***1.Права доступа + группа, владелец***

**Права доступа в Linux**

В Linux для файлов и директорий устанавливаются три категории прав доступа:

1. Владелец (User): Пользователь, который создал файл или директорию.

2. Группа (Group): Набор пользователей, имеющих общие права доступа.

3. Остальные (Others): Все остальные пользователи, не являющиеся владельцем или членом группы.

Для каждой категории могут быть заданы следующие права:

- Чтение (r, 4): Право на просмотр содержимого файла или списка файлов в директории.

- Запись (w, 2): Право на изменение содержимого файла или добавление и удаление файлов в директории.

- Выполнение (x, 1): Право на выполнение файла как программы или вход в директорию.

Способ задания: буквенный или цифровой(rwx-------,700)

**Настройка прав доступа**

Вы можете просмотреть и изменить права доступа с помощью команд:

- ls -l: Показать текущие права доступа.

- chmod: Изменить права доступа. Например, chmod 755 file установит права rwxr-xr-x для файла.

**Использование sudo**

sudo — это утилита, позволяющая пользователям выполнять команды с правами других пользователей (по умолчанию с правами суперпользователя root). Это полезно для выполнения задач, которые требуют административных прав, без необходимости всегда работать под учетной записью root.

**Изменение владельца и группы-владельца**

- Изменение владельца: Используется команда chown. Например, chown user file изменяет владельца файла на user.

- Изменение группы: Используется команда chgrp. Например, chgrp group file устанавливает группу файла на group.

- Комбинированное изменение: chown user:group file сразу изменяет и владельца, и группу файла.

**Данные, кроме прав доступа**

Кроме прав доступа, файлы и директории в Linux имеют следующие атрибуты:

- Тип: Это может быть обычный файл, директория, символическая ссылка и так далее.

- Ссылка на inode: Содержит метаданные о файле, такие как права доступа, владельца, группу, размер, временные метки и указатели на данные.

- Размер: Размер файла в байтах.

- Количество жестких ссылок: Число ссылок на inode.

- Владелец и группа: Имя владельца и группы.

- Временные метки: Дата создания, последней модификации и последнего доступа.

- ID устройства: Определяет расположение файла в файловой системе.

***2. Переменные окружения***

***Что такое переменные окружения?***

Переменные окружения — это переменные, доступные в оболочке (шелле), которые могут использоваться для настройки среды выполнения программ. Они играют важную роль в конфигурации рабочей среды пользователя и определении параметров запуска приложений.

Переменные окружения в Linux (и в других операционных системах) представляют собой набор динамических значений, которые могут влиять на поведение работающих процессов и приложений. Они широко используются для хранения системных значений, путей к ресурсам и конфигурации приложений.

***Зачем нужны переменные окружения?***

- Конфигурация системных параметров: Например, переменная PATH хранит пути, в которых система ищет исполняемые файлы при запуске команд.

- Настройки пользователя: Можно задать пользовательские переменные для удобства работы, например, переменные для хранения пути к проектам.

- Работа с программами и скриптами: Передача настроек и конфигураций в программы через переменные окружения.

***Какие бывают переменные окружения?***

Существуют глобальные и локальные переменные окружения:

- Глобальные (со значением по умолчанию): Доступны всем процессам. Их изменение отразится на всех запущенных сессиях. Примеры: PATH, HOME, USER.

- Локальные (пользовательские): Доступны только в рамках текущей сессии или процесса. Можно задать и использовать их в скриптах.

***Где хранятся переменные окружения?***

Переменные окружения могут быть определены и сохранены в нескольких местах:

- В конфигурационных файлах оболочки: Например*,* ***.bashrc****,* ***.bash\_profile****,* ***.profile*** для bash-оболочки. Для других оболочек, таких как zsh, используются файлы .zshrc и другие.

- Системные файлы конфигурации: Файлы типа **/etc/environment** или **/etc/profile**.

- В сессии терминала: Путём непосредственного задания в командной строке.

***Как задать переменные окружения?***

1. В текущей сессии:

Используйте команду export. Например:

*export MY\_VARIABLE="some\_value"*

2. В конфигурационных файлах:

Добавьте запись в .bashrc или .bash\_profile:

*export MY\_VARIABLE="some\_value"*

Чтобы изменения вступили в силу, выполните source ~/.bashrc или перезапустите сессию.

3. Во время запуска программы:

Можно указать значение переменной при запуске программы:

*MY\_VARIABLE="some\_value" ./my\_program*

***Конфигурационные файлы***

1. ***/etc/profile:***

- Этот файл запускается при входе пользователя в систему с помощью оболочки логина (login shell).

