разработка модулей. Недостатки: Поскольку всё ядро работает в одном адресном пространстве, сбой в одном из компонентов может нарушить работоспособность всей системы.

Примеры: Традиционные ядра UNIX (такие как BSD), Linux; ядро MS-DOS, ядро KolibriOS.

Монолитные ядра имеют долгую историю развития и усовершенствования и, на данный момент, являются наиболее архитектурно зрелыми и пригодными к эксплуатации. Вместе с тем, монолитность ядер усложняет их отладку, понимание кода ядра, добавление новых функций и возможностей, удаление «мёртвого», ненужного, унаследованного от предыдущих версий кода. «Разбухание» кода монолитных ядер также повышает требования к объёму оперативной памяти, требуемому для функционирования ядра ОС. Это делает монолитные ядерные архитектуры малопригодными к эксплуатации в системах, сильно ограниченных по объёму ОЗУ, например, встраиваемых системах, производственных микроконтроллерах и т. д.

Виды архитектуры ядер ОС



Архитектуры ядра (что хранить в ядре; какие функции необходимо хранить в ядре):

Ссылка на изображение

Архитектуры ядра

2. **Многослойная монолитная архитектура (модульная)**.

Модульное ядро — современная, усовершенствованная модификация архитектуры монолитных ядер операционных систем. В отличие от «классических» монолитных ядер, модульные ядра, как правило, не требуют полной перекомпиляции ядра при изменении состава аппаратного обеспечения компьютера. Вместо этого модульные ядра предоставляют тот или иной механизм подгрузки модулей ядра, поддерживающих то или иное аппаратное обеспечение (например, драйверов). При этом подгрузка модулей может быть как динамической (выполняемой «на лету», без перезагрузки ОС, в работающей системе), так и статической (выполняемой при перезагрузке ОС после переконфигурирования системы на загрузку тех или иных модулей).

Все модули ядра работают в адресном пространстве ядра и могут пользоваться всеми функциями, предоставляемыми ядром. Поэтому модульные ядра продолжают оставаться монолитными. Модульность ядра осуществляется на уровне бинарного образа, а не на архитектурном уровне ядра, так как динамически подгружаемые модули загружаются в адресное пространство ядра и в дальнейшем работают как интегральная часть ядра. Модульные монолитные ядра не следует путать с архитектурным уровнем модульности, присущий микроядрам и гибридным ядрам. Практически, динамичная загрузка модулей, это просто более гибкий способ изменения образа ядра во время выполнения — в отличие от перезагрузки с другим ядром. Модули позволяют легко расширить возможности ядра по мере необходимости.

Модульные ядра удобнее для разработки, чем традиционные монолитные ядра, не поддерживающие динамическую загрузку модулей, так как от разработчика не требуется многократная полная перекомпиляция ядра при работе над какой-либо его подсистемой или драйвером. Выявление, локализация, отладка и устранение ошибок при тестировании также облегчаются. Примером может служить VFS — «виртуальная файловая система», совместно используемая многими модулями файловых систем в ядре Linux.

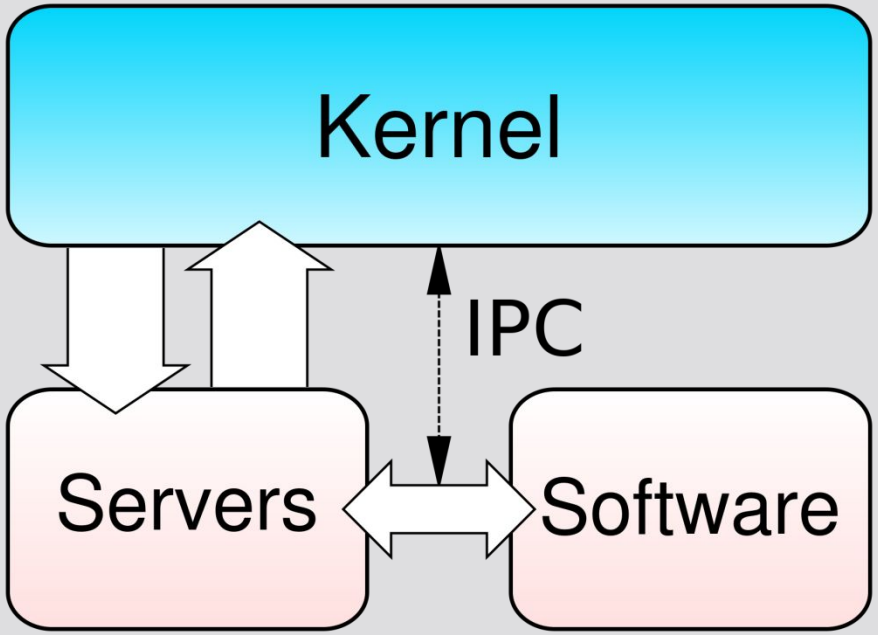
Архитектуры ядра

Архитектуры ядра (что хранить в ядре; какие функции необходимо хранить в ядре):

