Оглавление



# 1 Логическая организация файловой системы

## 1.1 Особенность файловой системы Linux:

- ***иерархическая структура***
- ***защита информации в файлах***
- ***трактовка периферийных устройств как файлов***
- ***существование “жестких ссылок” (hard link) и “символических ссылок” (soft link)***

Файловая система организована в виде дерева с одной исходной вершиной, которая называется корнем (записывается: "/");

**Каждая вершина в древовидной структуре файловой системы, кроме листьев, является каталогом.**

**Листья соответствуют либо обычным файлам, либо файлам устройств.**

(Что существенно отличается от таких файловых систем, как FAT и NTFS, в которых нет *единой* вершины файловой системы, ее роль выполняют корневые каталоги логических дисков, формально между собой независимые).

## 1.2 Логическая организация файловой системы

Имени файла предшествует указание пути поиска, который описывает место расположения файла в иерархической структуре файловой системы.

**Имя пути** поиска состоит из компонент, разделенных между собой наклонной чертой (/); каждая компонента представляет собой набор символов, составляющих имя вершины, которое является уникальным для каталога (предыдущей компоненты), в котором оно содержится. В имя могут включаться практически любые символы (даже перевод строки “\n” (Enter)!).

Полное имя пути поиска начинается с указания наклонной черты и идентифицирует файл (вершину), поиск которого ведется от корневой вершины дерева файловой системы с обходом тех ветвей дерева файлов, которые соответствуют именам отдельных компонент.

**Программы, выполняемые под управлением системы Linux, не содержат никакой информации относительно внутреннего формата, в котором ядро хранит файлы данных. Данные в программах представляются как бесформатный поток байтов. Программы могут интерпретировать поток байтов по своему желанию, при этом любая интерпретация никак не будет связана с фактическим способом хранения данных в операционной системе.**

Каталоги похожи на обычные файлы - система представляет информацию в каталоге набором байтов, но эта информация включает в себя имена файлов в каталоге в объявленном формате для того, чтобы операционная система и программы, такие как **ls** (выводит список имен и атрибутов файлов), могли их обнаружить.

**Права доступа к файлу** регулируются установкой специальных битов разрешения доступа, связанных с файлом. Устанавливая биты разрешения доступа, можно независимо управлять выдачей разрешений на **чтение**, **запись** и **выполнение** для трех категорий пользователей: **владельца (создателя)** файла, **группового пользователя прочих**.

**Пользователи** могут создавать файлы, если разрешен доступ на запись к каталогу, в котором создается файл. Вновь созданные файлы становятся листьями в древовидной структуре файловой системы.

## 1.3 Монтирование файловых систем

Иерархическое пространство имен обычно очень велико и включает пространства нескольких файловых систем, расположенных на разных устройствах.

Когда файловая система монтируется в существующее дерево имен, ее корневой каталог заменяет некоторый заданный каталог, для чего используется следующий вызов:

**mount (‘/bin’,’/dev/hda’,R/W)**

В результате устройство, представленное файлом **/dev/hda**, монтируется в каталог **/bin**.

Признак R/W указывает, что оно может использоваться как для чтения, так и для записи. После монтирования файл **/cc** получает имя **/bin/cc**.

Для удаления файловой системы из общего пространства имен можно применить системный вызов **unmount ().**

## 1.4 Каталоги и текущие каталоги.

Использовать для доступа к файлам их полные имена, заданные относительно корневого каталога, не всегда удобно.

Поэтому ядро UNIX связывает с каждым процессом каталог, называемый текущим или рабочим, и все имена интерпретируются как заданные относительно этого каталога, что не только упрощает работу пользователей, но еще и ускоряет поиск.

Кроме того, для каждого пользователя определен начальный каталог, который при регистрации пользователя в системе назначается рабочим каталогом.

Процесс может в любой момент сменить рабочий каталог с помощью системного вызова **chdir().**

Linux поддерживает особый способ навигации по дереву имен вверх от рабочего каталога к корню.

