# Лабораторная работа № 13

# Сетевая файловая система NFS.

## *Краткие теоретические сведения*

## 1. Сетевая файловая система NFS.

Network File System (NFS) – протокол сетевого доступа к файловым системам, первоначально разработан Sun Microsystems в 1984 году. Основан на протоколе вызова удалённых процедур (ONC RPC, Open Network Computing Remote Procedure Call, RFC 1057, RFC 1831). Позволяет подключать (монтировать) удалённые файловые системы через сеть, описан в RFC 1094, RFC 1813, и RFC 3530.

NFS абстрагирована от типов файловых систем как сервера, так и клиента, существует множество реализаций NFS-серверов и клиентов для различных операционных систем и аппаратных архитектур. В настоящее время используется наиболее зрелая версия NFS v.4 (RFC 3010), поддерживающая различные средства аутентификации (в частности, Kerberos и LIPKEY с использованием протокола RPCSEC_GSS) и списки контроля доступа (как POSIX, так и Windows-типов).

Компания Sun Microsystems представила NFS в 1985 году как средство обеспечения прозрачного доступа к удаленным файловым системам. Помимо публикации протокола Sun лицензировала его базовую реализацию, которая была использована различными поставщиками для портирования NFS на разные операционные системы. С тех пор NFS стала фактически промышленным стандартом, который поддерживается действительно всеми вариантами системы UNIX, а также некоторыми другими системами, например, VMS и MS-DOS.

Архитектура NFS базируется на модели клиент-сервер. Файл-сервер представляет собой машину, которая экспортирует некоторый набор файлов. Клиентами являются машины, которые имеют доступ к этим файлам. Одна машина может для различных файловых систем выступать как в качестве сервера, так и в качестве клиента. Однако программный код NFS разделен на две части, что позволяет иметь только клиентские или только серверные системы.

Клиенты и серверы взаимодействуют с помощью удаленных вызовов процедур (RPC – remote procedure call), которые работают как синхронные запросы. Когда приложение на клиенте пытается обратиться к удаленному файлу, ядро посылает запрос в сервер, а процесс клиента блокируется до получения ответа. Сервер ждет приходящие запросы, обрабатывает их и отсылает ответы назад клиентам.

### *Взгляд со стороны пользователя*

Сервер NFS экспортирует одну или несколько файловых систем. Каждая экспортируемая файловая система может быть либо целым разделом диска либо его поддеревом. (Различные варианты UNIX имеют свои собственные правила дробления экспортируемых систем. Некоторые из них могут, например, разрешать экспортировать только файловую систему целиком, другие – только одно поддерево в каждой файловой системе). Сервер может определить, обычно посредством строк в файле `/etc/exports`, какие клиенты могут иметь доступ к каждой экспортируемой файловой системе, а также разрешенный режим доступа к ней: «только чтение» или «чтение и запись».

Затем клиентские машины могут монтировать такую файловую систему или ее поддерево к любому каталогу в своей существующей иерархии файлов, точно так же, как они смогли бы смонтировать любую локальную файловую систему. Клиент может монтировать каталог в режиме «только чтение», даже если сервер экспортирует его в режиме «чтение и запись». NFS поддерживает два типа монтирования: жесткое и мягкое. От типа монтирования зависит поведение клиента в случае, если сервер не отвечает на запрос. Если файловая система смонтирована жестко, клиент продолжает повторные запросы до получения ответа. В случае мягкого монтирования клиент спустя некоторое время отказывается от повторных запросов и получает ошибку. Когда монтирование произведено, клиент может обращаться к файлам в удаленной файловой системе, используя те же самые операции, которые применяются к локальным файлам. Некоторые системы поддерживают также такой тип монтирования, поведение которого соответствует жесткому монтированию при организации повторных попыток смонтировать файловую систему, но оказывается мягким для последующих операций ввода/вывода.

Операции монтирования NFS менее ограничены по сравнению с операциями монтирования локальных файловых систем. Протокол не требует, чтобы вызывающий операцию монтирования пользователь был привилегированным, хотя большинству пользователей навязываются эти требования. (Например, ULTRIX компании Digital позволяет любому пользователю монтировать файловую систему NFS до тех пор, пока этот пользователь имеет права доступа по записи в каталог точки монтирования. Пользователь может монтировать ту же самую файловую систему к нескольким точкам дерева каталогов, даже к своему подкаталогу. Сервер может экспортировать только свои локальные файловые системы и не может пересекать свои собственные точки монтирования во время прохода по путевому имени. Таким образом, чтобы увидеть все файлы сервера, клиент должен смонтировать все его файловые системы.

На рисунке 1 приведен пример. Серверная система nfssrv имеет два диска. Диск 1 смонтирован к каталогу `/usr/local` диска 0 и экспортируются каталоги `/usr` и `/usr/local`. Предположим, что клиент выполняет следующие четыре операции `mount`:

```
mount -t nfs  nfssrv:/usr        /usr

mount -t nfs  nfssrv:/usr/u1     /u1

mount -t nfs  nfssrv:/usr        /users

mount -t nfs  nfssrv:/usr/local  /usr/local
```

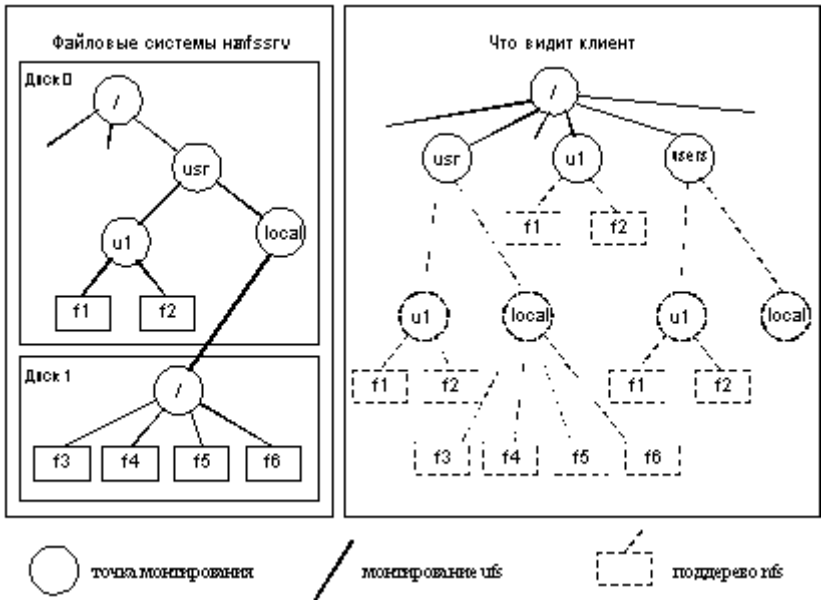


Рисунок 1. Монтирование файловых систем NFS

Все четыре операции монтирования будут успешно выполнены. На клиенте поддерево `/usr` отражает полное поддерево `/usr` на nfssrv, поскольку клиент также смонтировал `/usr/local`. Поддерево `/u1` на клиенте отображает поддерево `/usr/u1` на nfssrv. Этот пример иллюстрирует, что вполне законно можно монтировать подкаталог экспортированной файловой стстемы (это позволяют не все реализации). Наконец, поддерево `/users` на клиенте отображает только ту часть поддерева `/usr` сервера, которая размещена на диске 0. Файловая система диска 1 под `/users/local` не видна.

