Оглавление

1 Примеры сетевых файловых служб

В операционных системах семейства UNIX наибольшее распространение получили две сетевые файловые системы:

FTP (File Transfer Protocol)

NFS (Network File System).

Они первоначально разрабатывались для доступа к локальной файловой системе **s5/ufs**, являющейся основной для большинства ОС семейства UNIX. Эти сетевые файловые системы используют собственные протоколы клиент-сервер FTP и NFS, предоставляя интерфейс s5/ufs своим клиентам.

Со временем в крупных сетях стали одновременно применяться несколько сетевых файловых систем разных типов.

Протокол передачи данных (FTP*),* распределенные файловые системы и модель «клиент-сервер», использующая протокол совместного доступа к файлам, - примеры реализации сред совместного доступа к файлам.

1.1 Протокол FTP

FTP - это протокол «клиент-сервер», позволяющий передавать данные через сеть. FTP-сервер и FTP-клиент общаются друг с другом, используя TCP в качестве транспортного протокола. FTP, как определено стандартом, не является безопасным методом передачи данных, поскольку он использует передачу незашифрованных данных по сети. FTP поверх SSH-протокола добавляет параметр безопасности к изначальной спецификации FTP

1.2 Распределенная файловая система DFS

Распределенная файловая система (DFS) - файловая система, распределенная между несколькими хостами. DFS может предоставить хостам прямой доступ ко всей файловой системе, обеспечивая при этом эффективное управление и защиту данных.

1.3 Модели «клиент-сервер»

Традиционная *модель «клиент-сервер»,* осуществляемая посредством протоколов совместного доступа к файлам - еще один механизм удаленного совместного доступа к файлам. В этой модели клиент монтирует удаленные файловые системы, доступные на выделенных файловых серверах.

Стандартные протоколы «клиент-сервер» для совместного доступа к файлам — **NFS** для Unix и **CIFS** для Windows. Они позволяют владельцу файла задать для конкретного пользователя или группы пользователей требуемый тип доступа, например, только для чтения или для чтения и записи.

В обеих этих реализациях пользователи не знают о местонахождении файловой системы. Кроме того, *служба именования,* такая как система именования доменов (DNS), облегченный протокол доступа к каталогам (LDAP) и информационная служба сети (NIS), помогает, пользователям определить уникальный ресурс и получить к нему доступ через сеть. *Протокол службы именования* создает пространство имен, содержащее уникальное имя каждого сетевого ресурса и помогающее распознавать ресурсы в сети.

1.3.1 Сетевая файловая система NFS

Сетевая файловая система (NFS) — это протокол «клиент-сервер» для совместного использования файлов, чаще всего используемый в системах на основе UNIX.

Изначально NFS основывался на протоколе пользовательских датаграмм (UDP) без установления соединения. Он использует машинно-независимую модель для представления данных пользователя. Он также использует вызов удаленных процедур (RPC) как метод межпроцессного взаимодействия между двумя компьютерами. Протокол NFS обеспечивает набор RPC, чтобы получить

доступ к удаленной файловой системе для следующих операций;

- Поиск файлов и каталогов;
- Открытие, чтение, запись в файл и закрытие файлов;
- Изменение атрибутов файлов;
- Изменение ссылок на файлы и каталоги.

NFS использует механизм монтирования *для* создания соединения *между* клиентом и удаленной системой с целью передачи данных. NFS (NFSv3 и более ранние версии) — протокол *без поддержки хранения адресов,* а это означает, что он не поддерживает ни один тип таблиц для хранения информации об открытых файлах и ассоциированных указателях ссылок. Таким образом, каждый вызов предоставляет полный набор аргументов *ДЛЯ* ТОГО, чтобы получить доступ к файлам на сервере. Эти аргументы включают имя файла и местоположение, конкретную позицию для чтения или записи и версии NFS.

