Разделы носителя информации (partitions) в UNIX

Физические носители информации – магнитные или оптические диски, ленты и т. д., использующиеся как физическая основа для хранения файлов, в операционных системах принято логически делить на *разделы (partitions)* или *логические диски*. Причем слово «делить» не следует понимать буквально, в некоторых системах несколько физических дисков могут быть объединены в один раздел.

В операционной системе UNIX такое объединяющее разделение, как правило, невозможно. В UNIX физический носитель информации обычно представляет собой один раздел или несколько разделов. В большинстве случаев разбиение на разделы производится линейно, хотя некоторые варианты UNIX могут допускать некое подобие древовидного разбиения (Solaris). Количество разделов и их размеры определяются при форматировании диска. Поскольку форматирование диска относится к области администрирования операционных систем, оно в нашем курсе рассматриваться не будет.

Наличие нескольких разделов на диске может определяться требованиями операционной системы или пожеланиями пользователя. Допустим, пользователь хочет разместить на одном жестком диске несколько операционных систем с возможностью попеременной работы в них, тогда он размещает каждую операционную систему в своём разделе. Или другая ситуация: необходимость работы с несколькими видами файловых систем. Под каждый тип файловой системы выделяется отдельный логический диск. Третий вариант – это разбиение диска на разделы для размещения в разных разделах различных категорий файлов. Скажем, в одном разделе помещаются все системные файлы, а в другом разделе – все пользовательские файлы.

Для простоты далее в этих семинарах будем полагать, что у нас имеется только один раздел и, следовательно, одна файловая система. Вопросы взаимного сосуществования нескольких файловых систем в рамках одной операционной системы, к сожалению, выходят за рамки наших практических занятий.

Логическая структура файловой системы и типы файлов в UNIX

Мы не будем давать здесь определение файла, полагая, что интуитивное представление о файлах у вас имеется, а на лекциях вы получили понятие об абстрактных файлах, логических файлах и физических файлах.

В материалах семинаров 1–2 упрощенно говорилось о том, что файлы могут объединяться в директории, и что файлы и директории организованы в древовидную структуру. На нынешнем уровне знаний мы можем сформулировать это более аккуратно. В операционной системе UNIX существуют файлы нескольких типов, а именно:

- обычные или регулярные файлы;
- директории или каталоги;
- специальные файлы связи (link);
- файлы типа FIFO или именованные pip'ы;
- специальные файлы устройств (разбиваются на два подтипа: файлы символьных и блочных устройств);
- сокеты (sockets).

Что такое регулярные файлы и директории, вам должно быть хорошо известно из личного опыта. О способах их отображения в физическое дисковое пространство речь пойдет чуть позже. С файлами типа FIFO вы познакомились на семинарах 5-6, когда рассматривалась работа с именованными pip'ами. С файлами типа «связь» вы встретитесь позже на этом семинаре. Специальные файлы устройств используются в подсистеме ввода-вывода операционной системы UNIX, о них мы поговорим в следующей теме семинаров.

Под файлами типа “сокет” подразумеваются Unix Domain сокеты, они же локальные сокеты IPC. Они никак не связаны с сетевыми сокетами, которые мы рассмотрим в конце курса, за исключением общего интерфейса работы с ними. Несмотря на то, что работа с сетевым сокетом ведется через файловый дескриптор, большинство реализаций сетевых сокетов никак не связаны с файлами в файловой системе.

В целостной логической файловой системе файлы всех этих типов логически объединены в достаточно сложный граф с однонаправленными ребрами, существенно отличающийся от древовидной структуры, рассмотренной на семинарах 1-2.

Узлами в этом графе по-прежнему выступают файлы, а вот однонаправленные рёбра бывают двух видов, которые для удобства дальнейшего рассмотрения мы будем называть красными и чёрными. Соответственно, можно будет говорить о полном графе логической файловой системы, включающем все рёбра, и о чёрном графе, исключив из рассмотрения красные рёбра (см. рис.1).