### *Цели разработки*

Первоначальная разработка NFS имела следующие цели:

- NFS не должна ограничиваться операционной системой UNIX. Любая операционная система должна быть способной реализовать сервер и клиент NFS.
- Протокол не должен зависеть от каких либо определенных аппаратных средств.
- Должны быть реализованы простые механизмы восстановления в случае отказов сервера или клиента.
- Приложения должны иметь прозрачный доступ к удаленным файлам без использования специальных путевых имен или библиотек и без перекомпиляции.
- Для UNIX-клиентов должна поддерживаться семантика UNIX.
- Производительность NFS должна быть сравнима с производительностью локальных дисков.
- Реализация должна быть независимой от транспортных средств.

### *Компоненты NFS*

Реализация NFS состоит из нескольких компонент. Некоторые из них локализованы либо на сервере, либо на клиенте, а некоторые используются и тем и другим. Некоторые компоненты не требуются для обеспечения основных функциональных возможностей, но составляют часть расширенного интерфейса NFS:

- Протокол NFS определяет набор запросов (операций), которые могут быть направлены клиентом к серверу, а также набор аргументов и возвращаемые значения для каждого из этих запросов. Версия 1 этого протокола существовала только в недрах Sun Microsystems и никогда не была выпущена. Все реализации NFS (в том числе NFSv3) поддерживают версию 2 NFS (NFSv2), которая впервые была выпущена в 1985 году в SunOS 2.0. Версия 3 протокола была опубликована в 1993 году и реализована некоторыми фирмами-поставщиками.
- Протокол удаленного вызова процедур (RPC) определяет формат всех взаимодействий между клиентом и сервером. Каждый запрос NFS посылается как пакет RPC.

• Расширенное представление данных (XDR – Extended Data Representation) обеспечивает машинно-независимый метод кодирования данных для пересылки через сеть. Все запросы RPC используют кодирование XDR для передачи данных. Следует отметить, что XDR и RPC используются для реализации многих других сервисов, помимо NFS.

• Программный код сервера NFS отвечает за обработку всех запросов клиента и обеспечивает доступ к экспортируемым файловым системам.

• Программный код клиента NFS реализует все обращения клиентской системы к удаленным файлам путем посылки серверу одного или нескольких запросов RPC.

• Протокол монтирования определяет семантику монтирования и размонтирования файловых систем NFS.

• NFS использует несколько фоновых процессов-демонов. На сервере набор демонов `nfsd` ожидают запросы клиентов NFS и отвечают на них. Демон `mountd` обрабатывает запросы монтирования. На клиенте набор демонов `biod` обрабатывает асинхронный ввод/вывод блоков файлов NFS.

• Менеджер блокировок сети (NLM – Network Lock Manager) и монитор состояния сети (NSM – Network Status Monitor) вместе обеспечивают средства для блокировки файлов в сети. Эти средства, хотя формально не связаны с NFS, можно найти в большинстве реализаций NFS. Они обеспечивают сервисы не возможные в базовом протоколе. NLM и NSM реализуют функционирование сервера с помощью демонов `lockd` и `statd` соответственно.

### *Отсутствие сохранения состояния*

Возможно, наиболее важной характеристикой протокола NFS является то, что сервер, чтобы работать корректно, не запоминает состояний и не нуждается ни в какой информации о своих клиентах. Каждый запрос является полностью независимым от других запросов и содержит всю необходимую информацию для его обработки. Серверу не нужно поддерживать никаких записей о прошлых запросах клиентов, за исключением необязательных возможностей, которые могут использоваться с целью кэширования данных или для сбора статистики.

Например, в протоколе NFS отсутствуют запросы по открыванию и закрыванию файлов, поскольку они создали бы информацию о состоянии, которая должна запоминаться сервером. По этой же причине, запросы read и write передают в качестве параметра начальное смещение, в отличие от операций read и write с локальными файлами, которые получают смещение из объекта «открытый файл».

Протокол без сохранения состояний упрощает восстановление после краха системы. Если отказывает клиентская система, никакого восстановления не требуется, поскольку сервер не поддерживает никакой устойчивой информации о клиенте. Если клиент перезагрузился, он может перемонтировать файловые системы и запустить приложения, которые обращаются к удаленным файлам. Серверу не нужно ни знать, ни беспокоиться об отказе клиента.

Если отказывает сервер, то клиент увидит, что на свои запросы он не получает ответы. Тогда он продолжает повторно посылать запросы до тех пор, пока сервер не перезагрузится. (Это справедливо только в случае жесткого монтирования (которое выполняется по умолчанию). При мягком монтировании клиент спустя некоторое время прекращает посылку запросов и возвращает приложению сообщение об ошибке). С этого момента времени сервер начнет получать запросы и может их обрабатывать, поскольку запросы не зависят ни от какой более ранней информации о состоянии. Когда, наконец сервер ответит на запросы, клиент перестанет их повторно посылать. У клиента нет никаких средств определить, действительно ли сервер отказал и был перезагружен, или просто медленно выполняет операции.

Протоколы с сохранением состояния требуют реализации сложных механизмов восстановления после отказа. Сервер должен обнаруживать отказы клиента и ликвидировать все состояния, связанные с этим клиентом. Если отказывает и перезагружается сервер, он должен уведомить клиентов так, чтобы они могли заново создать свое состояние на сервере.

