inode

Inode (Index Node) в Linux — это структура данных, которая хранит информацию о файле или каталоге, такую как:

— владелец (идентификаторы владельца и группы),
— права доступа,
— дата и время создания и изменения,
— размер,
— расположение на жестком диске(указатели на блоки данных, где фактическое содержимое файла хранится на диске).

Каждый файл или каталог в системе имеет свой уникальный номер индексного узла (inode number), который можно использовать для выполнения различных операций с файлом или каталогом.

Inode имеют ограниченный размер, поэтому количество файлов или каталогов, которые могут быть созданы в файловой системе, ограничено.

Хранение: Айноды хранятся в специальных таблицах на диске, называемых **таблицами айнодов**. Каждая файловая система имеет свою таблицу айнодов, которая создается при создании/форматировании файловой системы. Айноды распределяются по группам блоков (block groups) в файловой системе. Например, в файловых системах ext2/ext3/ext4 каждый блок группы содержит свою таблицу айнодов. **Связь с памятью**: Айнод не хранит данные файла непосредственно, вместо этого он содержит указатели на блоки данных(страницы), где фактическое содержимое файла находится на диске. Когда файл открывается, операционная система загружает данные из этих блоков в оперативную память.

Структура inode

Как айнод управляет переполнением блока

1. **Размер блока**: В файловых системах, таких как ext4, размер блока обычно составляет 4 КБ. Если файл превышает этот размер, система должна использовать дополнительные блоки для хранения оставшейся части файла.
2. **Указатели на блоки данных**: Айнод содержит указатели на блоки данных. Каждый айнод может хранить несколько прямых указателей на блоки данных (например, 12 прямых указателей), а также один или несколько косвенных указателей:
   - **Прямые указатели**: Указывают на первые блоки данных файла.
   - **Косвенные указатели**: Указывают на блоки, которые в свою очередь содержат адреса других блоков данных. Это позволяет файловой системе обращаться к большему количеству данных.
3. **Алгоритм обработки переполнения**:
   - Когда файл создается или модифицируется и его размер превышает размер блока, система проверяет, достаточно ли свободных блоков для хранения данных.
   - Если текущий блок заполняется, файловая система выделяет новый блок и обновляет айнод, добавляя указатель на этот новый блок.
   - Если необходимо, используются косвенные указатели для доступа к дополнительным блокам.

Пример работы с айнодом

Предположим, у вас есть файл размером 10 КБ:

- Первые 4 КБ будут храниться в первом блоке (блок 1).
- Следующие 4 КБ будут храниться во втором блоке (блок 2).
- Оставшиеся 2 КБ будут храниться в третьем блоке (блок 3). Айнод будет содержать прямые указатели на блоки 1, 2 и 3

Заключение

Таким образом, айнод не предотвращает переполнение блока сам по себе, но обеспечивает механизм управления данными и метаданными, который позволяет эффективно обращаться к дополнительным блокам при необходимости. Это достигается за счет использования прямых и косвенных указателей на блоки данных, что позволяет файловым системам обрабатывать файлы любого размера.

Адресное пространство — это

множество адресов, которые могут быть использованы для обращения к данным в памяти. Оно определяет доступные диапазоны адресов, с которыми может работать процессор или программа. т. е. адресное пространство — это совокупность всех допустимых адресов ячеек памяти

Основные аспекты адресного пространства

1. Логическое (виртуальное) адресное пространство:
   - Это адреса, с которыми работает программа.
   - Они не связаны напрямую с физической памятью.
   - Пример: Программа "думает", что у неё есть непрерывный массив памяти от адреса `0x00000000` до, скажем, `0xFFFFFFFF` (в 32-разрядной системе).
   - Логическое адресное пространство создаётся операционной системой с использованием механизма виртуальной памяти.
2. Физическое адресное пространство:
   - Это реальные адреса, соответствующие физической памяти (RAM).
   - Пример: Адреса памяти чипов RAM на материнской плате компьютера.
   - Размер физического адресного пространства ограничен объёмом доступной оперативной памяти.
3. Адресное пространство процессов:
   - Каждому процессу предоставляется собственное виртуальное адресное пространство.

- Даже если несколько процессов используют одинаковые адреса (например, `0x1000`), они ссылаются на разные области в физической памяти или на страницы в файле подкачки.

Структура адресного пространства

Виртуальное адресное пространство обычно делится на несколько регионов, например:

- Код программы: Хранит исполняемый код.
- Данные: Глобальные переменные и статические данные.
- Стек: Используется для временного хранения данных, связанных с вызовами функций.
- Куча (heap): Динамическая память, выделяемая во время выполнения программы.

Пример на 32-разрядной системе

В 32-разрядной системе (4 байта на адрес):

- Логическое адресное пространство: от `0x00000000` до `0xFFFFFFFF` (всего 4 ГБ).
- Только часть из этого диапазона может быть реально сопоставлена с физической памятью (например, если в компьютере 2 ГБ RAM).

Пример на 64-разрядной системе

В 64-разрядной системе теоретическое адресное пространство огромное (16 эксабайт), но современные процессоры и операционные системы накладывают ограничения. Например:

- Практически доступное виртуальное адресное пространство может быть 48 или 57 бит, что ограничивает его до 256 ТБ или 128 ПБ.

Адресное пространство: Итог

Адресное пространство — это модель, которая определяет, какие адреса доступны для использования. Оно может быть:

- Виртуальным (программа видит свой собственный набор адресов, изолированный от других программ).
- Физическим (реальная память, к которой эти адреса сопоставлены).