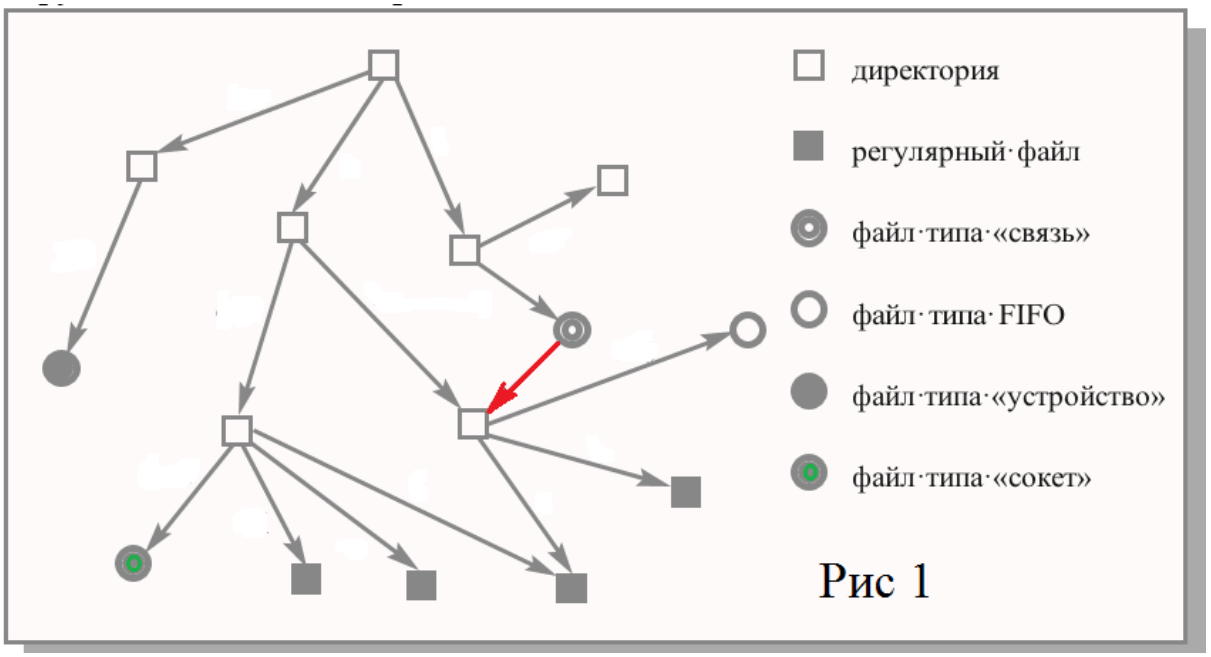

Рис 1

Чёрный граф получается из дерева в результате сращивания вместе нескольких терминальных узлов дерева (листьев), т.е. узлов, из которых не выходит ни одного чёрного ребра. В нетерминальных узлах такого чёрного графа (т.е. в узлах, из которых выходят рёбра) могут располагаться только файлы типа «директория». В терминальных узлах чёрного графа могут располагаться файлы любых типов. В узел, соответствующий фалу типа «директория», не может входить более одного чёрного ребра.

Чёрные рёбра — это ничто иное, как отношение «содержит файл», т.е. файл, в который входит ребро, непосредственно находится в директории, из которой ребро выходит. Обратите внимание, что файл, не являющийся директорией, может непосредственно входить в несколько директорий. Как это получается и зачем это нужно мы обсудим несколько позже.

Красные рёбра появляются в графе логической файловой системы при создании файлов типа «связь». В графе целостной логической файловой системы из каждого файла типа «связь» выходит ровно одно красное ребро, которое может вести к файлу произвольного типа.

В отличие от древовидной логической структуры набора файлов, где имена файлов связывались с узлами дерева, в таком графе имя файла связывается не с узлом, соответствующим файлу, а с входящим в него чёрным ребром — это имя, под которым файл входит в директорию. Красные рёбра, выходящие из узлов, соответствующих файлам типа «связь», являются неименованными (см. рис.2).

