**Текст лекции №6. Тема: «Организация файловой системы».**

**Вопрос 1. Логическая организация файловой системы**

*Файл* – это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные, а также собственно хранимые в этой области данные и набор атрибутов, позволяющих ОС манипулировать этими данными.

Долговременное и надежное хранение информации. Долговременность достигается за счет использования запоминающих устройств, не зависящих от питания, а высокая надежность определяется средствами защиты доступа к файлам и общей организацией программного кода ОС, при которой сбои аппаратуры чаще всего не разрушают информацию, хранящуюся в файлах.

**Совместное использование информации**. Файлы обеспечивают естественный и простой способ разделения информации между приложениями и пользователями за счет наличия понятного человеку символьного имени и постоянства хранимой информации и расположения файла. Пользователь должен иметь удобные средства работы с файлами, включая, каталоги и справочники, объединяющие файлы в группы, средства поиска файлов по признакам, набор команд для создания, модификации и удаления файлов. Файл может быть создан одним пользователем, а применяться совсем другим, при этом создатель файла или администратор могут определить права доступа к нему других пользователей. Эти цели реализуются в ОС файловой системой.

Файловая *система* (ФС) – это часть операционной системы, включающая:

* 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 различные операции с файлами, такие как создание, уничтожение, чтение, запись, именование и поиск файлов.

Файловая система позволяет программам обходиться набором достаточно простых операций для выполнения действий над некоторым абстрактным объектом, представляющим файл. При этом программистам не нужно иметь дело с деталями действительного расположения данных на диске, буферизацией данных и другими низкоуровневыми проблемами передачи данных с долговременного запоминающего устройства. Все эти функции ФС берет на себя.

Файловая система распределяет дисковую память, поддерживает именование файлов, отображает имена файлов на соответствующие адреса во внешней памяти, обеспечивает доступ к данным, поддерживает разделение, защиту и восстановление файлов.

Таким образом, файловая система играет роль промежуточного слоя, экранирующего все сложности физической организации долговременного хранилища данных и создающего для программ более простую логическую модель этого хранилища, а также предоставляя им набор удобных в использовании команд для манипулирования файлами.

Задачи, решаемые ФС, зависят от способа организации вычислительного процесса в целом.

Самый простой тип – это ФС в однопользовательских и однопрограммных ОС, к числу которых относится, например, *MS-DOS*.

Основные функции однопользовательских однопрограммных ФС нацелены на решение следующих задач:

* именование файлов;
* программный интерфейс для приложений;
* отображения логической модели файловой системы на физическую организацию хранилища данных;
* устойчивость файловой системы к сбоям питания, ошибкам аппаратных и программных средств.

Задачи ФС усложняются в операционных однопользовательских мультипрограммных ОС, которые, хотя и предназначены для работы одного пользователя, но дают ему возможность запускать одновременно несколько процессов. Одной из первых ОС этого типа стала *OS/2*. К перечисленным задачам добавляется новая задача совместного доступа к файлу из нескольких процессов. Файл в этом случае является разделяемым ресурсом, а значит, ФС должна решать весь комплекс проблем, связанных с такими ресурсами.

В частности, в однопользовательских мультипрограммных ФС должны быть предусмотрены следующие средства:

* блокировки файла и его частей;
* предотвращения гонок;
* исключения тупиков;
* согласования копий и т. п.

В многопользовательских системах ФС появляется еще одна задача – защита файлов одного пользователя от несанкционированного доступа другого пользователя.

Еще более сложными становятся функции ФС, которая работает в составе сетевой ОС.

Файловые системы поддерживают несколько функционально различных типов файлов, в число которых, как правило, входят:

* обычные файлы;
* файлы-каталоги;
* специальные файлы;
* именованные конвейеры;
* файлы, отображаемые на память.

Обычные файлы, или просто файлы, содержат информацию произвольного характера, которую заносит в них пользователь или которая образуется в результате работы системных и пользовательских программ. Большинство современных операционных систем никак не ограничивают и не контролируют содержимое и структуру обычного файла.

Содержимое обычного файла определяется приложением, которое с ним работает.

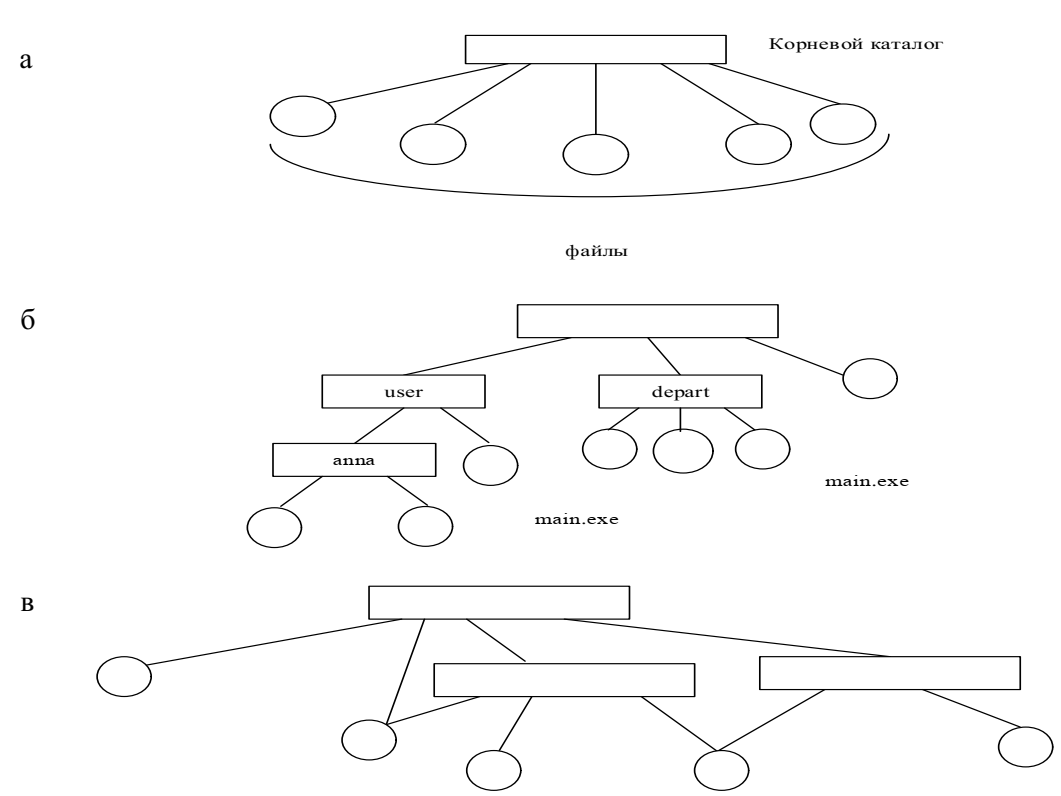
*Каталоги* – это особый тип файлов, которые содержат системную справочную информацию о наборе файлов, сгруппированных пользователями по какому-либо неформальному признаку. Во многих операционных системах в каталог могут входить файлы любых типов, в том числе другие каталоги, за счет чего образуется древовидная структура, удобная для поиска. Каталоги устанавливают соответствие между именами файлов и их характеристиками, используемыми файловой системой для управления файлами. В число таких характеристик входит, в частности, информация (или указатель на другую структуру, содержащую эти данные) о типе файла и расположении его на диске, правах доступа к файлу, датах его создания и модификации. Во всех остальных отношениях каталоги рассматриваются файловой системой как обычные файлы.