Swap — это

механизм расширения оперативной памяти за счёт дискового пространства. Он полезен для предотвращения сбоев из-за нехватки RAM, но может негативно повлиять на производительность из-за низкой скорости доступа. Правильная настройка swap позволяет системе работать стабильно и эффективно.

Как работает Swap

1. **Когда используется Swap**:
   - Когда объём физической RAM заканчивается, операционная система перемещает менее активные данные из RAM в область подкачки на диске.
   - Это освобождает RAM для данных и процессов, которые активно используются.
2. **Процесс подкачки (paging)**:
   - Данные перемещаются между RAM и swap по страницам (обычно 4 КБ).
   - Swap хранит страницы данных, которые временно не используются процессами.
3. **Доступ к Swap**:
   - Диск значительно медленнее оперативной памяти, поэтому использование swap замедляет работу системы.
   - Swap используется как временное решение для предотвращения сбоев из-за нехватки памяти, но не заменяет RAM.

Преимущества Swap

1. **Предотвращение сбоев**:
   - Когда RAM полностью заполнена, система продолжает работать за счёт swap.
2. **Обработка больших нагрузок**:
   - Позволяет запускать больше процессов, чем позволяет объём физической памяти.
3. **Изоляция данных** (в случае swap-раздела):
   - Данные подкачки хранятся отдельно, что может быть полезно для безопасности.

Недостатки Swap

1. **Медленная скорость**:
   - Диск (даже SSD) гораздо медленнее, чем RAM, поэтому операции с данными в swap сильно замедляют работу системы.
2. **Износ SSD**:
   - Частое использование swap может сократить срок службы SSD из-за интенсивной записи.
3. **Не заменяет RAM**:
   - Swap только временное решение и не может обеспечить ту же производительность, что и оперативная память.

Команда в Linux — это

инструкция, которую пользователь даёт операционной системе для выполнения определённого действия.

Исполняемые скрипты (создание и способы запуска)

Bash-скрипт — это файл, содержащий последовательность команд, которые выполняются программой bash построчно. Он позволяет выполнять ряд действий, таких как переход к определенному каталогу, создание папки и запуск процесса с помощью командной строки.

Абсолютный и относительный путь

В операционной системе Linux может быть несколько видов путей к файлу:

Полный, абсолютный путь linux от корня файловой системы — начинается от корня «/» и описывает весь путь к файлу. Например: «/home/user/myfile»

Относительный путь linux — это путь к файлу относительно текущей папки. Например (для файла находящегося в родительской папке): «../myfile».

Путь относительно домашний папки текущего пользователя — путь в файловой системе, только не от корня, а от папки текущего пользователя. Чтобы задать путь подобным образом он должен начинаться с «~/». Например: «~/myfile».

Символические и жесткие ссылки

Символические ссылки

Символические ссылки более всего похожи на обычные ярлыки. Они содержат адрес нужного файла в вашей файловой системе. Когда вы пытаетесь открыть такую ссылку, то открывается целевой файл или папка. Главное ее отличие от жестких ссылок в том, что при удалении целевого файла ссылка останется, но она будет указывать в никуда, поскольку файла на самом деле больше нет.

Вот основные особенности символических ссылок:

- Могут ссылаться на файлы и каталоги;
- После удаления, перемещения или переименования файла становятся недействительными;
- Права доступа и номер inode отличаются от исходного файла;
- При изменении прав доступа для исходного файла, права на ссылку останутся неизменными;

- Можно ссылаться на другие разделы диска;
- Содержат только имя файла, а не его содержимое.

Теперь давайте рассмотрим жесткие ссылки.

Жесткие ссылки

Этот тип ссылок реализован на более низком уровне файловой системы. Файл размещен только в определенном месте жесткого диска. Но на это место могут ссылаться несколько ссылок из файловой системы. Каждая из ссылок - это отдельный файл, но ведут они к одному участку жесткого диска. Файл можно перемещать между каталогами, и все ссылки останутся рабочими, поскольку для них неважно имя. Рассмотрим особенности:

- Работают только в пределах одной файловой системы;
- Нельзя ссылаться на каталоги;
- Имеют ту же информацию inode и набор разрешений что и у исходного файла;
- Разрешения на ссылку изменятся при изменении разрешений файла;
- Можно перемещать и переименовывать и даже удалять файл без вреда ссылке.

Перенаправление потоков

Перенаправление потоков — это возможность командной оболочки перенаправлять стандартные потоки в определённое пользователем место, например в файл. 3

В Linux существует три стандартных потока ввода/вывода данных: 1

1. **Стандартный поток ввода** (standard input, поток №0). Представляет собой информацию, передаваемую в терминал, в частности — инструкции, переданные в оболочку для выполнения. Обычно данные в этот поток попадают в ходе ввода их пользователем с клавиатуры. 1
2. **Стандартный поток вывода** (standard output, поток №1). Это поток данных, которые оболочка выводит после выполнения каких-то действий. Обычно эти данные попадают в то же окно терминала, где была введена команда, вызвавшая их появление. 1
3. **Стандартный поток ошибок** (standard error, поток №2). Этот поток похож на стандартный поток вывода, так как обычно то, что в него попадает, оказывается на экране терминала. 1

