Семинар 10. Файловые системы. Часть 1.

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. Об этом подробнее вы узнаете на лекции.

В операционной системе UNIX такое объединяющее разделение невозможно. В UNIX физический носитель информации может представлять собой один раздел или несколько разделов. В большинстве случаев разделение 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 не будет рассматриваться в нашем курсе.

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 своем собственном разделе. Другая ситуации, приводящая к желательности наличия нескольких разделов внутри одной операционной системы, - это 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. Примером операционной системы, внутренние требования которой приводят к появлению нескольких разделов на диске, могут служить ранние версии MS-DOS, для которых максимальный размер логического диска не мог превышать 32 MB.

Для простоты далее на 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 мы затронем на следующих семинарах перед обсуждением реализации подсистемы ввода-вывода.

Логическая структура файловой системы и типы файлов в UNIX. Мы не будем давать здесь определение понятию файла, считая что интуитивное представление о файлах у всех вас имеется, а на лекциях вы получили понятие о файлах, как об

именованных абстрактных объектах, обладающих определенными свойствами. При этом в пространстве имен файлов одному файлу могут соответствовать несколько имен.

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

- обычные или регулярные файлы;
- директории или каталоги;
- файлы типа FIFO или именованые pip'ы;
- специальные файлы устройств;
- сокеты (sockets);
- специальные файлы связи (link).

Что такое регулярные файлы и директории вам должно быть хорошо известно из личного опыта и из лекций. О способах их отображения в дисковое пространство мы поговорим чуть позже. С файлами типа FIFO мы познакомились на семинаре 5, когда говорили о работе с именованными pip'ами. С файлами типа "связь" мы встретимся на этом семинаре, когда будем обсуждать операции над файлами и соответствующие им системные вызовы. Специальные файлы устройств и файлы типа "сокет" будут рассмотрены на следующих семинарам.

Файлы всех этих типов логически объединены в ациклический граф с однонаправленными ребрами, получающийся из дерева в результате сращивания вместе нескольких терминальных узлов дерева или нескольких его нетерминальных узлов таким образом, чтобы полученный граф не содержал циклов. В нетерминальных узлах такого ациклического графа (т.е. в узлах, из которых выходят ребра) могут располагаться только файлы типов "директория" и "связь". Причем из узла, в котором располагается файл типа "связь" может выходить только ровно одно ребро. В терминальных узлах этого ациклического графа (т.е. в узлах, из которых не выходит ребер) могут располагаться файлы любых типов, хотя присутствие в теминальном узле файла типа "связь" обычно говорит о некотором нарушении целостности файловой системы.

В отличие от древовидной структуры набора файлов, где имена файлов связывались с узлами дерева, в таком ациклическом графе имя файла связывается не с узлом, соответствующим файлу, а с входящим в него ребром. Ребра, выходящие из узлов, соответствующих файлам типа "связь" являются неименованными. Надо отметить, что во всех практически существующих реализациях UNIX-подобных систем в узел графа,

соответствующий файлу типа "директория", не может входить более одного именованого ребра, хотя стандарт на операционную систему UNIX и не запрещает этого.

В качестве полного имени файла может использоваться любое имя, получающееся при прохождении по ребрам от корневого узла графа (т.е. узла, в который не входит ни одно ребро) до узла, соотвествующего этому файлу, по любому пути с помощью следующего алгоритма:

1. Если интересующему нас файлу соответствует корневой узел, то файл имеет имя "/".
2. Берем первое именованое ребро в пути и записываем его имя, которому предваряем символ "/".
3. Для каждого очередного именованного ребра в пути приписываем к уже получившейся строке справа символ "/" и имя соответствующего ребра.

Полное имя является уникальным для всей файловой системы и однозначно определяет соответствующий ему файл.

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 (System V file system).

Все 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 index node), который был подробно изложен на лекции и на котором здесь мы останавливаться не будем. Индексный узел содержит атрибуты файла и оставшуюся часть информации об 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). Все необходимое для работы с этими типами файлов содержится в их атрибутах.

Давайте перечислим часть атрибутов файлов, хранящихся в индексном узле и свойственных для большинства 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 еще 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 графового представления логической организации файловой системы это означает, что каждому узлу графа соотвествует ровно один номер индексного узла и никакие два узла графа не могут иметь одинаковые номера.

