# Операционные системы

Введение в файловые системы

# Определение файловой системы

- *Файловая система* – это часть ОС, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и/или процессами.

# Понятие файловой системы

- В широком смысле понятие “ФС” включает:
  - совокупность всех файлов на диске;
  - наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;
  - комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

# Файлы

- ***Файл*** – это именованный набор связанной информации, записанной во вторичную память.
- *Файлы* представляют собой абстрактные объекты, задача которых – хранить информацию, скрывая от пользователя детали работы с устройствами.

# Имена файлов

- Файлы идентифицируются символьными именами, которые им дают пользователи.
- Общий формат символьного имени файла:
  <ИМЯ>.<РАСШИРЕНИЕ>
- Символьные имена могут быть различной длины в зависимости от типа ФС.
- Поддержка национальных алфавитов – символьные имена в формате UNICODE.
- Присваивание нескольких символьных имен одному файлу.
- Возможное различие строчных и прописных букв.

# Типы файлов

- обычные файлы:
- специальные файлы;
- файлы-каталоги.

# Обычные файлы

- *Обычные файлы* в свою очередь подразделяются на текстовые и двоичные:
  - Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере.
  - Двоичные файлы не используют ASCII-коды, они часто имеют сложную внутреннюю структуру. Например, исполняемый *файл* в ОС Unix имеет пять секций: заголовок, текст, данные, биты реаллокации и символьную таблицу. ОС выполняет *файл*, только если он имеет нужный формат. Другим примером бинарного *файла* может быть архивный *файл.* Все операционные системы должны уметь распознавать хотя бы один тип файлов – их собственные исполняемые файлы.

# Специальные файлы

- *Специальные файлы* – это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла.
- Команды чтения/записи обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Так, например, клавиатура обычно рассматривается как текстовый *файл*, из которого компьютер получает данные в символьном формате.
- Специальные файлы, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.

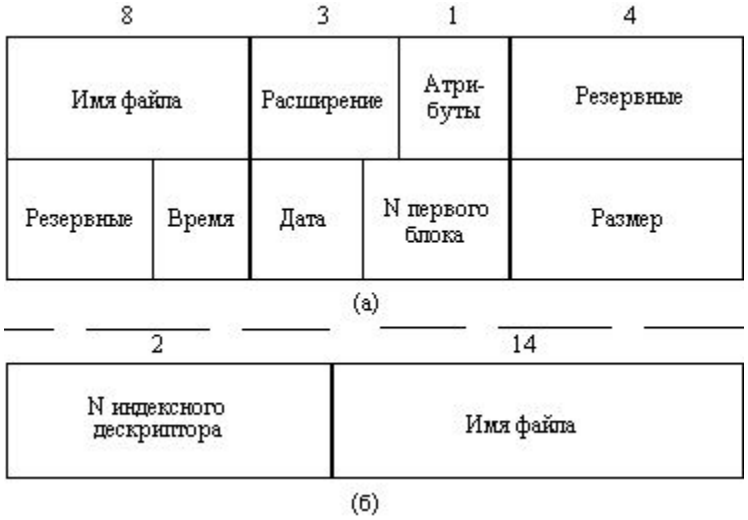
# Каталоги

- *Каталоги* – системные *файлы*, поддерживающие структуру файловой системы.
- Каталог это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр, или файлы, составляющие один программный пакет), а с другой стороны – это файл, содержащий системную информацию о группе файлов, его составляющих.
- В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

# Атрибуты файлов

- В разных файловых системах могут использоваться в качестве атрибутов разные характеристики:
  - информация о разрешенном доступе;
  - пароль для доступа к файлу;
  - владелец файла;
  - создатель файла;
  - флаги "только для чтения", "скрытый файл", "системный файл", "архивный файл", "двоичный/символьный", "временный" (удалить после завершения процесса), флаг блокировки;
  - времена создания, последнего доступа и последнего изменения;
  - текущий размер файла;
  - максимальный размер файла.

# Структура каталогов




*структура записи каталога FAT16 (32 байта)*

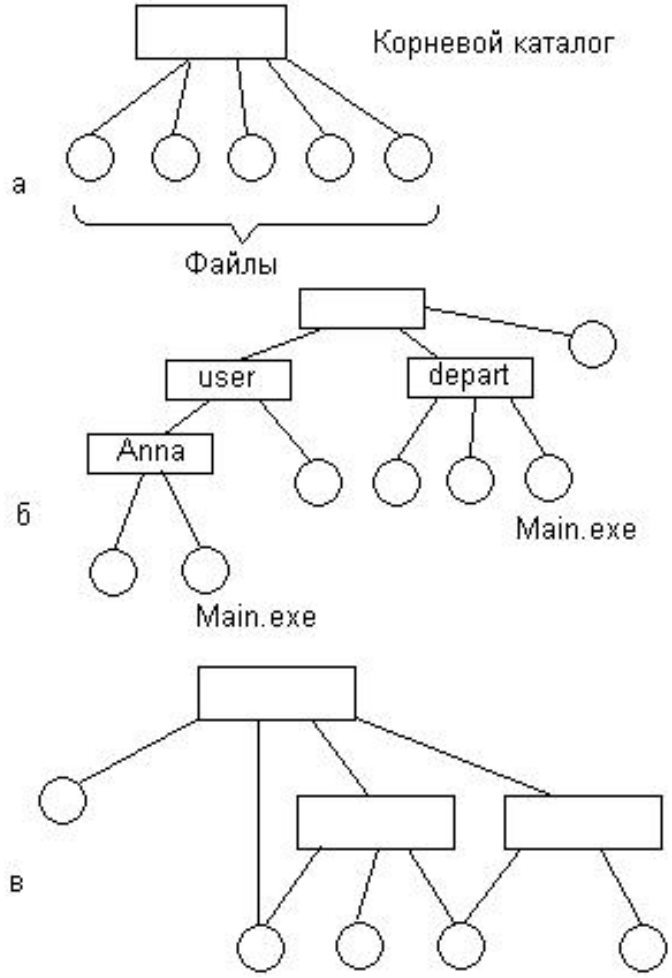
*структура записи каталога ОС UNIX*

Каталоги могут непосредственно содержать атрибуты файлов, как это сделано в файловой системе FAT16, или, как это реализовано в ОС UNIX, ссылаться на специальные индексные узлы (i-node), которые содержат атрибуты и информацию о размещении файлов.

# Логическая организация ФС

- Каталоги могут образовывать иерархическую структуру за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня.
- Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть - если файл может входить сразу в несколько каталогов.
- В MS-DOS каталоги образуют древовидную структуру, а в UNIX'е - сетевую. Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.

# Логическая организация ФС




- одноуровневая

- иерархическая (дерево)

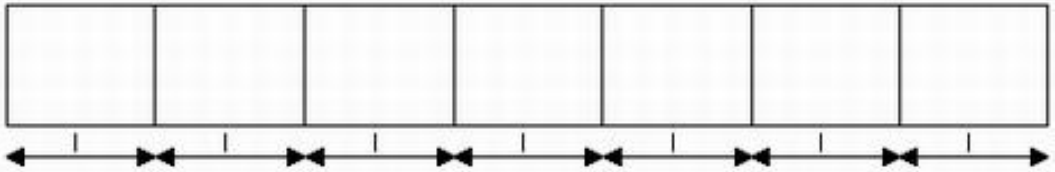
- иерархическая (сеть)

# Логическая организация файла

- Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. Логическая запись – это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи.
- На слайде показаны несколько схем логической организации файла. Записи могут быть фиксированной длины или переменной длины. Записи могут быть расположены в файле последовательно (последовательная организация) или в более сложном порядке, с использованием так называемых индексных таблиц, позволяющих обеспечить быстрый доступ к отдельной логической записи (индексно-последовательная организация). Для идентификации записи может быть использовано специальное поле записи, называемое ключом.
- В файловых системах ОС UNIX и MS Windows файл имеет простейшую логическую структуру – последовательность однобайтовых записей.