Для этой цели в каждый каталог включаются два служебных элемента-обозначения:

«**.**» - для данного каталога,

«**..**» - для родительского.

Например, если рабочим каталогом назначен **/home/tlh20**, то **../jmb** является путевым именем файла **jmb**, находящегося в каталоге, родительском по отношению к каталогу home. Если мы находимся в каталоге **"/dev",** то путь **"./tty01"** указывает файл, полное имя пути поиска для которого **"/dev/tty01".**

Однако вместо того чтобы задавать длинные относительные имена подобного типа, часто бывает удобнее сменить рабочий каталог.

## 1.5 Специальные файлы устройств

Система Linux трактует устройства так, как если бы они были файлами. Внешние накопители (жесткие и гибкие диски) также можно *смонтировать* (подключить) в файловую систему в *точку монтирования.*

Устройства, для которых назначены специальные файлы устройств, становятся вершинами в структуре файловой системы. Обращение программ к устройствам имеет тот же самый синтаксис, что и обращение к обычным файлам. **Способ защиты устройств совпадает со способом защиты обычных файлов: путем соответствующей установки битов разрешения доступа к ним (файлам устройств).**

Поскольку имена устройств выглядят так же, как и имена обычных файлов, и поскольку над устройствами и над обычными файлами выполняются одни и те же операции, большинству программ нет необходимости различать внутри себя типы обрабатываемых файлов. Например, программа копирования файлов будет одинаково копировать их, вне зависимости от того, чем физически являются файлы – портами ввода-вывода, терминалами или “обычными” файлами.

## 1.6 Жесткие ссылки (hard link) и символические ссылки (soft link).

Еще одной особенностью файловых систем Linux является наличие и поддержка “жестких ссылок” (hard link) и “символических ссылок” (soft link).

**Жесткие ссылки**

Жесткие ссылки можно создавать только для файлов (а не для файлов и каталогов) и только в пределах одной файловой системы (внешнего накопителя).

Жесткие ссылки для операционной системе Linux и реальное имя файла *идентичны*, после создания жесткой ссылки нельзя определить, какое имя первоначально являлось оригиналом (Поэтому любой файл всегда имеет как минимум одну жесткую ссылку – его имя, под которым он был создан.).

При удалении жесткой ссылки Linux подсчитывает оставшееся количество ссылок, указывающих на файл, и не освобождает блоки данных файла на физическом носителе до тех пор, пока не удалит его последнюю ссылку.

Символическая или косвенная ссылка обеспечивает возможность вместо имени файла (с путем) или каталога указывать имя ссылки; т.е. символическая ссылка представляет собой псевдоним (*текстовую подстановку*) для имени. Файл, на который указывает символическая ссылка, и сама ссылка представляют собой *разные* объекты файловой системы. Поэтому можно создавать ссылки на несуществующие файлы и каталоги, удалять оригинальные файлы, не удалив при этом ссылку. Можно создавать ссылки на ссылки и т.п.

Команда создания ссылок

**ln –l /usr/src /usr/linux-sources**

**ln** формирует символическую ссылку **“/usr/linux-source”** на каталог **/usr/src.**

Теперь для доступа к файлам в реальном каталоге **/usr/src** можно использовать две конструкции (будем считать, что каталог **/usr/src** содержит файл **Makefile**):

**/usr/src/Makefile**

или **/usr/linux-sources/Makefile**

## 1.7 Системные вызовы файловой системы в Linux

Системные вызовы

- **для работы с отдельными файлами**
- **для работы с каталогами**
- **для работы с файловой системой в целом.**

Для создания нового файла используется системный вызов **creat.** В качестве параметров этому системному вызову задается имя файла и режим защиты.

Так, команда

**fd = creat("abc", mode):**

создает файл *abc* с режимом защиты, указанном в переменной (или константе) *mode*. Биты *mode* определяют пользователей, которые могут получить доступ к файлу, а также уровень предоставляемого им доступа

Системный вызов **creat** не только создает новый файл, но также и открывает его для записи.