Перенаправление осуществляется с использованием символов > и < в различных комбинациях, применение которых зависит от того, куда в итоге должны попасть перенаправляемые данные. 1

Двойное перенаправление дописывает данные с вывода к данным целевого объекта. **Одинарный символ перенаправления** заменяет содержимое целевого объекта на перенаправляемый вывод. 2

Файловый дескриптор — это **неотрицательное число, которое является идентификатором потока ввода-вывода**. Дескриптор может быть связан с файлом, каталогом, сокетом. 1

Обычно файловые дескрипторы выделяются последовательно: есть пул свободных номеров. Когда создаётся новый файл или открывается существующий, ему присваивается номер. Следующий файл получает очередной номер — например, 101, 102, 103 и так далее. 1

Некоторые жёстко закреплённые индексы дескрипторов:

- **0** — стандартный ввод (stdin), место, из которого программа получает интерактивный ввод; 1
- **1** — стандартный вывод (stdout), на который направлена большая часть вывода программы; 1
- **2** — стандартный поток ошибок (stderror), в который направляются сообщения об ошибках. 1

Когда завершается работа с файлом, присвоенный ему дескриптор освобождается и возвращается в пул свободных номеров. Он снова доступен для выделения под новый файл. 1

Способы создания и заполнения файлов

1) `touch file` 2) `> file` - сразу открывается файл, для выхода control + d 3) `cat > file` 4) `nano file` 5) `vim file`

Основные команды в Linux

Знать всё с этого скрина![[{C725DF93-F893-4AFA-B255-9BC1BEB2D003}.png]] chmod chown chgroup usermod

Команды:

`man` (от англ. manual — руководство) — команда Unix, предназначенная для форматирования и вывода справочных страниц. Поставляется почти со всеми UNIX-подобными дистрибутивами.1

Синтаксис команды имеет общий вид: `man [option] [command]`. Здесь option относится к дополнительным флагам, которые изменяют поведение команды man, command — это команда Linux или утилита, для которой нужно получить доступ к руководству. `-[options]`:

- `-a`, `--all` - отображает все страницы руководства, связанные с заданной командой, вместо первой страницы. -
- `f`, `--whatis` - выводит краткое описание заданной команды.
- `-k`, `--apropos` - ищет команды, содержащие указанное ключевое слово в их описании.
- `-w`, `--where` - отображает путь к странице руководства, связанной с заданной командой.
- `-h`, `--help` - отображает справочную информацию о команде man и её опциях.

`echo` — это встроенная команда, которая позволяет пользователям отображать строки текста или строки, передаваемые в качестве аргументов 1.

Синтаксис команды:

`echo -[options] -[string]`

`-[options]` для лабы:

- `-n` - не выводить перевод строки;
- `-e` - включить поддержку вывода Escape последовательностей;
- `-E` - отключить интерпретацию Escape последовательностей. `- [string]` — это строка, которую нужно отобразить

Если включена опция -e, то вы можете использовать такие Escape последовательности для вставки специальных символов:

- `/c` — удалить перевод строки;
- `/t` — горизонтальная табуляция;
- `/v` — вертикальная табуляция;
- `/b` — удалить предыдущий символ;
- `/n` — перевод строки;
- `/r` — символ возврата каретки в начало строки.

Основное использование echo: отображение текста или строки в терминале Для этого нужно просто предоставить желаемый текст или строку в качестве аргумента команде echo **Примеры:**

- echo "Hello, Linux!" Эта команда выведет на терминал «Hello, Linux!» Опция `-e` включает интерпретацию escape-символов, в результате чего получится двухстрочный вывод

```
name="User"
echo -e "Hello\n$name!"
```

В результате получится

```
Hello
User!
```

путём подстановки значения имени переменной в строку 2.

Echo также может использоваться для перенаправления вывода в файл с помощью операторов > или >> Например:

```
echo "Save this line to a file" > output.txt
```

При этом выходные данные команды echo будут записаны в имя файла output.txt

`cat` — это утилита, которая читает данные из файла или стандартного ввода и выводит их на экран 1.

С помощью команды cat можно:

- посмотреть содержимое небольшого файла 1;
- склеить несколько файлов 21;
- исключить ошибки случайного изменения файла при просмотре 2.

Синтаксис команды cat:

```
cat -[options] -[file2] -[file2] ...
```

`-[options]` для лабы

- `-n` — нумеровать все строки

Все `-[options]`:

- `-b` — нумеровать только непустые строки;
- `-E` — показывать символ $ в конце каждой строки;
- `-s` — удалять пустые повторяющиеся строки;
- `-T` — отображать табуляции в виде ^I;
- `-h` — вывести на экран монитора справочную информацию;
- `-v` — отобразить текущую версию утилиты.

Примеры использования команды cat:

- Просмотр содержимого файла: `cat file
- Вывод нескольких файлов: `cat file1 file2 file3`
- Комбинирование вывода текста из файла и стандартного ввода: cat file - file1`
- При отсутствии параметров и опций: `cat`
- В этом случае команда будет отображать данные из стандартного ввода
- Направление данных в указанный файл на диске: `cat > file1`
- Запись данных из нескольких файлов в один: `cat file1 file2 > file3`

`grep` — утилита командной строки, которая находит на вводе строки, отвечающие заданному регулярному выражению, и выводит их, если вывод не отменён специальным ключом.