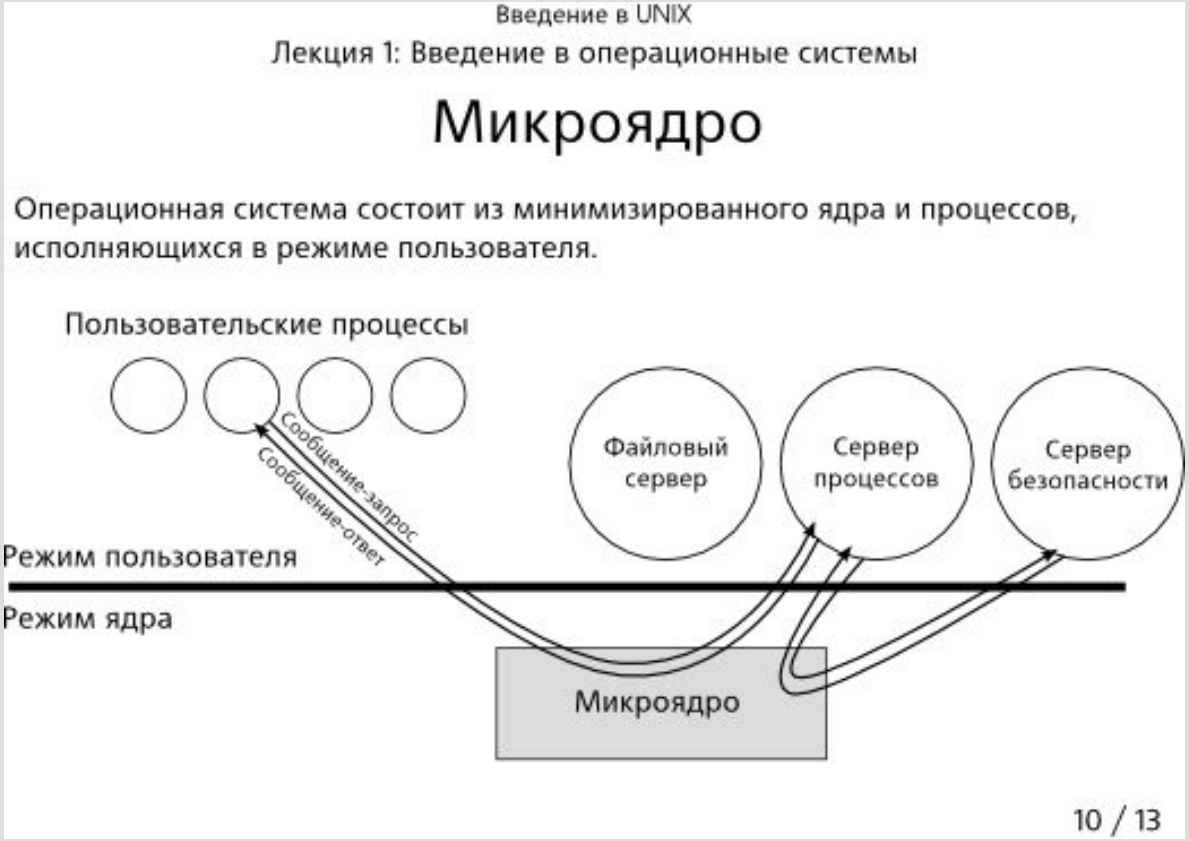
3. **Микроядерная архитектура** - в этом типе архитектуры ядра основные пользовательские службы, такие как управление драйверами устройств, управление стеком протоколов, управление файловой системой и управление графикой, присутствуют в пространстве пользователя, а остальные функции управление памятью, управление процессами присутствует внутри пространства ядра. Таким образом, всякий раз, когда система имеет потребность в услугах, присутствующих в пространстве ядра, ОС переключается в режим ядра, а для служб пользовательского уровня она переключается в режим пользователя. Этот тип архитектуры ядра уменьшает размер ядра, но скорость выполнения процессов и предоставления других услуг значительно ниже, чем у монолитных ядер.

Архитектуры ядра (что хранить в ядре; какие функции необходимо хранить в ядре):

3. **Микроядерная архитектура**



Архитектуры ядра (что хранить в ядре; какие функции необходимо хранить в ядре):



Ссылка на изображение

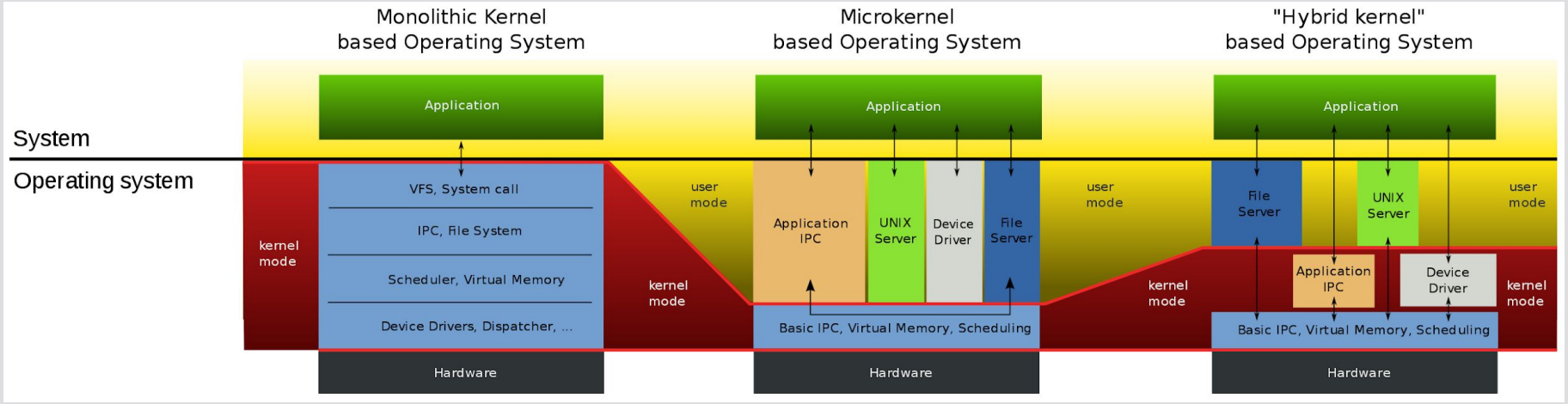
Архитектуры ядра (что хранить в ядре; какие функции необходимо хранить в ядре):

4. **Нано ядерная архитектура** - в ядро входят только машинно-зависимые модули. Данная архитектура используется в виртуализированной ОС

5. **Экза ядерная архитектура** (в режиме ядра работают только менеджеры ресурсов). Идея применяются в гетерогенных вычислительных системах

6. **Гибридное ядро** (англ. *Hybrid kernel*) — модифицированные микроядра, позволяющие для ускорения работы запускать модули ОС в пространстве ядра. Для наилучшей производительности системы нам требуется как высокая скорость, так и малый размер ядра, чтобы наша система могла иметь максимальную эффективность. Поэтому для решения этой задачи был разработан новый тип ядра, который представлял собой комбинацию монолитного ядра и микроядра. Этот тип ядра известен как гибридное ядро. Такой тип архитектуры используется практически во всех системах, которые производятся в настоящее время.

Архитектуры ядра (что хранить в ядре; какие функции необходимо хранить в ядре):



Краткое заключение по ядрам ОС

Монолитные ядра – все функции операционной системы загружены в **единый файл** и выполняются в качестве одного процесса в одном адресном пространстве.

Микроядра – все функции ядра разделяются на **несколько процессов**, которые обычно называют серверами. Все серверы поддерживаются независимыми друг от друга и выполняются каждый в своем адресном пространстве.

Гибридные ядра – различные **комбинации**, совмещающие оба вышеуказанных подхода.

Краткое заключение по ядрам ОС

	Плюсы	Минусы
Микроядра	• При ошибке одного из серверов можно избежать падения всей системы • Запуск процессов в режиме ядра только тех серверов, которым это необходимо • Низкое потребление памяти	• Много накладных расходов на обеспечение взаимодействия между серверами • Процессы должны ждать свою очередь, чтобы получить информацию
Монолитные	• Прямой запуск функций • Производительность	• При любом изменении требуется пересборка всего ядра • Любая ошибка в любой подсистеме приводит к kernel panic

Краткое заключение по ядрам ОС

Ядро Linux

На сегодняшний день Linux — **монолитное ядро с поддержкой загружаемых модулей.**

● Драйверы устройств и расширения ядра обычно запускаются в нулевом кольце защиты, с полным доступом к оборудованию.

