14. Управление внешней памятью. Основные операции с файлами

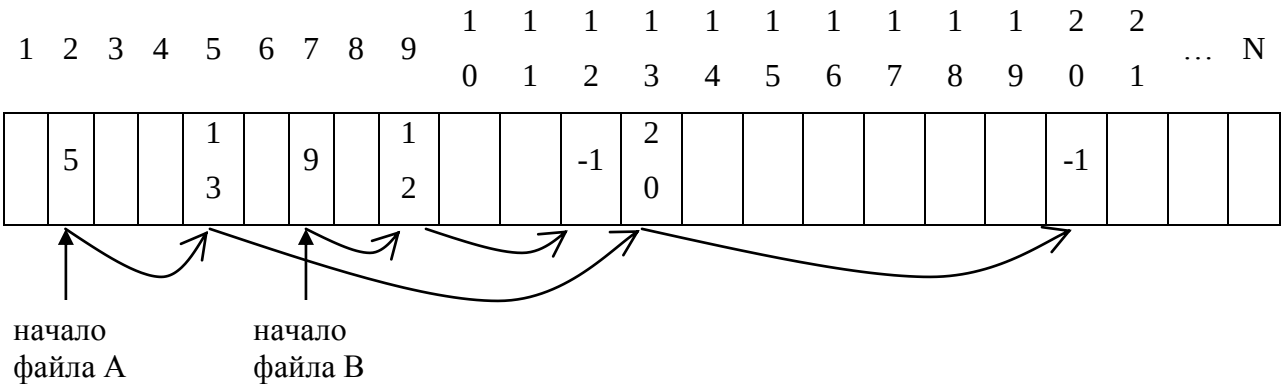
Файловая система распределяет дисковое пространство между файлами на уровне блоков/кластеров. Короткие файлы (например, до 1 Кб) будут целиком помещаться в блок, и поэтому для них достаточно запомнить номер выделенного файлу блока. Проблема возникает для достаточно длинных файлов, требующих нескольких, возможно сотен, блоков. Для решения этой проблемы существуют разные подходы, некоторые из которых рассматриваются ниже.

Наиболее простой подход – выделять новому файлу **непрерывный** набор блоков. Тогда для однозначного определения положения файла на диске достаточно двух чисел – номера первого блока и количества выделенных блоков. Огромное преимущество данного метода – высокая скорость операций чтения с минимальными затратами на поиск файла. Однако использование данного способа в общем случае приводит к ряду серьезных неприятностей, прежде всего – при удалении файлов. Между занятыми блоками возникают свободные области заранее неизвестного размера, что приводит к сильной фрагментации диска. Поэтому наиболее разумная область использования простейшего непрерывного распределения блоков – это носители с возможностью выполнения только операций чтения данных.

Для дисков общего назначения необходимы другие способы, реализующие следующую общую идею: отдельные фрагменты файла могут размещаться в **ЛЮБЫХ** свободных блоках, но файловая система должна уметь **связывать** эти блоки в единую логическую цепочку. Такой подход похож на организацию линейных динамических списков в основной памяти с помощью адресных указателей. Однако прямой перенос этой техники на дисковое пространство дает не очень хороший результат, прежде всего потому, что для поиска данных где-то в середине или конце файла придется последовательно просмотреть все предшествующие блоки. По этой причине были придуманы более хитрые механизмы, основными из которых в настоящее время являются следующие два.

Первый способ – **табличный.** Файловая система создает и поддерживает специальную таблицу распределения файлов **(File Allocation Table, FAT)**. Эта таблица является основой одноименных файловых систем FAT16 (MS DOS) и FAT32 (Windows 98/ME/XP). Таблица FAT – это массив целых чисел-номеров дисковых блоков. Число j в ячейке i говорит, что за блоком i должен идти блок j, и тем самым для задания цепочки блоков некоторого файла достаточно как-то определить номер первого блока (например, сохранив его в каталоге вместе с остальными параметрами файла) и установить факт окончания цепочки в последнем блоке (например, с помощью фиктивного номера -1). В этом случае легко восстанавливается вся цепочка выделенных файлу блоков без необходимости просмотра самих блоков.