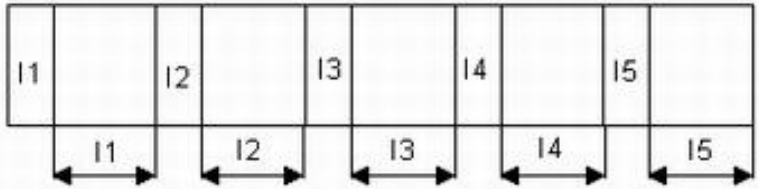
# Логическая организация файла

а



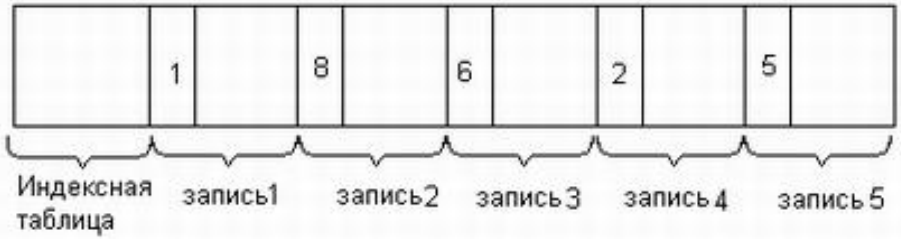
Последовательность логических записей фиксированной длины

б




Последовательность логических записей переменной длины

в



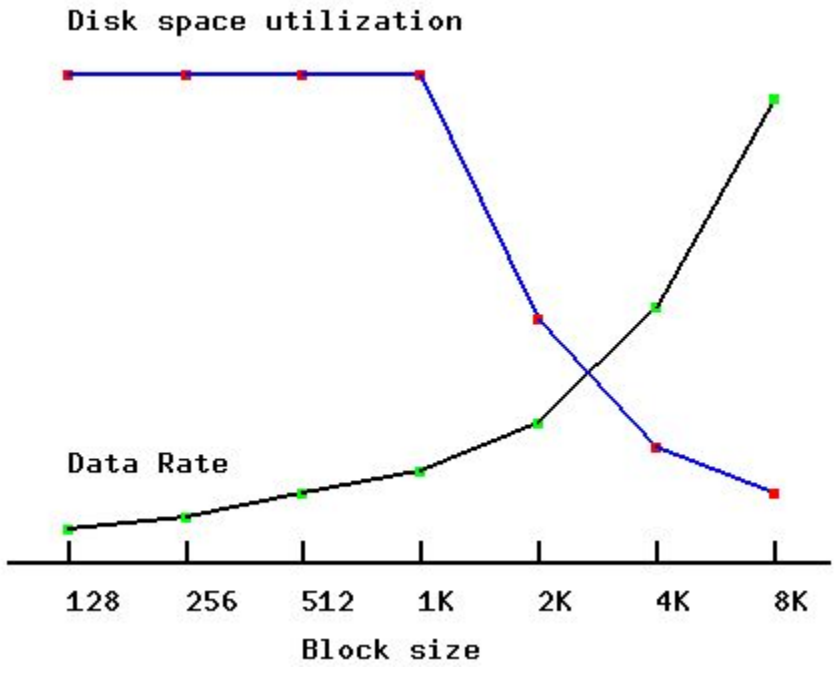

Индексная логическая организация

| Индекс | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Адрес | 21 | 201 | 315 | 661 | 670 | 715 |

Индекс ≡ ключ

# Физическая организация файла

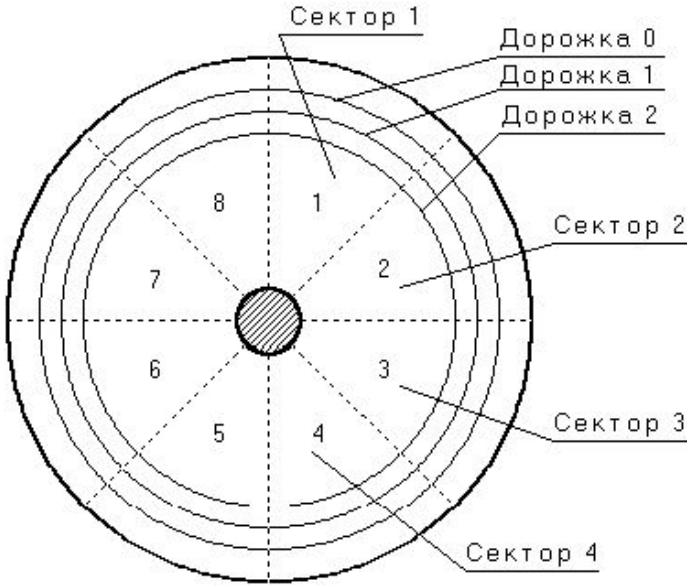
- Физическая организация файла описывает правила хранения файла на устройстве внешней памяти (например, магнитном диске).
- Файл состоит из физических записей – блоков.
- Блок – наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью.
- Примерами блоков являются сектора, кластеры и цилиндры.



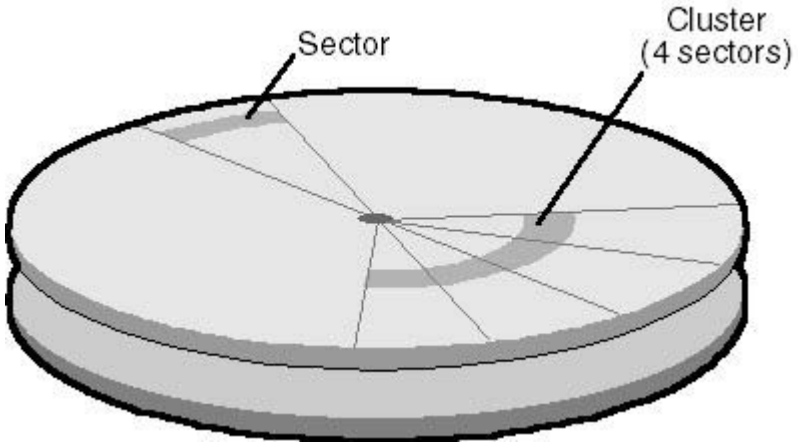
# Цилиндры и сектора

Магнитный диск представляет собой один или несколько объединенных дисков (блинов), по поверхностям (Sides) которых перемещаются головки (Heads).

Головки перемещаются по круговым дорожкам (Tracks), каждая дорожка разделена на **сектора** (Sectors). Дорожки, равноудаленные от центра диска и образующие цилиндрическую поверхность, называют **цилиндрами** (Cilinders).



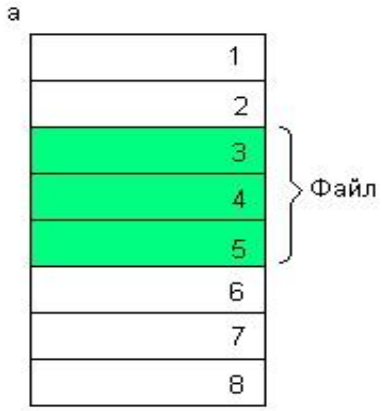
# Кластеры

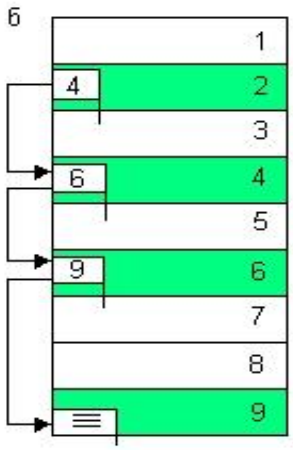



- Некоторые файловые системы (например, FAT и NTFS) в качестве единицы хранения информации используют логические блоки называемые кластерами.
- Кластер представляет группу смежных секторов. Число секторов в кластере всегда равно степени двойки ($2^n$).