● Все драйверы и подсистемы работают в своем адресном пространстве, отделенном от пользовательского.

● В **отличие от обычных монолитных ядер, драйверы** устройств легко собираются в виде модулей и загружаются или выгружаются во время работы системы.

Модули ядра Linux

● любой дополнительный функционал может быть загружен в виде модулей;

● Модули хранятся в системе в директории **/lib/modules/;**

● При работе с модулями используются утилиты **lsmod, insmod, modprobe и modinfo.**

Работы с модулями

● lsmod — информация обо всех загруженных модулях
● modinfo <MODULE-NAME> — просмотр информации о модуле
● modprobe <MODULE-NAME> — загрузка модуля
● insmod /lib/modules/.../<MODULE-NAME>.ko — загрузка модуля с помощью insmod
● rmmod <MODULE-NAME> — выгрузка модуля

Подсистемы ядра

- **Process Scheduler** (SCHED) — планировщик процессов, отвечает за контроль над доступом процессов к CPU;

- **Memory Manager** (MM) — менеджер памяти, обеспечивает различным процессам безопасный доступ к основной памяти системы;

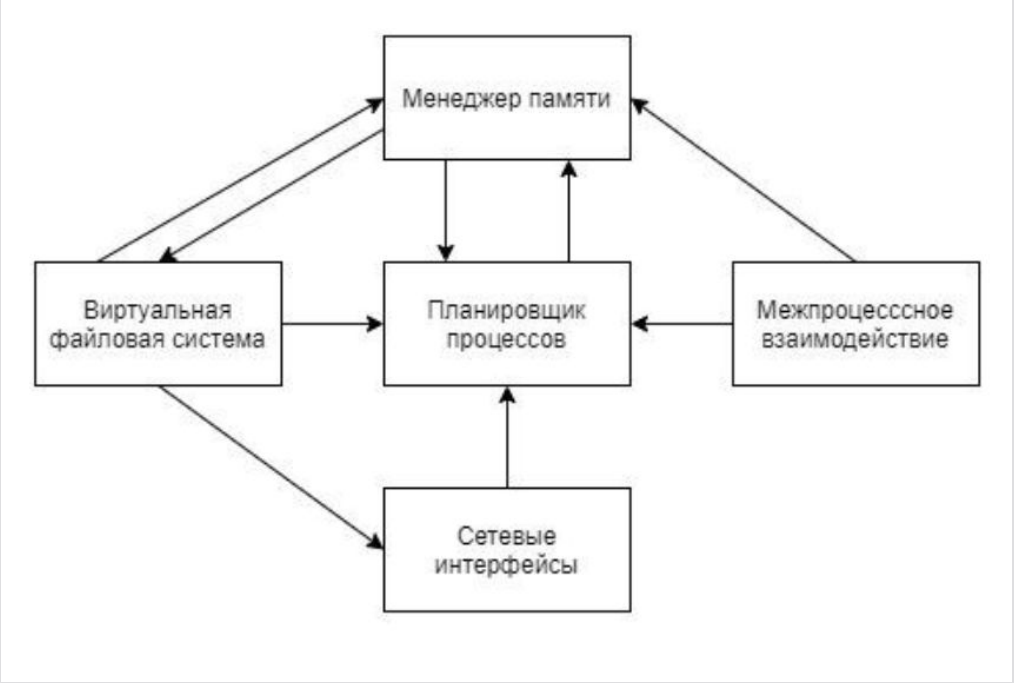
- **Virtual File System** (VFS) — виртуальная файловая система, создает абстрактный слой, скрывая детали оборудования, предоставляя общий файловый интерфейс для всех устройств;

- **Network Interface** (NET) — сетевые интерфейсы, обеспечивает работу с различными сетевыми стандартами и сетевым оборудованием;

- **Inter-Process Communication** (IPC) — межпроцессная подсистема, поддерживающая несколько механизмов для process-to-process связей в единой Linux-системе.

Зависимости подсистем друг от друга

Планировщик процессов — основная подсистема.
Все остальные зависят от нее, так как всем им необходимо приостанавливать
и возобновлять выполнение процессов.
Например, когда процесс пытается отправить некое сообщение по сети, сетевой интерфейс может приостановить выполнение процесса, пока сетевое оборудование выполняет отправку сообщения.



Выполнение в режиме ядра

Передача ядру контроля над процессом для выполнения необходимой задачи.

Например, открытие файла или передачи данных по сети.

Существует 2 события, при которых выполнение переходит в режим ядра:

- ***системные вызовы;***

Поступают от пользовательских программ и касаются работы с устройствами, памятью и т.п.

- ***аппаратные прерывания.***

Сигнализируют об окончании какого-либо действия со стороны устройства или о возникшей на устройстве ошибке.

Системные вызовы

Системный вызов — обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции.

Примеры системных вызовов:
open, read, write, close.

Документация доступна во втором разделе man:

man 2 <syscall-NAME> — поиск документации по системному вызову.

Strace — утилита для отслеживания системных вызовов при выполнении процесса.

strace <команда> — поиск документации по системному вызову..

Аппаратные прерывания

1. Когда ЦПУ получает прерывание, он останавливает любые Процессы

○ Если это не более приоритетное прерывание, тогда обработка пришедшего прерывания произойдет только тогда, когда более приоритетное будет завершено.

2. Сохраняет некоторые параметры в стеке

3. Вызывает обработчик прерывания.

➡ **Это означает, что не все действия допустимы внутри обработчика прерывания, потому что система находится в неизвестном состоянии.**

ВЫВОДЫ ПО ПОДТЕМЕ

- рассмотрели архитектуру ядра;
- рассмотрели основные виды ядер;
- рассмотрели модули ядра;
- рассмотрели планировщик процессов и что такое прерывания.

Загрузка ОС

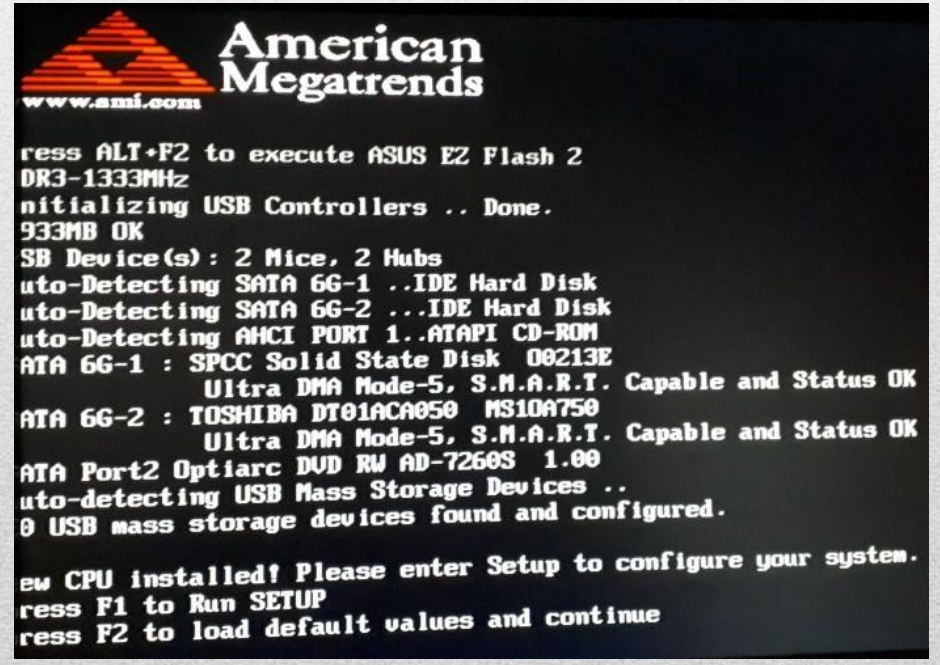
План темы:

