Лабораторная работа №10

Реализация файловой системы

Цель работы: **разработать модель файловой системы**

1. Теоретические сведения

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы организовать эффективную работу с данными, хранящимися во внешней памяти, и обеспечить пользователю удобный интерфейс при работе с такими данными.

1.1 Алгоритмы выделения дискового пространства

Главный вопрос, какой тип структур используется для учета отдельных блоков файла, то есть способ связывания файлов с блоками диска. В ОС используется несколько методов выделения файлу дискового пространства. Для каждого из методов запись в директории, соответствующая символьному имени файла, содержит указатель, следуя которому можно найти все блоки данного файла.

Выделение непрерывной последовательностью блоков

Простейший способ - хранить каждый файл как непрерывную последовательность блоков диска. При непрерывном расположении файл характеризуется адресом и длиной (в блоках). Файл, стартующий с блока b, занимает затем блоки b+1, b+2, ... b+n-1.

Эта схема имеет два преимущества. Во-первых, ее легко реализовать, так как выяснение местонахождения файла сводится к вопросу, где находится первый блок. Во-вторых, она обеспечивает хорошую производительность, так как целый файл может быть считан за одну дисковую операцию.

Этот способ распространен мало, и вот почему. В процессе эксплуатации диск представляет собой некоторую совокупность свободных и занятых фрагментов. Не всегда имеется подходящий по размеру свободный фрагмент для нового файла.

Кроме того, непрерывное распределение внешней памяти неприменимо до тех пор, пока неизвестен максимальный размер файла.

Единственным приемлемым решением перечисленных проблем является периодическое уплотнение содержимого внешней памяти, или "сборка мусора", цель которой состоит в объединении свободных участков в один большой блок. Но это дорогостоящая операция, которую невозможно осуществлять слишком часто.

Таким образом, когда содержимое диска постоянно изменяется, данный метод нерационален. Однако для стационарных файловых систем, например для файловых систем компакт-дисков, он вполне пригоден.

Связный список

Внешняя фрагментация - основная проблема рассмотренного выше метода - может быть устранена за счет представления файла в виде связного списка блоков диска. Запись в директории содержит указатель на первый и последний блоки файла (иногда в качестве варианта используется специальный знак конца файла - EOF). Каждый блок содержит указатель на следующий блок (см. рис. 1).

Рис. 1. Хранение файла в виде связного списка дисковых блоков

Внешняя фрагментация для данного метода отсутствует. Любой свободный блок может быть использован для удовлетворения запроса. Заметим, что нет необходимости декларировать размер файла в момент создания. Файл может расти неограниченно.
Связное выделение имеет, однако, несколько существенных недостатков:

- при прямом доступе к файлу для поиска i-го блока нужно осуществить несколько обращений к диску, последовательно считывая блоки от 1 до i-1, то есть выборка логически смежных записей, которые занимают физически несмежные секторы, может требовать много времени. Здесь мы теряем все преимущества прямого доступа к файлу.
- данный способ не очень надежен. Наличие дефектного блока в списке приводит к потере информации в оставшейся части файла и потенциально к потере дискового пространства, отведенного под этот файл.
- для указателя на следующий блок внутри блока нужно выделить место, что не всегда удобно. Емкость блока, традиционно являющаяся степенью двойки (многие программы читают и пишут блоками по степеням двойки), таким образом, перестает быть степенью двойки, так как указатель отбирает несколько байтов.

Поэтому метод связного списка обычно в чистом виде не используется.