Синтаксис команды:

```
grep -[options] template /path/to/the/file_or_directory
```

или

```
command |&amp;nbsp;grep&amp;nbsp;-[options]&amp;nbsp;template
```

`-[options]` для лабы: `-v` --invert-match - вывести только те строки, в которых шаблон поиска не найден

все `-[options]`:

Site Unreachable

`chown` (от англ. change owner) — утилита в UNIX-подобных операционных системах, которая изменяет владельца и/или группу для указанных файлов.

Синтаксис команды:

```
chown [user] /path/to/the/file
```

В поле пользователь надо указать пользователя, которому мы хотим передать файл. Также можно указать через двоеточие группу, например, `[user]:[group]`. Тогда изменится не только пользователь, но и группа.

Пример использования:

- Изменение владельца папки dir1 на root:

```
drwxr-xr-x 3 ipka23 staff 4096 авг 6 23:23 dir1
```

```
chown root ./dir1
```

```
drwxr-xr-x 3 root staff 4096 авг 6 23:23 dir1
```

- Можно поменять сразу владельца и группу каталога или файла

```
drwxr-xr-x 3 ipka23 staff 4096 авг 6 23:23 dir1
```

```bash chown root:root ./dir2
```

```
drwxr-xr-x 3 root root 4096 авг 6 23:23 dir1
```

cp — команда Unix , предназначенная для копирования файлов из одного в другие каталоги (возможно, с другой файловой системой).

Синтаксис команды:

```
cp -[options] path/to/source/file path/to/destinatoin/file
```

`-[options]` для лабы:

- `-l`, --link - создать жесткую ссылку;
- `-s`, --symbolic-link - создать символическую ссылку;
- `-r`, --recursive - копировать папку Linux рекурсивно;

Все `-[options]`:

- `--attributes-only` - не копировать содержимое файла, а только флаги доступа и владельца;
- `-b`, `--backup` - создать резервную копию файла назначения, если он существует;
- `--copy-contents` - копировать содержимое для специальных файлов (сокеты, файлы устройств);
- `-f`, `--force` - удалить файл назначения перед попыткой записи в него, если он существует;
- `-i`, `--interactive` - спрашивать, нужно ли перезаписывать существующие файлы;
- `-n`, `--no-clobber` - не перезаписывать существующие файлы;
- `-P`, `--no-dereference` - копировать сами символические ссылки, а не то, на что они указывают;
- `-L`, `--dereference` - копировать не символические ссылки, а то, на что они указывают;
- `-l`, `--link` - создавать жесткие ссылки вместо копирования;
- `--preserve` - переносить указанные атрибуты с файла источника в файл назначения. Возможные значения: mode, ownership, time-stamps, context, links, xattr, all;
- `--no-preserve` - не переносить указанные атрибуты;
- `--parents` - сохранять путь, указанный в файле источнике, в папке назначения;
- `-r`, `--recursive` - копировать папку Linux рекурсивно;
- `--reflink` - использовать Copy on Write, если это поддерживается файловой системой;
- `-s`, `--symbolic-link` - не выполнять копирование файлов в Linux, а создавать символические ссылки;
- `-S`, `--suffix` - указать суффикс для резервных копий файлов;
- `--sparse` - настройка работы с разреженными файлами;
- `-t`, `--target-directory` - считать файл назначения директорией и копировать файл источника или директорию источника в эту директорию с оригинальным именем;
- `-T`, `--no-target-directory` - считать директорию назначения файлом или директорией для записи данных. Если в качестве источника выбран файл, то он будет скопирован с новым именем. Если директория, то её содержимое будет скопировано в директорию назначения;
- `-u`, `--upgrade` - скопировать файл только если он был изменён;
- `-x`, `--one-file-system` - рекурсивное копирование не должно выходить за пределы этой файловой системы;
- `-v`, `--verbose` - максимально подробный вывод;
- `-p` - сохранять владельца, временные метки и флаги доступа при копировании, аналогично --preserve=mode,ownership,timestamps;
- `-d` - копировать символические и жесткие ссылки именно как ссылки, аналогично --no-dereference --preserve=links;
- `-a` - режим резервного копирования, при котором сохраняются все атрибуты и ссылки, а также выполняется резервное копирование папок, аналогично --recursive --preserve=all --no-dereference. **Примеры использования команды cp**:
- Копирование одного файла в другую папку: `cp файл.txt /путь/к/папке/`
- Копирование папки и её содержимого: `cp -r папка /путь/к/целевой/папке/`
- Копирование с заменой существующих файлов: `cp -f файл.txt /путь/к/папке/`
- Копирование с подтверждением перезаписи результирующего файла, если он существует: `cp -i файл.txt /путь/к/папке/`

Основа философии UNIX

- все объекты операционной системы - это файлы, для предоставления доступа к тем или иным возможностям системы мы просто даем доступ пользователю к нужным файлам или убираем. Для внутреннего хранения файла файловая система разбивает хранилище на **блоки**. Традиционным **размером блока** были 512 байт, но более актуальное значение — 4 кибибайта. Вообще же при выборе этого значения руководствуются поддерживаемым **размером** страницы на типовом оборудовании MMU (memory management unit, «устройство управления памятью» — прим. перев.). Чтобы понять, как файл может быть директорией в Unix и Linux, важно рассмотреть концепцию файловой системы и то, как она организована.