# Способы физической организации

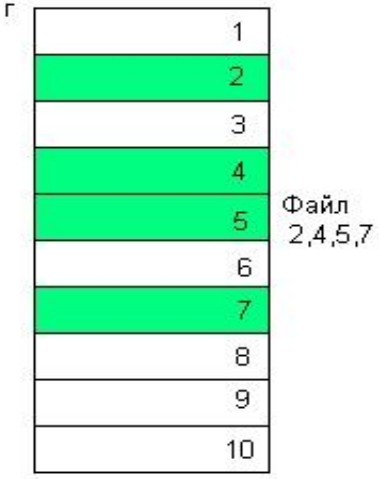
непрерывное размещение




связанный список блоков

связанный список индексов

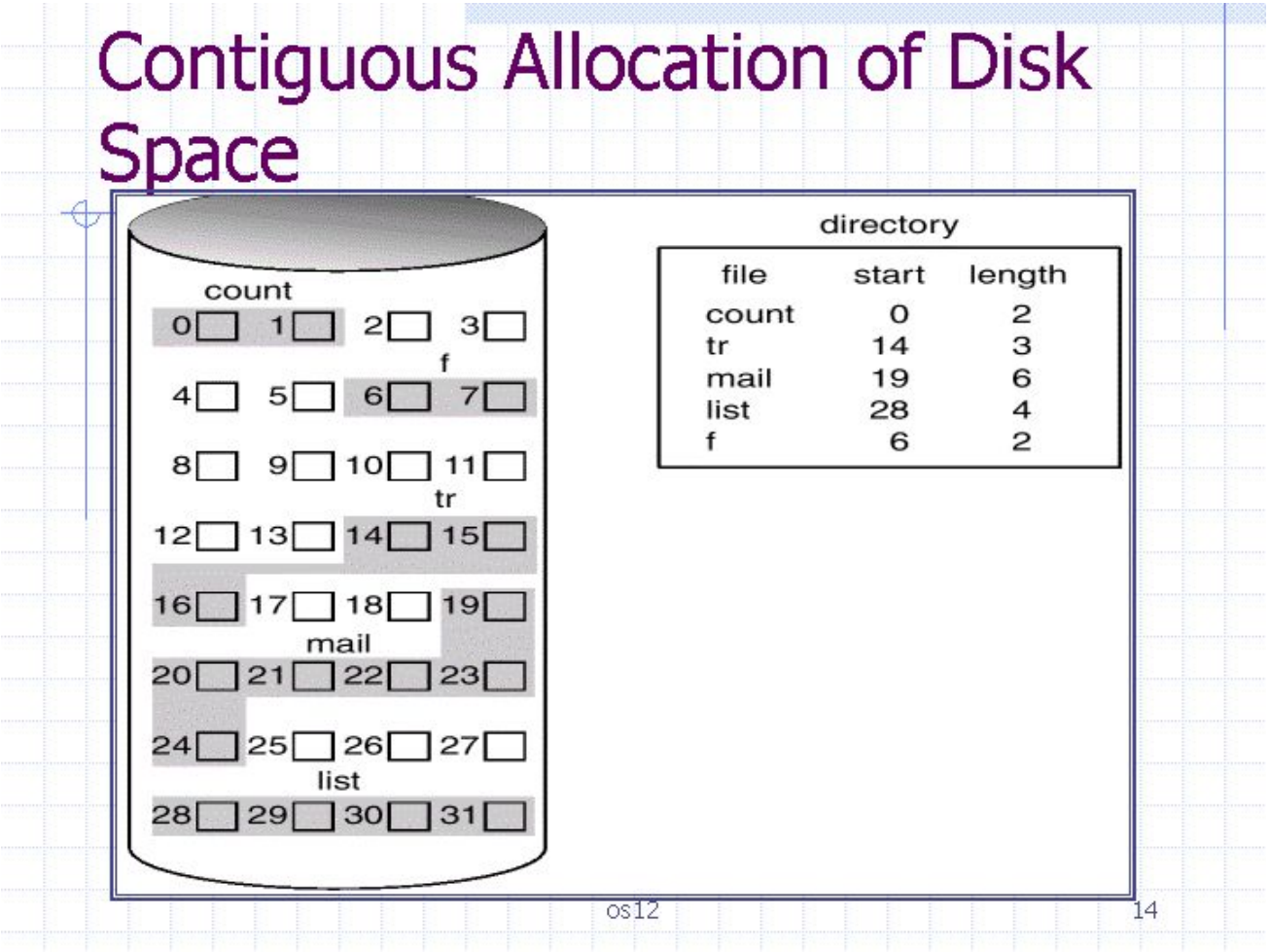




перечень номеров блоков

# Непрерывное размещение

- *Непрерывное размещение* – простейший вариант физической организации (рисунок а), при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти. Для задания адреса файла в этом случае достаточно указать только номер начального блока.
- В качестве примера обычно приводят файловую систему, реализованную в IBM OS/360.
- Достоинства этого метода:
  - отсутствие накладных расходов на хранение служебной информации;
  - высокая производительность при последовательном доступе;
  - простое вычисление адреса любого блока файла.
- Но имеются и два существенных недостатка:
  - Во-первых, во время создания файла заранее неизвестна его длина, а значит не известно, сколько памяти надо зарезервировать для этого файла. Поэтому при увеличении размера файла может потребоваться его перемещение.
  - Во-вторых, при таком порядке размещения неизбежно возникает фрагментация, и пространство на диске используется не эффективно, так как отдельные участки маленького размера (минимально 1 блок) могут остаться не используемыми.