Таблица отображения файлов

Одним из вариантов предыдущего способа является хранение указателей не в дисковых блоках, а в индексной таблице в памяти, которая называется таблицей отображения файлов (FAT - file allocation table) (см. рис. 2). Этой схемы придерживаются многие ОС (MS-DOS, OS/2, MS Windows и др.)
По-прежнему существенно, что запись в директории содержит только ссылку на первый блок. Далее при помощи таблицы FAT можно локализовать блоки файла независимо от его размера. В тех строках таблицы, которые соответствуют последним блокам файлов, обычно записывается некоторое граничное значение, например EOF.
Главное достоинство данного подхода состоит в том, что по таблице отображения можно судить о физическом соседстве блоков, располагающихся на диске, и при выделении нового блока можно легко найти свободный блок диска, находящийся поблизости от других блоков данного файла. Минусом данной схемы может быть необходимость хранения в памяти этой довольно большой таблицы.

Номера блоков диска		
1		
2	10	
3	11	Начало файла F_2
4		
5	EOF	
6	2	Начало файла F_1
7	EOF	
8		
9		
10	7	
11	5	

Рис. 2. Метод связного списка с использованием таблицы в оперативной памяти

Индексные узлы

Наиболее распространенный метод выделения файлу блоков диска - связать с каждым файлом небольшую таблицу, называемую индексным узлом (i-node), которая перечисляет атрибуты и дисковые адреса блоков файла (см. рис 3). Запись в директории, относящаяся к файлу, содержит адрес индексного блока. По мере заполнения файла указатели на блоки диска в индексном узле принимают осмысленные значения.

Индексирование поддерживает прямой доступ к файлу, без ущерба от внешней фрагментации. Индексированное размещение широко распространено и поддерживает как последовательный, так и прямой доступ к файлу.

Обычно применяется комбинация одноуровневого и многоуровневых индексов. Первые несколько адресов блоков файла хранятся непосредственно в индексном узле, таким образом, для маленьких файлов индексный узел хранит всю необходимую информацию об адресах блоков диска. Для больших файлов один из адресов индексного узла указывает на блок косвенной адресации. Данный блок содержит адреса дополнительных блоков диска. Если этого недостаточно, используется блок двойной косвенной адресации, который содержит адреса блоков косвенной адресации. Если и этого не хватает, используется блок тройной косвенной адресации.