Предположим, файлу A выделены блоки 2, 5, 13 и 20, а файлу B – блоки 7, 9 и 12. Тогда фрагмент таблицы FAT будет содержать следующие данные:



В данном способе большое значение имеет **разрядность** чисел, хранимых в FAT-таблице. Если все числа 16-разрядные (как это было в системе FAT16), то потенциально можно адресовать 2^{16} , т.е. 65536 (64 Кб) блоков. При размере блока в 2 Кб это дает общий объем адресуемой дисковой памяти, равный 128 Мб, что для современных дисков конечно же является мизерной величиной. Поэтому в системе FAT16 приходилось идти на увеличение размера блока вплоть до 32 Кб, что позволяло работать с разделами объемом в 2 Гб. Но с другой стороны, использование таких крупных блоков увеличивает потери памяти при хранении небольших файлов.

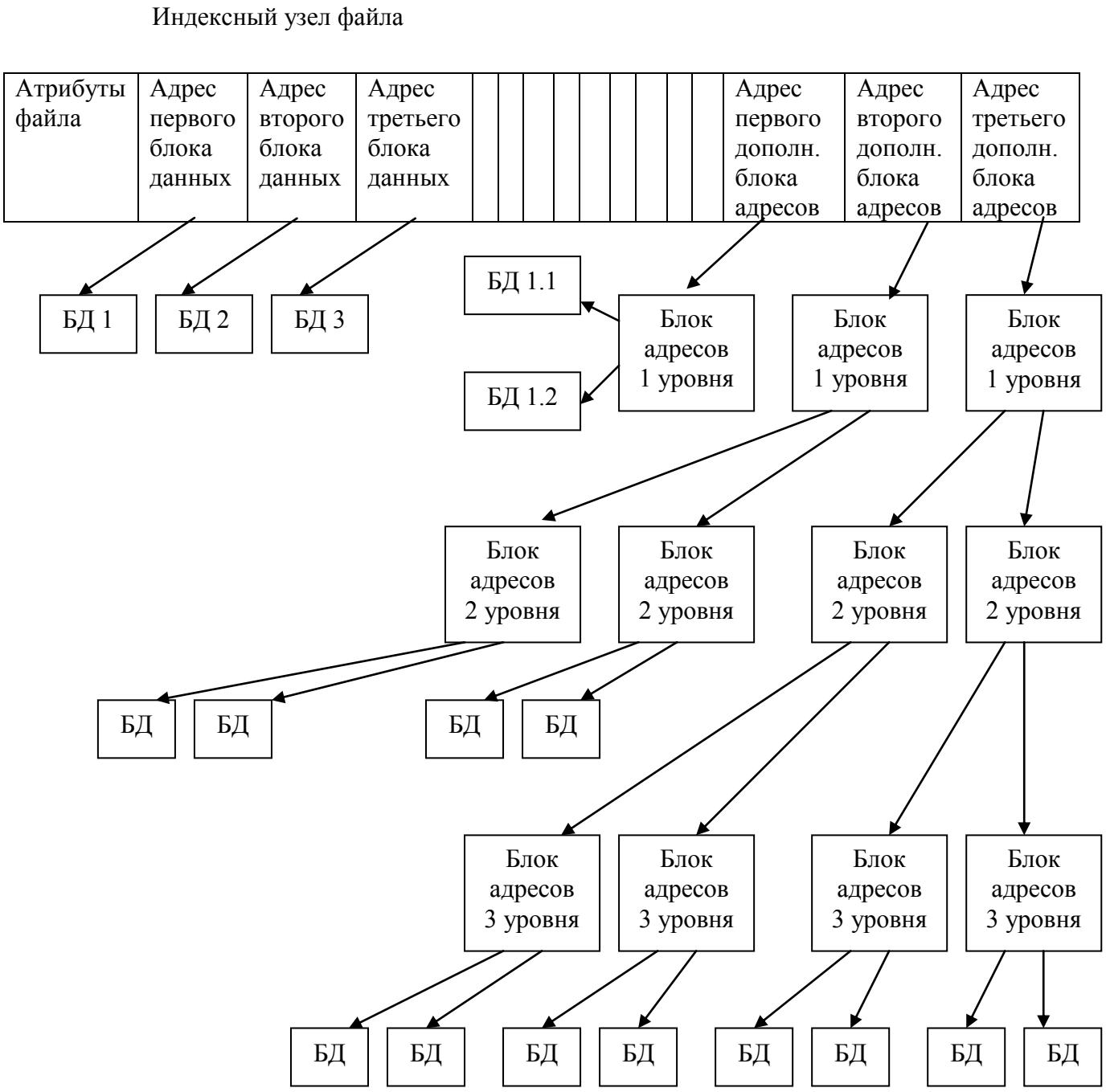
Поэтому при появлении дисков с большими объемами вместо FAT16 стала применяться система FAT32, в которой (как видно из названия) используются 4-байтовые номера блоков. Это позволяет потенциально работать с дисками объемом до 2 терабайт, используя при этом небольшие блоки/кластеры размером в 4 Кб. К сожалению, обратной стороной медали является резкое увеличение самой FAT-таблицы. Например, для диска объемом 40 Гб при использовании блоков размером 4 Кб необходимо поддерживать 10 млн. блоков, что при 4–х байтовых адресах блоков требует для таблицы 40 Мб памяти. При этом надо учесть, что для ускорения поиска адресов блоков FAT-таблица обычно постоянно хранится в оперативной памяти, отбирая ее существенную часть у приложений.