Главная проблема работы без сохранения состояния заключается в том, что сервер должен зафиксировать все изменения в стабильной памяти до посылки ответа на запрос. Это означает, что не только данные файла, но и все метаданные, такие как индексные дескрипторы или косвенные блоки должны быть сброшены на диск до возвращения результатов. В противном случае сервер может потерять данные, о которых клиент уверен, что они успешно записались на диск. (Отказ системы может привести к потере данных даже в локальной файловой системе, но в таких случаях пользователи знают об отказе и о возможности потерять данные). Работа без сохранения состояния связана также с другими недостатками. Она требует отдельного протокола (NLM) для обеспечения блокировки файлов. Кроме того, чтобы решить проблемы производительности операций синхронной записи большинство клиентов кэшируют данные и метаданные локально. Но это противоречит гарантиям протокола о соблюдении согласованного состояния.

<table>
<tr><td>Уровни OSI</td><td>Протоколы<br>NFS v2</td></tr>
<tr><td>Прикладной</td><td>NFS</td></tr>
<tr><td>Представления</td><td>XDR</td></tr>
<tr><td>Сеансовый</td><td>RPC</td></tr>
<tr><td>Транспортный</td><td>UDP</td></tr>
<tr><td>Сетевой</td><td>IP</td></tr>
<tr><td>Канальный</td><td rowspan="2">Ethernet</td></tr>
<tr><td>Физический</td></tr>
</table>

Рисунок 2. Система NFS в соответствии с сетевой моделью OSI.

На Рисунке 2 представлена сетевая модель NFS v2 в соответствии с эталонной моделью OSI. В отличие от большинства сетевых служб TCP/IP система NFS явным образом использует протоколы презентационного и сеансового уровня. Работа NFS опирается на концепцию вызовов удаленных процедур (Remote Procedure Call, RPC). Согласно этой концепции, при доступе к удаленному ресурсу (например, к файлу) программа на локальном компьютере выполняет обычный системный вызов (предположим, вызов функции открытия файла), но на самом деле процедура выполняется удаленно – на сервере ресурсов. При этом пользовательский процесс не в состоянии определить, как выполняется вызов – локально или удаленно. Установив, что процесс обращается к ресурсу на удаленном компьютере, выступающем в качестве сервера, ядро или специальный демон системы упаковывает аргументы процедуры вместе с ее идентификатором в сетевой пакет, открывает сеанс связи с сервером и пересылает ему данный пакет. Сервер распаковывает полученный пакет, определяет запрашиваемую процедуру и аргументы, а затем выполняет процедуру. Далее сервер пересылает клиенту код возврата процедуры, а тот передает его пользовательскому процессу. Таким образом, RPC полностью соответствует сеансовому уровню модели OSI.

Возникает справедливый вопрос: зачем в сетевой модели NFS нужен специальный протокол презентационного уровня? Дело в том, что Sun благоразумно рассчитывала на применение NFS в гетерогенных сетях, где компьютеры имеют различную системную архитектуру, в том числе различный порядок представления байтов в машинном слове, различное представление чисел с плавающей точкой, несовместимые границы выравнивания структур и т.д. Поскольку протокол RPC предполагает пересылку аргументов процедур, т.е. структурированных данных, то наличие протокола презентационного уровня является насущной необходимостью в гетерогенной среде. В качестве такового выступает протокол внешнего представления данных (eXternal Data Representation, XDR). Он описывает так называемую каноническую форму представления данных, не зависящую от системной архитектуры процессора. При передаче пакетов RPC клиент переводит локальные данные в каноническую форму, а сервер проделывает обратную операцию. Следует иметь в виду, что каноническая форма XDR соответствует представлению данных, принятому для семейства процессоров SPARC и Motorola. В серверах, где реализована аналогичная форма представления данных, это позволяет добиться некоторого (правда, скорее всего, микроскопического) преимущества в производительности над конкурентами в случаях интенсивного обращения к файловому серверу.

В NFS v2 в качестве транспортного протокола был выбран UDP. Разработчики объясняют это тем, что сеанс RPC длится короткий промежуток времени. Более того, с точки зрения выполнения удаленных процедур каждый пакет RPC является самодостаточным, т. е. каждый пакет несет полную информацию о том, что необходимо выполнить на сервере, или о результатах выполнения процедуры. Сервисы RPC обычно не отслеживают состояние связи (connectionless), т.е. сервер не хранит информацию о том, какие запросы клиента обрабатывались в прошлом: например, с какого места файла клиент считывал данные последний раз. Для сетевой файловой системы это определенное преимущество с точки зрения надежности, так как клиент может продолжать файловые операции сразу же после перезагрузки сервера. Но такая схема чревата возникновением проблем при записи и блокировке файлов, и, чтобы их обойти, разработчики NFS были вынуждены применять разные обходные маневры.

Важное отличие сервисов RPC, входящих в состав NFS, от других сетевых серверных служб состоит в том, что они не используют суперсервер inetd. Рядовые сетевые службы, наподобие telnet или rlogin, обычно не запускаются в виде демонов при старте системы, хотя это делать не возбраняется. Чаще всего они задействуют так называемый суперсервер inetd, который «прослушивает» программные порты протоколов TCP и UDP. Службы задаются в конфигурационном файле суперсервера (обычно `/etc/inetd.conf`). При поступлении запроса на программный порт со стороны клиента inetd запускает в качестве дочернего процесса соответствующую сетевую службу (например, in.rlogind), которая и обрабатывает запрос.

Службы RPC не пользуются суперсервером inetd, поскольку, как уже было сказано, сеанс RPC длится очень недолго, фактически лишь в течение обработки одного-единственного запроса. Т.е. для каждого запроса inetd был бы вынужден запускать новый дочерний процесс службы RPC, что весьма накладно для UNIX. По аналогичным соображениям процесс RPC не может порождать новые процессы и не может параллельно обслуживать сразу несколько запросов. Поэтому в целях повышения производительности сервисы RPC запускаются в виде нескольких одновременно работающих экземпляров демонов. При этом количество экземпляров конкретного демона напрямую не связано с количеством клиентов. Даже один демон может обслуживать множество клиентов, но за один раз он способен обрабатывать единственный запрос, остальные будут помещаться в очередь.

Еще одно важное отличие сервисов RPC от обычных сетевых служб состоит в том, что они не используют заранее заданных программных портов UDP. Вместо этого применяется так называемая система трансляции портов portmapper. Для ее поддержки при загрузке системы инициализируется специальный демон portmap. В рамках системы трансляции портов за каждым сервисом RPC закрепляется программный номер (program number), номер версии, номер процедуры (procedure number) и протокол (UDP или TCP). Программный номер однозначно идентифицирует конкретный сервис RPC. Взаимосвязь между именами сервисов RPC и программными номерами можно проследить на основании содержимого файла `/etc/rpc`. Каждая программа RPC поддерживает множество процедур, которые определяются по их номерам. Номера процедур можно узнать в соответствующих header-файлах: например, для сервиса NFS они задаются в файле `/usr/include/nfs/nfs.h`.