- Он применяется для задания системных параметров, которые будут использоваться всеми пользователями.

- Включает в себя глобальные настройки, такие как системные переменные окружения.

2. ***/etc/bash.bashrc:***

- Это системный файл конфигурации, который работает для всех интерактивных оболочек bash.

- Обычно используется для задания общих настроек и алиасов для всех пользователей системы.

3. ***~/.bash\_profile:***

- Этот файл исполняется для оболочек логина конкретного пользователя.

- Используется для пользовательских настроек, которые должны применяться при входе в систему.

- Если файл ~/.bash\_login или ~/.profile также существует, он может быть запущен вместо или посе ~/.bash\_profile.

4. ***~/.bashrc:***

- Исполняется для каждой интерактивной нелогин-оболочки.

- Используется для задания пользовательских настроек, таких как алиасы, функции, переменные окружения, которые должны быть доступны в каждой новой оболочке.

5. ***~/.profile:***

- Может использоваться в качестве альтернативы ~/.bash\_profile.

- Поддерживается для совместимости с другой UNIX-подобной оболочкой.

6. ***~/.bash\_logout:***

- Запускается при выходе из оболочки логина.

- Может использоваться для выполнения чистки или других задач перед выходом из системы.

***Переменные окружения***

***1. \*PATH\****

- \*Описание\*: Содержит список директорий, в которых оболочка ищет исполняемые файлы при вводе команды.

- \*Пример\*: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

- \*Использование\*: Оболочка проверяет эти директории в указанном порядке, чтобы найти и выполнить введённую команду.

***2. \*HOME\****

- \*Описание\*: Указывает на домашнюю директорию текущего пользователя.

- \*Пример\*: /home/username

- \*Использование\*: Часто используется программами для сохранения пользовательских настроек и данных.

***3. \*USER\****

- \*Описание\*: Имя текущего пользователя.

- \*Пример\*: username

***4. \*SHELL\****

- \*Описание\*: Полный путь к командной оболочке, которая используется пользователем.

- \*Пример\*: /bin/bash

***5. \*LANG\****

- \*Описание\*: Определяет параметры локализации, включая язык и кодировку.

- \*Пример\*: en\_US.UTF-8

***6. \*PWD\****

- \*Описание\*: Путь к текущей рабочей директории.

- \*Пример\*: /home/username/projects

***7. \*EDITOR\****

- \*Описание\*: Указывает на текстовый редактор по умолчанию, используемый в командной строке.

- \*Пример\*: vim или nano

8. ***\*LOGNAME\****

- \*Описание\*: Имя пользователя***, с которым выполнен вход в систему.***

***- \*Пример\*: username***

***3. Окружение и оболочка***

***Оболочка (или shell)***

***Оболочка*** — это интерфейс между пользователем и операционной системой. Она позволяет пользователю вводить команды и управлять системными ресурсами. В контексте UNIX и Linux, оболочка обычно представляет собой интерфейс командной строки (CLI). Однако, существуют и графические оболочки.

***Основные функции оболочки:***

- Выполнение команд: Интерпретация и выполнение вводимых пользователем команд.

- Пайплайнинг и перенаправление: Управление потоками ввода и вывода между командами и файлами.

- Скриптинг: Возможность написания сценариев для автоматизации задач.

***Часто используемые оболочки:***

- Bash (Bourne Again SHell) — наиболее распространённая оболочка, используемая в Linux.

- Zsh — популярная альтернатива bash с расширенными возможностями и конфигурацией.

- Fish (Friendly Interactive SHell) — современная оболочка, известная интуитивностью использования.

- Ksh (KornShell) и csh (C Shell) — другие примеры традиционных оболочек UNIX.

***Окружение (Environment)***

***Окружение*** в контексте операционной системы — это совокупность переменных и настроек, которые определяют условия выполнения программ. Окружение каждого процесса включает:

- Переменные окружения: Ключевые пары значений, которые могут влиять на выполнение скриптов и программ (например, PATH, HOME, USER).

- Текущая рабочая директория: Директория, в которой выполняются команды.

- Настройки локализации: Определяют язык и региональные параметры, такие как формат времени и денежные единицы.

***Разница между оболочкой и окружением***

- Оболочка (Shell):

- Это интерфейс, через который пользователи взаимодействуют с системой.

- Запускает и управляет программами, используя синтаксис командной строки.