- процесс загрузки ОС Linux;
- основные функции BIOS/UEFI;
- работу загрузчика GRUB**;**
- начнем изучение процесса инициализации init.

Загрузка

Загрузка (boot, booting) — процесс запуска устройства и загрузки ОС.

В этот момент происходит обнаружение и (само)настройка всех компонентов системы.



Этапы загрузки

1. Загрузка BIOS/UEFI (MBR/GPT).
2. Поиск, загрузка в память и запуск загрузчика (GRUB).
3. Поиск, загрузка в память и запуск ядра ОС.
4. Запуск системных служб.
5. Запуск пользовательских служб.

1. После нажатия кнопки включения или перезагрузки управление берет на себя Базовая система ввода/вывода (Basic Input/Output System - BIOS).

2. Главный загрузчик определяет, откуда следует загружать ОС. В зависимости от типа загрузчика управление будет передано либо загрузочному коду, находящемуся в активном разделе жесткого диска, либо менеджеру загрузки, либо сам загрузчик поместит ядро ОС в оперативную память и передаст ему управление.

3. Получивший управление загрузчик операционной системы инициирует загрузку ядра ОС в память.

4. Запускается сама операционная система.

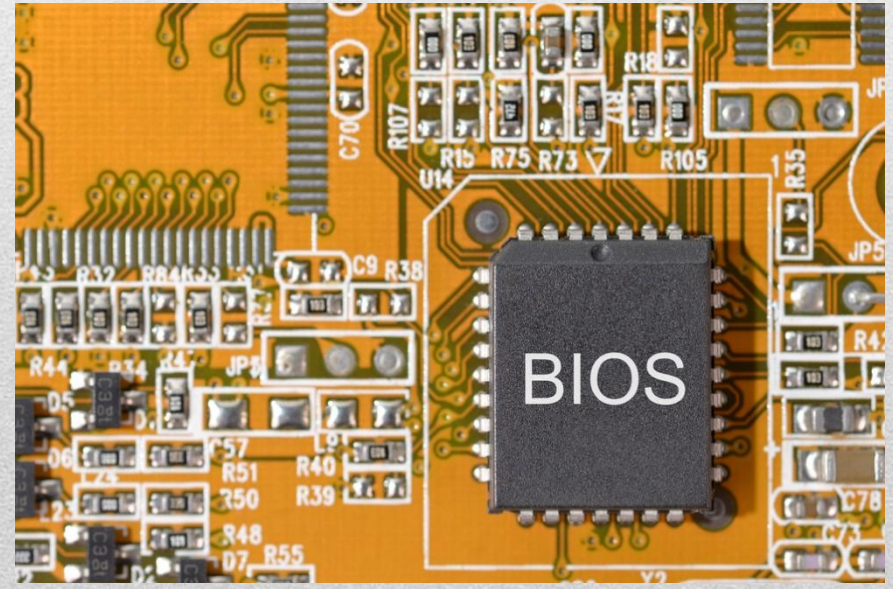
Как видно из вышесказанного, при загрузке компьютера используются абсолютные адреса, т.е. номера жестких дисков, цилиндров, головок, секторов, блоков. Выполнение этапов загрузки происходит по жестко заданной цепочке.

Для успешной загрузки ОС необходимо, чтобы все участники цепочки находились на своих строго определенных местах: главный загрузчик и основная таблица разделов в загрузочном секторе диска, загрузчик(и) ОС и таблицы разделов в остальных разделах диска.

Этапы загрузки

BIOS (basic input/output system — «базовая система ввода-вывода») - это программа, записанная в постоянной энергонезависимой памяти компьютера -ПЗУ (английская аббревиатура - CMOS).

BIOS производит тестирование и инициализацию всех устройств и, если они прошли успешно, считывает MBR по абсолютному адресу. Затем помещает считанный код главного загрузчика в оперативную память и передает ему управление.



Источник изображения: https://www.tomshardware.com/news/bios-firmware-definition,37646.html

BIOS

Изначально BIOS была собственностью IBM и применялась только IBM PC. После реверс-инжиниринга стала применяться во всех PC-совместимых компьютерах.

Ранее, помимо загрузки, BIOS использовалась для обработки прерываний от устройств ввода-вывода. Сейчас данный механизм полностью реализован на уровне операционной системы.

Также, в первых компьютерах BIOS была аппаратно реализована в виде ROM (ПЗУ), в последних — это flash-память с возможностью перезаписи.

Источник изображения: https://www.tomshardware.com/news/bios-firmware-definition,37646.html

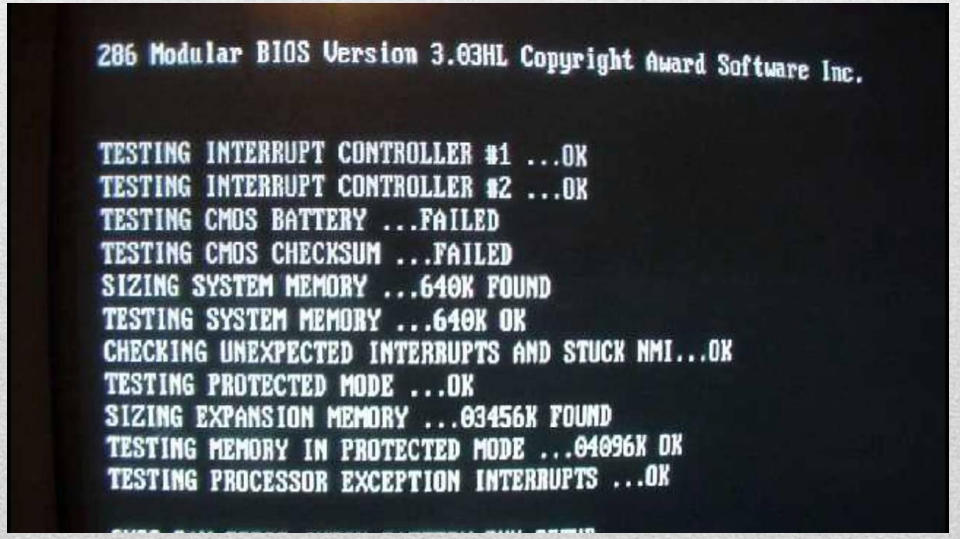
POST

POST (power-on self-test) — процесс первоначальной проверки оборудования сразу после его включения.