В настоящее время используются три версии NFS:

- **NFS версия 2 (NTSv2):** использует протокол пользовательских датаграмм (UDP) для обеспечения сетевого соединения между клиентом и сервером без поддержки хранения адресов. Такие функции как блокировка обрабатываются вне протокола;
- **NFS версия 3 (NFSv3)**: самая распространенная версия; использует протоколы UDP или TCP и работает на базе протокола без поддержки хранения адресов. Она включает в себя некоторые новые функции, такиекак 64-разрядный размер файла, асинхронные записи и дополнительные атрибуты файла в целях сокращения повторных выборок;
- **NFS версия 4 (NTSv4)**: Эта версия использует протокол TCP и работает на базе протокола с хранением адресов. Она обеспечивает повышенный уровень безопасности.

1.3.2 Сетевая файловая система CIFS

CIFS — это протокол приложения «клиент-сервер», позволяющий

клиентским программам запрашивать файлы и сервисы с удаленных компьютеров через TCP/ IP. Он является общественным (открытым) вариантом протокола блока серверных сообщений (SMB).

Протокол CIFS позволяет удаленным клиентам получать доступ к находящимся на сервере файлам. CIFS дает возможность использовать файлы совместно с другими клиентами при помощи специальных замков. Имена файлов в CIFS кодируются через символы Юникода. CIFS обеспечивает следующие возможности для обеспечения целостности данных:

- Он использует блокировку файла и записи, чтобы не дать пользователям осуществлять перезапись работы другого пользователя в файле или записи.
- Он работает поверх TCP.
- Он поддерживает отказоустойчивость и может автоматически восстановить соединения и заново открыть файлы, которые были открыты до сбоя. Функции отказоустойчивости CIFS зависят от того, написано ли приложение, позволяюшее воспользоваться этими функциями. Кроме того, CIFS — это протокол с хранением адресов, поскольку сервер CIFS поддерживает информацию о соединении *ДЛЯ* каждого подсоединенного клиента. В случае выхода сети из строя или сбоя сервера CIFS клиент получает уведомление об отключении. Ущерб клиенту минимален, если приложение имеет встроенный интеллект для восстановления связи. Однако если таковой отсутствует, пользователь должен принять меры, чтобы восстановить подключение CIFS.

Пользователи ссылаются на удаленные файловые системы с помощью простой в использовании схемы именования файлов.

1.4 Реализация протокола передачи файлов FTP

Сетевая файловая служба на основе протокола **FTP *(File Transfer Protocol)*** представляет собой одну из наиболее ранних служб, используемых для доступа к удаленным файлам.

До появления службы WWW это была самая популярная служба доступа к удаленным данным в Интернете и корпоративных IP-сетях.

Первые спецификации FTP относятся к 1971 году. Серверы и клиенты FTP имеются практически в каждой ОС семейства UNIX, а также во многих других сетевых ОС.

Клиенты FTP встроены сегодня в программы просмотра (браузеры) Интернета, так как архивы файлов на основе протокола FTP по-прежнему популярны и для доступа к таким архивам браузером используется протокол FTP.

Протокол FTP позволяет целиком переместить файл с удаленного компьютера на локальный и наоборот, то есть работает по схеме **загрузки-выгрузки**.

Кроме того, он поддерживает несколько команд **просмотра удаленного каталога и перемещения по каталогам удаленной файловой системы.**

Поэтому FTP особенно удобно использовать для доступа к тем файлам, данные которых нет смысла просматривать удаленно, а гораздо эффективней целиком переместить на клиентский компьютер (например, файлы исполняемых модулей приложений).

В протокол FTP встроены примитивные средства **аутентификации удаленных пользователей на основе передачи по сети пароля в открытом виде.**

Кроме того, поддерживается анонимный доступ, не требующий указания имени пользователя и пароля, который является более безопасным, так как не подвергает пароли пользователей угрозе перехвата.