В частности, сервис NFS имеет программный номер 100003 и включает такие процедуры, как «открытие файла», «чтение блока», «создание файла» и т. д. При вызове удаленных процедур вместе с аргументами процедуры в пакете RPC передается программный номер сервиса, номер процедуры и номер версии. Номер версии служит для идентификации возможностей сервиса. Дело в том, что разработчики постоянно совершенствуют службу NFS, при этом каждая новая версия полностью обратно совместима с предыдущими.

Принцип работы транслятора portmap достаточно прост. При инициализации (в частности, в момент загрузки ОС) какого-либо сервиса RPC он регистрируется с помощью демона portmap. При запуске на сервере сервис RPC ищет

незанятый программный порт, резервирует его за собой и сообщает номер порта демону portmap. Для того чтобы связаться с сервером, клиент RPC должен сначала обратиться к portmap сервера и узнать у него, какой программный порт занимает конкретный сервис RPC на сервере. Только затем клиент может непосредственно связаться с сервисом. В некоторых случаях клиент связывается с нужным сервисом косвенным образом, т. е. вначале он обращается к демону portmap, а тот запрашивает сервис RPC от лица клиента. В отличие от сервисов RPC, транслятор портов portmap всегда привязан к заранее заданному порту 111, так что клиент связывается с portmap стандартным способом.

## 2. Архитектура NFS.

Состоит из сервера NFS, клиентов NFS, сервера монтирования, клиентов монтирования, сервера автоматического монтирования на компьютере клиента, сервера и клиента блокировки, сервера определения статуса, сервера квотирования. Сервер NFS может хранить файлы локально или быть клиентом другого сервера NFS. Монтирование в этих точках возлагается на клиента. Сервер не разбирает составное имя, соответственно, интерпретация символьных ссылок также возлагается на клиента. Клиент должен обрабатывать случаи сбоя и перезагрузки сервера (сервиса) или сетевые проблемы (разрыв связности, потеря пакетов) используя интервалы ожидания и повторные передачи.

NFS2 и NFS3 сильно привязаны к семантике файловых операций в Unix (файлы ".", "..", права, uid/gid, имена в символьных ссылках и т.д.). Предполагается, что файловая система имеет иерархическую структуру, каждый файл или каталог имеет имя. Разделитель имён не важен, т.к. все процедуры NFS имеют дело только с простыми именами. Максимальная длина в NFS2 для простого имени файла – 255 байт, составного – 1024 байта. Протокол NFS3 не накладывает ограничения на длины простого и составного имён, но клиент и сервер могут иметь их. Максимальный размер блока данных – 8192 байта в NFS2, определяется процедурой FSINFO в NFS3. В NFS3 добавлены типы файлов: блочное устройство, символьное устройство, сокет, канал. Максимальный размер файла увеличен в NFS3 с $2^{31}$-1 до $2^{64}$-1 байта. В NFS3 добавлены средства слабой синхронизации (т.е. отсутствует гарантия) кэша клиента (weak cache consistency).

Построен над RPC, ранние версии используют UDP, поздние – TCP, обычно используется порт 2049 (т.е. rpcbind использовать необязательно). При работе в режиме TCP позволяет установить постоянное соединение для конкретной файловой системы (одно для всех клиентских процессов), а опция TCP keepalive позволяет определить сбой в работе сервера или перезагрузку и автоматически осуществлять повторное соединение и повтор «зависших» запросов.

Сервер NFS не хранит информацию о состоянии клиентов и их предыдущих запросах (stateless), это упрощает для клиента обработку сбоев сервера (перезагрузка сервера воспринимается как простая задержка). Процедуры open и close отсутствуют. Все операции в NFSv2 синхронные, в NFSv3 предусмотрено кэширование записи (асинхронный режим WRITE). Клиент может кэшировать запись самостоятельно. Большинство клиентов NFS3 кэшируют атрибуты файлов. Невозможно реализовать удаление открытого файла так, чтобы его можно было продолжать читать до закрытия. Аналогичная проблема с динамическим изменением прав доступа к файлу (из-за этого в реализациях считается, что владелец файла всегда имеет доступ к нему). Кстати, права доступа к файлу необходимо проверять при каждом обращении, т.к. сервер не способен определить, что файл уже был открыт. Сервер не способен отличить запрос на чтение файла от запроса на выполнение, поэтому разрешает чтение, если у пользователя есть права на чтение или исполнение.

Теоретически, NFS клиент и сервер возможно реализовать в виде приложения, а не в ядре. Для одновременного обслуживания множества запросов используется либо многопоточная реализация NFS сервера или запускается несколько экземпляров процесса nfsd/nfsd4 (их количество необходимо подгонять под нагрузку). Аналогично, на клиентской стороне можно использовать многопоточную реализацию или запустить требуемое количество процессов biod (rpciod). Как сервер, так и клиент может поддерживать несколько версий протокола, выбор версии производится с помощью соответствующего механизма RPC.

Для работы NFS, кроме собственно NFS (RPC программа 100003) требуется программа удалённого монтирования (RPC программа 100005), блокировки файлов (RPC программа 100021), проверки статуса (RPC программа 100024), квотирования (RPC программа 100011). При работе с файлами клиент использует указатели на файл (file handles, фиксированная длина 32 байта в NFS2, переменная длина до 64 байт в NFS3), т.е. при открытии файла сервер возвращает указатель, которым клиент пользуется в дальнейшем при работе с файлом. Содержимое указателя не должно интерпретироваться клиентом, но в Unix реализации там хранится major/minor устройства, на котором смонтирована файловая система, inode и версия inode (требуется при повторном использовании одного и того же inode для различных файлов). Если указатели совпадают, то они указывают на один и тот же файл. Если они различны, то это не означает, что файлы различны. Указатель может «зачервстветь» (stale), если между обращениями клиента к серверу он, например, перезагрузился.

NFSv4 (RFC 3010 – устарел, RFC 3530) не использует rpcbind, поддерживает NT ACL и сохраняет информацию о состоянии клиентов. Только режим TCP. Протоколы монтирования, блокировки и статуса встроены непосредственно в NFSv4. При монтировании аутентифицируется конечный пользователь, а не хост (причём по IP) как в NFSv3.