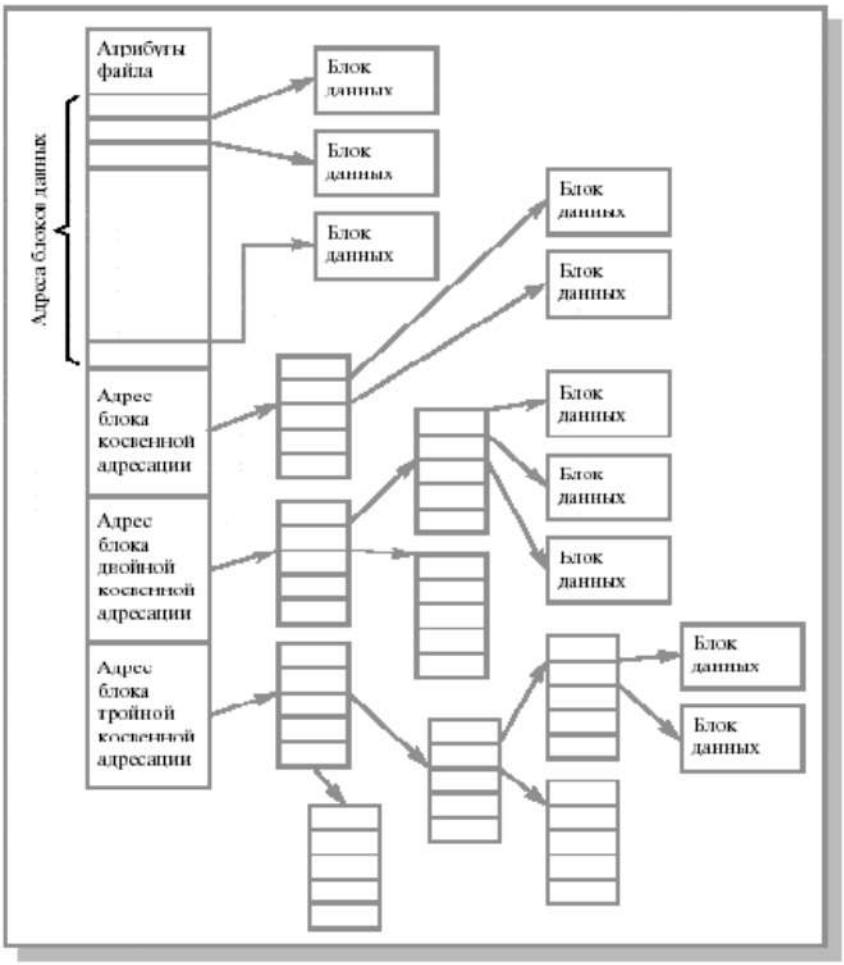

Рис. 3. Структура индексного узла

Данную схему используют файловые системы Unix (а также файловые системы HPFS, NTFS и др.). Такой подход позволяет при фиксированном, относительно небольшом размере индексного узла поддерживать работу с файлами, размер которых может меняться от нескольких байтов до нескольких гигабайтов. Существенно, что для маленьких файлов используется только прямая адресация, обеспечивающая максимальную производительность.

1.2 Управление свободным и занятым дисковым пространством

Дисковое пространство, не выделенное ни одному файлу, также должно быть управляемым. В современных ОС используется несколько способов учета используемого места на диске. Рассмотрим наиболее распространенные.

Учет при помощи организации битового вектора

Часто список свободных блоков диска реализован в виде битового вектора (bit map или bit vector). Каждый блок представлен одним битом, принимающим значение 0 или 1, в зависимости от того, занят он или свободен.

Например, 00111100111100011000001 ... .

Главное преимущество этого подхода состоит в том, что он относительно прост и эффективен при нахождении первого свободного блока или n последовательных блоков на диске. Многие компьютеры имеют инструкции манипулирования битами, которые могут использоваться для этой цели.

Несмотря на то что размер описанного битового вектора наименьший из всех возможных структур, даже такой вектор может оказаться большого размера. Поэтому данный метод эффективен, только если битовый вектор помещается в памяти целиком, что возможно лишь для относительно небольших дисков. Например, диск размером 4 Гбайт с блоками по 4 Кбайт нуждается в таблице размером 128 Кбайт для управления свободными блоками. Иногда, если битовый вектор становится слишком большим, для ускорения поиска в нем его разбивают на регионы и организуют резюмирующие структуры данных, содержащие сведения о количестве свободных блоков для каждого региона.

Учет при помощи организации связного списка

Другой подход - связать в список все свободные блоки, размещая указатель на первый свободный блок в специально отведенном месте диска, попутно кэшируя в памяти эту информацию. Подобная схема не всегда эффективна. Для трассирования списка нужно выполнить много обращений к диску. Однако, к счастью, нам необходим, как правило, только первый свободный блок.

Иногда прибегают к модификации подхода связного списка, организуя хранение адресов n свободных блоков в первом свободном блоке. Первые n-1 этих блоков действительно используются. Последний блок содержит адреса других n блоков и т. д.

Существуют и другие методы, например, свободное пространство можно рассматривать как файл и вести для него соответствующий индексный узел.

Размер блока

Размер логического блока играет важную роль. В некоторых системах (Unix) он может быть задан при форматировании диска. Небольшой размер блока будет приводить к тому, что каждый файл будет содержать много блоков. Чтение блока осуществляется с задержками на поиск и вращение, таким образом, файл из многих блоков будет читаться медленно. Большие блоки обеспечивают более высокую скорость обмена с диском, но из-за внутренней фрагментации (каждый файл занимает целое число блоков, и в среднем половина последнего блока пропадает) снижается процент полезного дискового пространства.

Для систем со страничной организацией памяти характерна сходная проблема с размером страницы.