Протокол FTP выполнен по схеме клиент-сервер.

1.4.1 Состав модулей FTP

Клиент FTP состоит из нескольких функциональных модулей:

- **User Interface** — пользовательский интерфейс, принимающий от пользователя символьные команды и отображающий состояние сеанса FTP на символьном экране.

- **User-PI** — интерпретатор команд пользователя. Этот модуль взаимодействует с соответствующим модулем сервера FTP.
- **User-DTP** — модуль, осуществляющий передачу данных файла по командам, получаемым от модуля User-PI по протоколу клиент-сервер. Этот модуль взаимодействует с локальной файловой системой клиента.

FTP-сервер включает следующие модули:

- **Server-PI** — модуль, который принимает и интерпретирует команды, передаваемые по сети модулем User-PI.
- **Server-DTP** — модуль, управляющий передачей данных файла по командам от модуля Server-PI. Взаимодействует с локальной файловой системой сервера.

Клиент и сервер FTP поддерживают параллельно два сеанса

- **управляющий сеанс**
- **сеанс передачи данных.**

Управляющий сеанс открывается при установлении первоначального FTP-соединения клиента с сервером, причем в течение одного управляющего сеанса может последовательно выполняться несколько **сеансов передачи данных**, в рамках которых передаются или принимаются несколько файлов.

1.4.2 Общая схема взаимодействия клиента и сервера

1. Сервер FTP всегда открывает управляющий порт **TCP 21** для прослушивания, ожидая приход запроса на установление управляющего сеанса FTP от удаленного клиента.
2. После установления управляющего соединения клиент отправляет на сервер команды, которые уточняют параметры соединения:
   - **имя и пароль клиента;**
   - **роль участников соединения (активная или пассивная);**
   - **порт передачи данных;**

- **тип передачи;**
- **тип передаваемых данных (двоичные данные или ASCII-код);**
- **директивы на выполнение действий (читать файл, писать файл, удалить файл и т. п.).**

3. После согласования параметров пассивный участник соединения переходит в режим ожидания открытия соединения на порт передачи данных. Активный участник инициирует это соединение и начинает передачу данных.
4. После окончания передачи данных соединение по портам данных закрывается, а управляющее соединение остается открытым. Пользователь может по управляющему соединению активизировать новый сеанс передачи данных.

Порты передачи данных выбирает клиент FTP (по умолчанию клиент может использовать для передачи данных порт управляющего сеанса), а сервер должен использовать порт, на единицу меньший порта клиента.

Протокол FTP использует при взаимодействии клиента с сервером несколько команд (не следует их путать с командами пользовательского интерфейса клиента, которые использует человек).

Эти команды делятся на три группы:

- **команды управления доступом к системе;**
- **команды управления потоком данных;**
- **команды службы FTP.**

В набор команд управления доступом входят следующие команды:

USER — доставляет серверу имя клиента. Эта команда открывает управляющий сеанс и может также передаваться при открытом управляющем сеансе для смены имени пользователя.

PASS — передает в открытом виде пароль пользователя.

CWD — изменяет текущий каталог на сервере

REIN — повторно инициализирует управляющий сеанс

QUIT — завершает управляющий сеанс.

Команды управления потоком устанавливают параметры передачи данных:

PORT — определяет адрес и порт хоста, который будет активным участником соединения при передаче данных. Например, команда PORT 194,85,135,126,7,205 назначает активным участником хост 194.85.135.126 и порт 1997 (вычисление номера порта не тривиально, но вполне однозначно).

PASV — назначает хост пассивным участником соединения по передаче данных. В ответ на эту команду должна быть передана команда PORT с указанием адреса и порта, находящегося в режиме ожидания.

TYPE — задает тип передаваемых данных (ASCII-код или двоичные данные)

STRU — определяет структуру передаваемых данных (файл, запись, страница)

MODE — задает режим передачи (потоком, блоками и т. п.).