## 3. Протокол NFS.

NFS предоставляет клиентам прозрачный доступ к файлам и файловой системе сервера. Это отличается от FTP, который обеспечивает передачу файлов. С помощью FTP осуществляется полное копирование файла. NFS осуществляет доступ только к тем частям файла, к которым обратился процесс, и основное достоинство NFS в том, что он делает этот доступ прозрачным. Это означает, что любое приложение клиента, которое может работать с локальным файлом, с таким же успехом может работать и с NFS файлом, без каких либо модификаций самой программы.

NFS это приложение клиент-сервер, построенное с использованием Sun RPC. NFS клиенты получают доступ к файлам на NFS сервере путем отправки RPC запросов на сервер. Это может быть реализовано с использованием обычных пользовательских процессов – а именно, NFS клиент может быть пользовательским процессом, который осуществляет конкретные RPC вызовы на сервер, который так же может быть пользовательским процессом. Однако, NFS обычно реализуется иначе, это делается по двум причинам. Во-первых, доступ к NFS файлам должен быть прозрачным для клиента. Поэтому, вызовы NFS клиента осуществляются операционной системой клиента от имени пользовательского процесса клиента. Во-вторых, NFS сервера реализованы внутри операционной системы для повышения эффективности работы сервера. Если бы NFS сервер являлся пользовательским процессом, каждый запрос клиента и отклик сервера (включая данные, которые будут считаны или записаны) должен пройти через разделитель между ядром и пользовательским процессом, что вообще довольно дорогое удовольствие.

На рисунке 3 показаны типичные настройки NFS клиента и NFS сервера. На этом рисунке необходимо обратить внимание на следующее.

1. Клиенту безразлично, получает ли он доступ к локальному файлу или к NFS файлу. Ядро определяет это, когда файл открыт. После того как файл открыт, ядро передает все обращения к локальным файлам в квадратик, помеченный как «доступ к локальным файлам», а все ссылки на NFS файлы передаются в квадратик «NFS клиент».
2. NFS клиент отправляет RPC запросы NFS серверу через модуль TCP/IP. NFS обычно использует UDP, однако более новые реализации могут использовать TCP.
3. NFS сервер получает запросы от клиента в виде UDP датаграмм на порт 2049. Несмотря на то, что NFS может работать с преобразователем портов, что позволяет серверу использовать динамически назначаемые порты, UDP порт 2049 жестко закреплен за NFS в большинстве реализаций.

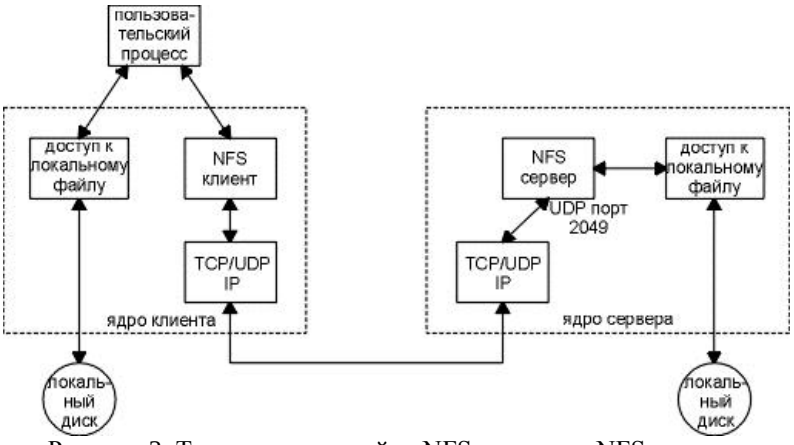

Рисунок 3. Типичные настройки NFS клиента и NFS сервера.

4. Когда NFS сервер получает запрос от клиента, он передаются локальной подпрограмме доступа к файлу, которая обеспечивает доступ к локальному диску на сервере.
5. Серверу может потребоваться время, для того чтобы обработать запросы клиента. Даже доступ к локальной файловой системе может занять некоторое время. В течение этого времени сервер не хочет блокировать запросы от других клиентов, которые также должны быть обслужены. Чтобы справиться с подобной ситуацией, большинство NFS серверов запускаются несколько раз, то есть внутри ядра существует несколько NFS серверов. Конкретные методы решения зависят от операционной системы. В большинстве ядер Unix систем не «живет» несколько NFS серверов, вместо этого запускается несколько пользовательских процессов (которые обычно называются `nfsd`), которые осуществляют один системный вызов и остаются внутри ядра в качестве процесса ядра.
6. Точно так же, NFS клиенту требуется время, чтобы обработать запрос от пользовательского процесса на хосте клиента. RPC выдается на узел сервера, после чего ожидается отклик. Для того, чтобы пользовательские процессы на узле клиента могли в любой момент воспользоваться NFS, существует несколько NFS клиентов, запущенных внутри ядра клиента. Конкретная реализация также зависит от операционной системы. Unix система обычно использует технику, напоминающую NFS сервер: пользовательский процесс, называемый biod, осуществляет один единственный системный вызов и остается внутри ядра как процесс ядра.

Большинство Unix узлов может функционировать как NFS клиент и как NFS сервер, или как и то и другое одновременно. Большинство PC реализаций (MS-DOS) имеют только реализации NFS клиента. Большинство IBM мейнфреймов предоставляет только функции NFS сервера.

NFS в действительности – это нечто большее, чем просто NFS протокол. На рисунке 4 показаны различные программы RPC, которые используются с NFS.

| Приложение | Номер программы | Номер версии | Количество процедур |
|---|---|---|---|
| преобразователь портов | 100000 | 2 | 4 |
| NFS | 100003 | 2 | 15 |
| программа mount | 100005 | 1 | 5 |
| менеджер блокирования | 100021 | 1,2,3 | 19 |
| монитор статуса | 100024 | 1 | 6 |

Рисунок 4. Различные RPC программы, используемые в NFS.

Демон монтирования вызывается на хосте NFS клиента, перед тем как клиент может получить доступ к файловой системе сервера.

Менеджер блокирования и монитор статуса позволяют клиенту заблокировать часть файлов, которые находятся на NFS сервере. Эти две программы не зависимы от протокола NFS, потому что блокирование требует идентификации клиента и на узле клиента, и на сервере, а NFS сам по себе «безразличен».

## 4. Аутентификация и авторизация.

Для авторизации запросов при использовании RPC аутентификации AUTH_UNIX используются uid и gid передаваемые клиентом, т.е. сервер полностью доверяет клиенту, а единственной возможностью проверить клиента является его IP адрес и таблицы преобразования идентификаторов (обычно производится на сервере по статической таблице или таблица создаётся в момент монтирования).

Обычно uid=0 клиента преобразуется в UID_NOBODY (nfsnobody, 65534, -2) на сервере, кроме случаев использования NFS в качестве корневой файловой системы.