● Если во время POST обнаруживаются ошибки, то они выдаются пользователю как набор звуковых сигналов и номер кода ошибки, доступный через специальные отладочные платы.

● POST запускается после включения компьютера или нажатия кнопки « Reset».

● Если перезагрузка происходит по нажатию *Alt-Ctrl-Del,* то процедура POST не запускается.



Источник изображения: https://www.tomshardware.com/news/bios-firmware-definition,37646.html

UEFI

UEFI, Unified Extensible Firmware Interface (интерфейс расширяемой прошивки) — спецификация интерфейса между ОС и аппаратными прошивками (firmware).

UEFI

Преимущества UEFI

● возможность использовать разделы больше 2 ТБ GUID Partition Table (GPT);
● работа напрямую с ФС;
● работа с сетью;
● удобное и функциональное графическое окружение;
● 32- или 64-битное окружение;
● разработка на языке C;
● модульная архитектура;
● большие возможности по совместимости и модернизации.

Недостатки UEFI

● бóльшая возможность для внедрения вредоносных программ;
● проблемы с гибкостью, т.к. каждая ОС должна иметь свой собственный драйвер в составе UEFI;
● плохая поддержка старых ОС;
● усложнение архитектуры усложняет разработку;
● возможна ситуация, когда устанавливается только определенная ОС (например, Windows).

UEFI Boot Manager

UEFI, в отличии от BIOS, содержит свой собственный загрузчик — **UEFI Boot Manager**.

Boot Manager осуществляет загрузку:

- UEFI-загрузчиков ОС;
- UEFI-драйверов;
- UEFI-приложений.

Boot Manager имеет свою собственную конфигурацию, записанную в NVRAM (Non-volatile RAM).

Фазы UEFI

1. SEC (Security) — проверяет цифровые подписи и передает управление доверенному коду.
2. PEI (Pre EFI Initialization) — инициализация устройств.
3. DXE (Driver eXecution Environment) — загрузка сервисов UEFI.
4. BDS (Boot Device Select) — поиск устройств загрузки.
5. RT (Run Time) — GRUB.

Загрузчик

Загрузчик операционной системы, в зависимости от типа
может выполнять такие функции как:

организация диалога с пользователем для выбора ОС;
выполнение подготовительных операций для запуска ядра ОС;

формирование параметров, передаваемых ядру ОС;
загрузка ядра (с диска или по сети) и передача ему управления.

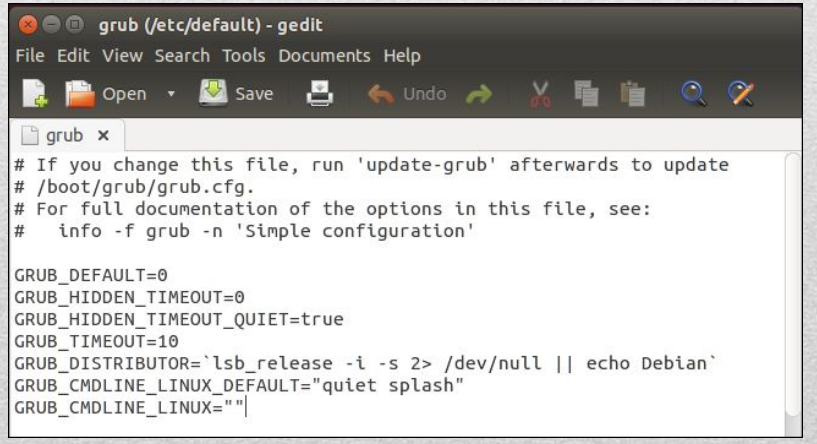
Наиболее часто используются загрузчики ОС:
NTLDR -Windows NT;
Windows Boot Manager -Windows Vista, Windows 7;
LILO (Linux LOader) - Linux;
GRUB (Grand Unified Bootloader);
OS/2 Boot Manager - OS/2;
BootX -Mac OS X;
Acronis OS selector - менеджер загрузки разных ОС.

Редактирование GRUB

Отредактируем /etc/default/grub
Открываем файл для редактирования редактором Nano
sudo nano /etc/default/grub

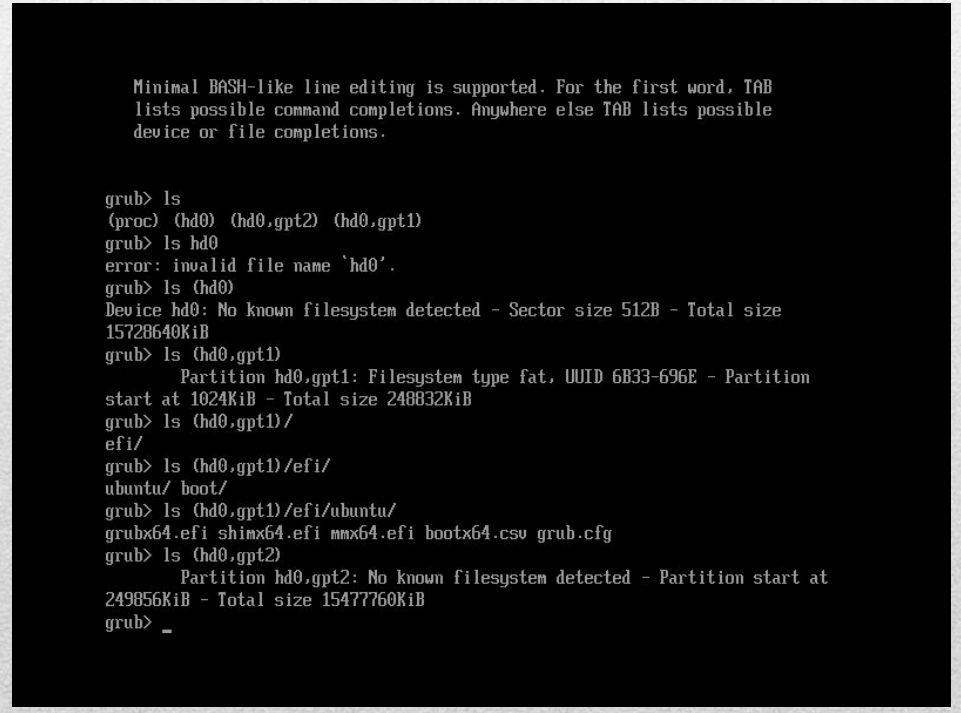
- закоммент им ожидание окна загрузки

Выполняем системную команду обновления данных загрузчика
sudo update-grub
Перезагружаем ОС «сразу»
shutdown -r now



Командная строка Grub

Выполнять команды grub возможно и до загрузки ОС (по аналогии в Windows – кнопка **F8**.
Для этого необходимо нажать «**c**» в меню GRUB (при использовании виртуальной машины может потребоваться при загрузке нажать «Shift и держать, пока не появится меню GRUB).