Надо отметить, что свойством уникальности номеров индексных узлов, идентифицирующих файлы, мы уже неявно пользовались при работе с именованными pip'ами и средствами System V IPC. Для именованного pip'а именно номер индексного узла, соответствующего файлу с типом FIFO, является той самой точкой привязки, пользуясь которой различные неродственные процессы могут получить данные об 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 дескриптора, который по определенному алгоритму объединяется с номером экземпляра средства связи.

Организация директорий (каталогов) в UNIX. Содержимое регулярных файлов (информация, находящаяся в них, и способ ее организации) всецело определяется программистом, создающим файл. В отличие от регулярных, остальные типы файлов, содержащих данные, т. е. директории и связи, имеют жестко 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 имена ребер, выходящих из узлов, соотвествующих директориям, вместе с индексными номерами узлов, к которым они ведут.

В файловой системе s5fs пространство имен файлов (ребер) содержит имена, не превышающие 14-и символов, а максимальное количество inode в одном разделе файловой системы не может превышать значения 65535. Эти ограничения не позволяют придавать файлам осмысленные имена и приводят к необходимости разбиения больших жестких дисков на несколько разделов. Зато они помогают упростить структуру хранения информации в директории. Все содержимое директории представляет собой таблицу, в которой каждый элемент имеет фиксированный размер в 16 байт. Из них 14 байт отводится под имя соответствующего файла (ребра), а 2 байта - под номер его индексного узла. При этом первый элемент таблицы дополнительно содержит ссылку на саму данную директорию под именем ".", а второй элемент таблицы - ссылку на родительский каталог, т.е. на узел графа, из которого выходит единственное именованное ребро, ведущее к текущему узлу, (если такой существует) под именем "..".

В более современной файловой системе FFS (Fast File System) размерность пространства имен файлов (ребер) увеличена до 255 символов. Это позволило использовать практически любые мыслимые имена для файлов (вряд ли найдется программист, которому будет не лень набирать для имени более 255 символов), но потребовало изменить структуру каталога (чтобы уменьшить его размеры и не хранить пустые байты). В системе FFS каталог представляет собой таблицу из записей переменной длины. В структуру каждой записи входят: номер индексного узла, длина этой записи, длина имени файла и собственно его имя. Две первых записи в каталоге, как и в s5fs, по прежнему адресуют саму данную директорию и ее родительский каталог.

Понятие суперблока. Мы с вами уже коснулись содержимого заголовка раздела, когда говорили о массиве индексных узлов файловой системы. Оставшуюся часть заголовка в s5fs принято называть суперблоком. Суперблок хранит информацию, необходимую для правильного функционирования файловой системы в целом. В нем содержатся, в частности, следующие данные:

- Тип файловой системы.
- Флаги состояния файловой системы.
- Размер логического блока в байтах (обычно кратен 512 байтам).

- Размер файловой системы в логических блоках (включая сам суперблок и массив inode).
- Размер массива индексных узлов (т.е. сколько файлов может быть размещено в файловой системе).
- Число свободных индексных узлов (сколько файлов еще можно создать).
- Число свободных блоков для размещения данных.
- Часть списка свободных индексных узлов.
- Часть списка свободных блоков для размещения данных.

В некоторых модификациях файловой системы s5fs последние два списка выносятся за пределы суперблока, но остаются в заголовке раздела. При первом же обращении к файловой системе суперблок обычно целиком считывается в адресное пространство ядра для ускорения последующих обращений. Поскольку количество логических блоков и индексных узлов в файловой системе может быть весьма большим, нецелесообразно хранить списки свободных блоков и узлов в суперблоке полностью. При работе с индексными узлами часть списка свободных узлов, находящаяся в суперблоке, постепенно исчерпывается. Когда список близок к исчерпанию, операционная система сканирует массив индексных узлов и заново заполняет список.

Часть списка свободных логических блоков, лежащая в суперблоке, содержит ссылку на его продолжение, расположенное где-либо в блоках данных. Когда эта часть оказывается использованной, операционная система загружает на освободившееся место продолжение списка, а блок использовавшийся для его хранения переводится в разряд свободных.

Операции над файлами и директориями. Хотя с точки зрения пользователя рассмотрение операций над файлами и директориями представляется достаточно простым и сводится к перечислению ряда системных вызовов и команд операционной системы, попытка систематического подхода к набору операций вызывает определенные затруднения. Далее речь у нас пойдет в основном о регулярных файлах и файлах типа "директория".