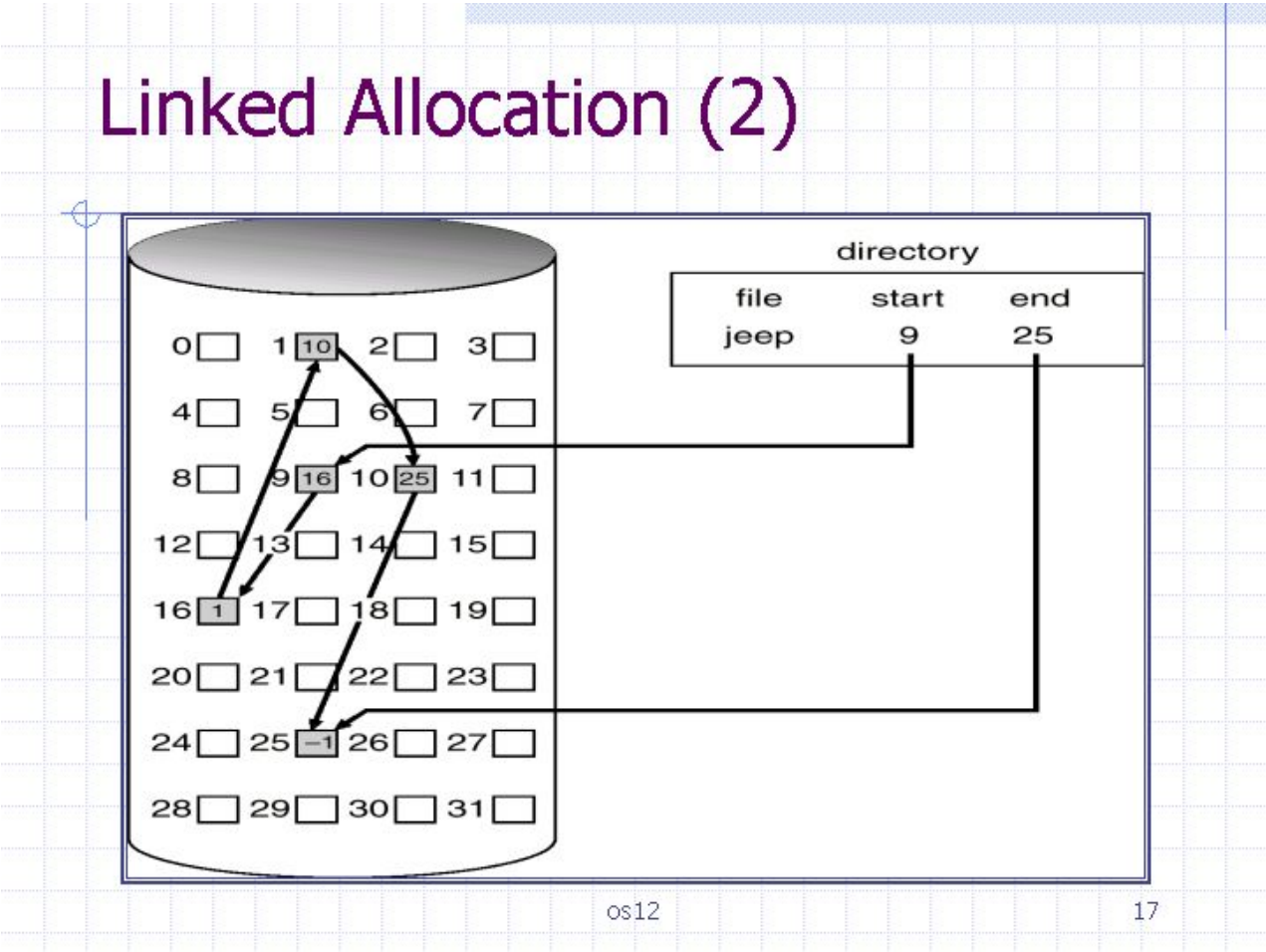
# Иллюстрация непрерывного размещения



# Связанный список блоков

- Следующий способ физической организации – *размещение в виде связанного списка блоков дисковой памяти* (рисунок б). При таком способе в начале каждого блока содержится указатель на следующий блок. В этом случае адрес файла также может быть задан одним числом – номером первого блока.
- В качестве примера обычно приводят операционную систему TOPS-10В качестве примера обычно приводят операционную систему TOPS-10 и компьютер Alto.
- Достоинством метода является отсутствие фрагментации. В отличии от предыдущего метода файл может безболезненно увеличивать число занимаемых блоков.
- Недостатками данного метода являются:
  - сложность реализации доступа к произвольно заданному месту файла: для того, чтобы прочитать *n*-ый по порядку блок, необходимо проследить список блоков прочитав *(n-1)* первых блока;
  - совместное хранение служебной информации и данных в одном блоке, при повреждении блока файла дальнейшая цепочка «обрывается»;
  - при этом способе количество данных файла, содержащихся в одном блоке, не равно степени двойки (одно слово израсходовано на номер следующего блока), а многие программы читают данные блоками, размер которых равен степени двойки.

# Иллюстрация связанного списка блоков



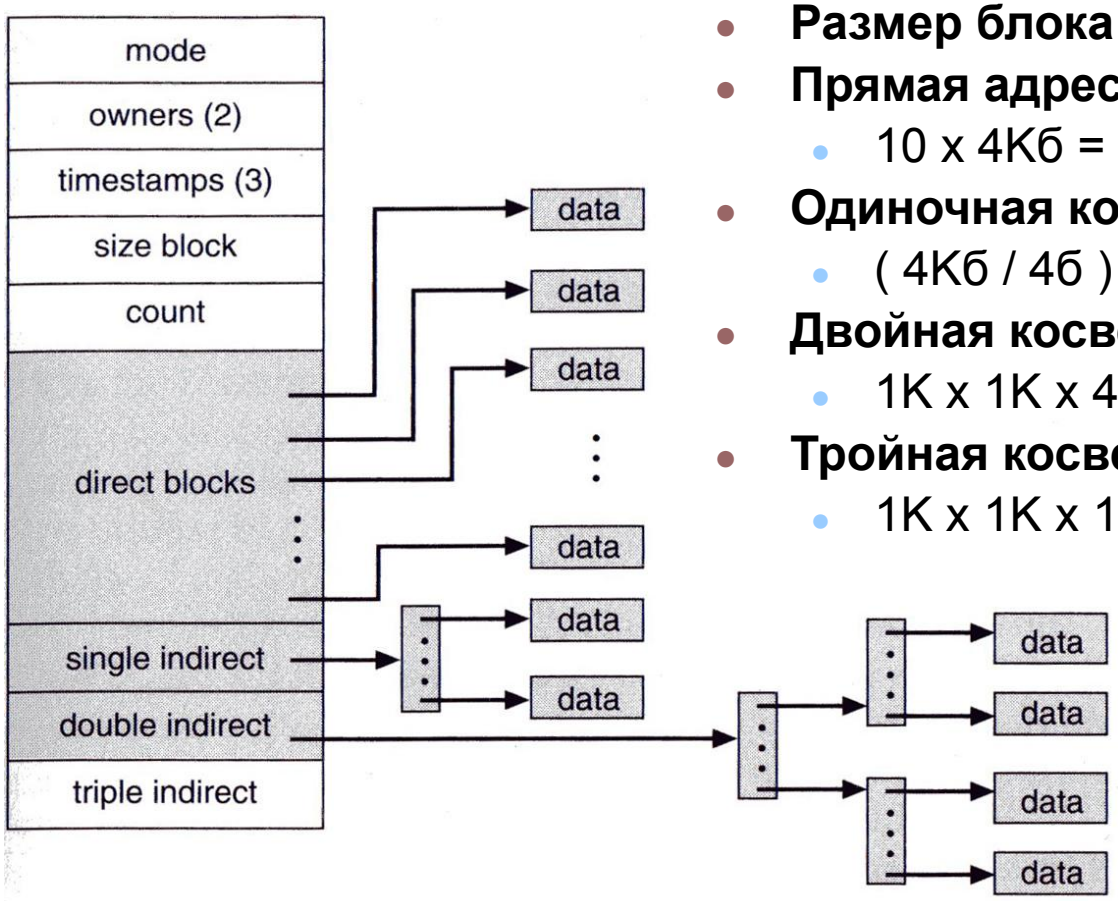
# Связанный список индексов

- Популярным способом, используемым, например, в файловой системе FAT операционной системы MS-DOS, является использование *связанного списка индексов*. С каждым блоком связывается некоторый элемент – индекс. Индексы располагаются в отдельной области диска (в MS-DOS это таблица FAT). Если некоторый блок распределен некоторому файлу, то индекс этого блока содержит номер следующего блока данного файла.
- При такой физической организации сохраняются все достоинства предыдущего способа, и снимаются оба отмеченных недостатка:
  - во-первых, для доступа к произвольному месту файла достаточно прочитать только блок индексов, отсчитать нужное количество блоков файла по цепочке и определить номер нужного блока;
  - во-вторых, данные файла занимают блок целиком, а значит имеют объем, равный степени двойки.