Предполагается возможность использовать аутентификацию AUTH_DES или AUTH_KERB.

В NFS аутентификация производится исключительно на этапе монтирования файловой системы и только на основании доменного имени (или IP-адреса) клиентской машины. Т.е. если клиент NFS (здесь подразумевается компьютер, а не пользователь компьютера) обращается к серверу с запросом на монтирование, то сервер определяет права доступа по таблице `/etc/exports`, при этом клиент идентифицируется по имени (IP-адресу) компьютера. Если клиенту разрешено производить те или иные операции над экспортируемым ресурсом, то ему сообщается некое «магическое число» (`magic cookie`). В дальнейшем для подтверждения своих полномочий клиент должен включать это число в каждый запрос RPC.

Вот, собственно, и весь нехитрый набор средств аутентификации клиентов, пользователи же никак не аутентифицируются. Тем не менее, каждый запрос RPC содержит идентификатор пользователя uid, инициировавшего запрос, и список идентификаторов групп gid, куда входит пользователь. Но эти идентификаторы используются не для аутентификации, а для определения прав доступа конкретного пользователя к файлам и каталогам.

Обратите внимание, что uid и gid определяются на стороне клиента, а не сервера. Поэтому перед администраторами встает проблема согласования содержимого `/etc/passwd` (и `/etc/group`) между клиентами и серверами NFS, чтобы пользователю Васе на сервере не присвоили права пользователя Пети. Для больших сетей это представляет серьезные трудности. Обеспечить согласованность пользовательской базы данных, а также таких системных файлов, как `/etc/hosts`, `/etc/rpc`, `/etc/services`, `/etc/protocols`, `/etc/aliases` и др., можно с помощью сетевой информационной службы (Network Information System, NIS), разработанной компанией Sun еще в 1985 году и входящей в состав большинства версий UNIX (более продвинутая ее разновидность NIS+ не нашла широкого применения). NIS представляет собой информационную службу, в первом приближении напоминающую службу каталогов Windows NT, и позволяет централизованно хранить и обрабатывать системные файлы. Между прочим, NIS построена по тому же принципу, что и NFS, в частности она использует протоколы RPC и XDR.

Еще одна важная особенность NFS состоит в том, что в каждом запросе RPC передается список групп gid пользователя. Для ограничения размера пакета RPC в большинстве реализаций NFS количество групп не может превышать 8 или 16. Если пользователь входит в состав большего количества групп, то это может привести к ошибкам при определении прав доступа на сервере. Данная проблема весьма актуальна для корпоративных файловых серверов. Радикальным решением является использование списков контроля доступа ACL, но, к сожалению, далеко не все разновидности UNIX их поддерживают.

Принятая в NFS система аутентификации весьма убога и не обеспечивает надежной защиты. Любой, кто имел дело с NFS, знает, как просто обойти ее систему безопасности. Для этого даже не обязательно применять методы подделки IP-адресов (IP-spoofing) или имен (DNS-spoofing). Злоумышленнику достаточно перехватить «магическое число», и в дальнейшем он может проводить действия от имени клиента. К тому же «магическое число» не меняется до следующей перезагрузки сервера.

Исходя из этих соображений, Sun разработала протокол SecureRPC с использованием как несимметричных, так и симметричных ключей шифрования. При этом криптографические методы применяются для аутентификации не только узлов, но и пользователей. Однако сами данные не шифруются.

## 5. Протокол монтирования.

Клиент использует NFS протокол монтирования, чтобы смонтировать файловую систему сервера, перед тем как получить доступ к NFS файлам. Обычно это происходит при загрузке клиента. В результате клиент получает описатель файла файловой системы сервера.

На рисунке 5 описана последовательность действий Unix клиента при исполнении команды `mount(8)`.

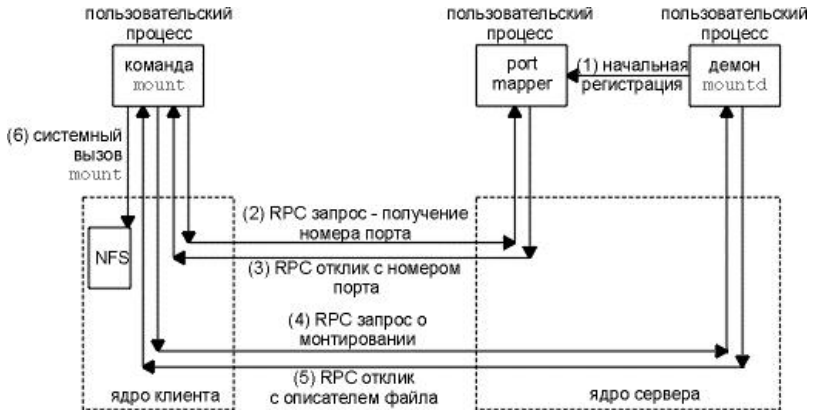


Рисунок 5. Протокол монтирования, используемый Unix командой mount.

При этом осуществляются следующие шаги.

1. При загрузке сервера на нем стартует преобразователь портов.
2. После преобразователя портов на сервере стартует демон монтирования (mountd). Он создает конечную точку TCP и конечную точку UDP, а также назначает каждой из них динамически назначаемый номер порта. Затем он регистрирует эти номера у преобразователя портов.
3. Клиент исполняется команду `mount`, которая выдает RPC вызов на преобразователь портов сервера, чтобы получить номер порта от демона монтирования на сервере. Для обмена между клиентом и преобразователем портов могут быть использованы и TCP и UDP, однако обычно используется UDP.
4. Преобразователь портов сообщает номер порта.
5. Команда `mount` выдает RPC вызов демону монтирования, чтобы смонтировать файловую систему сервера. И снова может быть использован как TCP, так и UDP, однако обычно используется UDP. Теперь сервер может проверить «годность» клиента основываясь на его IP адресе и номере порта, чтобы убедиться, можно ли этому клиенту смонтировать указанную файловую систему.
6. Демон монтирования откликается описателем файла указанной файловой системы.
7. Команда `mount` клиента выдает системный вызов mount, чтобы связать описатель файла, полученный в шаге 5, с локальной точкой монтирования на узле клиента. Описатель файла хранится в коде NFS клиента, и с этого момента любое обращение пользовательских процессов к файлам на файловой системе сервера будет использовать описатель файла как стартовую точку.

Подобная реализация отдает весь процесс монтирования, кроме системного вызова mount на клиенте, пользовательским процессам, а не ядру. Три программы, которые показаны – команда `mount`, преобразователь портов и демон монтирования – пользовательские процессы.