- Окружение (Environment):

- Набор состояния и параметров, которые влияют на поведение процессов.

- Может быть изменено и настроено, чтобы задать условия выполнения команд и приложений.

***4. Сессия***

В контексте Linux, сессия (или сеанс) относится к периоду времени, в течение которого пользователь подключается и взаимодействует с системой, обычно через терминал или графический интерфейс. Сессия начинается, когда пользователь заходит в систему, и заканчивается, когда он выходит.

***Основные аспекты сессии***

**1. Идентификация пользователя:**

- Каждая сессия связана с определённым пользователем. Пользователь идентифицируется логином и, как правило, паролем.

**2. Переменные окружения:**

- Переменные окружения задаются для каждой сессии. Каждый раз, когда вы входите в систему или открываете новый терминал, новая сессия создаётся с набором переменных окружения. Эти переменные влияют на поведение команд и программ, выполняемых в течение этой сессии.

**3. Процессы сессии:**

- Во время сессии пользователь может запускать множество процессов, таких как приложения или различные службы. Эти процессы будут работать в контексте текущей сессии.

**4. Настройки и конфигурации:**

- Настройки пользователя, такие как настройки оболочки, редактора или рабочего стола, обычно загружаются с началом сессии.

- Повторяемые настройки можно сохранить в конфигурационных файлах (например, ~/.bashrc для Bash), и они будут автоматически применяться при входе в систему.

**5. Закрытие сессии:**

- Сессия завершается, когда пользователь выходит из системы. Это может быть сделано через команду exit в терминале или через интерфейс выхода в графической среде. Завершение сессии обычно приводит к завершению всех процессов, запущенных в её рамках.

**6. Локальные и удаленные сессии:**

*- Локальная сессия*создаётся, когда пользователь входит в систему напрямую на машине (через физическую клавиатуру и экран).

*- Удалённая сессия* может быть установлена через такие протоколы, как SSH, что позволяет подключаться к системе, находящейся в другой сети или в интернете.

***Примеры работы с сессией***

- **SSH сессия**: Подключение к удаленному серверу через SSH создаёт новую сессию, в которой пользователь может выполнять команды как будто он физически находится за компьютером.

ssh user@remote-server.com

***6. inode***

***1. Определение:***

- **Inode (Index Node)** — это структура данных в файловой системе, которая хранит информацию о файле или директории, кроме его имени или фактического содержимого. Inode содержит данные о правах доступа, владельце, размер файла, временных метках и месте расположения данных файла.

***Максимальное количество inode в системе***

**1. Определение количества inode:**

- Количество inode задается при создании файловой системы. Обычно это фиксированное значение, которое определяется в качестве одного inode на определённый объем дискового пространства.

- Например, при создании файловых систем ext3/ext4, по умолчанию генерируется один inode на каждые 4 KB пространства, но это значение можно изменить в зависимости от предполагаемого использования дисков при помощи инструментов, таких как mkfs.

**2. Ограничения:**

- Если все inode заняты, система не сможет создавать новые файлы, даже если на диске остается свободное пространство.

***Что произойдет, если inode закончатся:***

**1. Невозможность создания новых файлов:**

- Как и в случае нехватки логического пространства, если заканчиваются inode, система не может создать новые файлы или директории, выдавая ошибку нехватки inode.

**2. Решения:**

- Удаление ненужных файлов для освобождения inode.

- Перемещение данных в другую файловую систему.

- Редко возможно динамическое увеличение числа inode без форматирования, поскольку для этого требуется изменение структуры файловой системы.

***Присваивание номера inode***

**1. Назначение:**

- При создании файла или директории файловая система выделяет первый доступный inode из пула свободных inode. Номера inode назначаются последовательно по мере использования системы.

**2. Перераспределение inode:**

- Когда файл удаляется, inode освобождается и становится доступным для назначения новому файлу. Таким образом, inode номера могут переиспользоваться.

***Где хранятся inode?***

**1. Местоположение:**

- Inode хранятся в специальных структурах данных внутри файловой системы. Эти структуры разбросаны по всему диску, чтобы уменьшить перемещения головки чтения/записи и ускорить доступ.

**2. Блочные группы:**

- В системах ext, inode распределены по группам блоков. Каждая группа содержит одинаковое количество inode. Это позволяет распределить нагрузку и позволяет файловой системе быстро находить свободный inode.

***7.Память в linux***

***Основные компоненты управления памятью***

**1. Физическая память:**

- RAM (Оперативная память): Высокоскоростная память, которая непосредственно используется процессором для выполнения текущих задач.