# Перечень номеров блоков

- В заключение рассмотрим задание физического расположения файла путем простого перечисления номеров блоков, занимаемых этим файлом. Этот способ используется в ФС UNIX, также похожий способ реализован в файловой системе NTFS.
- Например, в UNIX используется вариант данного способа, позволяющий обеспечить фиксированную длину адреса, независимо от размера файла. Для хранения адреса файла в индексном узле (i-node) выделено 13 полей:
  - Если размер файла меньше или равен 10 блокам, то номера этих блоков непосредственно перечислены в первых десяти полях адреса.
  - Если размер файла больше 10 блоков, то следующее 11-е поле содержит адрес блока, в котором могут быть расположены *n* номеров следующих блоков файла.
  - Если файл больше чем 10 + *n* блоков, то используется 12-е поле, в котором находится номер блока, содержащего *n* номеров блоков, которые содержат по *n* номеров блоков данного файла.
  - И, наконец, если файл больше 10+ *n* + *n* * *n*, то используется последнее 13-е поле для тройной косвенной адресации, что позволяет описать размещение файла, имеющего размер максимум 10 + *n* + *n* * *n* + *n* * *n* * *n* блоков.

# Иллюстрация использования индексного узла (i-node)

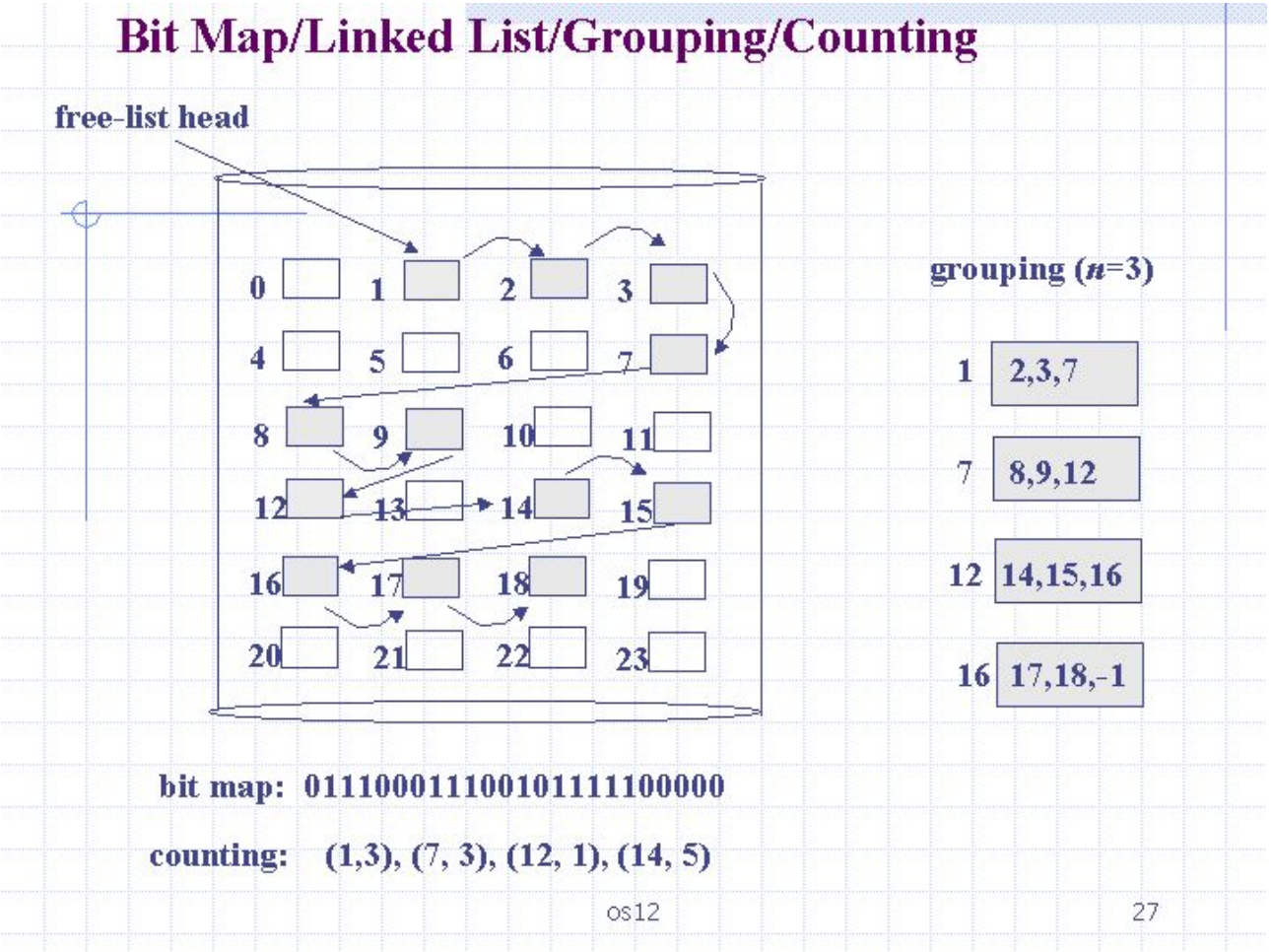



- **Размер блока – 4Кб, адрес – 4 байта**
- **Прямая адресация**:
  - 10 x 4Кб = 40Кб
- **Одиночная косвенная адресация**:
  - ( 4Кб / 4б ) x 4Кб = 1K x 4Кб = 4Мб
- **Двойная косвенная адресация** :
  - 1K x 1K x 4Кб = 4Гб
- **Тройная косвенная адресация**:
  - 1K x 1K x 1K x 4Кб = 4Тб

# Способы учета свободного дискового пространства

- связанный список (linked list) – в этом случае все свободные блоки связываются в общий список, указатель на первый свободный блок размещается в специально отведенном месте диска. Недостаток этого решения состоит в том, что для просмотра всего списка нужно выполнить много обращений к диску. Однако, обычно, необходим только первый свободный блок.
- битовая карта (bit map) – в этом случае список свободных блоков диска реализован в виде битового вектора. Каждый блок представлен одним битом, принимающим значение 0 или 1, в зависимости от того, занят он или свободен.
- групповой способ (grouping) – модификации подхода связанного списка, в первом свободном блоке хранятся адреса *n* свободных блоков. Первые *n-1* этих блоков действительно используются, а последний блок содержит адреса других *n* блоков и т. д.
- способ «счетчика» (counting) – каждая группа последовательных свободных блоков описывает парой чисел (номер стартового свободного блока; число последовательных свободных блоков).

# Иллюстрация учета свободного дискового пространства



# Права доступа к файлу

- Определить права доступа к файлу – значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа.
- Этот список может включать следующие операции:
  - создание файла
  - уничтожение файла
  - открытие файла
  - закрытие файла
  - чтение файла
  - запись в файл
  - дополнение файла
  - поиск в файле
  - получение атрибутов файла
  - установление новых значений атрибутов
  - переименование
  - выполнение файла
  - чтение каталога

# Управление правами доступа

- В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории: владельца файла, членов его группы и всех остальных.
- Различают два основных подхода к определению прав доступа:
  - избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
  - мандатный подход, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости от того, к какой группе пользователь отнесен.

# Матрица прав доступа

Строки соответствуют всем пользователям

Столбцы соответствуют всем файлам системы

*Имена файлов*

| *Имена пользователей* | modern.txt | win.exe | class.dbf | unix.ppt |
|---|---|---|---|---|
| kira | читать | выполнять | – | выполнять |
| genya | читать | выполнять | – | выполнять<br>читать |
| nataly | читать | – | – | выполнять<br>читать |
| victor | читать<br>писать | – | создать | – |

На пересечении строк и столбцов указываются разрешенные операции

# Кэширование диска

- Перехват запросов к внешним блочным ЗУ, промежуточным программным слоем – подсистемой буферизации (ПБ).
- ПБ представляет собой буферный пул, располагающийся в ОЗУ, и комплекс программ, управляющих этим пулом по принципу кэш-памяти.
- Размер каждого буфера пула равен размеру одного блока.
- Кэширование может выполняться как при выполнении операций чтения диска, так и при выполнения операций записи на диск.