Единая модель файловой системы

1. **Все — это файлы**: В Unix-подобных операционных системах существует принцип, согласно которому все объекты, включая обычные файлы, устройства и директории, рассматриваются как файлы. Это означает, что для взаимодействия с любым объектом используется одинаковый набор системных вызовов и команд. Директория — это просто специальный тип файла, который выполняет уникальную функцию.
2. **Структура директории**: Директория содержит записи о других файлах и подкаталогах. Каждая запись включает имя файла и указатель на его inode — структуру данных, содержащую информацию о файле (например, местоположение на диске). Таким образом, директория не хранит данные в привычном смысле, а лишь ссылки на другие файлы[2][4].

Иерархия файловой системы

3. **Древовидная структура**: Файловая система в Unix организована в виде дерева, где корневая директория обозначается как `/`. Все остальные файлы и директории располагаются внутри этой корневой директории. Директории могут содержать как обычные файлы, так и другие директории (подкаталоги), создавая иерархическую структуру[1][3].

4. **Типы файлов**: В Unix существуют различные типы файлов, включая обычные файлы и каталоги. Директория обозначается специальным символом `d` в выводе команды `ls -l`, что указывает на её статус как каталога[2][5].

Преимущества такой структуры

5. **Упрощение управления**: Рассматривая директории как файлы, операционная система может использовать одни и те же механизмы для работы с файлами и директориями. Это упрощает разработку программного обеспечения и управление данными, так как все объекты обрабатываются по единым правилам[4][5].

6. **Навигация и управление**: Пользователи могут легко перемещаться по файловой системе с помощью команд, таких как `cd` (для перехода между директориями) и `ls` (для просмотра содержимого). Это делает работу с файлами и каталогами более интуитивной[1][3].

Таким образом, хотя файл и директория выполняют разные функции (файл хранит данные, а директория организует их), концепция директории как файла позволяет создать более простую и унифицированную модель для работы с данными в Unix-подобных системах.

права доступа, какие бывают, как назначается все, что знаешь про права доступа абсолютно всё

Основные права доступа к файлам в Linux

Изначально каждый файл имел три параметра доступа. Вот они:

- **Чтение** - разрешает получать содержимое файла, но на запись нет. Для каталога позволяет получить список файлов и каталогов, расположенных в нем;
- **Запись** - разрешает записывать новые данные в файл или изменять существующие, а также позволяет создавать и изменять файлы и каталоги;
- **Выполнение** - вы не можете выполнить программу, если у нее нет флага выполнения. Этот атрибут устанавливается для всех программ и скриптов, именно с помощью него система может понять, что этот файл нужно запускать как программу. Но все эти права были бы бессмысленными, если бы применялись сразу для всех пользователей. Поэтому каждый файл имеет три категории пользователей, для которых можно устанавливать различные сочетания прав доступа:
- **Владелец** - набор прав для владельца файла, пользователя, который его создал или сейчас установлен его владельцем. Обычно владелец имеет все права, чтение, запись и выполнение.
- **Группа** - любая группа пользователей, существующая в системе и привязанная к файлу. Но это может быть только одна группа и обычно это группа владельца, хотя для файла можно назначить и другую группу.
- **Остальные** - все пользователи, кроме владельца и пользователей, входящих в группу файла.

Права доступа к каталогам

Каталог можно представить как таблицу, содержащую много записей, каждая из которых состоит из двух полей: имя и номер индексного дескриптора. В этой модели **право на запись в каталог означает право на создание и удаление записей**, т.е. создание файлов в каталоге, создание новых имен для существующих файлов (**link**), удаление имен файлов (возможно вместе с файлами) (**unlink**). Для удаления файла нет необходимости иметь право на операции с файлом, достаточно иметь право на запись в каталог, в котором хранится его последнее имя.1

Право на чтение означает для каталога право на получение списка имён (левой колонки в нашей модели), а **право на исполнение – доступ к номерам индексных дескрипторов** (правой колонке). В норме оба права должны использоваться одновременно. Если отсутствует право на выполнение, то имеющий право на чтение получит список имен файлов в каталоге, но не сможет ни узнать их метаданные (владелец, размер и т.п.), ни получить доступ к данным. Если отсутствует право на чтение, то становится невозможно узнать имена файлов в каталоге. Однако, если имя известно из других источников, то доступ к файлу можно получить стандартным образом.

Специальные права доступа к файлам в Linux

Для того, чтобы позволить обычным пользователям выполнять программы от имени суперпользователя без знания его пароля была придумана такая вещь, как SUID и SGID биты. Рассмотрим эти полномочия подробнее.

- **SUID** - если этот бит установлен, то при выполнении программы, id пользователя, от которого она запущена заменяется на id владельца файла. Фактически, это позволяет обычным пользователям запускать программы от имени суперпользователя;
- **SGID** - этот флаг работает аналогичным образом, только разница в том, что пользователь считается членом группы, с которой связан файл, а не групп, к которым он действительно принадлежит. Если SGID флаг установлен на каталог, все файлы, созданные в нем, будут связаны с группой каталога, а не пользователя. Такое поведение используется для организации общих папок;
- **Sticky-bit** - этот бит тоже используется для создания общих папок. Если он установлен, то пользователи могут только создавать, читать и выполнять файлы, но не могут удалять файлы, принадлежащие другим пользователям.

команда ls -l, что выводит подробно про каждую колонку.

![[Pasted image 20241021235856.png]] (размер файла в байтах) **Группа staff в macOS** — это стандартная группа, в которую автоматически входят все пользователи без прав администратора(root).