*Специальные файлы* – это фиктивные файлы, ассоциированные с устройствами ввода-вывода, которые используются для унификации механизма доступа к файлам и внешним устройствам. Специальные файлы позволяют пользователю выполнять операции ввода-вывода посредством обычных команд записи в файл или чтения из файла. Эти команды обрабатываются сначала программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются операционной системой в команды управления соответствующим устройством.

Современные файловые системы поддерживают и другие типы файлов, такие как символьные связи, именованные конвейеры, отображаемые на память файлы. Они будут рассмотрены позже.

**Иерархическая структура файловой системы**. Большинство файловых систем имеет иерархическую структуру, в которой уровни создаются за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня, при этом частным случаем иерархической структуры является одноуровневая организация, когда все файлы входят в один каталог (рис. 1 а).

Граф, описывающий иерархию 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.



*Рис. 1. Иерархия файловых систем.*

Например, в *MS-DOS* и *Windows* каталоги образуют древовидную структуру, а в *Unix* – сетевую. В древовидной структуре каждый файл является листом. Каталог самого верхнего уровня называется корневым каталогом, или корнем (*root*). При такой организации пользователь освобожден от необходимости помнить имена всех файлов, ему достаточно примерно представлять, к какой группе может быть отнесен тот или иной файл, чтобы путем последовательного просмотра каталогов найти его. Иерархическая структура удобна для многопользовательской работы: каждый пользователь со своими файлами локализуется в своем каталоге или поддереве каталогов, и вместе с тем все файлы в системе логически связаны.

**Имена файлов**. Все типы файлов имеют символьные имена.

В иерархически организованных файловых системах обычно используются следующие типы имен файлов:

* + простое (короткое) символьное имя;
  + полное (составное) символьное имя;
  + относительное символьное имя;
  + уникальное имя (числовой идентификатор).

*Простое*, или короткое, символьное имя идентифицирует файл в пределах одного каталога. Простые имена присваивают файлам пользователи и программисты, при этом они должны учитывать ограничения ОС как на номенклатуру символов, так и на длину имени. До сравнительно недавнего времени эти границы были весьма узкими. Так, в известной файловой системе *FAT MS-DOS* длина имен ограничивалась схемой 8.3 (8 символов – собственно имя, 3 символа – расширение имени), а в файловой системе s5, поддерживаемой многими версиями ОС *Unix*, простое символьное имя не могло содержать более 14 символов.

Однако пользователю гораздо удобнее работать с длинными именами, поскольку они позволяют давать файлам легко запоминающиеся названия, ясно говорящие о том, что содержится в том или ином файле. Поэтому современные файловые системы, а также усовершенствованные варианты уже существовавших файловых систем, как правило, поддерживают длинные простые символьные имена файлов. Например, в файловых системах *NTFS* и *FAT32*, входящих в состав ОС семейства *Windows*, имя файла может содержать до 255 символов.

Примеры простых имен файлов и каталогов:

* + *quest\_ul.doc;*
* *task-entran.exe;*
* *приложение к CD 254L на русском языке-doc;*
* *installable filesystem manager.doc*.

В иерархических файловых системах разным файлам разрешено иметь одинаковые простые символьные имена при условии, что они принадлежат разным каталогам. То есть здесь работает схема много файлов – одно простое имя. Для однозначной идентификации файла в таких системах используется так называемое полное имя.

*Полное* имя представляет собой цепочку простых символьных имен всех каталогов, через которые проходит путь от корня до данного файла.

Таким образом, полное имя является составным именем, в котором простые имена отделены друг от друга принятым в ОС разделителем. Часто в качестве разделителя используется прямой или обратный слэш, при этом принято не указывать имя корневого каталога. На рис. 1 б два файла имеют простое имя *Main.exe*, однако их составные имена /*depart/Maiп.ехе и /user/Anna/Main.exe* различаются.

В древовидной файловой системе между файлом и его полным именем имеется взаимно однозначное соответствие один файл – одно полное имя. В файловых системах имеющих сетевую структуру, файл может входить в несколько каталогов, а значит, иметь несколько полных имен, здесь справедливо соответствие один файл – много полных имен. В обоих случаях файл однозначно идентифицируется полным именем. Файл может быть идентифицирован также относительным именем.

*Относительное* имя файла определяется через понятие «текущий каталог». Для каждого пользователя в каждый момент времени один из каталогов файловой системы является текущим, причем этот каталог выбирается самим пользователем по команде ОС. Файловая система фиксирует имя текущего каталога, чтобы затем использовать его как дополнение к относительным именам для образования полного имени файла. При применении относительных имен пользователь идентифицирует файл цепочкой имен каталогов, через которые проходят маршрут от текущего каталога до данного файла. Например, если текущим каталогом является каталог/*user*, то относительное имя файла /*user/Anna/Main.ехе* выглядит следующим образом: *Anna/Main.exe*.

В некоторых операционных системах разрешено присваивать одному и тому же файлу несколько простых имен, которые можно интерпретировать как псевдонимы. В этом случае, так же как в системе с сетевой структурой, устанавливается соответствие один файл – много полных имен, так как каждому простому имени файла соответствует, по крайней мере, одно полное имя.

*Символьные* имена удобны для пользователя, но не для операционной системы. Для своих внутренних целей ОС присваивает файлу уникальное имя, так что справедливо соотношение один файл – одно уникальное имя. Уникальное имя существует наряду с одним или несколькими символьными именами, присваиваемыми файлу пользователями или приложениями. Уникальное имя представляет собой числовой идентификатор и предназначено только для использования операционной системой. Примером такого уникального имени файла является номер индексного дескриптора в системе *Unix*.

**Монтирование**. В общем случае вычислительная система может иметь несколько дисковых устройств. Даже типичный персональный компьютер обычно имеет один накопитель на жестком диске, один накопитель на гибких дисках и накопитель для компакт-дисков. Мощные же компьютеры, как правило, оснащены большим количеством дисковых накопителей, на которые устанавливаются пакеты дисков. Более того, даже одно физическое устройство с помощью средств операционной системы может быть представлено в виде нескольких логических устройств, в частности, путем разбиения дискового пространства на разделы.