Существуют два основных вида файлов, различающихся по методу доступа: файлы последовательного доступа и файлы прямого доступа. Если рассматривать файлы прямого и последовательного доступа как абстрактные типы данных, то они представляются как нечто, содержащее информацию, над которым можно совершать следующие операции:

- Для последовательного доступа: чтение очередной порции данных (read), запись очередной порции данных (write) и позиционирование на начале файла (rewind).

- Для прямого доступа: чтение очередной порции данных (read), запись очередной порции данных (write) и позиционирование на требуемой части данных (seek).

Работа с объектами этих абстрактных типов подразумевает наличие еще двух необходимых операций: создание нового объекта (new) и уничтожение уже существующего объекта (free).

Расширение математической модели файла за счет добавления к хранимой информации атрибутов, присущих файлу, (права доступа, учетные данные) влечет за собой появление еще двух операций: прочитать атрибуты (get attribute) и установить их значения (set attribute).

Наделение файлов какой-либо внутренней структурой (как у файла типа "директория") или наложение на набор файлов внешней логической структуры (объединение в ациклический направленный граф) приводит к появлению других наборов операций, составляющих интерфейс работы с файлами, которые тем не менее будут являться комбинациями вышеперечисленных базовых операций.

Для директории, например, такой набор операций, определяемый ее внутренним строением, может выглядеть так: операции new, free, set attribute и get attribute остаются без изменений, а операции read, write и rewind (seek) заменяются на более высокоуровневые:

- прочитать запись, соответствующую имени файла, - get record;
- добавить новую запись - add record;
- удалить запись, соответствующую имени файла, - delete record;

Неполный набор операций над файлами, связанный с их логическим объединением в директорную структуру, будет выглядеть следующим образом:

- Операции для работы с атрибутами файлов - get attribute, set attribute.
- Операции для работы с содержимым файлов - read, write, rewind(seek) для регулярных файлов и get record, add record, delete record для директорий.
- Операция создания регулярного файла в некоторой директории (создание нового узла графа и добавление в граф нового именованного ребра, ведущего в этот узел из некоторого узла, соответствующего директории) - create. Эту операцию можно рассматривать как суперпозицию двух операций: базовой операции new для регулярного файла и add record для соответствующей директории.
- Операция создания поддиректории в некоторой директории - make directory. Эта операция отличается от предыдущей операции create занесением в файл новой директории информации о файлах с именами "." и "..", т.е. по сути дела она есть суперпозиция операции create и двух операций add record.

- Операция создания файла типа "связь" - symbolic link.
- Операция создания файла типа "FIFO" - make FIFO.
- Операция добавления к графу нового именованного ребра, ведущего от узла, соответствующего директории, к узлу, соответствующему любому другому типу файла, - link. Это просто add record с некоторыми ограничениями.
- Операция удаления файла, не являющегося директорией или "связью" (удаление именованного ребра из графа, ведущего к терминальной вершине с одновременным удалением этой вершины, если к ней больше не ведет именованных ребер), - unlink.
- Операция удаления файла типа "связь" (удаление именованного ребра, ведущего к узлу, соответствующему файлу типа "связь", с одновременным удалением этого узла и выходящего из него неименованного ребра, если к этому узлу больше не ведет именованных ребер), - unlink link.
- Операция рекурсивного удаления директории со всеми входящими в нее файлами и поддиректориями - remove directory.
- Операция переименования файла (ребра графа) - rename.
- Операция перемещения файла из одной директории в другую (перемещается точка выхода именованного ребра, которое ведет к узлу, соответствующему данному файлу) - move.

Возможны и другие подобные операции.

Способ реализации файловой системы в реальной операционной системе также может добавлять новые операции. Если часть информации файловой системы или отдельного файла кэшируются в адресном пространстве ядра, то появляются операции синхронизации данных в кэше и на диске для всей системы в целом (sync) и для отдельного файла (sync file).

Естественно, что все перечисленные операции могут быть выполнены процессом только при наличии у него определенных полномочий (прав доступа и т.д.). Для выполнения операций над файлами и директориями операционная система предоставляет процессам интерфейс в виде системных вызовов, библиотечных функций и команд операционной системы. Часть этих системных вызовов, функций и команд мы рассмотрим в следующих разделах.

Системные вызовы и команды для выполнения операций над файлами и директориями. На предыдущих семинарах мы уже говорили о некоторых командах и системных вызовах, позволяющих выполнять операции над файлами в операционной системе UNIX. На семинаре 1 мы рассмотрели ряд команд, позволяющих изменять

атрибуты файла - chmod, chown, chgrp команду копирования файлов и директорий - cp, команду удаления файлов и директорий - rm, команду переименования и перемещения файлов и директорий - mv, команду просмотра содержимого директорий - ls.