Как видно из описания, служба FTP может применяться для работы как со структурированными файлами, разделенными на записи или страницы, так и с неструктурированными.

Команды службы FTP инициируют действия по передаче файлов или просмотру удаленного каталога:

RETR — запрашивает передачу файла от сервера на клиентский хост. Параметрами команды является имя файла. Может быть задано также смещение от начала файла — это позволяет начать передачу файла с определенного места при непредвиденном разрыве соединения (этот параметр используется в команде reget пользовательского интерфейса).

STOR — инициирует передачу файла от клиента на сервер. Параметры аналогичны команде RETR.

RNFR и **RNTO** — команды переименования удаленного файла. Первая в качестве аргумента получает старое имя файла, а вторая — новое.

DELE, MKD, RMD, LIST — эти команды соответственно удаляют файл, создают каталог, удаляют каталог и передают список файлов текущего каталога.

Каждая команда протокола FTP передается в текстовом виде по одной команде в строке. Строка заканчивается символами CR и LF ASCII-кода.

Пользовательский интерфейс клиента FTP зависит от его программной реализации. Наряду с традиционными клиентами, работающими в символьном режиме, имеются и графические оболочки, не требующие от пользователя знания символьных команд.

Символьные клиенты обычно поддерживают следующий основной набор команд:

- **open** ***имя_хоста*** — открытие сеанса с удаленным сервером.
- **bye** — завершение сеанса с удаленным хостом и завершение работы утилиты ftp.
- **close** — завершение сеанса с удаленным хостом, утилита ftp продолжает работать.
- **ls (dir)** — печать содержимого текущего удаленного каталога.
- **get** ***имя_файла*** — копирование удаленного файла на локальный хост.
- **put** ***имя_файла*** — копирование удаленного файла на удаленный сервер.

1.5 Реализация файловой системы NFS

Файловая система NFS *(Network File System)* создана компанией Sun Microsystems. В настоящее время это стандартная сетевая файловая система для ОС семейства UNIX, кроме того, клиенты и серверы NFS реализованы для многих других ОС.

Принципы ее организации на сегодня стандартизованы сообществом Интернета, последняя версия NFS v.4 описывается спецификацией RFC 3010, выпущенной в декабре 2000 года.

NFS представляет собой систему, поддерживающую схему **удаленного доступа к файлам.**

- Работа пользователя с удаленными файлами после выполнения операции монтирования становится полностью прозрачной - поддерево файловой системы сервера NFS становится поддеревом локальной файловой системы.
- Одной из целей разработчиков NFS была поддержка неоднородных систем с клиентами и серверами, работающими под управлением различных ОС на различной аппаратной платформе.

Этой цели способствует реализация NFS на основе механизма Sun RPC, поддерживающего по умолчанию средства XDR для унифицированного представления аргументов удаленных процедур.

Для обеспечения устойчивости клиентов к отказам серверов в NFS принят подход **stateless**, то есть серверы при работе с файлами не хранят данных об открытых клиентами файлах.

Основная идея NFS — позволить произвольной группе пользователей разделять общую файловую систему.

Чаще всего все пользователи принадлежат одной локальной сети, но не обязательно.

Можно выполнять NFS и на глобальной сети.

Каждый NFS-сервер предоставляет один или более своих каталогов для доступа удаленным клиентам.

Каталог объявляется доступным со всеми своими подкаталогами.

Список каталогов, которые сервер передает, содержится в файле **/etc/exports**, так что эти каталоги экспортируются сразу автоматически при загрузке сервера.

Клиенты получают доступ к экспортируемым каталогам путем монтирования.

Многие рабочие станции Sun бездисковые, но и в этом случае можно монтировать удаленную файловую систему к корневому каталогу, при этом вся файловая система целиком располагается на сервере.

Выполнение программ почти не зависит от того, где расположен файл: локально или на удаленном диске.