Другой способ используется в файловых системах различных версий ОС Unix и в файловой системе NTFS. Он основан на использовании так называемых **индексных узлов** – специальной структуры, которая связывается с каждым файлом. Вместо одной большой FAT-таблицы, одновременно содержащей информацию о распределении блоков по ВСЕМ файлам, создается и поддерживается множество небольших таблиц отдельно для каждого файла. Все эти таблицы хранятся на диске и загружаются в память только при обращении к файлу, т.е. при его открытии.

Индексный узел – это небольшая структура, содержащая все основные атрибуты файла и 10-15 адресов начальных блоков файла. Это позволяет легко адресовать небольшие (10 – 30 Кб) файлы, число которых всегда достаточно велико. Проблемы начинаются тогда, когда файл слишком большой, чтобы уложиться в небольшое число блоков. В этом случае один из последних блоков в индексном узле используется не для хранения данных файла, а для хранения еще одного, уже вторичного, набора адресов блоков для второй части большого файла. При среднем размере блока 2 Кб и 4–хбайтовом адресе блока один блок может хранить адреса еще 512 блоков файла, что дает в среднем 1 Мб дисковой памяти. Это приводит к **двухуровневой** схеме определения блока.

Если файл еще больше, можно включить **трехуровневую** схему: индексный узел указывает на блок, содержащий адреса блоков, каждый из которых, наконец, содержит адреса блоков с данными файла.

Ну, а для сверхбольших файлов включается **четырехуровневая** схема адресации.