Чтобы последующие системные вызовы могли получить доступ к файлу, успешный системный вызов creat возвращает небольшое неотрицательное целое число, называемое дескриптором файла ***(fd*** в приведенном выше примере). Если системный вызов выполняется с уже существующим файлом, длина этого файла уменьшается до 0, а все содержимое теряется.

Чтобы прочитать данные из существующего файла или записать данные в существующий файл, файл сначала нужно открыть с помощью системного вызова **open**. Этому системному вызову следует указать имя файла, а также режим, в котором он должен быть открыт: для чтения, для записи или и для того и для другого. Также можно указать различные дополнительные параметры. Системный вызов open возвращает дескриптор файла, который может быть использован для чтения или записи журнала. Затем файл может быть закрыт при помощи системного вызова **close.**

. У вызовов **read и write** по три параметра:

- дескриптор файла (указывающий, с каким из открытых файлов будет производиться операция чтения или записи),
- адрес буфера (сообщающий, куда поместить данные или откуда их взять),
- счетчик байтов (указывающий, сколько байтов следует прочитать или записать).

Пример типичного вызова:

**п = read(fd. buffer, nbytes)**

С каждым открытым файлом связан указатель на текущую позицию в файле. При последовательном чтении или записи он указывает на следующий байт, который будет прочитан или записан. Например, если указатель установлен на 4096-й байт, то после успешного чтения 1024 байт из этого файла при помощи системного вызова read он будет указывать на 5120-й байт.

Указатель в файле можно переместить с помощью системного вызова **lseek**, что позволяет при следующих обращениях к **системным** вызовам read и write читать данные из файла или писать их в файл в произвольной позиции в файле и даже за концом файла.

У системного вызова **lseek** три параметра:

- это дескриптор файла,
- новая позиция в файле,
- указывается ли эта позиция относительно начала файла, конца файла или относительно текущей позиции.

Значение, возвращаемое системным вызовом **lseek**, представляет собой абсолютную позицию в файле после того, как указатель был перемещен.

Для каждого файла операционная система UNIX хранит такие сведения, как тип (режим) файла (обычный, каталог, специальный файл), его размер, время последнего изменения и т. д. Программы могут получить эту информацию при помощи системного вызова **stat**. Первый параметр вызова представляет собой имя файла. Второй является указателем на структуру, в которую

следует поместить требуемую информацию. Системный вызов **fstat** представляет собой то же самое, что и системный вызов **stat**, с той разницей, что он работает с уже открытым файлом (имя которого может быть неизвестно), а не с путем.

Системный вызов **pipe** используется для создания каналов. Он создает псевдофайл для буферирования данных, которыми обмениваются компоненты канала, и возвращает дескрипторы файлов для чтения и записи буфера.

Системный вызов в **fcntl** используется для блокировки и разблокирования файлов, а также некоторых других специфических для файлов операций.

Каталоги создаются и удаляются при помощи системных вызовов **mkdi г** и **rmdir** соответственно. Каталог может быть уничтожен, только если он пуст.

При создании связи с файлом создается новая запись в каталоге, указывающая на существующий файл. Связь создается при помощи системного вызова **link**. В параметрах этого системного вызова указываются исходное и новое имя. Записи в каталоге удаляются системным вызовом **unlink**. Когда удаляется последняя связь с файлом, файл также автоматически удаляется.

Рабочий каталог можно изменить при помощи системного вызова **chdir.** После выполнения этого системного вызова будут по-другому интерпретироваться относительные имена путей.

Каталоги могут открываться, закрываться и читаться аналогично обычным файлам. Каждое обращение к системному вызову **readdir** возвращает ровно одну запись каталога фиксированного формата.

Пользователям запрещено писать в каталоги (это делается, чтобы пользователи случайно не нарушили целостности системы). Файлы могут добавляться к каталогу при помощи системных вызовов **creat** и **link**, а удаляться с помощью системного вызова unlink.

В операционной системе Linux нет способа обращаться к файлу по расположению его описателя в каталоге, но есть системный вызов **rewinddir**, позволяющий начать читать открытый каталог с начала.