Возникает вопрос, каким образом организовать хранение файлов в системе, имеющей несколько устройств внешней памяти? Первое решение состоит в том, что на каждом из устройств размещается автономная файловая система, то есть файлы, находящиеся на этом устройстве; описываются деревом каталогов, никак не связанным с деревьями каталогов на других устройствах. В таком случае для однозначной идентификации файла пользователь наряду с составным символьным именем файла должен указывать идентификатор логического устройства. Примером такого автономного существования файловых систем является операционная система *MS-DOS*, в которой полное имя файла включает буквенный идентификатор логического диска. Так, при обращении к файлу, расположенному на диске *А*, пользователь должен указать имя этого диска: *A:\privat\letter\uni\letl.doc1*.

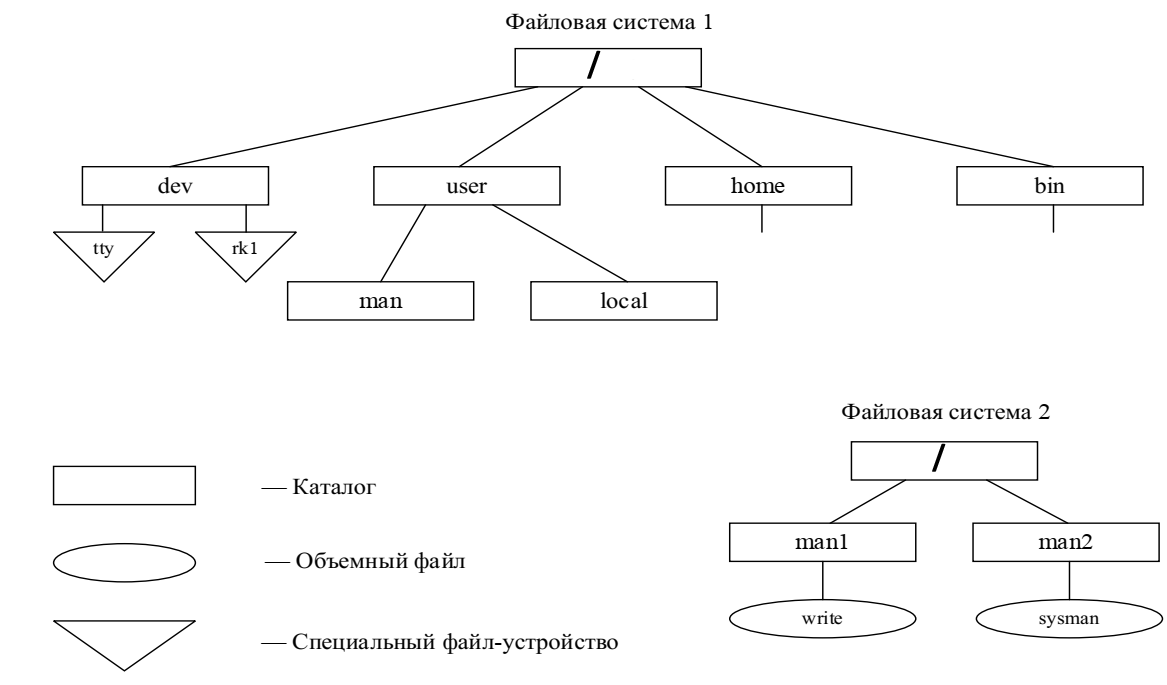
Другим вариантом является такая организация хранения файлов, при которой пользователю предоставляется возможность объединять файловые системы, находящиеся на разных устройствах, в единую файловую систему, описываемую единым деревом каталогов. Такая операция называется монтированием.

Рассмотрим, как осуществляется эта операция на примере ОС *Unix*.

Среди всех имеющихся в системе логических дисковых устройств операционная система выделяет одно устройство, называемое системным. Пусть имеются две файловые системы, расположенные на разных логических дисках, причем один из дисков является системным (рис. 2).

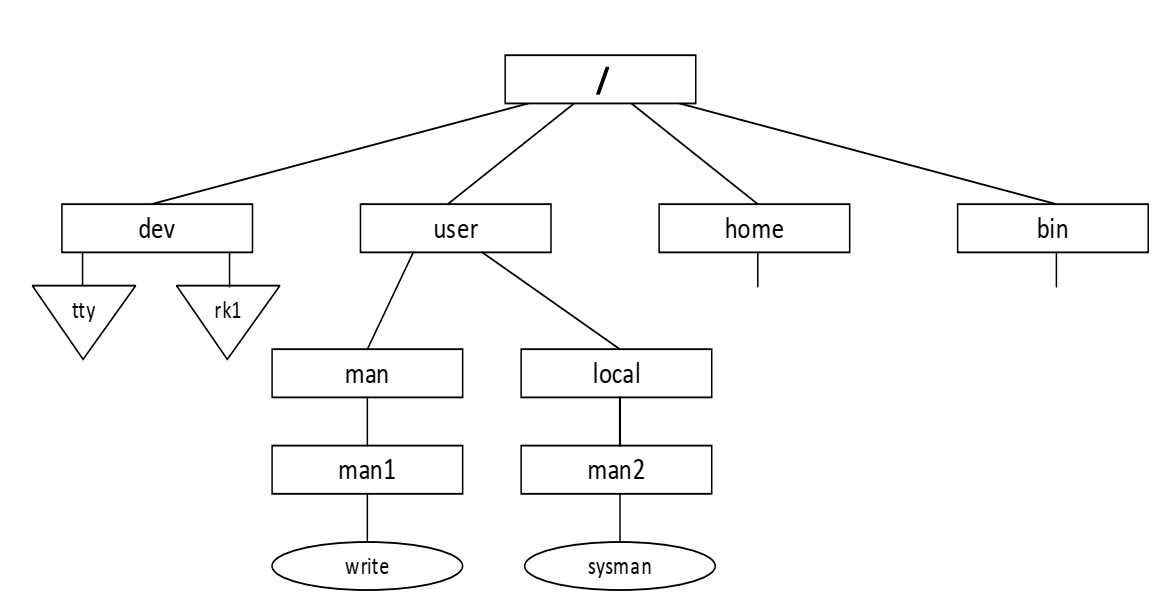
Знак «/» обозначает корень файловой системы. Это главный каталог в системе Linux. По сути, это и есть файловая система Linux. Здесь нет дисков или чего-то подобного, как в Windows. Вместо этого, адреса всех файлов начинаются с корня, а дополнительные разделы, флешки или оптические диски подключаются в папки корневого каталога.

Только пользователь root имеет право читать и изменять файлы в этом каталоге.



*Рис. 2. Файловая система до монтирования.*

Файловая система, расположенная на системном диске, назначается корневой. Для связи иерархий файлов в корневой файловой системе выбирается некоторый существующий каталог, в данном примере – каталог *man*. После выполнения монтирования выбранный каталог man становится корневым каталогом второй файловой системы. Через этот каталог монтируемая файловая система подсоединяется как поддерево к общему дереву (рис. 3).