Если два или более клиента одновременно смонтировали один и тот же каталог, то они могут связываться путем разделения файла.

1.5.1 Задачи NFS.

1. Регулярное резервное копирование всех данных на сервере
2. Использование RAID-массивов на сервере
3. Уменьшение стоимости хранения данных
4. Обеспечение доступа к одним и тем же данным с любого компьютера

1.5.2 Протоколы NFS

В своей работе файловая система NFS использует два протокола.

Первый NFS-протокол управляет монтированием. Клиент посылает серверу полное имя каталога и запрашивает разрешение на монтирование этого каталога в какую-либо точку собственного дерева каталогов.

При этом серверу не указывается, в какое место будет монтироваться каталог сервера.

Получив имя, сервер проверяет законность этого запроса и возвращает клиенту **дескриптор файла**, являющегося удаленной точкой монтирования.

Дескриптор включает

описатель типа файловой системы,

номер диска,

номер индексного дескриптора (inode) каталога, который является удаленной точкой монтирования,

информацию безопасности.

Операции **чтения и записи** файлов из монтируемых файловых систем используют дескрипторы файлов вместо символьного имени.

Монтирование может выполняться автоматически, с помощью командных файлов при загрузке.

Существует другой вариант автоматического монтирования: при загрузке ОС на рабочей станции удаленная файловая система не монтируется, но при первом открытии удаленного файла ОС посылает запросы каждому серверу и после обнаружения этого файла монтирует каталог того сервера, на котором расположен найденный файл.

Второй NFS-протокол используется для доступа к удаленным файлам и каталогам. Клиенты могут послать запрос серверу для выполнения какого-либо действия над каталогом или операции чтения или записи файла. Кроме того, они могут запросить атрибуты файла, такие как тип, размер, время создания и модификации.

Файловой системой NFS поддерживается большая часть системных вызовов UNIX, за исключением **open** и **close**.

Исключение open и close не случайно.

Вместо операции открытия удаленного файла клиент посылает серверу сообщение, содержащее имя файла, с запросом отыскать его (lookup) и вернуть дескриптор файла.

В отличие от вызова **open** вызов **lookup** не копирует никакой информации во внутренние системные таблицы.

Вызов **read** содержит дескриптор того файла, который нужно читать, смещение в уже читаемом файле и количество байт, которые нужно прочитать.

Преимуществом такой схемы является то, что сервер не запоминает ничего об открытых файлах. Таким образом, если сервер откажет, а затем будет

восстановлен, информация об открытых файлах не потеряется, потому что она не поддерживается.

При отказе сервера клиент просто продолжает посылать на него команды чтения или записи в файлы, однако не получив ответа и исчерпав тайм-аут, клиент повторяет свои запросы.

После перезагрузки сервер получает очередной повторный запрос клиента и отвечает на него.

Таким образом, **крах сервера вызывает только некоторую паузу в обслуживании клиентов, но никаких дополнительных действий по восстановлению соединений и повторному открытию файлов от клиентов не требуется.**

К сожалению, NFS затрудняет блокировку файлов.

Во многих ОС файл может быть открыт и заблокирован так, чтобы другие процессы не имели к нему доступа. Когда файл закрывается, блокировка снимается.

В системах stateless, подобных NFS, блокирование не может быть связано с открытием файла, так как сервер не знает, какой файл открыт. Следовательно, NFS требует специальных дополнительных средств управления блокированием.

В NFS используется кэширование на стороне клиента, данные в кэш переносятся поблочно и применяется упреждающее чтение, при котором чтение блока в кэш по требованию приложения всегда сопровождается чтением следующего блока по инициативе системы.

Изменения данных в кэшируемом клиентом файле видны другому клиенту, в зависимости от временных соотношений.

Клиент при очередном открытии файла, имеющегося в его кэше, проверяет у сервера, когда файл был в последний раз модифицирован. Если это произошло после того, как файл был помещен в кэш, файл удаляется из кэша и от сервера получается новая копия файла.