# Кэширование диска – чтение

- При запросе на чтение некоторого блока подсистема буферизации (ПБ) просматривает свой буферный пул.
- Если требуемый блок находится, то ПБ копирует его в буфер запрашивающего процесса. Таким образом, операция В/В считается выполненной без физического обмена с устройством.
- При отсутствии свободного буфера на диск вытесняется наименее используемая информация.
- В результате очевиден выигрыш во времени доступа к файлу. Если же нужный блок в буферном пуле отсутствует, то он считывается с устройства и одновременно с передачей запрашивающему процессу копируется в один из буферов подсистемы буферизации.

# Кэширование диска – запись

- отложенная запись (lazy commit) – сразу запись производится только в буферный пул, синхронизация буферов пула и блоков диска производится в фоновом режиме, в итоге запись на диск выполняется почти незаметно для пользователя, однако в случае сбоя электропитания информация из буферного пула может быть потеряна;
- сквозная запись – запись на диск производится одновременно с записью в буферный пул, при этом снижается реактивность операционной системы на запросы пользователя, но повышает надежность.

# Общая модель файловой системы

- Функционирование любой файловой системы можно представить многоуровневой моделью, в которой каждый уровень предоставляет некоторый интерфейс (набор функций) вышележащему уровню, а сам, в свою очередь, для выполнения своей работы использует интерфейс (обращается с набором запросов) нижележащего уровня.

# Символьный и базовый уровень

- Задачей символьного уровня является определение по символьному имени файла его уникального имени. В файловых системах, в которых каждый файл может иметь только одно символьное имя (например, FAT), этот уровень отсутствует, так как символьное имя, присвоенное файлу пользователем, является уникальным. В других ФС, в которых один и тот же файл может иметь несколько символьных имен, на данном уровне просматривается цепочка каталогов для определения уникального имени файла. В файловой системе UNIX, например, уникальным именем является номер индексного дескриптора файла (i-node).
- На базовом уровне по уникальному имени файла определяются его характеристики: права доступа, адрес, размер и другие. Как уже было сказано, атрибуты файла могут входить в состав каталога или храниться в отдельных таблицах. При открытии файла его характеристики перемещаются с диска в оперативную память, чтобы уменьшить среднее время доступа к файлу. В некоторых ФС (например, HPFS) при открытии файла вместе с его атрибутами считываются несколько первых блоков файла, содержащих данные.

# Уровень проверки прав доступа и логический уровень

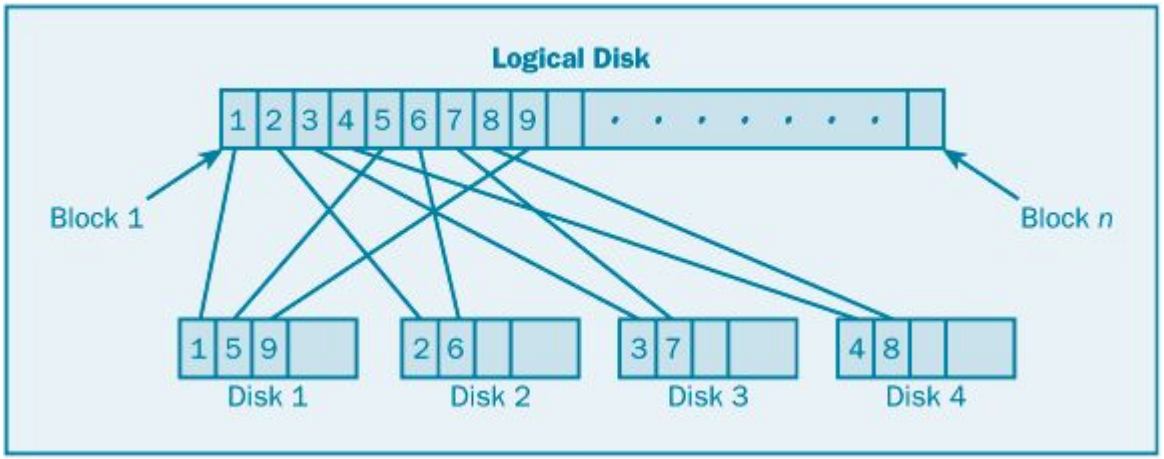
- Следующим этапом реализации запроса к файлу является проверка прав доступа к нему. Для этого сравниваются полномочия пользователя или процесса, выдавших запрос, со списком разрешенных видов доступа к данному файлу. Если запрашиваемый вид доступа разрешен, то выполнение запроса продолжается, если нет, то выдается сообщение о нарушении прав доступа.
- На логическом уровне определяются координаты запрашиваемой логической записи в файле, то есть требуется определить, на каком смещении от начала файла находится требуемая логическая запись. Алгоритм работы данного уровня зависит от логической организации файла. Например, если файл организован как последовательность логических записей фиксированной длины $l$, то n-ая логическая запись имеет смещение $l$ *(n-1) байт.

# Физический уровень

- На физическом уровне файловая система определяет номер физического блока, который содержит требуемую логическую запись, и смещение логической записи в физическом блоке. Для решения этой задачи используются результаты работы логического уровня - смещение логической записи в файле, адрес файла на внешнем устройстве, а также сведения о физической организации файла, включая размер блока.
- После определения номера физического блока, файловая система обращается к системе ввода-вывода для выполнения операции обмена с внешним устройством. В ответ на этот запрос в буфер файловой системы будет передан нужный блок, в котором на основании полученного при работе физического уровня смещения выбирается требуемая логическая запись.

# RAID-системы

- ***RAID*** – Redundant Array of Independent (Inexpensive) Disks – избыточный массив независимых (недорогих) дисков.
- ***RAID система*** – набор физических дисковых устройств, рассматриваемых ОС, как единое логическое дисковое устройство.



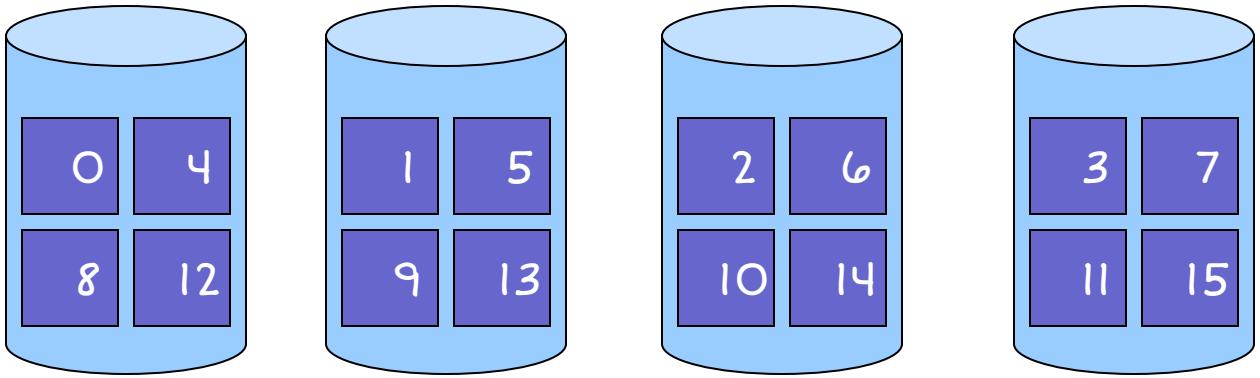

- Данные распределяются по физическим устройствам, образуется избыточная информация, используемая для контроля и восстановления информации.