Проведенные исследования показали, что большинство файлов имеют небольшой размер. Например, в Unix приблизительно 85% файлов имеют размер менее 8 Кбайт и 48% - менее 1 Кбайта.

Можно также учесть, что в системах с виртуальной памятью желательно, чтобы единицей пересылки диск-память была страница (наиболее распространенный размер страниц памяти - 4 Кбайта). Отсюда обычный компромиссный выбор блока размером 512 байт, 1 Кбайт, 2 Кбайт, 4 Кбайт.

1.3 Структура файловой системы на диске

Рассмотрение методов работы с дисковым пространством дает общее представление о совокупности служебных данных, необходимых для описания файловой системы. Структура служебных данных типовой файловой системы, например Unix, на одном из разделов диска, таким образом, может состоять из четырех основных частей (см. рис. 4).

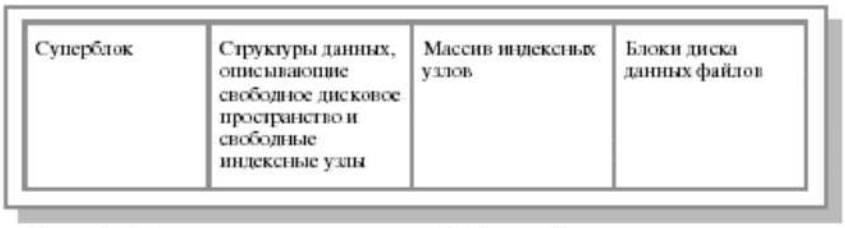

Рис. 4. Примерная структура файловой системы на диске

В начале раздела находится суперблок, содержащий общее описание файловой системы, например:

- тип файловой системы;
- размер файловой системы в блоках;
- размер массива индексных узлов;
- размер логического блока.

Описанные структуры данных создаются на диске в результате его форматирования (например, утилитами format, makefs и др.). Их наличие позволяет обращаться к данным на диске как к файловой системе, а не как к обычной последовательности блоков.
В файловых системах современных ОС для повышения устойчивости поддерживается несколько копий суперблока. В некоторых версиях Unix суперблок включал также и структуры данных, управляющие распределением дискового пространства, в результате чего суперблок непрерывно подвергался модификации, что снижало надежность файловой системы в целом. Выделение структур данных, описывающих дисковое пространство, в отдельную часть является более правильным решением.
Массив индексных узлов (ilist) содержит список индексов, соответствующих файлам данной файловой системы. Размер массива индексных узлов определяется администратором при установке системы. Максимальное число файлов, которые могут быть созданы в файловой системе, определяется числом доступных индексных узлов.
В блоках данных хранятся реальные данные файлов. Размер логического блока данных может задаваться при форматировании файловой системы. Заполнение диска содержательной информацией предполагает использование блоков хранения данных для файлов директорий и обычных файлов и имеет следствием модификацию массива индексных узлов и данных, описывающих пространство диска. Отдельно взятый блок данных может принадлежать одному и только одному файлу в файловой системе.

Реализация директорий

Как уже говорилось, директория или каталог - это файл, имеющий вид таблицы и хранящий список входящих в него файлов или каталогов. Основная задача файлов-директорий - поддержка иерархической древовидной структуры файловой системы. Запись в директории имеет определенный для данной ОС формат, зачастую неизвестный пользователю, поэтому блоки данных файла-директории заполняются не через операции записи, а при помощи специальных системных вызовов (например, создание файла).
Для доступа к файлу ОС использует путь (pathname), сообщенный пользователем. Запись в директории связывает имя файла или имя поддиректории с блоками данных на диске (см. рис. 5). В зависимости от способа выделения файлу блоков диска (см. раздел "Методы выделения дискового пространства") эта ссылка может быть номером первого блока или номером индексного узла. В любом случае обеспечивается связь символьного имени файла с данными на диске.