В этом примере на узле sun (NFS клиент) была исполнена команда

```
sun # mount -t nfs bsdi:/usr /nfs/bsdi/usr
```

Эта команда монтирует директорию `/usr` на узле bsdi (NFS сервер) как локальную файловую систему `/nfs/bsdi/usr`. На рисунке 6 показан результат.

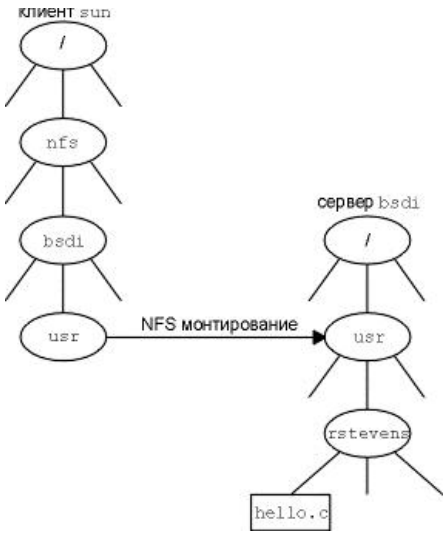


Рисунок 6. Монтирование директории bsdi:/usr как /nfs/bsdi/usr на хосте sun.

После чего при обращении к файлу `/nfs/bsdi/usr/rstevens/hello.c` на клиенте sun, происходит обращение к файлу `/usr/rstevens/hello.c` на сервере bsdi.

Прежде чем обратиться к файлу на удалённой файловой системе клиент (ОС клиента) должен смонтировать её и получить от сервера указатель на неё с помощью явной команды `mount` или с помощью одного из автоматических монтировщиков (amd, autofs, automount, supermount, superpupermount). Для этого клиентская программа монтирования (mount, amd) с помощью rpcbind определяет номер порта, на котором на сервере слушает программа RPC mount нужной версии (1, 2, 3), обращается к ней за указателем на корень требуемой файловой системы, далее этот указатель хранится на клиентской машине до размонтирования и используется при открытии файлов на удалённой файловой системе. Протокол RPC mount 1 используется совместно с NFS2, аутентификация AUTH_UNIX или AUTH_NONE (RFC 1094). Состояния клиентов хранятся. Может использовать как UDP, так и TCP. Сервер проверяет права клиента, основываясь на его IP адресе и «привилегированности» исходящего порта.

Процедуры протокола RPC mount 1 (RFC 1094):

- NULL
- MNT (смонтировать; принимает имя каталога; возвращает указатель на корень файловой системы, который можно использовать в NFS; сервер пополняет таблицу смонтированных файловых систем с указанием клиента: IP, имя узла)
- UMNT (размонтировать; принимает имя каталога)
- UMNTALL (размонтировать все файловые системы, смонтированные для этого клиента; а кто клиент-то: IP, имя узла)
- EXPORT (выдать список всех экспортируемых файловых систем и имён групп, которые смогут импортировать данную файловую систему)
- DUMP (выдать список смонтированных файловых систем с указанием узлов-клиентов)

Процедуры протокола RPC mount 3 (RFC 1813):

- NULL
- MNT (смонтировать; принимает имя каталога (ASCII) и список допустимых методов авторизации; возвращает указатель на корень файловой системы, который можно использовать в NFS; сервер пополняет таблицу смонтированных файловых систем с указанием имени узла клиента)
- DUMP (выдать список смонтированных файловых систем с указанием имён узлов клиентов в виде пар: имя каталога, имя узла; клиент останется в списке, если не выдаст явно UMNT/UMNTALL)
- UMNT (размонтировать; принимает имя каталога; иногда список неразмонтированных файловых систем используется для извещения клиентов о предстоящем выключении сервера)
- UMNTALL (размонтировать все файловые системы, смонтированные для этого клиента)
- EXPORT (выдать список всех экспортируемых файловых систем и имена, которые смогут импортировать данную файловую систему; имена могут представлять имена хостов или групп хостов, не могут быть интерпретированы клиентами)

## 6. Протокол блокировки.

Протокол блокировки файлов Network Lock Manager (NLM, RFC 1813, X/OpenNFS) версии 3 (для NFS2) и 4 (для NFS3) обеспечивает возможность блокировки файлов и записей. Клиент NFS должен быть готов к его отсутствию на сервере. После запуска сервера некоторое время (grace period, 45 секунд) принимаются только восстановления блокировок, новые блокировки не принимаются. Блокировки делятся на отслеживаемые и неотслеживаемые. Отслеживаемые блокировки должны восстанавливаться после перезапуска сервера без участия клиента и сбрасываться

при сбое клиента. Для реализации отслеживаемых блокировок требуется наличие серверов_мониторинга состояния (NSM) на сервере и клиенте. Перед запросом блокировки клиент NLM выдаёт запрос к своему NSM на мониторинг сервера. Сервер перед выполнением запроса на блокировку выдаёт запрос к своему NSM на мониторинг клиента. По снятию блокировки или отказу от неё мониторинг клиента снимается. При перезагрузке сервера его NSM извещает все клиентские NSM об изменении состояния. Те, в свою очередь, извещают заинтересованные локальные NLM клиенты, которые восстанавливают блокировки во время «льготного» периода. При перезагрузке клиента его NSM извещает все серверные NSM об изменении состояния. Те, в свою очередь, извещают заинтересованный локальный NLM сервер, который снимает все блокировки от данного клиента.

Неотслеживаемые блокировки (SHARE, UNSHARE, NM_LOCK, FREE_ALL) предназначены для совместимости с однозадачными ОС типа DOS. Клиент должен самостоятельно заботиться о восстановлении блокировок после перезапуска сервера и освобождать старые блокировки при своей перезагрузке.

## 7. Протокол мониторинга состояния.

Сервер NSM (Network Status Monitor) обеспечивает приложения информацией о состоянии узлов, установивших блокировки, и узлов, держащих блокировки. NSM сервер хранит информацию о текущем состоянии своего узла и предоставляет информацию об изменении состояния другим NSM серверам (NSM сервера равноправны). Состояние представляет собой монотонно увеличивающееся число. Чётное – когда сервер остановлен, нечётное – когда он работает. NSM не мониторит другие серверы, а терпеливо ждёт от них извещения (NOTIFY). Каждый сервер NSM держит у себя список соседей (имена хостов, хранится в постоянной памяти), состояние которых необходимо отслеживать для локальных клиентов (добавляются запросом MON), который хранится в постоянной памяти. Соседей из этого же списка NSM сервер извещает процедурой NOTIFY при изменении состояния своего хоста.