- Кэш: Быстрая память, встроенная в процессор, используемая для быстрого доступа к часто используемым данным.

**2. Виртуальная память:**

- Виртуальная память предоставляет каждому процессу отдельное адресное пространство, что изолирует процессы друг от друга и улучшает безопасность. Виртуальная память может превышать физическую память за счет использования своп-файлов или своп-разделов.

**3. Своп (Swap):**

- Своп пространство используется для временного хранения страниц памяти. Когда физическая память недостаточна, Linux перемещает неактивные страницы на диск. Swap может быть реализован в виде раздела или файла на диске.

***Управление памятью в Linux***

**1. Страничный подход:**

- Linux использует страничную организацию памяти, где физическая память разбита на страницы (обычно 4 КБ каждая). Это минимальная единица работы с памятью.

**2. Процесс управления страницами:**

- Работает на основе страниц с использованием механизма, такого как страничное вытеснение, для управления тем, какие страницы находятся в памяти, а какие находятся в свопе.

**3. Кеш и буферы:**

- Кеш страницы: кэширование страниц диска для быстрого повторного доступа.

- Буфер-кеш: управляет доступом к файловой системе и поддерживает буферизацию операций ввода-вывода. Это улучшает производительность за счет хранения недавно использованных данных.

***Команды для управления и мониторинга памяти***

**1. free**: Показ памяти

- Выводит информацию о свободной и используемой памяти в системе, включая кеш и своп.

**2. top и htop**: Мониторинг процессов

- Показывают, сколько памяти используют отдельные процессы и какие процессы наиболее активны.

**3. vmstat**: Статистика виртуальной памяти

- Предоставляет отчеты о процессах, памяти, переполнении и использовании центрального процессора.

**4. dmesg**: Диагностические сообщения ядра

- Может быть использовано для проверки сообщений ядра, связанных с управлением памятью.

**5. /proc/meminfo**: Информация о памяти

- Этот виртуальный файл содержит большое количество статистических данных о памяти.

***Оптимизация использования памяти***

**1. Настройка свопа:**

- Параметр swappiness определяет, как агрессивно система должна использовать своп. Может быть отрегулирован для баланса между производительностью и использованием памяти.

**2. Очистка кеша:**

- В некоторых ситуациях может понадобиться очистка кеша или буферов с использованием echo 3 > /proc/sys/vm/drop\_caches.

**3. Программное управление памятью:**

- Разработка программного обеспечения с учетом эффективного использования памяти: управление выделением и освобождением, использование подходящих структур данных.

***8.4KB страница в linux***

В Linux, как и в других операционных системах, виртуальная память обычно делится на страницы. Размер страницы — это минимальная единица управления памятью в операционной системе. В большинстве современных систем на архитектуре x86 и x86\_64 размер страницы составляет 4 КБ (кибибайта, или 4096 байт).

***Почему именно 4 КБ:***

**1. Аппаратная поддержка:**

- Архитектура x86, изначально разработанная Intel, поддерживает страницу размером 4 КБ на аппаратном уровне с момента выпуска 80386 в 1985 году. Это значительно упростило широкое принятие этого размера в системах, работающих на этой архитектуре.

**2. Оптимальный компромисс:**

- Меньшие размеры страницы (например, 1 КБ): Они могут увеличить накладные расходы на управление таблицами страниц, поскольку требуется большее количество записей.

- Более крупные размеры страницы (например, 64 КБ): Они могут привести к большему количеству «внутренних фрагментаций», когда небольшие объемы данных занимают целую страницу, что в итоге ведет к неэффективному использованию памяти.

**3. Совместимость и стандартизация:**

- Установление стандарта на 4 КБ позволяет создать программное обеспечение, которое гарантированно будет работать на всех системах с аналогичными архитектурными характеристиками. Это также упростило развитие программного обеспечения, поскольку разработчикам не нужно было учитывать разнообразие размеров страниц.

**4. Производительность и эффективность:**

- 4 КБ предоставляет достаточно гранулярной контроль над памятью для эффективного управления страницами, предоставляя хороший баланс между затратами на управление (большое количество записей в таблицах страниц мешало бы) и эффективной загрузкой необходимых данных в кэш памяти.

Важно отметить, что в современных системах часто поддерживаются и страницы большего размера, такие как 2 МБ или 1 ГБ (так называемые «огромные страницы») для задач, которые требуют больших непрерывных блоков памяти, что улучшает производительность за счет сокращения затрат на управление страницами и более эффективного использования кэша процессора.