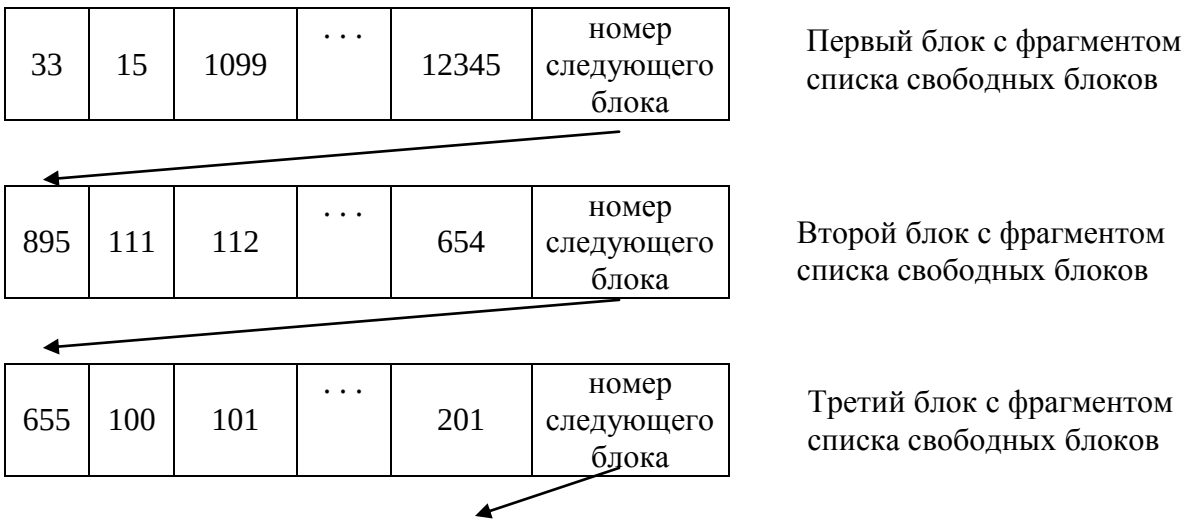


Все индексные узлы перенумерованы 3-байтовыми числами, что потенциально дает возможность поддерживать 2^{24}, т.е. более 16 млн. файлов.

При создании нового файла ему выделяется свободный индексный узел, номер которого заносится в каталог и связывается с именем созданного файла. При уничтожении файла номер его индексного узла становится свободным. Отсюда следует, что файловая система должна вести учет свободных и занятых индексных узлов. Большим достоинством данного способа является простота прямого доступа к любому фрагменту файла без необходимости просмотра всех предыдущих фрагментов. Недостаток – некоторое замедление доступа к фрагментам больших файлов за счет использования косвенной адресации.

Задача учета свободного и занятого дискового пространства аналогична учету свободных/занятых страниц оперативной памяти и решается похожими методами. Два основных метода – это **список свободных блоков** и **битовый массив блоков.**

Первый способ – это просто последовательность номеров свободных блоков. Если диск содержит 10 млн. блоков по 4 Кб, то для пустого диска список будет занимать около 40 Мб, поэтому разумно разбить его на блоки и хранить их в дисковой памяти. Один блок средних размеров (4 Кб) может хранить около 1000 номеров, т.е. всего максимально для хранения данных о свободных блоках требуется 10 тыс. блоков (0,1% объема диска).



В основной памяти достаточно иметь лишь один блок, т.е. часть списка свободных блоков. При создании файла или его расширении новые блоки

берутся именно из него, а когда этот фрагмент списка закончится, в память загружается следующий блок со следующим набором номеров свободных блоков. Аналогично, при удалении файла занимаемые им блоки освобождаются и добавляются в находящийся в памяти фрагмент списка. Когда этот фрагмент полностью заполнится, он записывается на диск, а на его место загружается следующий. Эта простейшая схема имеет ряд усовершенствований [1].

Второй способ – это создание и поддержка битового массива для ВСЕХ блоков диска. Значение бита 1 соответствует занятому состоянию блока, а значение 0 – свободному (можно и наоборот). Для 10 млн. блоков надо 10 млн. битов, т.е. немного больше 1 Мб. Такой массив можно постоянно хранить в основной памяти, хотя при необходимости его легко разбить на блоки и хранить на диске. Для этого может потребоваться, например, 256 дисковых блоков.

Преимуществом данного метода является его простота. При удалении файла и освобождении его блоков просто обнуляются соответствующие биты. При запросе на выделение файлу новых блоков система в битовом массиве ищет нулевые биты и инвертирует их значения.