- Административные права, также известные как права суперпользователя или root, представляют собой наивысший уровень доступа в операционной системе. Эти права позволяют пользователю выполнять любые действия, включая: • Изменение системных настроек. • Установку и удаление программного обеспечения. • Управление учетными записями пользователей. • Доступ к защищенным файлам и ресурсам. В macOS и Linux доступ к этим правам обычно осуществляется через учетную запись root или с помощью команд, таких как `sudo`, что позволяет временно повышать уровень привилегий для выполнения административных задач.

Переменные окружения — это

значения, которые используются в различных командах и программных сценариях, выполняемых в операционной системе. Принципиально они работают точно так же, как переменные в языках программирования. Они представляют знакомые нам пары «ключ-значение» и используются для хранения параметров, настроек приложений, хранения ключей и других информационных данных. Посмотреть установленные переменные можно командой `env` (environment) или `printenv`. [Что такое переменные окружения, зачем они нужны и как их использовать / Skillbox Media](https://skillbox.ru/media/code/kak-poyavilis-i-menyalis-peremennye-okruzheniya/

Виды переменных окружения

Если смотреть более широко, переменная окружения может быть трех типов:1

1. Локальные переменные окружения

Эти переменные определены только для текущей сессии. Они будут безвозвратно стерты после завершения сессии, будь то удаленный доступ или эмулятор терминала. Они не хранятся ни в каких файлах, а создаются и удаляются с помощью специальных команд.

2. Пользовательские переменные оболочки

Эти переменные оболочки в Linux определяются для конкретного пользователя и загружаются каждый раз когда он входит в систему при помощи локального терминала, или же подключается удаленно. Такие переменные, как правило, хранятся в файлах конфигурации: .bashrc, .bash*profile*, .*bash*login, .profile или в других файлах, размещенных в директории пользователя.

3. Системные переменные окружения

Эти переменные доступны во всей системе, для всех пользователей. Они загружаются при старте системы из системных файлов конфигурации: /etc/environment, /etc/profile, /etc/profile.d/ /etc/bash.bashrc.

какое время жизни у переменной окружения? как перед исполнением в команде заменяются названия переменных на их значения, звездочки на списки файлов и т.д.

Время жизни переменной окружения

Сеансовая природа: Переменные окружения существуют в рамках сеанса терминала. Это означает, что они создаются и доступны только в том сеансе, в котором были заданы. Когда вы закрываете терминал или выходите из сеанса, все переменные окружения, созданные в этом сеансе, исчезают.