***9. Сегментно-страничная виртуальная память***

***Сегментно-страничная виртуальная память*** — это комбинированный подход к управлению памятью, который используется для оптимизации использования памяти и повышения гибкости работы с памятью в операционной системе. Он объединяет концепции сегментирования и страничной организации памяти, чтобы воспользоваться преимуществами обеих методик.

***Концепции***

**1. Сегментирование**: Память разделяется на сегменты, которые представляют собой логические единицы, такие как код, данные или стек. Каждый сегмент имеет переменную длину и может начать с любого адреса. Сегменты предоставляют логическую организацию программ и данных, что позволяет, например, отдельным компиляторным модулям или библиотекам использовать разные сегменты.

**2. Страничная организация**: Каждый сегмент, в свою очередь, делится на страницы фиксированного размера (обычно 4 КиБ в большинстве систем). Это позволяет управлять памятью более детально, минимизируя фрагментацию и упрощая задачи распределения памяти.

***Преимущества комбинированного подхода***

**- Гибкость**: Сегментирование позволяет программистам и компиляторам более интуитивно организовать память, в то время как страничная организация помогает эффективно использовать физическую память, исключая проблемы с фрагментацией.

**- Распределение памяти**: Страницы позволяют системе легко распределять и освобождать маленькие блоки памяти, что особенно важно при работе с ограниченными ресурсами.

**- Защита и изоляция**: Сегментно-страничная модель увеличивает уровень защиты, поскольку каждая страница в сегменте может иметь свои права доступа, такие как только чтение или чтение/запись.

**- Разделение и совместное использование**: Модель поддерживает совместное использование сегментов между процессами, если это предусматривает логика выполнения программ, и делает это с минимальными изменениями в физической памяти.

***Применение***

Сегментно-страничная организация памяти применялась в ряде операционных систем, чтобы достичь лучших показателей по использованию памяти и упрощению модели программирования. На практике каждая система может реализовывать ее по-своему в зависимости от требований и архитектуры.

***10. Виртуальная память***

***Виртуальная память*** — это механизм управления ресурсами, используемый в компьютерных системах для расширения объема доступной системы памяти путем использования как физической оперативной памяти (RAM), так и пространства на диске. Этот метод позволяет приложениям работать с большим объемом информации, чем фактически имеется в физической памяти, и упрощает многозадачность.

***Основные концепции виртуальной памяти***

**1. Абстракция памяти*:*** Виртуальная память абстрагирует фактическую физическую память, предоставляя каждому процессу уникальное виртуальное адресное пространство. Это упрощает управление памятью на уровне операционной системы и делает программирование более удобным, так как каждый процесс работает в своей "иллюзорной" среде.

**2. Страничная организация и пагинация**: Виртуальная память часто реализуется с помощью страничной организации, где память делится на страницы фиксированного размера, как в физической, так и в виртуальной памяти. Пагинация позволяет загружать только необходимые страницы данных в оперативную память, сохраняя оставшиеся на диске до момента, когда они понадобятся.

**3. Сегментирование**: В некоторых системах используется также сегментирование, которое разделяет память на сегменты различной длины. Это может быть полезно для разделения различных логических запасов, таких как код, данные и стек.

**4. Свитчинг страниц (переключение):** Когда процессу требуется страница, которая не находится в физической памяти, происходит "пейдж фолт" (ошибка страницы). Система загружает необходимую страницу из дискового пространства в оперативную память, возможно заменяя при этом другую страницу, что позволяет эффективно использовать ограниченные ресурсы RAM.

***Преимущества виртуальной памяти***

**- Расширение доступного пространства**: Программы получают возможность работать с большими объемами данных, чем те, которые доступны в физической памяти.

**- Изоляция процессов:** Каждый процесс работает в своем собственном виртуальном адресном пространстве, обеспечивая его изоляцию и защищая программы друг от друга.

**- Гибкость**: Программы могут работать вне зависимости от того, сколько физической памяти доступно, что особенно полезно в многозадачных системах.

**- Упрощение программирования**: Программистам не нужно учитывать физическое расположение данных в памяти, поскольку работа идет в виртуальных адресах.

***Недостатки и ограничения***

**- Производительность**: Использование виртуальной памяти может замедлить выполнение программ из-за времени доступа к диску, когда страницы попадают в своп-файл.

**- Сложность реализации**: Реализация и управление виртуальной памятью требует сложных алгоритмов и дополнительного специального оборудования, такого как MMU (Memory Management Unit).