# Основные уровни RAID систем

- **RAID 0** (stripping, дисковый массив без избыточности)
- **RAID 1** (mirroring, зеркалирование)
- **RAID 2 (**отказоустойчивый массив с использованием кода Хэмминга, исправляет одинарные ошибки «на лету» и выявляет двойные ошибки)
- **RAID 3** (отказоустойчивый массив с параллельной передачей данных и четностью)
- **RAID 4** (отказоустойчивый массив независимых дисков с разделяемым диском четности)
- **RAID 5** (отказоустойчивый массив независимых дисков с распределенной четностью)
- **RAID 6** (отказоустойчивый массив независимых дисков с двумя независимыми распределенными схемами четности: N+2 дисков)

# RAID-0

- Представляет собой дисковый массив, в котором данные разбиваются на блоки, и каждый блок записываются (или же считывается) на отдельный диск.
- Таким образом, можно осуществлять несколько операций ввода-вывода одновременно.

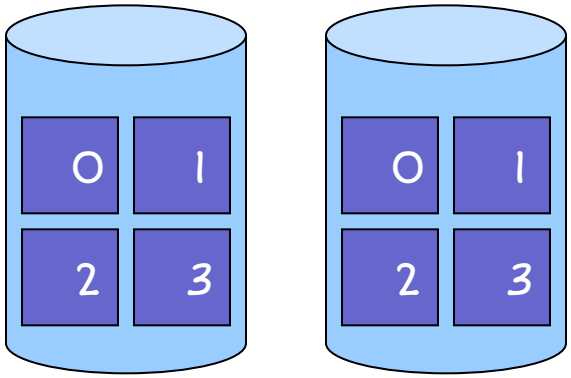


# RAID-0

- **Преимущества**:
  - наивысшая производительность для приложений требующих интенсивной обработки запросов ввода/вывода и данных большого объема;
  - простота реализации;
  - низкая стоимость на единицу объема.
- **Недостатки**:
  - не отказоустойчивое решение;
  - отказ одного диска влечет за собой потерю всех данных массива.

# RAID-1

- Зеркалирование - традиционный способ для повышения надежности дискового массива небольшого объема.
- В простейшем варианте используется два диска, на которые записывается одинаковая информация, и в случае отказа одного из них остается его дубль, который продолжает работать в прежнем режиме.



# RAID-1

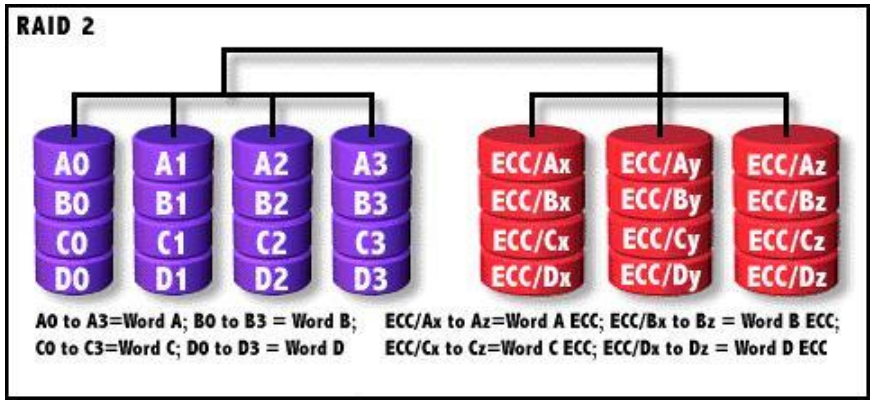
- **Преимущества**:
  - простота реализации;
  - простота восстановления массива в случае отказа (копирование);
  - достаточно высокое быстродействие для приложений с большой интенсивностью запросов.
- **Недостатки**:
  - высокая стоимость на единицу объема - 100% избыточность;
  - невысокая скорость передачи данных.

# Рекомендации по применению RAID-1

- Применяйте RAID 1 для диска, на котором содержится ваша ОС, потому что ее восстановление занимает очень много времени. RAID 1 хорошо подходит для этой задачи еще и потому, что ОС обычно занимает не более одного диска.
- Применяйте RAID 1 для журнала транзакций. Обычно журнал транзакций SQL Server может уместиться на одном диске. Кроме того, для журнала транзакций применяется в основном последовательная запись. Чтение из журнала транзакций производиться только из-за операций отката. Поэтому, если вы выделите для журнала транзакций отдельный том RAID 1, то вы достигните высокой производительности.
- Для томов RAID 1, как и для других RAID-систем, следует применять кэширование в режиме отложенной записи, это позволит повысить производительность, но обязательно защищайте вычислительную систему при помощи бесперебойного электропитания.

# RAID-2

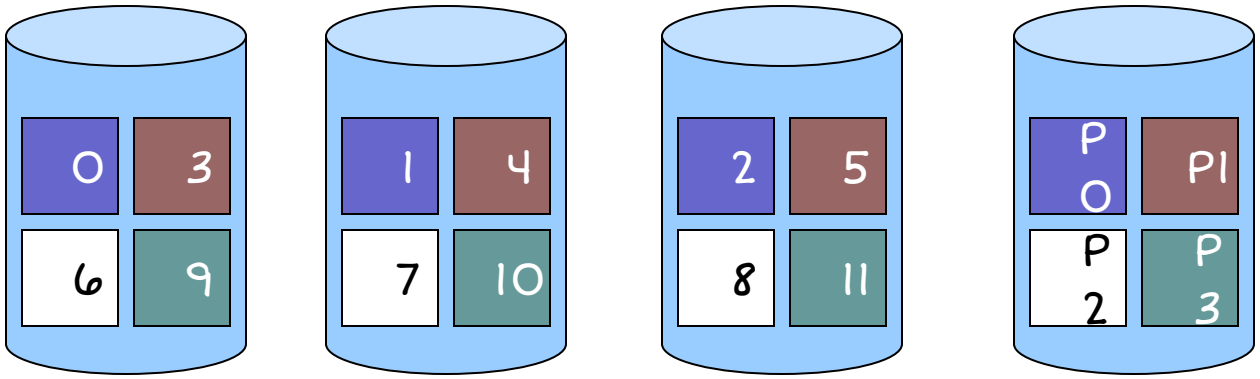
- **RAID-2** основан на разбиении входных данных на уровне битов и вычислении кода Хэмминга для контроля возможных ошибок (Hamming ECC - Error Correction Code).
- Исходные биты данных распределяются между дисками массива, а параллельно с ними на специальные выделенные диски (ECC диски) записываются вычисленные коды.




- Минимальное число p требуемых контрольных битов для d битов исходных данных определяется неравенством $d+p+1<=2p$. Несложно убедиться, что для d=4 (битов данных) требуется p=3 (контрольных битов), для d от 5 до 11 уже p=4.

# RAID-4

- Данные разбиваются на блочном уровне. Каждый блок данных записывается на отдельный диск и может быть прочитан отдельно. Четность для группы блоков генерируется при записи и проверяется при чтении.
- Главное отличие между RAID 3 и 4 состоит в том, что в последнем, расслоение данных выполняется на уровне секторов, а не на уровне битов или байтов.