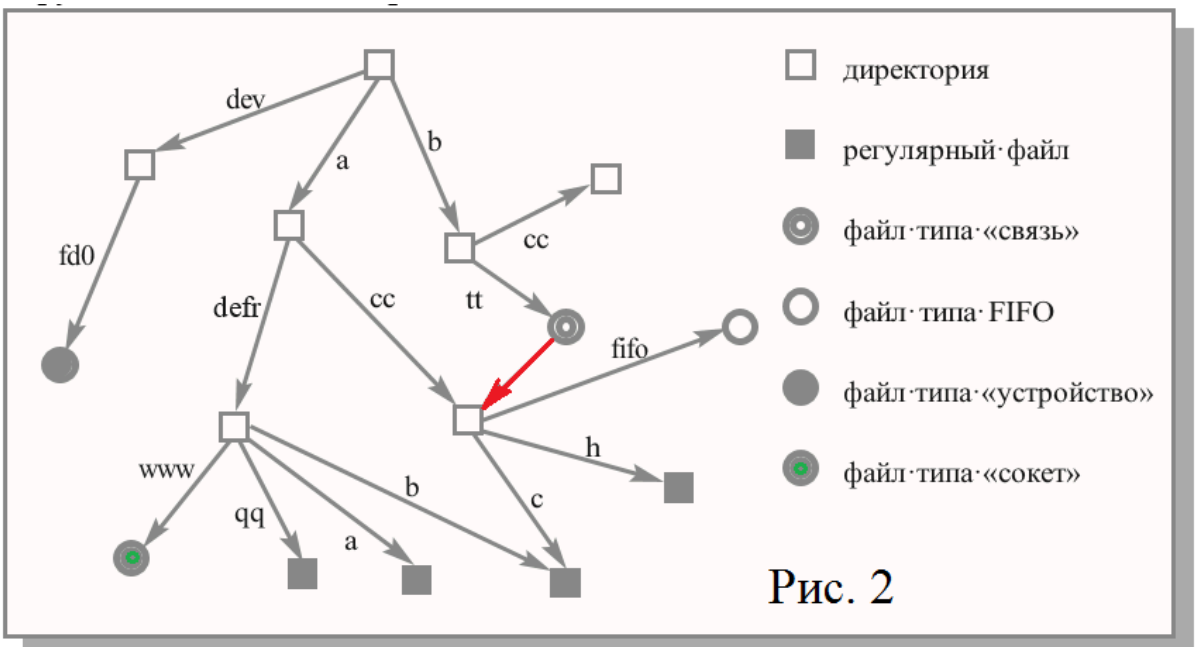

Рис. 2

Правила построения имён чёрных рёбер (имён файлов в директории) рассматривались на семинарах 1–2. В качестве полного имени файла может использоваться любое имя, получающееся при прохождении по рёбрам любого цвета от корневого узла графа (т.е. узла, в который не входит ни одно ребро) до узла, соответствующего этому файлу, по любому пути с помощью следующего алгоритма:

1. Если интересующему нас файлу соответствует корневой узел, то файл имеет имя «/».
2. Иначе берём первое именованное (чёрное) ребро в пути и записываем его имя, которому предваряем символ "/".
3. Для каждого очередного именованного (чёрного) ребра в пути приписываем к уже получившейся строке справа символ "/" и имя соответствующего ребра.
4. Для красных рёбер никаких добавлений в строку не делаем.

Полное имя является уникальным для всей файловой системы и однозначно определяет соответствующий ему файл. Взаимно однозначного соответствия нет. У одного файла может быть несколько полных имён. Аналогичным образом изменяется понятие относительного имени файла.

Организация файла на диске в UNIX на примере файловой системы s5fs. Понятие индексного узла (inode)

Рассмотрим, как организуется на физическом носителе любой файл в UNIX на примере простой файловой системы, впервые появившейся в вариантах операционной системы System V и носящей поэтому название s5fs (**S**ystem **V** **f**ile **s**ystem).

Всё дисковое пространство раздела в файловой системе s5fs логически разделяется на две части: *заголовок раздела* и *логические блоки данных*. Заголовок раздела содержит служебную информацию, необходимую для работы файловой

системы, и обычно располагается в самом начале раздела. Логические блоки хранят собственно содержательную информацию файлов и часть информации о размещении файлов на диске (т. е. какие логические блоки и в каком порядке содержат информацию, записанную в файл).