Клиенты распространяют модификации, сделанные в кэше, с периодом в 30 секунд, так что сервер может получить обновления с большой задержкой. В результате работы механизмов удаления данных из кэша и распространения модификаций данные, получаемые каким-либо клиентом, не всегда, являются самыми свежими.

Репликация в NFS не поддерживается.

1.5.3 Реализация NFS

В большинстве систем UNIX используется трехуровневая реализация, сходная с изображенной на **рис. 30**.

Верхний уровень представляет собой **уровень системных вызовов**. Он управляет такими системными вызовами, как **open**, **read** и **close**.

После анализа системного вызова и проверки его параметров он вызывает второй уровень, уровень **VFS (Virtual File System — виртуальная файловая система**).

Задача уровня VFS заключается в управлении таблицей, содержащей по одной записи для каждого открытого файла, аналогичной таблице i-узлов для открытых файлов в системе UNIX.

В обычной системе **UNIX** i-узел однозначно указывается парой (**устройство, номер i-узла**).

Вместо этого уровень VFS содержит для каждого открытого файла записи, называемые **v**-узлами (**virtual i-node - виртуальный i-узел)**.

V-узлы используются, чтобы отличать локальные файлы от удаленных

Для удаленных файлов предоставляется информация, достаточная для доступа к ним. Для локальных файлов записываются сведения о файловой системе и i-узле так как современные системы UNIX могут поддерживать несколько файловые систем (например, V7, Berkeley FFS, ext2fs, /proc, FAT и т. д.).

Хотя уровень VFS был создан для поддержки файловой системы NFS, сегодня он поддерживается большинством современных систем UNIX как составная часть операционной системы, даже если NFS не используется.

Чтобы понять, как используются v-узлы, рассмотрим выполнение последовательности системных вызовов **mount**, **open** и **read**.

Чтобы смонтировать файловую систему, системный администратор (или сценарий ***/etc/rc)*** вызывает программу ***mount***, указывая ей удаленный каталог, локальный каталог, в котором следует смонтировать удаленный каталог, и прочую информацию.

Программа ***mount*** анализирует имя удаленного каталога и обнаруживает имя сервера NFS, на котором располагается удаленный каталог. Затем она соединяется с этой машиной, запрашивая у нее дескриптор удаленного каталога.

Если этот каталог существует и его удаленное монтирование разрешено, сервер возвращает его дескриптор. Наконец, программа ***mount*** обращается к системному вызову **mount**, передавая ядру полученный от сервера дескриптор каталога.

Затем ядро формирует для удаленного каталога v-узел и просит программу клиента NFS (**рис. 30**) создать в своих внутренних таблицах **r-узел** (удаленный i-узел) для хранения дескриптора файла. V-узел указывает на r-узел.

Каждый **v-узел** на уровне VFS будет в конечном итоге содержать либо указатель на **r-узел** в программе клиента NFS, либо указатель на **i-узел** в одной из локальных файловых систем (показанный на рис. 30 в виде штриховой линии).

По содержимому **v-узла** можно понять, является ли файл или каталог локальным или удаленным.

Если он локальный, то может быть найдена соответствующая файловая система и i-узел. Если файл удаленный, может быть найден удаленный хост и дескриптор файла.

- Когда на клиенте открывается удаленный файл, при анализе пути файла ядро обнаруживает каталог, в котором смонтирована удаленная файловая система.

- Ядро видит, что этот каталог удаленный, а в **v-узле** каталога находит указатель на **г-узел.**
- Затем ядро просит программу клиента NFS открыть файл.
- Программа **клиента NFS** просматривает оставшуюся часть пути на удаленном сервере, ассоциированном с монтированным каталогом, и получает обратно **дескриптор файла** для него. Она создает в своих таблицах г-узел для удаленного файла и докладывает об этом уровню VFS, который помещает в свои таблицы v-узел для файла, указывающий на г-узел.