*Рис. 3. Общая файловая система после монтирования.*

После монтирования общей файловой системы для пользователя нет логической разницы между корневой и смонтированной файловыми системами, в частности, именование файлов производится так же, как если бы система с самого начала была единой.

**Атрибуты файлов**. Понятие «файл» включает не только хранимые им данные и имя, но и атрибуты. Атрибуты – это информация, описывающая свойства файла. Примеры возможных атрибутов файла:

* тип файла (обычный файл, каталог, специальный файл и т. п.);
* владелец файла;
* создатель файла;
* пароль для доступа к файлу;
* информация о разрешенных операциях доступа к файлу;
* времена создания, последнего доступа и последнего изменения;
* текущий размер файла;
* максимальный размер файла;
* признак «только для чтения»;
* признак «скрытый файл»;
* признак «системный файл»;
* признак «архивный файл»;
* признак «двоичный/символьный»;
* признак «временный» (удалить после завершения процесса);
* признак блокировки;
* длина записи в файле;
* указатель на ключевое поле в записи;
* длина ключа.

Набор атрибутов файла определяется спецификой файловой системы: в файловых системах разного типа для характеристики файлов могут использоваться разные наборы атрибутов.

В однопользовательской ОС в наборе атрибутов будут отсутствовать атрибуты, имеющие отношение к пользователям и защите, такие как владелец файла, создатель файла, пароль для доступа к файлу, информация о разрешенном доступе к файлу.

Пользователь может получать доступ к атрибутам с помощью средств, предоставленных для этих целей файловой системой. Обычно разрешается читать значения любых атрибутов, а изменять – только некоторые.

Например, пользователь может изменить права доступа к файлу (при условии, что он обладает необходимыми для этого полномочиями), но изменять дату создания или текущий размер файла ему не разрешается.

Значения атрибутов файлов могут непосредственно содержаться в каталогах, как это было сделано в файловой системе *MS-DOS*.

На рис. 4а представлена структура записи в каталоге, содержащая простое символьное имя и атрибуты файла. Здесь буквами обозначены признаки файла: *R* – только для чтения, *А* – архивный, *Н* – скрытый, *S* – системный.



*Рис. 4. Структура каталогов:*

*а – структура записи каталога MS-DOS;*

*б – структура записи каталога ОС Unix.*

Другим вариантом является размещение атрибутов в специальных таблицах, когда в каталогах содержатся только ссылки на эти таблицы.

Такой подход реализован, например, в файловой системе *ufs* ОС *Unix*. В этой файловой системе структура каталога очень простая. Запись о каждом файле содержит короткое символьное имя файла и указатель на индексный дескриптор файла – так называется в *ufs* таблица, в которой сосредоточены значения атрибутов файла (рис. 2.4 б).

В том и другом вариантах каталоги обеспечивают связь между именами файлов и собственно файлами. Однако подход, когда имя файла отделено от его атрибутов, делает систему более гибкой. Например, файл может быть легко включен сразу в несколько каталогов. Записи об этом файле в разных каталогах могут содержать разные простые имена, но в поле ссылки будет указан один и тотже номер индексного дескриптора.

**Логическая организация файла**. В общем случае данные, содержащиеся в файле, имеют некую логическую структуру. Эта структура является базой при разработке программы, предназначенной для обработки этих данных. Признаками, отделяющими один структурный элемент от другого, могут служить определенные кодовые последовательности или просто известные программе значения смещений этих структурных элементов относительно начала файла. Поддержание структуры данных может быть либо целиком возложено на приложение, либо в той или иной степени эту работу может взять на себя файловая система.

В первом случае, когда все действия, связанные со структуризацией и интерпретацией содержимого файла, целиком относятся к ведению приложения, файл представляется ФС неструктурированной последовательностью данных. Приложение формулирует запросы к файловой системе на ввод-вывод, используя общие для всех приложений системные средства, например, указывая смещение от начала файла и количество байтов, которые необходимо считать или записать. Поступивший к приложению поток байтов интерпретируется в соответствии с заложенной в программе логикой. Например, компилятор генерирует, а редактор связей воспринимает вполне определенный формат объектного модуля программы. При этом формат файла, в котором хранится объектный модуль, известен только этим программам. Подчеркнем, что интерпретация данных никак не связана с действительным способом их хранения в файловой системе.

Модель файла, в соответствии с которой содержимое файла представляется неструктурированной последовательностью (потоком) байтов, стала популярной вместе с ОС *Unix*, а теперь она широко используется и в большинстве современных ОС, в том числе во всех версиях ОС *Windows* и *NetWare*. Неструктурированная модель файла позволяет легко организовать разделение файла между несколькими приложениями: разные приложения могут по-своему структурировать и интерпретировать данные, содержащиеся в файле. В этом случае поддержание структуры файла поручается файловой системе. Файловая система видит файл как упорядоченную последовательность логических записей. Приложение может обращаться к ФС с запросами на ввод-вывод на уровне записей, например, «считать запись 25 из файла *FILE.DOC*».

Файловая система должна обладать информацией о структуре файла, достаточной для того, чтобы выделить любую запись. ФС предоставляет приложению доступ к записи, а вся дальнейшая обработка данных, содержащихся в этой записи, выполняется приложением. Развитием этого подхода стали системы управления базами данных (СУБД), которые поддерживают не только сложную структуру данных, но и взаимосвязи между ними.

Логическая запись является наименьшим элементом данных, которым может оперировать программист при организации обмена с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система должна обеспечивать программисту доступ к отдельной логической записи. Файловая система может использовать два способа доступа к логическим записям: читать или записывать логические записи последовательно (последовательный доступ) или позиционировать файл на запись с указанным номером (прямой доступ).

Очевидно, что ОС не может поддерживать все возможные способы структурирования данных в файле, поэтому в тех ОС, в которых вообще поддерживается логическая структуризация файлов, поддержка реализована для небольшого числа широко распространенных схем логической организации файла.

К числу таких способов структуризации относится представление данных в виде записей фиксированной длины (рис.5а). В этом случае доступ к *n*-й записи осуществляется либо путем последовательного чтения (*n*-1) предшествующих записей, либо прямо по адресу, вычисленному по ее порядковому номеру.