На семинаре, посвященном потокам ввода-вывода, мы говорили о хранении информации о файлах внутри адресного пространства процесса с помощью таблицы открытых файлов, о понятии файлового дескриптора, о необходимости введения операций открытия и закрытия файлов (системные вызовы open() и close()) и об операциях чтения и записи (системные вызовы read() и write()). Мы обещали вернуться к более подробному рассмотрению затронутых вопросов на текущих семинарах. Пора выполнять обещанное. Далее в этом разделе, если не будет оговоренно особо, под словом файл будет подразумеваться регулярный файл.

Вся информация об атрибутах файла и его расположении на физическом носителе содержится в соответствующем файлу индексном узле и, возможно, в нескольких логических блоках, связанных с индексным узлом. Для того, чтобы при каждой операции над файлом не считывать эту информацию с физического носителя заново, кажется логичным, считав ее один раз при первом обращении к файлу, хранить ее в адресном пространстве процесса или в части адресного пространства ядра, характеризующей данный процесс.

С точки зрения пользовательского процесса каждый файл представляет собой линейный набор байт, снабженный указателем текущей позиции процесса в этом наборе. Все операции чтения из файла и записи в файл производятся в этом наборе с места, на которое показывает указатель текущей позиции. После операции чтения или записи указатель текущей позиции помещается после конца прочитанного или записанного участка файла. Значение этого указателя является динамической характеристикой файла для использующего его процесса и также хранится в PCB.

Некоторые файлы могут использоваться одновременно более чем одним процессом. Для того, чтобы не хранить дублирующуюся информацию об атрибутах файлов и их расположении для каждого процесса отдельно, такие данные обычно размещаются в адресном пространстве ядра операционной системы в единственном экземпляре, а доступ к ним процессы получают только при выполнении соотвествующих системных вызовов для операций над файлами.

Как видим, вся информация о файле, необходимая процессу для работы с ним, может быть разбита на две части: данные, специфичные для этого процесса, - например, указатель текущей позиции, и данные, являющиеся общими для различных процессов, - атрибуты и расположение файла. Естественно, что для хранения этой информации

применяются две различные связанные структуры данных, лежащие, как правило, в адресном пространстве ядра операционной системы, - системная таблица файлов и таблица индексных узлов открытых файлов. Для доступа к этой информации в управляющем блоке процесса заводится таблица открытых файлов, каждый непустой элемент которой содержит ссылку на структуру, хранящую данные о файле, характерные для данного процесса, и из которой, в свою очередь, мы можем по ссылке доступиться к общим данным о файле. Индекс элемента в этой таблице (небольшое целое неотрицательное число) или файловый дескриптор является той величиной, характеризующей файл, которой может оперировать процесс при работе на уровне пользователя. В эту же таблицу открытых файлов помещаются и ссылки на данные, описывающие другие потоки ввода-вывода, такие как pipe и FIFO (об этом мы уже говорили на семинаре 5). Как мы увидим позже (на семинарах, посвященных сетевому программированию), эта же таблица будет использоваться и для размещения ссылок на структуры данных, необходимых для передачи информации от процесса к процессу по сети.

Системный вызов open(). Для выполнения большинства операций над файлами через системные вызовы пользовательский процесс обычно должен указать в качестве одного из параметров системного вызова дескриптор файла, над которым нужно совершить операцию. Поэтому, прежде чем совершать операции, мы должны поместить информацию о файле в наши таблицы файлов и определить соответствующий файловый дескриптор. Для этого, как мы уже говорили, применяется процедура открытия файла, осуществляемая системным вызовом open(). При открытии файла операционная система проверяет, соотвествуют ли права, которые запросил процесс для операций над файлом, правам доступа, установленным для этого файла. В случае соответсвия, она помещает необходимую информацию в системную таблицу файлов и, если этот файл не был ранее открыт другим процессом, в таблицу индексных дескрипторов открытых файлов, устанавливает необходимую связь между всеми тремя таблицами и возвращает на пользовательский уровень дескриптор этого файла.

По сути дела, с помощью операции открытия файла операционная система осуществляет отображение из пространства имен файлов в дисковое пространство файловой системы, подготавливая почву для совершения других операций.