# *2 Физическая реализация файловой системы*

## 2.1 Физическая организация файловых систем

Вместо термина «кластер» в файловых системах Linux используется термин «блок». **Размер блока кратен 512 байт.** В ранних системах размер блока обычно составлял 512 байт, теперь же, как правило, используются блоки большего размера, что позволяет ускорить доступ к хранящимся на диске данным за счет потери некоторого объема его пространства, связанной с внутренней фрагментацией.

Раздел ФС делится на 4 области:

- **загрузочный блок** (не используется файловой системой)**;**
- **суперблок(superblock) (** блок 1**) содержит самую общую информацию о файловой системе**
  - количество блоков в файловой системе,
  - количество свободных блоков в файловой системе,
  - список свободных блоков,
  - размер таблицы г-узлов,
  - количество свободных i-узлов,
  - список свободных i-узлов,
  - флаги, используемые для синхронизации доступа к свободным блокам и i-узлам.

Обычно копии суперблока содержатся в нескольких местах диска, поскольку хранящаяся в нем информация критически важна для файловой системы.

- **Массив индексных дескрипторов (inode list**)(группа блоков 2-N) содержит таблицу i-узлов
- **Блоки хранения данных файла (**блоки, начиная с блока N+ 1**),** либо выделены для файлов, либо свободны. Они занимают основную часть дискового пространства. Информация о свободных блоках организована в виде цепочки таблиц свободных блоков. Операция чтения возвращает одну такую таблицу и указатель на следующую таблицу.

## 2.2 Индексные узлы

Основная особенность физической организации – **отделение имени файла от его характеристик**, хранящихся в отдельной структуре, называемой индексным дескриптором(**inode**)

**inode имеет размер 64 байта:**

- идентификатор владельца;
- разрешенные способы доступа,
- время последнего обращения с использованием каждого вида доступа,
- 
- тип файла (обычного типа, каталог, спец.файл, конвейер, символьная связь)
- число ссылок на данный индексный дескриптор, равный количеству псевдонимов файла;
- адреса дисковых блоков, содержащих данные файла
- размер файла

**Обычные файлы Linux** представляют собой неструктурированные потоки байтов, поэтому их тип указывается только с той целью, чтобы система могла отличать файлы данных от каталогов и специализированных файлов.

**Для специальных файлов устройств** дисковые блоки не выделяются, а их i-узлы идентифицируют соответствующие им устройства.

Физические адреса блоков хранятся в виде массива из 13 элементов.

- Первые 10 элементов адресуют непосредственно блоки хранения данных файла.
- 11-й элемент адресует блок, в свою очередь содержащий адреса блоков хранения данных
- 12-й элемент указывает на дисковый блок, также хранящий адреса блоков, каждый из которых адресует блок хранения данных файла.
- 13-й элемент используется для тройной косвенной адресации, ко-

гда для нахождения адреса блока хранения данных используется три дополнительных блока

- При размере блока 1024 б и файлах до 10 Кб используется прямая индексация, обеспечивающая максимальную производительность
- Для файлов до 266 Кб( 10 Кб + 256 Х 1-24) достаточно простой косвенной адресации
- При использовании тройной косвенной адресации можно обеспечить доступ к 16 777 216 блокам(256 Х 256 Х 256 )

Размер дискового блока в разных системах UNIX, равно как и количество непосредственных и косвенных указателей, может быть различным, но общая структура метаданных одинакова, она позволяет работать с файлами большого размера при фиксированном размере i-узлов.

Хранение всей этой информации в единой таблице i-узлов упрощает работу с файлами, а также процесс реализации алгоритмов для проверки согласованности файловой системы.

**Однако у него имеется и существенный недостаток: при повреждении части поверхности диска хранящаяся в таблице информация может быть утеряна. Поэтому обычно таблицу реплицируют.**

**Файлы каталогов**

Запись о файле в каталоге состоит из двух полей: **символьного имени** и **номера индексного дескриптора**.