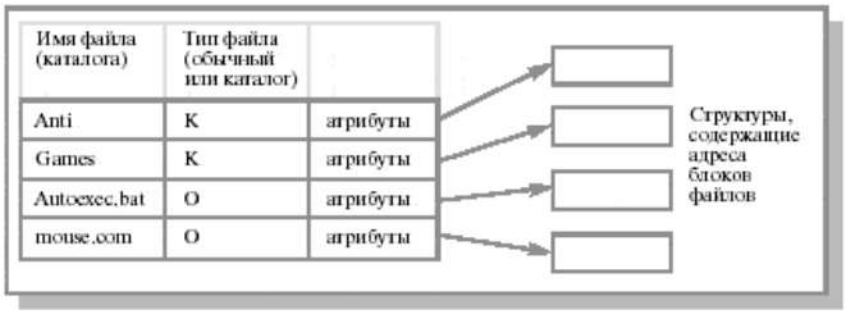

Рис. 5. Реализация директорий

Когда система открывает файл, она ищет его имя в директории. Затем из записи в директории или из структуры, на которую запись в директории указывает, извлекаются атрибуты и адреса блоков файла на диске. Эта информация помещается в системную таблицу в главной памяти. Все последующие ссылки на данный файл используют эту информацию. Атрибуты файла можно хранить непосредственно в записи в директории, как показано на рис. 5. Однако для организации совместного доступа к файлам удобнее хранить атрибуты в индексном узле, как это делается в Unix.

Примеры реализации директорий в некоторых ОС

Директории в ОС MS-DOS

В ОС MS-DOS типовая запись в директории имеет вид, показанный на рис. 6.

Рис. 6. Вариант записи в директории MS-DOS

В ОС MS-DOS, как и в большинстве современных ОС, директории могут содержать поддиректории (специфицируемые битом атрибута), что позволяет конструировать произвольное дерево директорий файловой системы.
Номер первого блока используется в качестве индекса в таблице FAT. Далее по цепочке в этой таблице могут быть найдены остальные блоки.

Директории в ОС Unix
Структура директории проста. Каждая запись содержит имя файла и номер его индексного узла (см. рис. 7). Вся остальная информация о файле (тип, размер, время модификации, владелец и т. д. и номера дисковых блоков) находится в индексном узле.

Рис. 7. Вариант записи в директории Unix

В более поздних версиях Unix форма записи претерпела ряд изменений, например имя файла описывается структурой. Однако суть осталась прежней.

1.4 Поиск в директории

Список файлов в директории обычно не является упорядоченным по именам файлов. Поэтому правильный выбор алгоритма поиска имени файла в директории имеет большое влияние на эффективность и надежность файловых систем.

Линейный поиск

Существует несколько стратегий просмотра списка символьных имен. Простейшей из них является линейный поиск. Директория просматривается с самого начала, пока не встретится нужное имя файла. Хотя это наименее эффективный способ поиска, оказывается, что в большинстве случаев он работает с приемлемой производительностью. Например, авторы Unix утверждали, что линейного поиска вполне достаточно. По-видимому, это связано с тем, что на фоне относительно медленного доступа к диску некоторые задержки, возникающие в процессе сканирования списка, несущественны. Метод прост, но требует временных затрат. Для создания нового файла вначале нужно проверить директорию на наличие такого же имени. Затем имя нового файла вставляется в конец директории (если, разумеется, файл с таким же именем в директории не существует, в противном случае нужно информировать пользователя). Для удаления файла нужно также выполнить поиск его имени в списке и пометить запись как неиспользуемую. Реальный недостаток данного метода - последовательный поиск файла. Информация о структуре директории используется часто, и неэффективный способ поиска будет заметен пользователями. Можно свести поиск к бинарному, если отсортировать список файлов. Однако это усложнит создание и удаление файлов, так как требуется перемещение большого объема информации.