Для размещения любого файла на диске используется метод *индексных узлов* (inode – от **i**ndex **node**), который был подробно изложен на лекции и на котором здесь мы останавливаться не будем. Индексный узел содержит атрибуты файла и оставшуюся часть информации о его размещении на диске. Необходимо, однако, отметить, что такие типы файлов, как «связь», «сокет», «устройство», «FIFO» не занимают на диске никакого иного места, кроме индексного узла (им не выделяется логических блоков). Всё необходимое для работы с этими типами файлов содержится в их атрибутах.

Перечислим часть атрибутов файлов, хранящихся в индексном узле и свойственных большинству типов файлов. К таким атрибутам относятся:

- Тип файла и права различных категорий пользователей для доступа к нему.
- Идентификаторы владельца-пользователя и владельца-группы.
- Размер файла в байтах (только для регулярных файлов, директорий и файлов типа «связь»).
- Время последнего доступа к файлу.
- Время последней модификации файла.
- Время последней модификации самого индексного узла.

Существует ещё один атрибут, о котором мы поговорим позже, когда будем рассматривать операцию связывания файлов.

Количество индексных узлов в разделе является постоянной величиной, определяемой на этапе генерации файловой системы. Все индексные узлы системы организованы в виде массива, хранящегося в заголовке раздела. Каждому файлу соответствует только один элемент этого массива и, наоборот, каждому непустому элементу этого массива соответствует только один файл. Таким образом, каждый файл на диске может быть однозначно идентифицирован номером своего индексного узла (его индексом в массиве).

На языке представления логической организации файловой системы в виде графа это означает, что каждому узлу графа соответствует только один номер индексного узла, и никакие два узла графа не могут иметь одинаковые номера.

Надо отметить, что свойством уникальности номеров индексных узлов, идентифицирующих файлы, вы уже неявно пользовались при работе с именованными pip'ами (семинар 5) и средствами System V IPC (семинар 7). Для именованного pip'а именно номер индексного узла, соответствующего файлу с типом FIFO, является той самой точкой привязки, пользуясь которой, неродственные процессы могут получить данные о расположении pip'а в адресном пространстве ядра и его состоянии и связаться друг с другом. Для средств System V IPC при генерации IPC-ключа с помощью функции ftok() в действительности используется не имя заданного файла, а номер

соответствующего ему индексного узла, который по определённому алгоритму объединяется с номером экземпляра средства связи.

Организация директорий (каталогов) в UNIX

Содержимое регулярных файлов (информация, находящаяся в них, и способ её организации) всецело определяется программистом, создающим файл. В отличие от регулярных, остальные типы файлов, содержащих данные, т. е. директории и связи, имеют жёстко заданную структуру и содержание, определяемые типом используемой файловой системы.

Основным содержимым файлов типа «директория», если говорить на пользовательском языке, являются имена файлов, лежащих непосредственно в этих директориях, и соответствующие им номера индексных узлов. Более строго, в терминах графового представления, содержимое директорий представляет собой список записей, где находятся имена рёбер, выходящих из узлов, соответствующих директориям, вместе с индексными номерами узлов, к которым они ведут. При этом первая запись в содержимом директории дополнительно содержит ссылку на саму данную директорию под именем ".", а вторая запись — ссылку на родительский каталог (если родительский каталог существует), т.е. на узел графа, из которого выходит единственное именованное ребро, ведущее к текущему узлу, под именем "..".

Операции связывания файлов. Команда ln, системные вызовы link() и symlink()

С операциями, позволяющими изменять логическую структуру файловой системы, такими как создание файла, вы уже сталкивались в этом разделе. Однако операции создания связи служат для проведения новых именованных (чёрных) рёбер в уже существующей структуре без добавления новых узлов или для проведения пути к уже существующему узлу через чёрное (именованное) ребро к новому файлу типа «связь» и неименованное (красное) ребро.