Например, если *L* – длина записи, то начальный адрес *n*-й записи равен

*Lхn*. Заметим, что при такой логической организации размер записи фиксирован

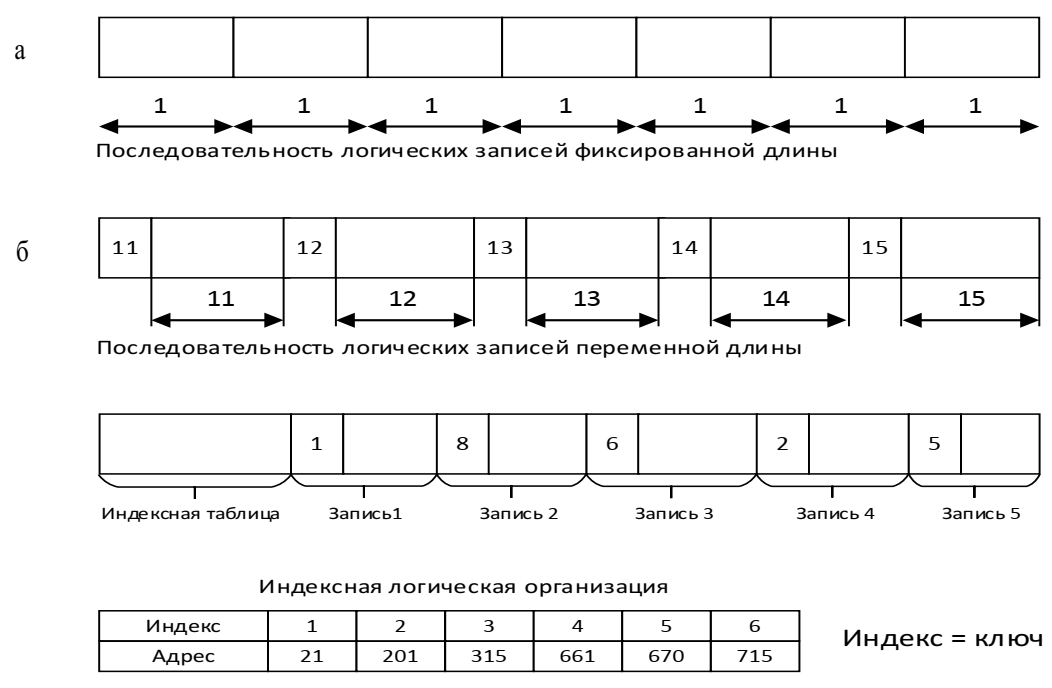
в пределах файла, а записи в различных файлах, принадлежащих одной и той же файловой системе, могут иметь различный размер.

Другой способ структуризации состоит в представлении данных в виде последовательности записей переменного размера.

Если расположить значения длин записей так, как это показано на рис. 5б, то для поиска нужной записи система должна последовательно считать все предшествующие записи. Вычислить адрес нужной записи по ее номеру при такой логической организации файла невозможно, а следовательно, не может быть применен более эффективный метод прямого доступа. Файлы, доступ к записям которых осуществляется последовательно, по номерам позиций, называются неиндексированными, или последовательными, файлами.

Другим типом файлов являются индексированные файлы; они допускают более быстрый прямой доступ к отдельной логической записи.

В индексированном файле записи имеют одно или более ключевых (индексных) полей и могут адресоваться путем указания значений этих полей. Для быстрого поиска данных в индексированном файле предусматривается специальная индексная таблица, в которой значениям ключевых полей ставится в соответствие адрес внешней памяти. Этот адрес может указывать либо непосредственно на искомую запись, либо на некоторую область внешней памяти, занимаемую несколькими записями, в число которых входит искомая запись.



*Рис. 5. Способы логической организации файлов.*

В последнем случае говорят, что файл имеет индексно-последовательную организацию, так как поиск включает два этапа: прямой доступ по индексу к указанной облаете диска, а затем последовательный просмотр записей в указанной области. Ведение индексных таблиц берет на себя файловая система. Понятно, что записи в индексированных файлах могут иметь произвольную длину.

Все сказанное в большей степени относится к обычным файлам, которые могут быть как структурированными, так и неструктурированными. Что же касается других типов файлов, таких как каталоги или файлы типа «символьная связь», то они обладают вполне определенной специфической для своего типа структурой, которая известна и «понятна» файловой системе.

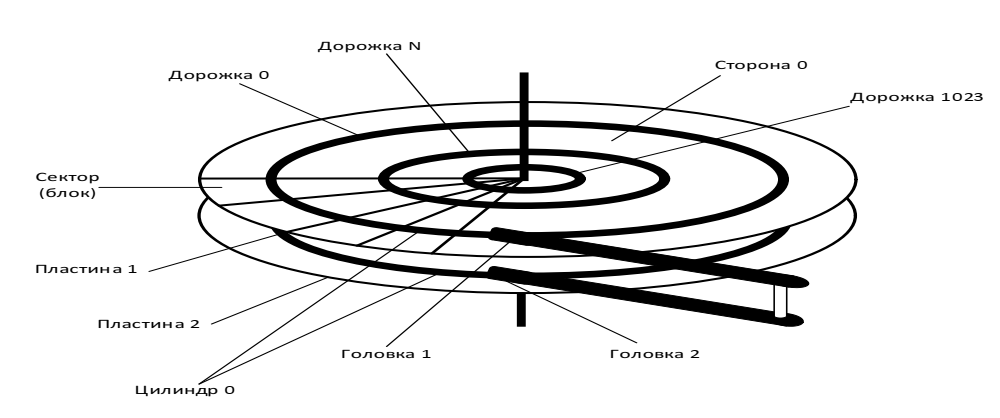
**Вопрос 2. Физическая организация файловой системы.**

Представление пользователя о файловой системе, как об иерархически организованном множестве информационных объектов, имеет мало общего с порядком хранения файлов на диске. Файл, имеющий образ цельного, непрерывающегося набора байтов, на самом деле очень часто разбросан «кусочками» по всему диску, причем это разбиение никак не связано с логической структурой файла, например, его отдельная логическая запись может быть расположена в несмежных секторах диска. Логически объединенные файлы из одного каталога совсем не обязаны соседствовать на диске.

Принципы размещения файлов, каталогов и системной информации на реальном устройстве описываются физической организацией файловой системы. Очевидно, что разные файловые системы имеют разную физическую организацию.

**Диски, разделы, секторы, кластеры**. Основным типом устройства, которое используется в современных вычислительных системах для хранения файлов, является дисковый накопитель. Дисковые накопители предназначены для считывания и записи данных на жесткие и гибкие магнитные диски.

Жесткий диск состоит из одной или нескольких стеклянных или металлических пластин, каждая из которых покрыта с одной или двух сторон магнитным материалом. Таким образом, диск в общем случае состоит из пакета пластин (рис. 6).

На каждой стороне каждой пластины размечены тонкие концентрические кольца − дорожки (*traks*), на которых хранятся данные. Количество дорожек зависит от типа диска. Нумерация дорожек начинается с 0 от внешнего края к центру диска. Когда диск вращается, элемент, называемый головкой, считывает двоичные данные с магнитной дорожки или записывает их на магнитную дорожку.