Замена названий переменных и других символов

1. **Подстановка значений**: Когда вы вводите команду в терминале, оболочка автоматически заменяет названия переменных на их значения перед выполнением команды. Например:

```
echo $MY_VAR

Если `MY_VAR` содержит значение `Hello`, то команда будет выполнена как:

    echo Hello
    ```


2. **Расширение метасимволов**: Звездочка (`*`) и другие метасимволы также обрабатываются оболочкой перед выполнением команды. Например:bash
    `ls *.txt`
    Здесь `*.txt` будет заменено на список всех файлов с расширением `.txt` в текущей директории.
3. **Обработка аргументов**: Оболочка обрабатывает специальные символы и конструкции (например, `$`, `*`, `?`) для замены их на соответст
Таким образом, время жизни переменной окружения ограничено сеансом терминала, а замена названий переменных и других символов происходит автоматически пр
Окружение — это
список пар «ключ — значение», который передаётся выполняемой программе так же, как и обычный список аргументов.
Посмотреть текущее окружение можно командой `env` (environment) или `printenv`
Что такое оболочка
- Shell (оболочка, также известная как терминал, консоль или командная строка) — это **текстовый интерфейс пользователя, через который команды отпр
Переменные оболочки
Переменные оболочки в Unix — это **локальные переменные, которые содержатся исключительно в оболочке, в которой они были установлены или с

Для просмотра переменных оболочки нужно ввести команду **set** без параметров. [1](https://dzen.ru/a/Y0kywyJbshsSZWj9)
разница окружения и оболочки
- **Окружение** — это область, которую оболочка создаёт каждый раз при запуске сеанса, где содержатся переменные, определяющие свойства системы. Он

Файловая система это
способ хранить и организовывать информацию на каком-нибудь носителе.
От файловой системы зависит, как файлы будут кодироваться, храниться на диске и читаться компьютером. [1](https://blog.skillfactory.ru/glossary/fay

Файловая система есть на каждом жёстком диске, SSD-накопителе или флешке. [1](https://blog.skillfactory.ru/glossary/faylovaya-sistema/) Она от

Основные функции файловой системы:

- размещение и упорядочивание данных; [2](https://www.roksis.ru/articles/chto-takoe-faylovaya-sistema/)
- создание, чтение и удаление файлов; [2](https://www.roksis.ru/articles/chto-takoe-faylovaya-sistema/)
- назначение и изменение атрибутов файлов (размер, время создания и изменения, владелец и создатель); [2](https://www.roksis.ru/articles/chto-takoe
- предоставление доступа прикладного программного обеспечения к накопителю; [2](https://www.roksis.ru/articles/chto-takoe-faylovaya-sistema/)
- защита данных от случайных повреждений; [2](https://www.roksis.ru/articles/chto-takoe-faylovaya-sistema/)
- поиск файлов по запросу программного обеспечения или пользователя. [2](https://www.roksis.ru/articles/chto-takoe-faylovaya-sistema/)
4 кибибайта страница, что это такое?
- **Размер блока**: Ext4 использует блоки размером 4 КБ по умолчанию, что позволяет эффективно управлять пространством на диске. Это размер страницы, ко

Виртуальная память
- Виртуальная память (Virtual Memory) - Linux
использует концепцию виртуальной памяти,
которая создает иллюзию наличия у
каждого процесса своего личного
пространства памяти. Виртуальная память
позволяет системе исполнять код
приложений, используя больший объем
памяти, чем физически доступно. Это
достигается путем сброса неиспользуемых
блоков памяти приложений на диск

Страница
- Система страница (Paging) - физическая и
виртуальная память разделены на блоки
фиксированного размера, которые
называются страницами. Система страниц
позволяет эффективно управлять памятью
и активирует механизм обмена данными
между ОЗУ и диском (swap)

Swap в UNIX-системах — это
механизм виртуальной памяти, при котором отдельные фрагменты памяти (обычно неактивные) перемещаются из ОЗУ во вторичное хранилище (жёсткий диск или

В качестве swap в UNIX-подобных системах используют **отдельные разделы жёсткого диска или swap-файлы**. [1](https://acm.bsu.by/wiki/Unix2017b

Swap используется для загрузки приложений, которым требуется больше памяти, чем физически присутствует в компьютере. Когда ОЗУ полна, данные активных пр

Чтобы узнать текущие тип и расположение swap, можно использовать команду **swapon -s**
С точки зрения пользователя сессия - это
процесс работы с текстовым терминалом с момента ввода имени и пароля и до выхода из системы командой logout** (exit, нажатие ^D в пустой строке).

Что такое сегментно страничная виртуальная память
При реализации страничной виртуальной памяти виртуальное адресное пространство делится на части одинакового фиксированного размера, называемые виртуальн
Страничное распределение памяти предусматривает только механическое разбиение на страницы, оно не позволяет различным образом организовать работу со стр

ОЗУ
Оперативная память может быть представлена как совокупность **сегментов** и **адресов**, но эти понятия зависят от контекста использования. Рассмотрим э

Адреса в оперативной памяти

- **Линейное адресное пространство**:

    - Оперативная память организована в виде непрерывного массива адресов, начиная с `0x00000000` (или другого минимального адреса) и заканчивая максима
    - Каждый адрес соответствует конкретной ячейке памяти, способной хранить минимальную единицу данных (обычно 1 байт).
    - В 32-разрядных системах адресное пространство может быть до 4 ГБ (2^{32}), а в 64-разрядных — теоретически до 16 эксабайт.
- **Физические и виртуальные адреса**:

    - Физические адреса соответствуют реальным ячейкам памяти в RAM.
    - Виртуальные адреса — это логические адреса, используемые программами. Операционная система и процессор сопоставляют их с физическими адресами с по

Сегменты в оперативной памяти

Сегментация — это способ организации памяти, который использовался в некоторых архитектурах (например, в x86) для удобства работы с большими адресны

- **Сегменты в x86-архитектуре**:

    - В 16-разрядной и 32-разрядной архитектурах x86 память делилась на сегменты:
        - **Кодовый сегмент (Code Segment, CS)**: Хранит инструкции программы.
        - **Данные (Data Segment, DS)**: Хранятся глобальные и статические данные.
        - **Сегмент стека (Stack Segment, SS)**: Хранит адреса возврата и временные данные, используемые в вызовах функций.
        - **Дополнительные сегменты**: Например, ES, FS, GS для расширения возможностей.
- **Современные системы**:

    - В современных 64-разрядных системах сегментация практически не используется для управления памятью. Вместо этого применяется **постраничная органи

Связь между адресами и сегментами

- В архитектурах с сегментацией каждый сегмент имеет свой **базовый адрес** (начало) и **размер**.
- Адрес в памяти получается как сумма:
    - Базового адреса сегмента.
    - Смещения (offset), указывающего конкретное местоположение внутри сегмента.

Пример (в старых системах):

- Сегмент `0x1000`, смещение `0x0020` → итоговый адрес: `0x1000 + 0x0020 = 0x1020`.

Память в современных системах

В современных системах память организована следующим образом:

1. **Стек**:

    - Используется для временных данных, таких как локальные переменные и адреса возврата.
    - Работает по принципу "последний пришёл — первый ушёл" (LIFO).
2. **Куча (heap)**:

    - Для динамически выделяемой памяти.
    - Управляется вручную программистом через функции, такие как `malloc()` или `new`.
3. **Глобальные данные**:

    - Содержат глобальные и статические переменные.
    - Разделяются на инициализированные и неинициализированные данные.
4. **Код программы**:

    - Содержит исполняемый код, который процессор выполняет.
5. **Пустые или зарезервированные области**:

    - Например, адресное пространство может содержать области, зарезервированные для системы.

Итог

- **Оперативная память** состоит из **адресов**, которые являются линейным представлением ячеек памяти.
- В некоторых архитектурах память может быть организована в **сегменты**, каждый из которых отвечает за определённую область данных или кода.
- В современных системах сегментация заменена постраничной организацией, что делает управление памятью более гибким.

Что значит total 16?

ipka23@Air-Ila-3 Desktop % ls -lai total 16 260804 drwx------@ 6 ipka23 staff 192 19 ноя 21:38 . 260799 drwxr-x---+ 39 ipka23 staff 1248 19 ноя 22:31 .. 596768 -rw-r--r--@ 1 ipka23 staff 6148 19 ноя 21:36 .DS_Store 2454186 drwxr-xr-x 8 ipka23 staff 256 26 окт 12:48 .idea 260805 -rw-r--r-- 1 ipka23 staff 0 30 авг 21:01 .localized 3015321 drwxr-xr-x@ 8 ipka23 staff 256 24 окт 13:56 ITMO

Подробности о значении "total 16":

1. **Общее количество блоков**: Значение `16` указывает на то, что для хранения файлов и подкаталогов в данной директории используется 16 блок
2. **Блоки и размер**: Чтобы понять, сколько это в байтах, вы можете умножить количество блоков на размер блока. Например, если размер блока составляет