Системный вызов close(). Обратным системным вызовом, по отношению к системному вызову open(), является системный вызов close(), с которым мы тоже уже знакомились. После завершения работы с файлом процесс освобождает выделенные ресурсы операционной системы и, возможно, синхронизирует информацию о файле,

содержащуюся в таблице индексных узлов открытых файлов, с информацией на диске, используя этот системный вызов. Надо отметить, что место в таблице индексных узлов открытых файлов не освобождается по системному вызову close() до тех пор, пока в системе существует хотя бы один процесс, использующий этот файл. Для обеспечения такого поведения в ней для каждого индексного узла заводится счетчик числа открытий, увеличивающийся на 1 при каждом системном вызове open() для данного файла и уменьшающийся на 1 при каждом его закрытии. Очищение элемента таблицы индексных узлов открытых файлов с окончательной синхронизацией данных в памяти и на диске происходит только в том случае, если при очередном закрытии файла этот счетчик становится равным 0.

Операция создания файла. Системный вызов creat(). При обсуждении системного вызова open() мы подробно говорили о его использовании для создания нового файла. Для этих же целей можно использовать системный вызов creat(), являющийся, по существу, урезанным вариантом вызова open() (о значение флага O_TRUNC для системного вызова open() см. чуть ниже).

Операция чтения атрибутов файла. Системные вызовы stat(), fstat() и lstat(). Для чтения всех атрибутов файла в специальную структуру могут применяться системные вызовы stat(), fstat() и lstat(). Разъяснение понятий жесткая и мягкая (символическая) связь, встречающихся в описании системных вызовов, будет дано позже при рассмотрении операций связывания файлов.

Операции изменения атрибутов файла. Большинство операций изменения атрибутов файла обычно выполняется пользователем в интерактивном режиме с помощью команд операционной системы. Мы уже говорили о них на семинарах 1-2 и не будем возвращаться к ним вновь. Отметим только операцию изменения размеров файла, а точнее операцию его обрезания, без изменения всех других атрибутов, кроме, быть может, времен последнего доступа к файлу и его последней модификации. Для того, чтобы уменьшить размеры существующего файла до 0, не затрагивая его остальных характеристик (прав доступа, даты создания, учетной информации и т.д.), можно при открытии файла использовать в комбинации флагов системного вызова open() флаг O_TRUNC. Для изменения размеров файла до любой желаемой величины (даже для его увеличения во многих вариантах UNIX, хотя изначально этого не предусматривалось!) может служить системный вызов ftruncate(). При этом, если размер файла мы уменьшаем, то вся не влезающая в новый размер информация в конце файла будет потеряна, если же размер файла мы увеличиваем, то это будет выглядеть так, как будто мы дополнили его до недостающего размера нулевыми битами.

Операции чтения из файла и записи в файл. Для операций чтения из файла и записи в файл применяются системные вызовы read() и write(), которые мы уже обсуждали ранее. Надо отметить, что их поведение при работе с файлами имеет определенные особенности, связанные с понятием указателя текущей позиции в файле.

Операция изменения указателя текущей позиции. Системный вызов lseek(). С точки зрения процесса все регулярные файлы являются файлами прямого доступа. В любой момент процесс может изменить положение указателя текущей позиции в открытом файле с помощью системного вызова lseek().

Особенностью этого системного вызова является возможность помещения указателя текущей позиции в файле за конец файла (т.е. возможность установления значения указателя большего, чем длина файла). При любой последующей операции записи в таком положении указателя файл будет выглядеть так, как будто возникший промежуток от конца файла до текущей позиции, где начинается запись, был заполнен нулевыми битами. Если операции записи в таком положении указателя не производится, то никакого изменения файла, связанного с необычным значением указателя, не произойдет (например, операция чтения будет возвращать нулевое значение для количества прочитанных байтов).

Операция добавления информации в файл. Флаг O_APPEND. Хотя эта операция по сути дела является комбинацией двух уже рассмотренных операций, мы считаем нужным упомянуть ее особо.

Если открытие файла системным вызовом open() производилось с установленным флагом O_APPEND, то любая операция записи в файл будет всегда добавлять новые данные в конец файла, независимо от предыдущего положения указателя текущей позиции (так, как если бы непосредственно перед записью был выполнен вызов lseek() для установки указателя на конец файла).

Операции создания связей. Команда ln, системные вызовы link() и symlink(). С операциями, позволяющими изменять логическую структуру файловой системы, такими как, скажем создание файла, мы уже сталкивались в этом разделе. Однако операции создания связи служат для проведения новых именованных ребер в уже существующей структуре без добавления новых узлов или для опосредованного проведения именованного ребра к уже существующему узлу через файл типа "связь" и неименованное ребро. Такие операции мы с вами до сих пор не рассматривали, поэтому давайте остановимся на них подробнее.