Номер блока	1	2	3	4	5	6	7	8	9	10	. . .	999999	1000000
Состояние блока	1	1	1	0	0	0	1	0	1	0		0	0

Основные операции с файлами – это **создание** и **удаление, открытие** и **закрытие, чтение** и **запись, получение** или **изменение атрибутов**. Все они требуют установления соответствия символьного имени файла и его физического расположения на диске. Поскольку все основные современные ФС поддерживают иерархическую организацию файлов в виде дерева каталогов, необходимо сначала кратко рассмотреть возможные структуры каталогов.

Ранее уже было отмечено, что каталог – это просто специальный файл, в котором собирается информация о включенных в него файлах. Фактически,

каталог – это набор записей, по одной для каждого файла. Структура записей различна для разных ФС. Например, записи каталога системы FAT16/32 для каждого файла содержат символьное имя файла, атрибуты файла и номер первого дискового блока, выделенного файлу.

Наоборот, в Unix-системах атрибуты файла хранятся в его индексном узле, а запись в каталоге содержит лишь 2 поля – символьное имя файла (куда же пользователю без него!) и номер выделенного системой индексного узла.

Теперь более понятным становится алгоритм **поиска файла** по его полному имени в дереве каталогов:

- прежде всего из полного имени файла выделяется первая компонента (разделитель – прямая или обратная косая черта);
- система обращается к корневому каталогу, положение которого на диске всегда ей известно и который загружается в память при запуске системы;
- в корневом каталоге отыскивается запись, первое поле которой совпадает с выделенной компонентой полного имени;
- из найденной записи напрямую (для FAT-систем) или косвенно через индексный узел определяется номер дискового блока с очередной порцией информации;
- в память считываются данные из этого блока, из полного имени выделяется вторая компонента и организуется ее поиск в данном списке с определением номера нового дискового блока и т.д.

Для поиска в каталоге как массиве записей могут использоваться разные методы: простейший линейный просмотр, двоичный поиск в отсортированном наборе, хеш-поиск и даже поиск с использованием B-деревьев.

Следующая пара операций – это создание и уничтожение файлов. Запрос на создание файла обычно включает полное имя создаваемого файла и ряд его параметров. Отработка этого запроса включает в себя:

- поиск каталога для хранения имени файла и просмотр его на случай наличия заданного имени;
- включение новой записи в каталог и заполнение ее полей;
- поиск для создаваемого файла либо первого свободного блока (FAT-системы), либо свободного индексного узла с занесением этой информации в соответствующее поле записи каталога;
- если размер создаваемого файла известен заранее, то выделяется необходимое число свободных дисковых блоков и связывание их в цепочку.

Обратный запрос на уничтожение файла может включать в себя:

- поиск файла по его полному имени;
- определение в каталоге адреса первого блока файла или номера индексного узла;
- просмотр цепочки выделенных файлу блоков и освобождение их;
- удаление записи о файле из каталога.

Важнейшая пара операций – **открытие** и **закрытие** существующих файлов. Открытие обычно производится перед выполнением основных операций чтения или записи данных и состоит в выполнении подготовительных операций, упрощающих и ускоряющих последующий обмен данными. Для хранения данных об открытых файлах система поддерживает одну или несколько связанных таблиц. После получения запроса на открытие файла выполняются следующие действия:

- поиск файла по его полному имени с определением текущего каталога;
- выполнение необходимых проверок на возможность открытия файла;
- формирование новой записи в системной **таблице открытых файлов**, важнейшим компонентом которой является индексный узел файла;
- создание новой записи в таблице дескрипторов файлов процесса;
- возврат пользовательскому процессу созданного дескриптора для использования в последующих операциях

Очевидно, что закрытие файла означает лишь уничтожение соответствующих записей в системных таблицах.

Запрос на чтение данных из открытого файла обычно включает следующие параметры: дескриптор файла, адрес буфера ввода для временного хранения прочитанных данных, количество считываемых байтов. По дескриптору файла с помощью таблицы открытых файлов легко определяется набор необходимых дисковых блоков. Аналогично отрабатывается запрос на запись файлов, за исключением того, что если выделенных файлу блоков недостаточно, система должна выделить дополнительные блоки.