У каждого открытого файла или каталога есть v-узел, указывающий на г-узел или i-узел.

Вызывающему процессу выдается дескриптор удаленного файла. Этот дескриптор файла отображается на v-узел при помощи таблиц уровня VFS.

На сервере не создается никаких записей в таблицах. Хотя сервер готов предоставить дескрипторы файлов по запросу, он не следит за состоянием дескрипторов файлов.

Когда **дескриптор файла** присылается **серверу** для доступа к файлу, сервер проверяет дескриптор и использует его, если дескриптор действителен. При проверке может проверяться ключ **аутентификации**, содержащийся в заголовках вызова удаленной процедуры RPC.

Когда дескриптор файла используется в последующем системном вызове, например **read,** уровень VFS находит соответствующий **v-узел** и по нему определяет, является ли он локальным или удаленным, а также какой **i-узел** или **г-узел** его описывает.

Затем он посылает серверу сообщение, содержащее дескриптор, смещение в файле (хранящееся на стороне клиента, а не сервера) и количество байтов.

Для повышения эффективности обмен информацией между клиентом и сервером выполняется большими порциями, как правило, по 8192 байт, даже если запрашивается меньшее количество байтов.

Когда сообщение с запросом прибывает на сервер, оно передается там уровню VFS, который определяет файловую систему, содержащую файл.

Затем уровень VFS обращается к этой файловой системе, чтобы прочитать и вернуть байты. Эти данные передаются клиенту.

После того как уровень VFS клиента получает 8-килобайтную порцию данных, которую запрашивал, он автоматически посылает запрос на следующую порцию, чтобы она была под рукой, когда понадобится.

Такая функция, называемая **опережающим чтением**, позволяет значительно увеличить производительность.

При записи в удаленный файл проходится аналогичный путь от клиента к серверу. Данные также передаются **8-килобайтными** порциями. Если системному вызову **write** подается менее 8 Кбайт данных, данные просто накапливаются локально. Только когда порция в 8 Кбайт готова, она посылается серверу. Если файл закрывается, то весь остаток немедленно посылается серверу.

1.5.4 Кэширование

Для увеличения производительности **применяется кэширование**, как в обычной системе UNIX.

Серверы кэшируют данные, чтобы снизить количество обращений к дискам, но это происходит незаметно для клиентов.

Клиенты управляют **двумя кэшами**, одним для атрибутов файлов (i-узлов) и одним для данных.

Когда требуется либо i-узел, либо блок файла, проверяется, нельзя ли получить эту информацию из кэша. Если да, то обращения к сети можно избежать.

Хотя кэширование на стороне клиента во много раз повышает производительность, оно также приводит к появлению новых непростых проблем.

Предположим, что два клиента сохранили в своих кэшах один и тот же блок файла, и что один из них его модифицировал. **Когда другой клиент считывает этот блок, он получает из кэша старое значение блока.**

Учитывая серьезность данной проблемы, реализация NFS пытается смягчить ее остроту несколькими способами.

Во-первых, с каждым блоком кэша ассоциирован таймер. Когда время истекает, запись считается недействительной. Как правило, для блоков с данными таймер устанавливается на **3 с**, а для блоков каталога — **30 с.** Таким образом, риск несколько снижается.

Кроме того, при каждом открытии кэшированного файла серверу посылается сообщение, чтобы определить, когда в последний раз был модифицирован этот файл.

Если последнее изменение произошло после того, как была сохранена в кэше локальная копия файла, эта копия из кэша удаляется, а с сервера получается новая копия.

Наконец, каждые 30 с истекает время таймера, и все «грязные» (модифицированные) блоки кэша посылаются на сервер.

Хотя такая схема и далека от совершенства, подобные «заплатки» позволяют пользоваться этой системой в большинстве практических случаев.