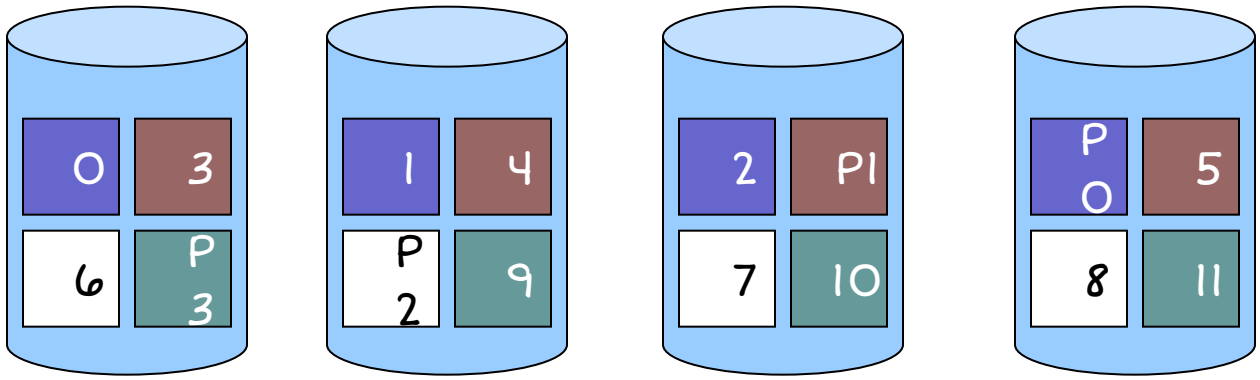


# RAID-4

- **Преимущества**:
  - очень высокая скорость чтения данных больших объемов;
  - высокая производительность при большой интенсивности запросов чтения данных;
  - малые накладные расходы для реализации избыточности.
- **Недостатки**:
  - достаточно сложная реализация;
  - очень низкая производительность при записи данных;
  - сложное восстановление данных;
  - низкая скорость чтения данных малого объема при единичных запросах;
  - асимметричность быстродействия относительно чтения и записи.

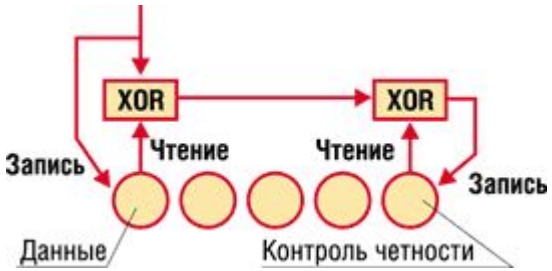
# RAID-5

- Этот уровень похож на RAID 4, но в отличие от предыдущего четность распределяется циклически по всем дискам массива.



# RAID-5

- **Преимущества**:
  - высокая производительность при большой интенсивности запросов чтения/записи данных;
  - малые накладные расходы для реализации избыточности.
- **Недостатки**:
  - скорость записи данных ниже, чем в RAID 1 – в RAID-5 каждая операция записи требует 2-х чтений старых значений, выполнения 2-х операций XOR и 2-х новых записей;




  - скорость чтения данных ниже, чем в RAID 4;
  - достаточно сложная реализация (обычно аппаратная);
  - при выходе из строя одного диска производительность системы резко падает;
  - сложное восстановление данных.

# Сравнение RAID-систем

| RAID | Минимум дисков | Потребность в дисках | Отказо-устойчивость | Скорость передачи данных | Интенсивность обработки запросов | Практическое использование |
|---|---|---|---|---|---|---|
| 0 | 2 | N | < 1 диск | < RAID 3 | очень высокая до N x 1 диск | Графика, видео |
| 1 | 2 | 2N | < RAID 6 | R > 1 диск<br>W = 1 диск | до 2 x 1 диск<br>W = 1 диск | малые файл-серверы |
| 2 | 7 | 2N >X>N+1 | < RAID 1 | ~ RAID 3 | Низкая | мейнфреймы |
| 3 | 3 | N+1 | < RAID 1 | низкая | Низкая | Графика, видео |
| 4 | 3 | N+1 | < RAID 1 | R < RAID 3<br>W < RAID 5 | R = RAID 0<br>W << 1 диск | файл-серверы |
| 5 | 3 | N+1 | < RAID 1 | R < RAID 4<br>W < RAID 3 | R = RAID 0<br>W < 1 диск | серверы баз данных (обработка транзакций) |
| 6 | 4 | N+2 | самая высокая | низкая | R > 1 диск<br>W < RAID 4 | используется крайне редко |

# Составные RAID системы

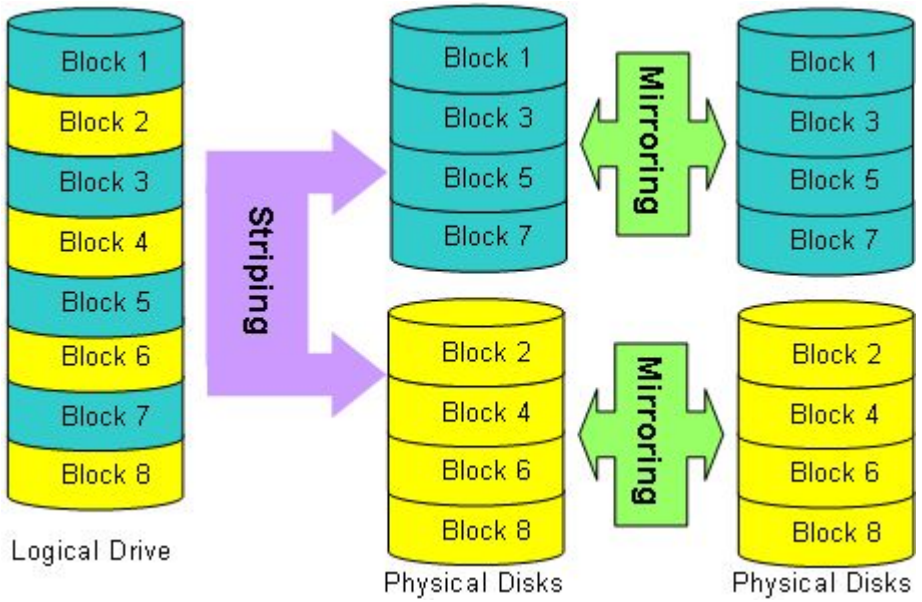
- **RAID 0+1 / RAID 1+0**
- **RAID 0+3 / RAID 3+0**
- **RAID 0+5 / RAID 5+0**
- **RAID 1+5 / RAID 5+1**
- **…**

# Сравнение RAID 0+1 и RAID 1+0

- В **RAID 0+1** формируется 2 идентичных массива RAID-0 (только striping), запись на которые ведется зеркально (mirroring), т.е. согласно RAID-1. Для примера, если у нас есть 8 идентичных дисков, то мы получаем 2 массива по 4 диска в каждом. Данные в каждом массиве пишутся параллельно сразу на 4 диска (без защиты данных), но между массивами данные полностью дублируются.
- В варианте **RAID 1+0** наоборот формируется 4 массива по 2 диска в каждом. Теперь в каждом массиве осуществляется зеркальное дублирование информации, но входной поток данных распараллеливается между 4 массивами.
- В итоге по надежности RAID 1+0 несколько лучше. Так, массив из 8 дисков (4 по 2) может остаться работоспособным при отказе до 4 жестких дисков!

# Рекомендации по применению RAID 1+0

- Уровень RAID 10 является наилучшим отказоустойчивым решением, он обеспечивает хорошую защиту данных и высокую производительность, однако затраты на него тоже большие (50% суммарной емкости всех накопителей).



# Реализация RAID-систем

- программная (software-based);
- аппаратная - шинно-ориентированная (bus-based);
- аппаратная - автономная подсистема (subsystem-based).

# Программная реализация RAID

- Главное преимущество программной реализации - низкая стоимость.
- Но при этом у нее много недостатков:
  - низкая производительность,
  - загрузка дополнительной работой центрального процессора,
  - увеличение шинного трафика.
- Программно обычно реализуют простые уровни RAID - 0 и 1, так как они не требуют значительных вычислений.
- Учитывая эти особенности, RAID системы с программной реализацией используются в серверах начального уровня.

# Программная реализация RAID

- Ядро GNU/Linux 2.6.28 (последнее из вышедших в 2008 году) поддерживает программные RAID следующих уровней: 0, 1, 4, 5, 6, 10.  Загрузка поддерживается только с диска RAID 1.
- Файловая система *ZFS* поддерживает уровни RAID: 0, 1, 5, 6, а также составные уровни.
- Серверные версии MS Windows 2000 и старше поддерживают программный RAID 0, RAID 1 и RAID 5.

# Аппаратная реализация RAID