Федеральное государственное образовательное бюджетное учреждение

высшего образования «Финансовый университет при Правительстве

Российской Федерации»

**Доклад по дисциплине «Операционные системы»**

**«Управление свободным и занятым дисковым пространством»**

Выполнил студент

группы 2ПКС-116

Зайцев Н. В.

Москва 2017

**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. Hапример, 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. Например, компьютеры семейств Intel и Motorola имеют инструкции, при помощи которых можно легко локализовать первый единичный бит в слове.

Описываемый метод учета свободных блоков используется в Apple Macintosh.

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.

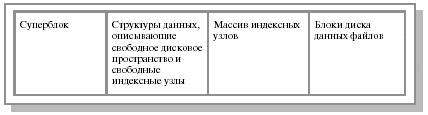
Для систем со страничной организацией памяти характерна сходная проблема с размером страницы.

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.

**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.



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.