Источник изображения

Пример:
Параметры ядра при работе с Grub

● *Просмотр параметров ядра:*
sysctl -a
● *Просмотр конфигурационного файла:*
cat /etc/sysctl.conf | more
● *Просмотр сетевых настроек IPv4:*
cat /etc/sysctl.conf | grep ipv4
● *Включение пересылки пакетов (функций шлюза):*
sudo sysctl net.ipv4.ip_forward=1

Система инициализации

Init (инициализация)

init — специальный процесс (демон) управления системой и службами.

Расположение:

/sbin/init

Режимы работы init:

- однопользовательский (службы не запускаются);
- многопользовательский (режим запуска по умолчанию);
- сервер (аналогичен многопользовательскому, но без GUI).

Виды систем инициализации:

- **System V:**
  - Все службы запускаются последовательно.
- **BSD init**:
  - FreeBSD, NetBSD, OpenBSD.
- **systemd:**
  - упрощенный процесс загрузки;
  - параллельный запуск служб;
  - запись событий в системный журнал.

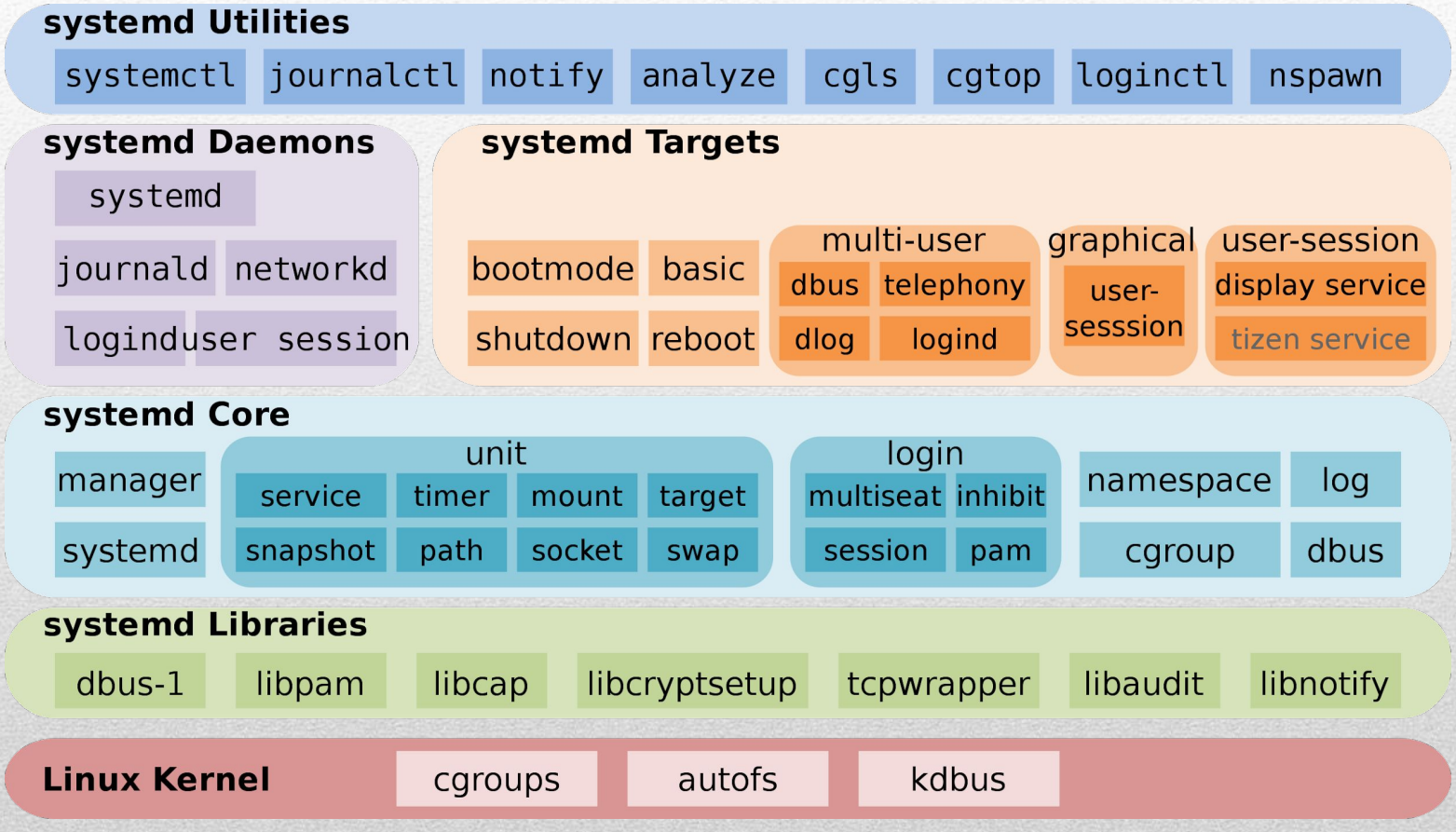
Рассмотрим systemd

systemd — подсистема инициализации и управления службами в Linux, фактически вытеснившая в 2010-е годы традиционную подсистему init. Основная особенность — интенсивное распараллеливание запуска служб в процессе загрузки системы, что позволяет существенно ускорить запуск операционной системы. Основная единица управления — модуль, одним из типов модулей являются «службы» — аналог демонов — наборы процессов, запускаемые и управляемые средствами подсистемы и изолируемые контрольными группами.