*Рис. 6.Схема устройства жесткого диска.*

Головка может позиционироваться над заданной дорожкой. Головки перемещаются над поверхностью диска дискретными шагами, каждый шаг соответствует сдвигу на одну дорожку. Запись на диск осуществляется благодаря способности головки изменять магнитные свойства дорожки. В некоторых дисках вдоль каждой поверхности перемещается одна головка, а в других – имеется по головке на каждую дорожку.

В первом случае для поиска информации головка должна перемещаться по радиусу диска. Обычно все головки закреплены на едином перемещающем механизме и двигаются синхронно. Поэтому, когда головка фиксируется на заданной дорожке одной поверхности, все остальные головки останавливаются над дорожками с такими же номерами. В тех же случаях, когда на каждой дорожке имеется отдельная головка, никакого перемещения головок с одной дорожки на другую не требуется, за счет этого экономится время, затрачиваемое на поиск данных.

Совокупность дорожек одного радиуса на всех поверхностях всех пластин пакета называется цилиндром (*cylinder*). Каждая дорожка разбивается на фрагменты, называемые секторами (*sectors*), или блоками (*blocks*), так что все дорожки имеют равное число секторов, в которые максимально можно записать одно и то же число байтов. Сектор имеет фиксированный для конкретной системы размер, выражающийся степенью двойки. Чаще всего размер сектора составляет 512 байт. Учитывая, что дорожки разного радиуса имеют одинаковое число секторов, плотность записи становится тем выше, чем ближе дорожка к центру.

Для того чтобы контроллер мог найти на диске нужный сектор, необходимо задать ему все составляющие адреса сектора:

* номер цилиндра;
* номер поверхности;
* номер сектора на дорожке.

*Сектор* – наименьшая адресуемая единица обмена данными дискового устройства с оперативной памятью. Это означает, что даже в тех случаях, когда программе требуется прочитать с диска только один байт, в действительности будет считан целый сектор и передан системе для выборки нужных данных.

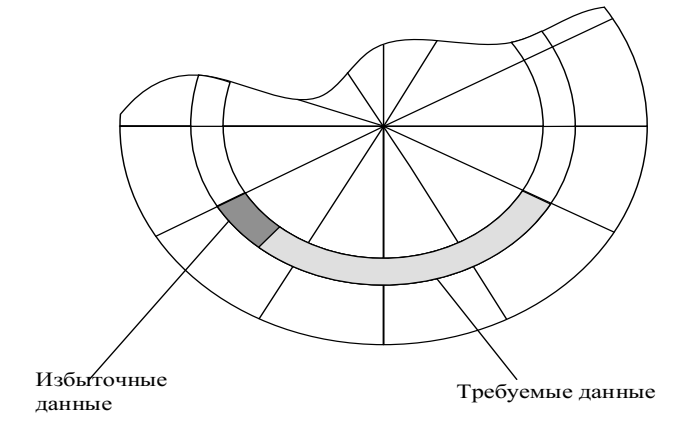
Так как прикладная программа оперирует не секторами, а байтами, к тому же заданный объем требуемых данных не обязательно оказывается кратным размеру сектора, то типичный запрос включает чтение нескольких секторов, содержащих нужную информацию, и одного или двух секторов, содержащих наряду с требуемыми избыточные данные (рис. 7).

Операционная система при работе с диском использует собственную единицу дискового пространства, называемую кластером (*cluster*). При создании файла ОС запрашивает для него область на диске, размер которой кратен принятому в этой операционной системе размеру кластера.

Например, если файл имеет размер 2560 байт, а размер кластера в файловой системе определен в 1024 байта, то файлу будет выделено на диске 3 кластера.

*Кластер* – наименьшая единица дискового пространства, которой оперирует файловая система при распределении памяти на диске.

Операционная система при работе с диском использует собственную единицу дискового пространства, называемую кластером (cluster). При создании файла ОС запрашивает для него область на диске, размер которой кратен принятому в этой операционной системе размеру кластера.



*Рис. 7. Считывание избыточных данных при обмене с диском.*

Например, если файл имеет размер 2560 байт, а размер кластера в файловой системе определен в 1024 байта, то файлу будет выделено на диске 3 кластера.

*Кластер* – наименьшая единица дискового пространства, которой оперирует файловая система при распределении памяти на диске.

Иногда кластер называют блоком, что может привести к терминологической путанице. Вообще, терминология, используемая при описании форматов дисков и файловых систем, зависит от аппаратной платформы (*RISC*, *Wintel* и т. п.) и операционной системы. Это нужно учитывать и трактовать термины в зависимости от контекста.

Дорожки и секторы создаются в результате выполнения процедуры физического, или низкоуровневого, форматирования диска, предшествующей использованию диска. Для определения границ блоков на диск записывается идентификационная информация. Низкоуровневый формат диска не зависит от типа операционной системы, которая этот диск будет использовать.

Разметку диска под конкретный тип файловой системы выполняют процедуры высокоуровневого, или логического, форматирования. При высокоуровневом форматировании определяется размер кластера, и на диск записывается информация, необходимая для работы файловой системы, в том числе: информация о доступном и неиспользуемом пространстве, о границах областей, отведенных под файлы и каталоги, о поврежденных областях. Кроме того, на диск записывается загрузчик операционной системы – небольшая программа, которая начинает процесс инициализации операционной системы после включения питания или рестарта компьютера.

Прежде чем форматировать диск под определенную файловую систему, он может быть разбит на разделы. *Раздел* – это непрерывная часть физического диска, которую операционная система представляет пользователю как логическое устройство.

В представлении прикладного программиста логическое устройство функционирует так, как если бы это был отдельный физический диск.

Именно с логическими устройствами работает пользователь, обращаясь к ним по символьным именам, например *А*, *В*, *С*, *SYS* и т. п. Операционные системы разного типа используют единое для всех них представление о разделах, но создают на его основе логические устройства, специфические для каждого типа ОС.

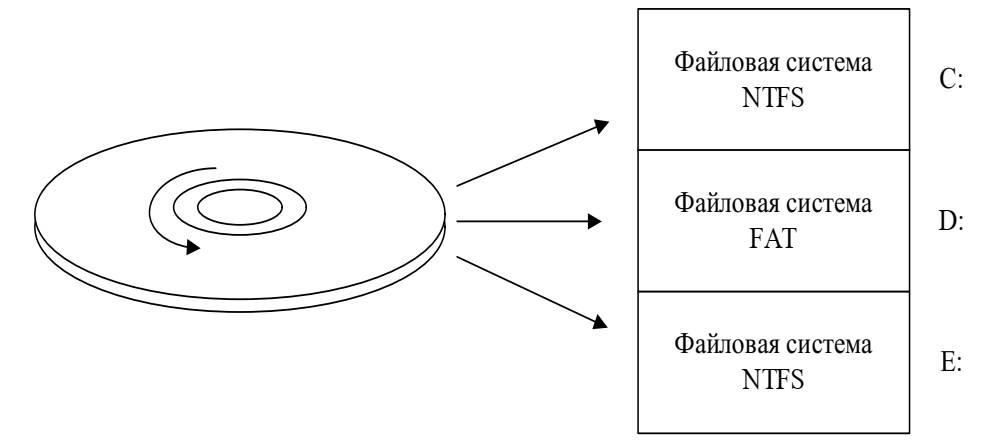
На каждом логическом устройстве может создаваться только одна файловая система.