## 8. Правила экспортирования.

Файловые системы и каталоги, которые клиенты могут удаленно монтировать на сервере NFS, должны быть явно заданы. Данная процедура называется в NFS «экспортированием» ресурсов. В то же время сервер NFS, в отличие от, скажем, сервера SMB, не занимается широковещательной рассылкой списка своих экспортируемых ресурсов. Тем не менее клиент может запросить у сервера такой список. На стороне сервера за обслуживание запросов монтирования отвечает демон rpc.mountd.

Экспортирование файловых ресурсов NFS производится в соответствии с четырьмя основными правилами.

1. Файловую систему можно экспортировать как целиком, так и по частям, каковыми являются каталоги и файлы. При этом следует помнить, что самой крупной экспортируемой единицей является файловая система. Если на сервере некая файловая система (/usr/bin) смонтирована по иерархии ниже другой файловой системы (/usr), то экспортирование системы /usr систему /usr/bin не затронет.
2. Экспортировать можно только локальные файловые ресурсы, иными словами, если на сервере смонтирована чужая файловая система, т. е. находящаяся на другом сервере, то ее нельзя реэкспортировать.
3. Нельзя экспортировать подкаталоги уже экспортированной файловой системы, если только они не представляют собой самостоятельных файловых систем.
4. Нельзя экспортировать родительские каталоги уже экспортированного каталога, если только родительский каталог не представляет собой независимую файловую систему.

Любое нарушение этих правил приведет к ошибке в работе NFS.

## 9. Описатели файлов.

Одна из основ NFS реализуется описателями файлов. Для обращения к файлу или директории на сервере объекта используется opaque. Термин opaque обозначает, что сервер создает описатель файла, передает его обратно клиенту, который клиент затем использует при обращении к файлу. Клиент никогда не просматривает содержимое описателя файла – его содержимое представляет интерес только для сервера.

NFS клиент получает описатель файла каждый раз когда открывает файл, который в действительности находится на NFS сервере. Когда NFS клиент читает или пишет в этот файл (по поручению пользовательского процесса), описатель файла передается обратно серверу. Это указывает на то, что доступ к файлу был осуществлен.

Обычно пользовательский процесс не работает с описателями файлов. Обмен описателями файлов осуществляют NFS клиент и NFS сервер. В версии 2 NFS описатель файла занимает 32 байта, а в версии 3 он вырос до 64 байт.

Unix серверы обычно хранят в описателе файла следующую информацию: идентификатор файловой системы (major и minor номера устройства файловой системы), номер инода (inode) (уникальный номер внутри файловой системы), номер поколения инода (номер, который изменяется каждый раз, когда инод повторно используется для другого файла).

# *Реализация NFS сервера на базе ОС FreeBSD UNIX*

1. Настройка NFS сервера заключается в запуске определенных процессов и редактировании конфигурационного файла `/etc/exports`. Список процессов, которые должны быть запущены можно включить в файл `/etc/rc.conf`, который читается при старте системы, для этого в `/etc/rc.conf` необходимо добавить такие строки:

```
rpcbind_enable="YES"
nfs_server_enable="YES"
nfs_server_flags="-u -t -n 4"
mountd_flags="-r"
```

На клиенте в файле `/etc/rc.conf` должна присутствовать такая строка:

```
nfs_client_enable="YES"
```

2. Для запуска NFS сервера без перезагрузки системы необходимо выполнить следующую последовательность команд:

```
# rpcbind
# nfsd -u -t -n 4
# mountd -r
```

3. Файл `/etc/exports` определяет, какие файловые системы экспортируются на сервере NFS. Каждая строка в `/etc/exports` задаёт экспортируемую файловую систему, а также другие параметры, влияющие на характеристики доступа.

4. Для применения изменений в файле `/etc/exports` необходимо перезапустить демон `mountd`. Это можно выполнить посылкой сигнала `HUP` процессу `mountd`:

```
# kill -HUP `cat /var/run/mountd.pid`
```

или вызовом скрипта `mountd` подсистемы `rc(8)` с соответствующим параметром:

```
# /etc/rc.d/mountd reload
```

## *Задание на работу*

1. Настроить сервер NFS.

2. Обеспечить возможность монтирования файловых систем согласно варианту.

3. Проверить работу NFS сервера.

***Варианты заданий.***

| **№ варианта** | **Экспортируемый ресурс для сети 10.18.51.0/24 в режиме только чтение** | **Экспортируемый ресурс в режиме чтения и записи** | |
|---|---|---|---|
| | | **Ресурс** | **Клиент** |
| 1 | /usr/home/public | /usr/home/private | 10.18.51.2 |
| 2 | /opt/public | /opt/share | 10.18.51.3 |
| 3 | /usr/local/public | /usr/local/incoming | 10.18.51.4 |
| 4 | /opt/local/public | /opt/local/private | 10.18.51.5 |
| 5 | /usr/public | /usr/share | 10.18.51.6 |
| 6 | /usr/home/appl | /usr/home/incoming | 10.18.51.7 |
| 7 | /opt/appl | /opt/private | 10.18.51.8 |
| 8 | /usr/local/appl | /usr/local/share | 10.18.51.9 |
| 9 | /opt/local/appl | /opt/local/incoming | 10.18.51.10 |
| 10 | /usr/appl | /usr/private | 10.18.51.11 |
| 11 | /usr/home/docs | /usr/home/share | 10.18.51.2 |
| 12 | /opt/docs | /opt/incoming | 10.18.51.3 |

| № варианта | Экспортируемый ресурс для сети 10.18.51.0/24 в режиме только чтение | Экспортируемый ресурс в режиме чтения и записи | |
|---|---|---|---|
| | | Ресурс | Клиент |
| 13 | /usr/local/docs | /usr/local/private | 10.18.51.4 |
| 14 | /opt/local/docs | /opt/local/share | 10.18.51.5 |
| 15 | /usr/docs | /usr/incoming | 10.18.51.6 |

## *Литература*


1. . Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 4-е изд. – СПб.: Питер, 2010. – 944 с.

2. Стивенс У.Р. Протоколы TCP/IP. Практическое руководство/ Пер. с англ. и коммент. А.Ю. Глебовского. – СПб.: «Невский диалект» – «БХВ–Петербург», 2003. – 672 с.: ил.

3. Семенов Ю.А. Телекоммуникационные технологии v3.0, 17 августа 2007 года.

4. http://www.freebsd.org/doc/ru_RU.KOI8-R/books/handbook/network-nfs.html