Хеш-таблица

Хеширование (см. например, [Ахо, 2001]) - другой способ, который может использоваться для размещения и последующего поиска имени файла в директории. В данном методе имена файлов также хранятся в каталоге в виде линейного списка, но дополнительно используется хеш-таблица. Хеш-таблица, точнее построенная на ее основе хеш-функция, позволяет по имени файла получить указатель на имя файла в списке. Таким образом, можно существенно уменьшить время поиска.

В результате хеширования могут возникать коллизии, то есть ситуации, когда функция хеширования, примененная к разным именам файлов, дает один и тот же результат. Обычно имена таких файлов объединяют в связные списки, предполагая в дальнейшем осуществление в них последовательного поиска нужного имени файла. Выбор подходящего алгоритма хеширования позволяет свести к минимуму число коллизий. Однако всегда есть вероятность неблагоприятного исхода, когда непропорционально большому числу имен файлов функция хеширования ставит в соответствие один и тот же результат. В таком случае преимущество использования этой схемы по сравнению с последовательным поиском практически утрачивается.

2. Индивидуальные задания

Разработать приложение, создающее виртуальный файл и позволяющее

- форматировать виртуальный файл с возможностью задания размера кластера;
- создавать каталоги в виртуальном файле;
- производить учёт свободного пространства;
- реализовывать поиск файлов и директорий;
- сохранять в виртуальный файл файлы с жёсткого диска;
- удалять файлы из виртуального файла;
- записывать на жёсткий диск файлы из виртуального файла;
- создавать в виртуальном файле текстовые файлы;
- предоставлять возможность редактировать текстовые файлы внутри виртуального файла.

Файловую систему внутри виртуального файла выбрать согласно варианта.

Таблица 13. Варианты заданий

Вариант	Условие задачи	Учёт свободных блоков	Поиск файлов и папок
1	Последовательная файловая система. Выделение непрерывной последовательности блоков. (Best Fit).	Связанный список	Хэш-таблицы
2	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	Бинарный поиск
3	Одноуровневые индексные узлы.	Файловый	B-дерево
4	Индексно-последовательная файловая система. Таблица отображения файлов.	Связанный список	Линейный поиск
5	Многоуровневые индексные узлы.	Файловый	Линейный поиск
6	Индексно-последовательная файловая система. Связанный список.	Связанный список	Хэш-таблицы
7	Последовательная файловая система. Выделение непрерывной последовательности блоков. (Worst Fit).	Связанный список	Бинарный поиск
8	Индексно-последовательная файловая система. Таблица отображения файлов.	Файловый	Бинарный поиск
9	Одноуровневые индексные узлы.	Битовый вектор	Бинарный поиск
10	Многоуровневые индексные узлы.	Битовый вектор	Линейный поиск
11	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	Хэш-таблицы
12	Последовательная файловая система. Выделение непрерывной последовательности блоков. (First Fit).	Связанный список	Хэш-таблицы
13	Индексно-последовательная файловая система. Связанный список.	Битовый вектор	B-дерево
14	Одноуровневые индексные узлы.	Связанный список	Хэш-таблицы
15	Индексно-последовательная файловая система. Таблица отображения файлов.	Битовый вектор	Хэш-таблицы

Лабораторная работа №11

Управление файловой системой

Цель работы:

1. Теоретические сведения

См. условие задания лабораторной работы №7

2. Индивидуальные задания

Дополнить программу, разработанную при выполнении лабораторной работы №13 вспомогательной утилитой, согласно варианта (см. табл. 14).

Таблица 14. Варианты заданий

Вариант	Дополнительная утилита
1	Дефрагментация
2	Скандиск
3	Управление доступом
4	Журнализация
5	Журнализация
6	Кэширование
7	«Сборка мусора»
8	Рекласторизатор
9	Дефрагментация
10	Дефрагментация
11	Управление доступом
12	Рекласторизатор
13	Управление доступом
14	Дефрагментация
15	Рекласторизатор