Позволяет выполнять следующие действия со службами:

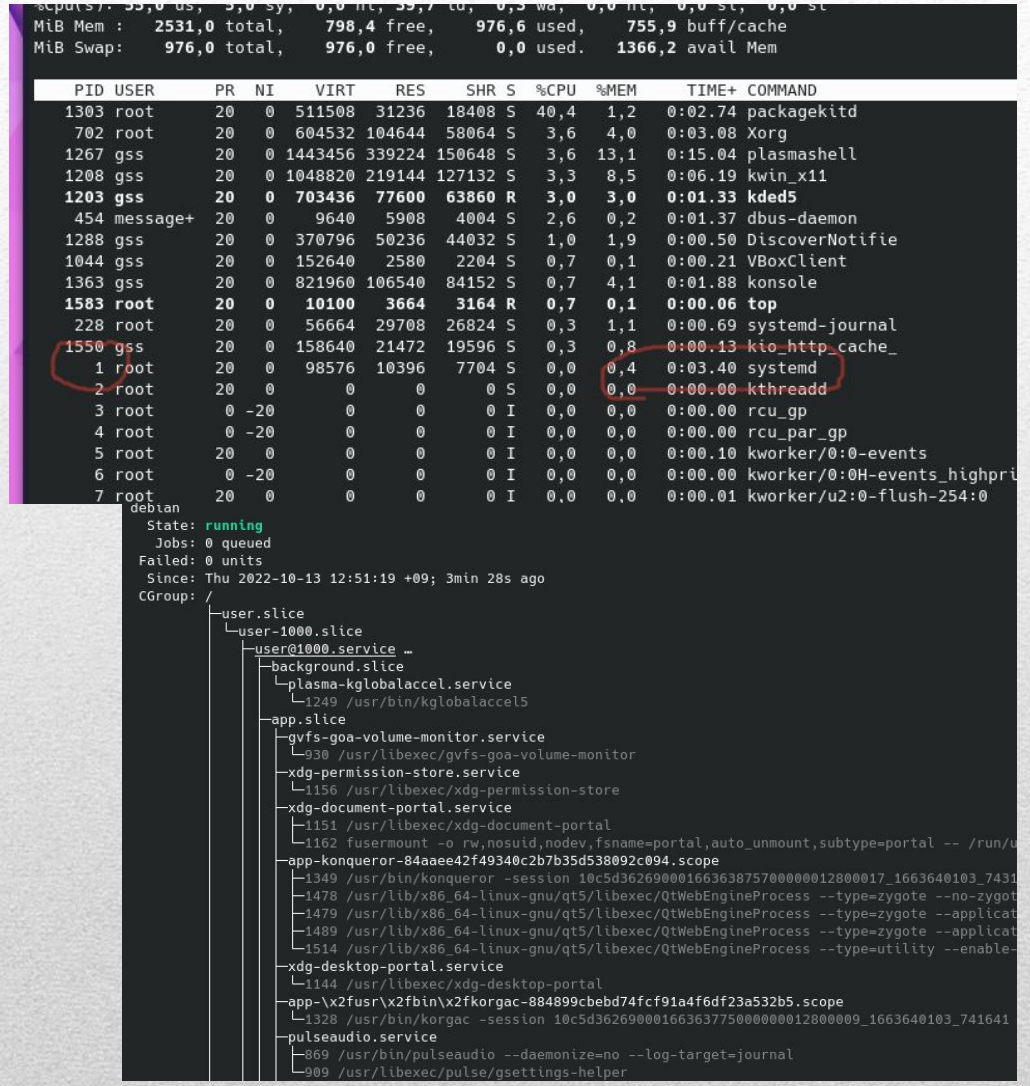
- запускать/останавливать;
- добавлять/удалять;
- редактировать;
- собирать логи.

Рассмотрим systemd



Рассмотрим systemd

Введите команду
Top



Введите команду
systemctl status

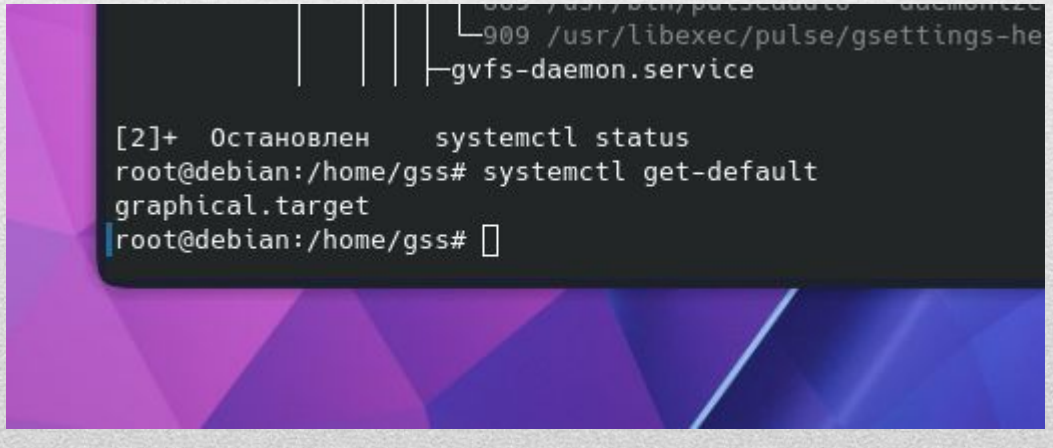
systemd - target

Цель (target) — нужное состояние системы; ссылка на файл, содержащий зависимости (службы).

➡ Systemd запускает все зависимости из соответствующего target-файла.

➡ Когда все зависимости будут запущены, то система будет работать на соответствующем target-уровне.

systemctl get-default
graphical.target



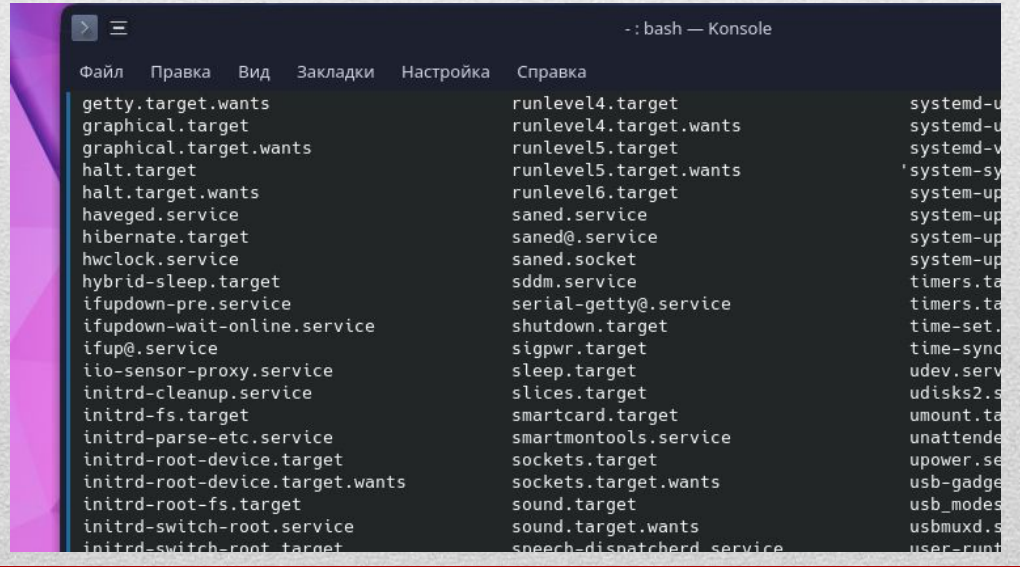
systemd - unit

Модуль (unit) — описывает запускаемую службу, устройство и т.п.
Каждый модуль описан в своем файле (unit file):

- /usr/lib/systemd/system/ – модули из пакетов (Nginx, Apache, MySQL);

- /run/systemd/system/ — модули, созданные во время работы ОС;

- /etc/systemd/system/ — модули, созданные пользователем.