В частном случае, когда все дисковое пространство охватывается одним разделом, логическое устройство представляет физическое устройство в целом. Если диск разбит на несколько разделов, то для каждого из этих разделов может быть создано отдельное логическое устройство. Логическое устройство может быть создано и на базе нескольких разделов, причем эти разделы не обязательно должны принадлежать одному физическому устройству. Объединение нескольких разделов в единое логическое устройство может выполняться разными способами и преследовать разные цели, основные из которых: увеличение общего объема логического раздела, повышение производительности и отказоустойчивости.

Примерами организации совместной работы нескольких дисковых разделов являются так называемые *RAID*-массивы, подробнее о которых рассказано далее.

На разных логических устройствах одного и того же физического диска могут располагаться файловые системы как одного и того же, так и разных типов. На рис. 8 показан пример диска, разбитого на три раздела, в которых установлено две файловых системы *NTFS* (разделы *С* и *Е*) и одна файловая система *FAT* (раздел *D*).

Операционная система может поддерживать разные статусы разделов, особым образом отмечая разделы, которые могут быть использованы для загрузки модулей операционной системы, и разделы, в которых можно устанавливать только приложения и хранить файлы данных. Один из разделов диска помечается как загружаемый, или активный. Именно из этого раздела считывается загрузчик операционной системы.



*Рис. 8. Разбиение диска на разделы.*

**Физическая организация и адресация файла**. Важным компонентом физической организации файловой системы является физическая организация

файла, то есть способ размещения файла на диске. Основными критериями эффективности физической организации файлов являются:

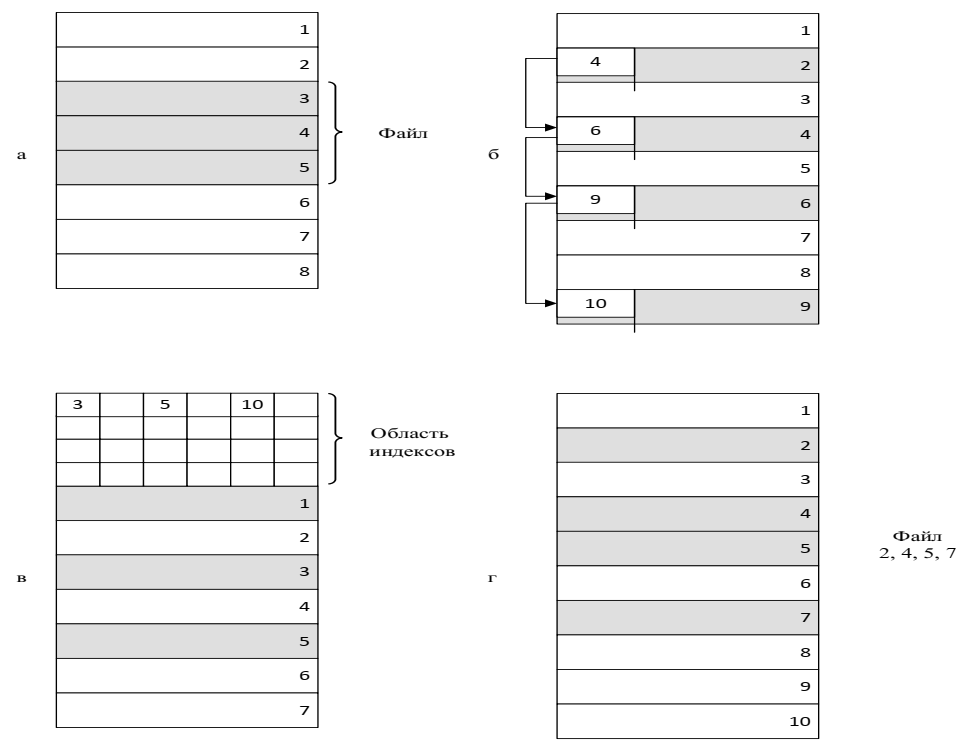
1. скорость доступа к данным;
2. объем адресной информации файла;
3. степень фрагментированности дискового пространства;
4. максимально возможный размер файла.

*Непрерывное размещение* – простейший вариант физической организации (рис. 9а), при котором файлу предоставляется последовательность кластеров диска, образующих непрерывный участок дисковой памяти.

Основным достоинством этого метода является высокая скорость доступа, так как затраты на поиск и считывание кластеров файла минимальны. Также минимален объем адресной информации – достаточно хранить только номер первого кластера и объем файла. Данная физическая организация максимально возможный размер файла не ограничивает.

Однако этот вариант имеет существенные недостатки, которые затрудняют его применимость на практике, несмотря на всю его логическую простоту. При более пристальном рассмотрении оказывается, что реализовать эту схему не так уж просто. Действительно, какого размера должна быть непрерывная область, выделяемая файлу, если файл при каждой модификации может увеличить свой размер?

Еще более серьезной проблемой является фрагментация. Спустя некоторое время после создания файловой системы, в результате выполнения многочисленных операций создания и удаления файлов пространство диска неминуемо превращается в «лоскутное одеяло», включающее большое число свободных областей небольшого размера.



*Рис. 9 Физическая организация файла: непрерывное размещение (а); связанный список кластеров (б); связанный список индексов (в); перечень номеров кластеров (г).*

Зачастую при фрагментации суммарный объем свободной памяти может быть очень большим, а выбрать место для размещения файла целиком – невозможно. Поэтому на практике используются методы, в которых файл размещается в нескольких в общем случае несмежных областях диска.

Следующий способ физической организации файла – размещение в виде связанного списка кластеров дисковой памяти (рис. 9б). При таком способе в начале каждого кластера содержится указатель на следующий кластер. В этом случае адресная информация минимальна: расположение файла может быть задано одним числом – номером первого кластера. В отличие от предыдущего способа, каждый кластер может быть присоединен к цепочке кластеров какого-либо файла, следовательно, фрагментация на уровне кластеров отсутствует.