```
    16 ×4096 =65536 (или 64 )16 ×4096 =65536 (или64 )

    Это означает, что файлы и подкаталоги в этой директории занимают примерно 64 КБ дискового пространства.
3. **Контекст использования**: Эта информация полезна для понимания того, сколько места занимают файлы в директории, и может помочь при управлении диско
Таким образом, строка `total 16` дает представление о том, сколько блоков используется для хранения содержимого текущей директории.
##### Почему есть различия между действительным размером и размером с блоками?
1. **Блоки на диске**: Файловые системы обычно выделяют пространство на диске в блоках фиксированного размера (обычно 512 байт или 4 КБ). Даже если файл
2. **Общее количество блоков**: Значение `total` показывает общее количество блоков, которые занимают все файлы и каталоги в директории. Это число может
3. **Разные метрики**: Размер директории (6674 Б) — это фактический размер данных, хранящихся в ней, тогда как `total 24` — это количество блоков, котор
Таким образом, разница между фактическим размером директории и значением `total` объясняется тем, как файловая система управляет пространством на диске
    команда для просмотра размера блока на macOS:
```

sh diskutil info / | grep "Block Size" ```

какие атрибуты есть у пользователя в linux?

У пользователя в Linux есть следующие атрибуты:

- **Логин** для входа в систему. 2
- **Пароль**. 2
- **Оболочка терминала** (bash, shell и др.). 2
- **Персональные данные** (имя, фамилия, компания и т. д.). 2
- **Принадлежность к группам** в системе. 2

Также может быть **атрибут комментария**, содержание которого произвольно (например, настоящие имя и фамилия пользователя). В BSD-системах есть ещё и **атрибут класса пользователя**. 3

Виртуальная память

У систем с виртуальной памятью четыре основных свойства:

1. Пользовательские процессы изолированы друг от друга и, умирая, не тянут за собой всю систему
2. Пользовательские процессы изолированы от физической памяти, то есть знать не знают, сколько у вас на самом деле оперативки и по каким адресам она находится.
3. Операционная система гораздо сложнее, чем в системах без виртуальной памяти
4. Никогда нельзя знать заранее, сколько времени займет выполнение следующей команды процессора **MMU (Memory Management Unit)** — это аппаратный компонент, расположенный между процессором и основной памятью. 3 **Основная функция MMU** заключается в преобразовании адресов, формируемых в процессе выполнения команд (так называемых виртуальных адресов), в физические адреса, по которым и выполняются обращения к памяти. Кроме того, MMU контролирует допустимость доступов по тем или иным виртуальным адресам. 1

TLB (Translation Lookaside Buffer) — это **кэш-память, которая хранит наиболее часто используемые записи таблиц переадресации**. Она входит в состав MMU и используется, чтобы избежать необходимости постоянно обращаться к памяти для выборки строк таблиц переадресации. 1 TLB может располагаться между процессором и кэшем процессора, между кэшем процессора и основной памятью или между разными уровнями многоуровневого кэша. 3

TLB - это

ассоциативный массив фиксированного размера, который по предъявлении ключа - номера виртуальной страницы умеет находить соответствующий номер физической страницы (если он ему известен). За содержимым TLB присматривает ядро ОС.

Операционная система — это

комплекс программ, который управляет работой компьютера и предоставляет удобный интерфейс для пользователя.

Ядро — это

центральный компонент ОС, который отвечает за взаимодействие с аппаратным обеспечением и выполнение системных функций.

Прошивка — это

базовый уровень программного обеспечения, предназначенный для управления аппаратным обеспечением. Она запускается первой и обеспечивает работу устройства на базовом уровне.

Сигналы в Linux — это

программные прерывания, которые посылаются процессу, когда случается некоторое событие. 1 Они обеспечивают способ для пользователя или процесса напрямую общаться с процессом.

Задачка от Жака Фреско №1:

Есть директория у директории по дефолту размер 4096 байт, она занимает 1 блок. Если мы посмотрим на файл, он тоже занимает 1 блок. Если немножечко переступить размером файла 1 блок, то размер будет 8 кибибайт. Почему? **Ответ:** Потому что под каждый файл в UNIX-системе выделяется минимум 1 блок памяти = 4096 байт = 4 кибибайт, т.е. если размер файла > 4 кибибайт, под него выделяется еще один блок размером 4 кибибайт, значит итоговый размер = 8 кибибайт

Задачка от Жака Фреско №2:

Создаем директорию -> переходим в нее -> создаем файл в директории -> создаем 10 жестких ссылок на созданный файл -> удаляем “исходный” файл. Вопрос: сколько жестких ссылок в директории? **Ответ:** 12, так как было 1(от созданного файла)+10(от жестких ссылок)-1(удалили файл)+2(от директории “.” + “..”)