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 работы каждый из разработчиков, конечно, хотел бы, чтобы эти файлы находились в его собственной поддиректории. Подобного можно было бы добиться, копируя по мере изменения новые версии соответствующих файлов из поддиректории одного исполнителя в поддиректорию другого исполнителя. Однако тогда, во-первых, возникнет ненужное дублирование информации на диске, во-вторых, появится необходимость решения тяжелой задачи синхронизации обновления всех копий этих файлов новыми версиями.

Существует другое решение этой проблемы. Достаточно разрешить файлам иметь несколько имен. 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 существовашего 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 жесткой связи (hard link). С точки зрения логической структуры файловой системы этому способу соответствует проведение нового именованного ребра из узла, соответствующего некоторой директории, к узлу, соответствующему файлу любого типа, 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 существующего!). 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 жестких 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 ее содержимого записи, соответствующей этому файлу, мы не можем освободить логические блоки, занимаемые файлом, и его индексный узел, не убедившись, что у файла нет дополнительных имен (к его индексному узлу не ведут ссылки из других директорий), иначе мы нарушим целостность файловой системы. Для решения этой проблемы файлы получают дополнительный атрибут - счетчик жестких связей (или именованых ребер), ведущих к

ним, который, как и другие атрибуты, располагается в их индексных узлах. При создании файла этот счетчик получает значение 1. При создании каждой новой жесткой связи, ведущей к файлу, он увеличивается на 1. Когда мы удаляем файл из некоторой директории, то из ее содержимого удаляется запись об этом файле, и счетчик жестких 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 логической структуры файловой системы из ациклического графа в циклический и с возможной неопределенностью толкования записи с именем ".." в содержимом директорий. Для их предотвращения во всех существующих вариантах операционной системы UNIX запрещено создание жестких связей, ведущих к уже существующим директориям (несмотря на то, что POSIX стандарт для операционной системы UNIX разрешает подобную операцию для суперпользователей). Поэтому мы и говорили о том, что в узел, соответствующий файлу типа "директория", не может вести более одного именованого ребра. (В операционной системе Linux по непонятной причине дополнительно запрещено создание жестких связей, ведущих к специальным файлам устройств.)

Для создания жестких 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.

Второй способ создания связи получил название способа создания мягкой (soft) или символической (symbolic) связи (link). В то время как жесткая 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 все 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, чье имя содержится в нем (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 скрытно от пользователя), в реальности будут совершаться не над файлом типа "связь", а над тем файлом, имя которого содержится в нем (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 именованного ребра к уже существующему узлу через файл типа "связь" и неименованное 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) приведет к ошибке "Файл с таким именем не существует", которая может быть аккуратно обработана приложением, т.е. удаление связанного объекта, как упоминалось ранее, лишь отчасти и нефатально нарушит целостность файловой системы.

Неаккуратное использование 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 жесткой связью, если бы не был введен запрет на образование жестких связей к директориям. Поскольку мягкие связи принципиально отличается от жестких связей и связей, возникающих между директорией и файлом при его создании, 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 мягких связей. 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 нам 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 рассматривает его содержимое.

Операция удаления связей и файлов. Системный вызов unlink(). При рассмотрении операции связывания файлов мы уже почти полностью рассмотрели, как собственно производится операция удаления жестких связей и файлов. При удалении мягкой связи,

т.е. фактически файла типа "связь", все происходит, как и для обычных файлов. Единственным изменением, с точки зрения логической структуры файловой системы, является то, что при действительном удалении узла, соответствующего файлу типа "связь", вместе с ним удаляется и выходящее из него неименованное ребро.

Дополнительно необходимо отметить, что условием реального удаления регулярного файла с диска является не только равенство 0 значения его счетчика жестких связей, но и отсутствие процессов, держащих этот файл открытым. Если такие процессы есть, то удаление регулярного файла произойдет в ходе его полного закрытия последним использующим файл процессом.

Для осуществления операции удаления жестких связей и/или файлов можно использовать уже известную вам с семинаров 1-2 команду операционной системы rm или системный вызов unlink(). Заметим, что системный вызов unlink() также не требует предварительного открытия удаляемого файла, поскольку после его удаления совершать над ним операции бессмысленно.