ФС не накладывает особых ограничений на размер корневого каталога, т.к. он расположен в области данных и может увеличиваться как обычный файл

Каждый файл каталога соответствует определенному каталогу файловой системы и содержит перечень всех его элементов с указанием номеров соответствующих им i-узлов.

Данная структура файловой системы обеспечивает возможность доступа к файлу по **нескольким именам**, поскольку каждому i-узлу могут соответствовать два и более элемента каталога. Таким образом, один и тот же файл может быть доступен из разных мест пространства имен.

## *2.3 Системные структуры данных в основной памяти и на диске*

Во время работы системы информация о файлах, открытых ее процессами, хранится в основной памяти.

У каждого процесса имеется **таблица открытых файлов**, в которой указаны системные идентификаторы файлов, соответствующие их дескрипторам, полученным процессом в результате системных вызовов.

Поскольку управление вводом-выводом осуществляется посредством интерфейса файловой системы, здесь же представлены и терминальные устройства.

В еще одной, **общесистемной, таблице открытых файлов** содержится как минимум по одной строке для каждого открытого файла, где указаны файловая система, и i-узел, а также текущее смещение в файле.

В **таблице активных i-узлов** содержатся копии i-узлов всех открытых файлов.

Когда файл закрывается, соответствующая ему информация записывается на диск (если в нее были внесены какие-либо изменения). В процессе использования файла эта информация тоже может периодически записываться на диск.

Благодаря тому, что значение смещения в файле хранится в централизованной таблице, такая информация может совместно использоваться несколькими процессами, и когда один из процессов, скажем, выполняет запись, другие видят, как изменяется текущая позиция в файле.

Когда разные процессы открывают файл независимо, им соответствуют различные записи в системной таблице, но, если родительский процесс создает

дочерний, по умолчанию у них будет одна запись, и они смогут совместно использовать указатель текущей позиции в файле.

В любом случае у файла, с которым работают процессы, i-узел всегда один. Классическая Linux не требует установки монопольной блокировки для записи: она предполагает, что связанные друг с другом процессы будут тесно взаимодействовать.

Кроме того, в памяти содержится копия суперблока для каждой монтированной файловой системы, применяемая с целью ускорения доступа к начальным элементам списка свободных блоков и i-узлов.

**Схема считывания файла.**

Обращение к библиотечной процедуре для запуска системного вызова read :

**n = read(fd. buffer, nbytes):**

Когда ядро получает управление, ему подаются только эти три параметра. Все остальные необходимые данные оно может получить из внутренних таблиц, относящихся к пользователю. Одной из таких таблиц является **массив дескрипторов файла**. Он проиндексирован по номерам дескрипторов файла и содержит по одной записи для каждого открытого файла (до некоторого максимума, как правило, около 20 файлов).

По дескриптору файла файловая система должна найти **i-узел** соответствующего файла.

По таблице открытых файлов, между таблицей дескрипторов файлов **и** таблицей i-узлов, и хранить указатели в файле, а также бит чтения/записи в ней.

На рисунке родительский процесс представляет собой оболочку, а дочерний процесс сначала является процессом ***p1***, а затем процессом ***p2***. Когда оболочка создает процесс ***p1***, его пользовательская структура (включая таблицу дескрипторов файлов) представляет собой точную копию такой же структуры оболочки, поэтому обе они содержат указатели на одну и ту же таблицу открытых файлов.

Когда процесс ***p1*** завершает свою работу, таблица дескрипторов файлов оболочки продолжает указывать на таблицу открытых файлов, в которой содержится позиция в файле процесса ***p1***.

Когда теперь оболочка создает процесс ***p2,*** новый дочерний процесс автоматически наследует позицию в файле. При этом ни сам новый процесс, ни оболочка даже не должны знать текущее значение этой позиции.

Однако если какой-нибудь посторонний процесс откроет файл, он получит свою собственную запись в таблице открытых файлов со своей позицией в файле, что как раз и нужно.

Таким образом, задача таблицы открытых файлов заключается в том, чтобы позволить родительскому и дочернему процессам совместно использовать один указатель в файле, но для посторонних процессов выделять отдельные указатели.