Допустим, что несколько программистов совместно ведут работу над одним и тем же проектом. Файлы, относящиеся к этому проекту, вполне естественно могут быть выделены в отдельную директорию так, чтобы не смешиваться с файлами других пользователей и другими файлами программистов, участвующих в проекте. Для удобства каждый из разработчиков, конечно, хотел бы, чтобы эти файлы находились в его собственной директории. Этого можно было бы добиться, копируя по мере изменения новые версии соответствующих файлов из директории одного исполнителя в директорию другого исполнителя. Однако тогда, во-первых, возникнет ненужное дублирование информации на диске. Во-вторых, появится необходимость решения тяжёлой задачи: синхронизации обновления всех копий этих файлов новыми версиями.

Существует другое решение проблемы. Достаточно разрешить файлам иметь несколько имён. Тогда одному физическому экземпляру данных на диске могут

соответствовать различные имена файла, находящиеся в одной или в разных директориях. Подобная операция присвоения нового имени файлу (без уничтожения ранее существовавшего имени) получила название операции создания связи.

В операционной системе UNIX связь может быть создана двумя различными способами.

Первый способ, наиболее точно следующий описанной выше процедуре, получил название способа создания *жёсткой связи (hard link)*. С точки зрения логической структуры файловой системы этому способу соответствует проведение нового чёрного именованного ребра из узла, соответствующего некоторой директории, к узлу, соответствующему некоторому существующему файлу, получающему дополнительное имя. С точки зрения структур данных, описывающих строение файловой системы, в эту директорию добавляется запись, содержащая дополнительное имя файла и номер его индексного узла (уже существующий!). При таком подходе и новое имя файла, и его старое имя или имена абсолютно равноправны для операционной системы и могут взаимозаменяемо использоваться для осуществления всех операций.

Использование жёстких связей приводит к возникновению двух проблем.

Первая проблема связана с операцией удаления файла. Если мы хотим удалить файл из некоторой директории, то после удаления из её содержимого записи, соответствующей этому файлу, мы не можем освободить логические блоки, занимаемые файлом, и его индексный узел, не убедившись, что у файла нет дополнительных имён (к его индексному узлу не ведут ссылки из других директорий), иначе мы нарушим целостность файловой системы. Для решения этой проблемы файлы получают дополнительный атрибут – счётчик жёстких связей (или чёрных рёбер), ведущих к ним, который, как и другие атрибуты, располагается в их индексных узлах. При создании файла этот счётчик получает значение 1. При создании каждой новой жёсткой связи, ведущей к файлу, он увеличивается на 1. Когда мы удаляем файл из некоторой директории, то из её содержимого удаляется запись об этом файле, и счётчик жёстких связей уменьшается на 1. Если его значение становится равным 0, происходит освобождение логических блоков и индексного узла, выделенных этому файлу.

Вторая проблема связана с опасностью превращения чёрного графа логической структуры файловой системы из ациклического в циклический (что может привести к зацикливанию ряда системных утилит) и с возможной неопределенностью толкования записи с именем ".." в содержимом директорий. Для их предотвращения во всех существующих вариантах операционной системы UNIX запрещено создание жёстких связей, ведущих к уже существующим директориям. Поэтому мы и говорили о том, что в узел, соответствующий файлу типа «директория», не может вести более одного именованного (чёрного) ребра.

Для создания жёстких связей применяются команда операционной системы ln без опций и системный вызов *link()*. **Надо отметить, что системный вызов *link()* является одним из немногих системных вызовов, совершающих операции над файлами, которые не требуют предварительного открытия файла, поскольку он подразумевает выполнение единичного действия только над содержимым индексного узла, выделенного связываемому файлу, и директорией, в которой создается новая жёсткая связь.**

```
$ ln TARGET LINK_NAME
```