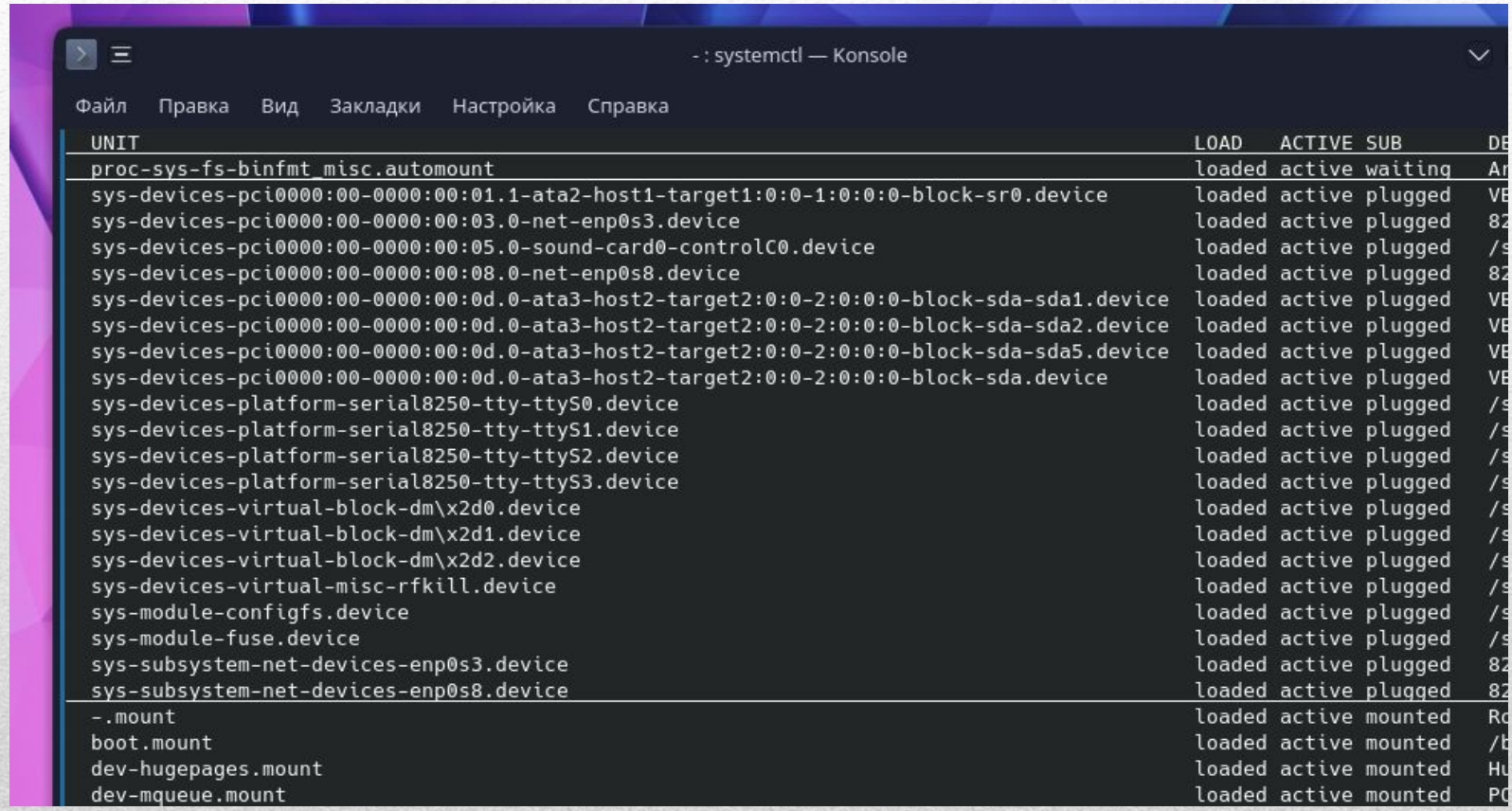
systemd – unit. Типы модулей (юнитов)

- модули служб — обычные службы ОС;
- модули монтирования — монтируют ФС;
- целевый модули/цели — группируют другие модули;

systemd-analyze plot > test.svg
sudo systemctl list-dependencies

systemctl list-units — **список модулей**
systemctl list-units --type=service — **список модулей-служб**
systemctl status <модуль> — **состояние выбранного модуля**
systemctl enable\disable <модуль> — **разрешить/запретить модуль**
systemctl start\stop\restart <модуль> — **запустить/остановить/… модуль**
systemctl daemon-reload — **перезапуск конфигурации systemd**

systemd – unit. Типы модулей (юнитов)



journalctl

Журнал — база данных, в которой хранятся сообщения ядра и служб, начиная с загрузки и заканчивая завершением работы.

journalctl

Настройки журнала:

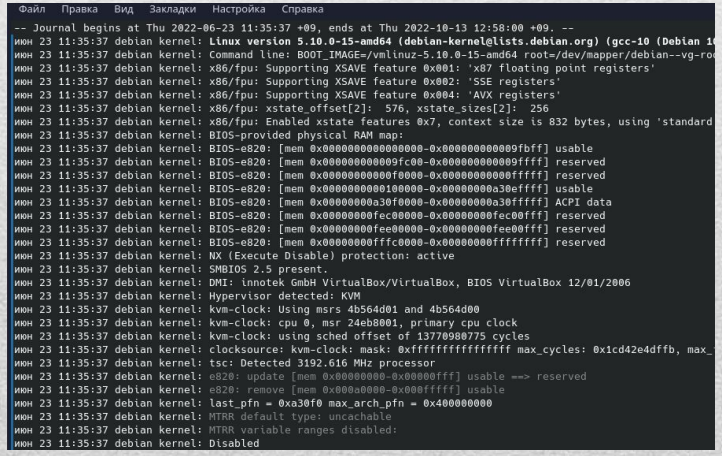
nano /etc/systemd/journald.conf

journalctl -u=sshd — сообщения для модуля ssh

journalctl -b 0 -u ssh — … только в текущем сеансе

journalctl -n 100 /usr/sbin/sshd — показать внешний лог

journalctl --since=yesterday --until=now — временной период



ИТОГИ

- рассмотрели как происходит загрузка современного ПК с ОС Linux;
- получили представление о работе загрузчика, в том числе GRUB;
- рассмотрели что такое инициализация системы и их основные виды;
- начали работу с менеджером системы инициализации systemd, journalctl