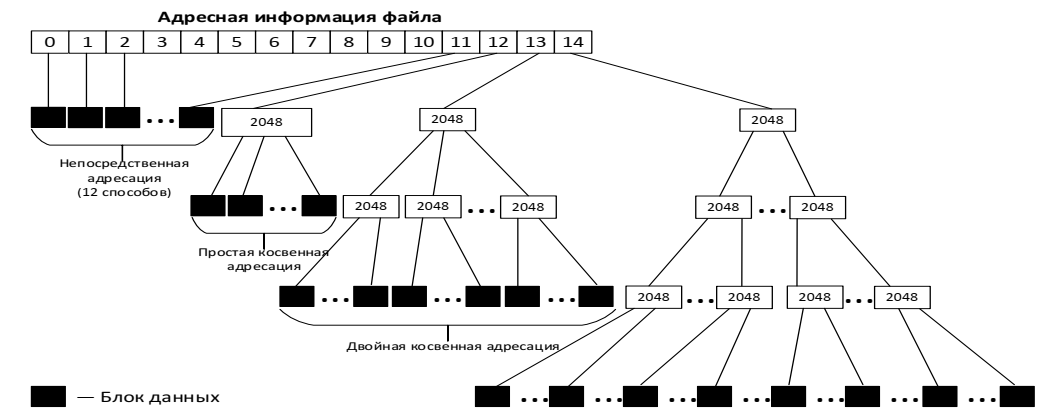
Файл может изменять свой размер во время своего существования, наращивая число кластеров. Недостатком является сложность реализации доступа к произвольно заданному месту файла – чтобы прочитать пятый по порядку кластер файла, необходимо последовательно прочитать четыре первых кластера, прослеживая цепочку номеров кластеров.

Популярным способом, применяемым, например, в файловой системе *FAT*, является размещение в виде связанного списка индексов (рис. 9в). Этот способ является некоторой модификацией предыдущего. Файлу также выделяется память в виде связанного списка кластеров. Номер первого кластера запоминается в записи каталога, где хранятся характеристики этого файла. Остальная адресная информация отделена от кластеров файла. С каждым кластером диска связывается некоторый элемент – индекс. Когда память свободна, все индексы имеют нулевое значение. Если некоторый кластер *N* назначен некоторому файлу, то индекс этого кластера становится равным либо номеру *М* следующего кластера данного файла, либо принимает специальное значение, являющееся признаком того, что этот кластер является для файла последним. Индекс же предыдущего кластера файла принимает значение *N*, указывая на вновь назначенный кластер.

При такой физической организации сохраняются все достоинства предыдущего способа: минимальность адресной информации, отсутствие фрагментации, отсутствие проблем при изменении размера. Кроме того, данный способ обладает дополнительными преимуществами. Во-первых, для доступа к произвольному кластеру файла не требуется последовательно считывать его кластеры, достаточно прочитать только секторы диска, содержащие таблицу индексов, отсчитать нужное количество кластеров файла по цепочке и определить номер нужного кластера. Во-вторых, данные файла заполняют кластер целиком, а значит, имеют объем, равный степени двойки.

Еще один способ задания физического расположения файла заключается в простом перечислении номеров кластеров, занимаемых этим файлом (рис. 9 г). Этот набор номеров и служит адресом файла. Недостаток данного способа очевиден: длина адреса зависит от размера файла и для большого файла может составить значительную величину. Достоинством же является высокая скорость доступа к произвольному кластеру файла, так как здесь применяется прямая адресация, которая исключает просмотр цепочки указателей при поиске адреса произвольного кластера файла. Фрагментация на уровне кластеров в этом способе также отсутствует.

Последний подход с некоторыми модификациями используется в традиционных файловых системах *S*5 и *ufs* ОС *Unix*. В файловой системе *ufs* для сокращения объема адресной информации используется комбинированная схема адресации кластеров, позволяющая дополнять прямой способ адресации косвенным. Для хранения адреса файла выделено 15 полей, каждое из которых состоит из 4 байт (рис. 10).



*Рис. 10. Схема адресации файловой системы ufs.*

Если размер файла меньше или равен 12 кластерам, то номера этих кластеров непосредственно перечисляются в первых двенадцати полях адреса. Если кластер имеет размер 8 Кбайт (максимальный размер кластера, поддерживаемого в *ufs*), то таким образом можно адресовать файл размером до 8192 х 12 = 98304 байт.

Если размер файла превышает 12 кластеров, то следующее 13-е поле содержит не номер следующего кластера, а номер кластера, в котором могут быть расположены номера следующих кластеров файла.

Таким образом, 13-й элемент адреса используется для косвенной адресаций. При размере в 8 Кбайт кластер, на который указывает 13-й элемент, может содержать 2048 номеров следующих кластеров данных файла, и размер файла может возрасти до 8192 х (12 + 2048) = 16 875 520 байт.

Если размер файла превышает 12 + 2048 = 2060 кластеров, то используется 14-е поле. В нем находится номер кластера, содержащего 2048 номеров кластеров, каждый из которых хранит 2048 номеров кластеров данных файла. Здесь применяется уже двойная косвенная адресация.

С ее помощью можно адресовать кластеры в файлах, содержащих до 8192 х (12 + 2048 + 20482) − 3,43766 х 1010 байт.

И наконец, если файл включает более 12 + 2048 + 20482 – 4 196 364 кластеров, то используется последнее 15-е поле для тройной косвенной адресации, что позволяет задать адрес файла, имеющего следующий максимальный размер:

8192 х (12 + 2048 + 20482 + 20483) − 7,0403 х 1013 байт.

Та к им обр а зом , файловая система ufs при размере кластера в 8 Кбайт поддерживает файлы, состоящие максимум из 70 триллионов байтов данных, хранящихся в 8 миллиардах кластеров. Как видно на рис. 2.10, для задания адресной информации о максимально большом файле требуется 15 элементов по 4 байта (60 байт) в центральной части адреса плюс 1 + (1 + 2048) + (1 + 2048

+ + 20482) – 4 198 403 кластера в косвенной части адреса. Несмотря на огромную величину, это число составляет всего около 0,05 % от объема адресуемых данных.

Файловая система *ufs* поддерживает дисковые кластеры и меньших размеров, при этом максимальный размер файла будет другим. Используемая в более ранних версиях *Unix*, файловая система *S*5 имеет аналогичную схему адресации, но она рассчитана на файлы меньших размеров, поэтому в ней применяется 13 адресных элементов вместо 15.

Метод перечисления адресов кластеров файла характерен и для файловой системы *NTFS*, используемой в ОС семейства *Windows NT*. Здесь он дополнен достаточно естественным приемом, сокращающим объем адресной информации: адресуются не кластеры файла, а непрерывные области, состоящие из смежных кластеров диска.

Каждая такая область, называемая отрезком (*run*), или экстентом (*extent*), описывается с помощью двух чисел: начального номера кластера и количества кластеров в отрезке. Так как для сокращения времени операции обмена ОС старается разместить файл в последовательных кластерах диска, то в большинстве случаев количество последовательных областей файла будет меньше количества кластеров файла.

Для того чтобы корректно принимать решение о выделении файлу набора кластеров, файловая система должна отслеживать информацию о состоянии всех кластеров диска: свободен/занят. Эта информация может храниться как отдельно от адресной информации файлов, так и вместе с ней.