Второй способ создания связи получил название способа создания *мягкой (soft)* или *символической (symbolic) связи (link)*. В то время как жёсткая связь файлов является аналогом использования прямых ссылок (указателей) в современных языках программирования, символическая связь, до некоторой степени, напоминает косвенные ссылки (указатель на указатель). При создании мягкой связи с именем **symlink** из некоторой директории к файлу, заданному полным или относительным именем **linkpath**, в этой директории действительно **создается новый файл типа «связь»** с именем **symlink** со своими собственными индексным узлом и логическими блоками. При тщательном рассмотрении можно обнаружить, что всё его содержимое составляет только символьная запись имени **linkpath**. Операция открытия файла типа «связь» устроена таким образом, что в действительности открывается не сам этот файл, а тот файл, чьё имя содержится в нём (при необходимости рекурсивно!). Поэтому операции над файлами, требующие предварительного открытия файла (как, впрочем, и большинство команд операционной системы, совершающих действия над файлами, где операция открытия файла присутствует, но скрыта от пользователя), в реальности будут совершаться не над файлом типа «связь», а над тем файлом, имя которого содержится в нём (или над тем файлом, который, в конце концов, откроется при рекурсивных ссылках). Отсюда, в частности, следует, что попытки прочитать **реальное** содержимое файлов типа «связь» с помощью системного вызова *read()* обречены на неудачу. Как видно, создание мягкой связи, с точки зрения изменения логической структуры файловой системы, эквивалентно опосредованному проведению пути к уже существующему узлу через именованное чёрное ребро к новому файлу типа «связь» и неименованное красное ребро.

Создание символической связи не приводит к проблеме, связанной с удалением файлов. Если файл, на который ссылается мягкая связь, удаляется с физического носителя, то попытка открытия файла мягкой связи (а, следовательно, и удаленного файла) приведёт к ошибке «Файла с таким именем не существует», которая может быть аккуратно обработана приложением. Таким образом, удаление связанного объекта, как упоминалось ранее, лишь отчасти и не фатально нарушит целостность файловой системы.

Неаккуратное применение символических связей пользователями операционной системы может привести к превращению логической структуры файловой системы из ациклического графа в циклический граф. Это, конечно, нежелательно, но не носит столь разрушительного характера, как циклы, которые могли бы быть созданы жёсткой

связью, если бы не был введён запрет на образование жёстких связей к директориям. Поскольку мягкие связи (наличие красных рёбер) принципиально отличаются от жёстких связей и связей, возникающих между директорией и файлом при его создании, — чёрных рёбер — мягкая связь легко может быть идентифицирована операционной системой или программой пользователя. Для предотвращения зацикливания программ, выполняющих операции над файлами, обычно ограничивается глубина рекурсии по прохождению красных рёбер. Превышение этой глубины приводит к возникновению ошибки «Слишком много мягких связей», которая может быть легко обработана приложением. Поэтому ограничения на тип файлов, к которым может вести мягкая связь, в операционной системе UNIX не вводятся.

Для создания мягких связей применяются уже знакомая вам команда операционной системы ln с опцией **-s** и системный вызов *symlink()*. **Надо отметить, что системный вызов *symlink()* также не требует предварительного открытия связываемого файла, поскольку он вообще не проверяет его существования.**

```
$ ln -s TARGET LINK_NAME
```

Задачи на семинар

***Задача 1 (15 баллов)**:*

Напишите программу, определяющую глубину рекурсии для символьных связей при открытии файлов. При запуске программа должна создавать некоторый регулярный файл, допустим, с именем «а». Затем она должна создать символьную связь с именем, предположим, «аа» на файл «а» и попытаться открыть файл «аа». Если это удалось, то должна создаваться символьная связь с именем, скажем, «аб» на файл «аа» и производиться попытка открытия файла «аб». И так далее, пока открыть файл вам не удастся. Это и будет означать, что вы достигли глубины рекурсии (количества допустимых символьных связей) при открытии файла. Значение глубины рекурсии должно быть выведено на экран.

***Подзадача 1 (2 бонусных балла)**:*

Создавайте все файлы в новой отдельной директории, чтобы в случае ошибки можно было легко удалить большое количество образовавшихся файлов. Создавать директорию автоматически можно при помощи функции mkdir(), объявленной в заголовочном файла <fcntl.h>, автоматически удалять создаваемые ссылки не требуется.

***Примечания**:*

Полученный ответ должен быть точным - максимально допустимое количество символьных связей в цепочке.

Не забывайте проверять возвращаемые значения из системных вызовов и своевременное закрывать файлы!