

1. Старт системы. **init.d**

На этапе начальной загрузки системы загружается и начинает выполняться ее ядро, затем запускается ряд инициализационных задач.

Обычно процесс загрузки состоит из следующих этапов:

- Загрузка и инициализация ядра;
- Распознавание и конфигурирование устройств;
- Создание основных процессов;
- Выполнение командных файлов загрузки системы;
- Переход в многопользовательский режим.

Практически все этапы проходят без контроля со стороны администратора. Процессом загрузки можно управлять, редактируя файлы запуска системы.

Первым этапом идет загрузка в память ядра системы и его инициализация. Обычно файл, содержащий ядро системы находится в каталоге **/boot**. После завершения инициализации ядро запускает стартовые процессы, основным из которых является процесс **init**. Этот процесс всегда имеет PID равный 1 и находится в памяти на протяжении всего периода жизни операционной системы.

Ядро ищет **init** в нескольких местах, в которых он располагается по традиции. В Linux обычно используется имя **/sbin/init**. Если ядро не нашло **init**, оно пробует запустить **/bin/sh**. Если и тут происходит неудача, система выдает ошибку и останавливается.

Процесс **init** поддерживает различные «уровни выполнения». Командные файлы, обеспечивающие процесс загрузки системы, объединяются в несколько групп и вызываются по мере продвижения **init** по уровням выполнения.

Точное определение уровней выполнения задается в файле **/etc/inittab**. Если этот файл отсутствует или поврежден, то запустить систему можно только в однопользовательском режиме. Строка файла **inittab** имеет следующий формат:

<id>:<уровни выполнения>:<действие>:<процесс>

Поле **<id>** – это имя элемента файла **inittab**. Данное поле представляет собой одно- или двух символьную строку. В некоторых случаях это поле может быть пустым. Для терминалов в качестве **<id>** используют номер терминала.

В поле **<уровни выполнения>** указывается, к какому уровню выполнения должен применяться данный элемент. Для одного элемента может быть задано несколько уровней выполнения. Процесс, указанный в поле **<процесс>** запускается, когда **init** переключается на один из уровней выполнения, указанных в поле **<уровень>**. Если **init** переходит на уровень выполнения, отсутствующий в списке, то процесс завершается.

Поле **<действие>** содержит информацию о том, что должен делать процесс **init**:

- **wait** – ждать завершения процесса;
- **once** – выполнить процесс только один раз;
- **respawn** – перезапустить процесс в случае его «смерти»;
- **off** – игнорировать данный элемент.

Также имеется несколько особых категорий действий.

В таблице приведен список возможных действий.

Значение	Ждать?	Назначение
initdefault	-	Задаёт исходный уровень выполнения
boot	Нет	Выполняется при первом чтении файла inittab
bootwait	Да	Выполняется при первом чтении файла inittab
ondemand	Нет	Всегда поддерживает процесс в работающем состоянии
powerfail	Нет	Выполняется при получении сигнала сбоя питания
powerwait	Да	Выполняется при получении сигнала сбоя питания
sysinit	Да	Выполняется перед обращением к консоли
once	Нет	Запускает процесс однократно
wait	Да	Запускает процесс однократно
respawn	Нет	Всегда поддерживает процесс в работающем состоянии
off	-	Завершает процесс, если он выполняется

В столбце «Ждать?» указано, в каких случаях перед продолжением ожидается выполнение команды.

Пример:

```
1:2345:respawn:/sbin/getty 9600 tty1
```

Если команда при запуске терпит неудачу, а **init** сконфигурирован на ее перезапуск, то будет занято много ресурсов системы постоянными перезапусками команды. Во избежание такой ситуации **init** хранит протокол перезапусков команды, и если частота перезапусков слишком высока, делает паузу в 5 минут перед очередным перезапуском команды.

При работающей системе одна из самых важных задач программы **init** - это принятие на себя роли родительского процесса для процессов, чей настоящий родительский процесс был уничтожен. В ОС Linux, как и во всех системах семейства UNIX, *все* процессы *должны* принадлежать одному дереву процессов.

Существует восемь уровней выполнения: от 0 до 6 и уровень **S** для однопользовательского режима. Существует уже устоявшееся разделение уровней выполнения. Так уровень 0 и уровень 6 соответствуют останову/перезагрузке ОС. Уровень 1 аналогичен уровню **S**, на уровне 2-4 выполняется загрузка набора приложений, обеспечивающих работу компьютера в сети (настройка сетевых устройств и загрузка соответствующих программ). Эти уровни настраиваются администратором системы. Чаще всего используется только уровень 3. Уровень 5 обычно полностью повторяет уровень 3, и в дополнение загружаются приложения графической оболочки X-Windows.

Когда **init** запускается, он ищет в **inittab** строку, в которой указан уровень выполнения, указанный по умолчанию:

```
id:2:initdefault:
```

Можно указать, чтобы **init** при запуске установил не уровень выполнения по умолчанию, передав при загрузке в качестве параметра аргумент `single` или `emergency`. Параметры командной строки ядра могут быть переданы, через загрузчик (например LILO). Это позволяет выбрать, например, однопользовательский режим (уровень выполнения 1 или **S**). Как показывает практика, главное следить за тем, чтобы по умолчанию не был задан уровень 0 или 6.

Когда система работает, можно поменять уровень выполнения командой **telinit** или **init <новый уровень>**. При смене уровня выполнения, **init** выполнит соответствующие команды из файла **inittab**. Также, если был изменен файл **inittab** и требуется перечитать его и применить новые значения для текущего уровня выполнения можно выполнить команду **init q**.

Также файл **inittab** имеет особые возможности взаимодействия в сложных случаях. Строки для таких возможностей помечены специальными ключевыми словами и имеют по три, а не по четыре поля.

Примеры:

powerwait, powerfail – передает **init** сигнал на начало завершения работы системы по причине сбоев в сети питания. Выдается при совместном использовании UPS и программ, которые наблюдают за состоянием UPS и информируют **init** об отключении внешнего питания.

powerok – если при получении состояния **powerwait** питание восстановилось, будет отработано данное состояние. Полезно для возвращения к нормальной работе, если система еще не успела завершить все процессы.

ctrlaltdel – при нажатии на ctrl-alt-del на клавиатуре консоли, дает команду **init** перезагрузить систему. Администратор может поменять реакцию на нажатие ctrl-alt-del, например, на игнорирование или запуск какой-либо произвольной программы.

sysinit – команда, выполняемая при запуске системы. Например, может чистить каталог /tmp.

Очень важным является уровень выполнения 1 или **S** (*однопользовательский режим*), в котором системный администратор использует только необходимый минимум системных средств. Данный уровень выполнения нужен при выполнении особо важных задач системного администрирования, таких как запуск **fsck** на монтируемых файловых системах, для чего их надо размонтировать.

Запущенная система может быть переведена в однопользовательский режим командой **telinit** с запросом уровня выполнения 1. При загрузке он может быть установлен передачей ядру в командной строке параметра **single** или **emergency**. В этом случае **init** не будет использовать уровень выполнения, заданный по умолчанию.

Загрузка в однопользовательском режиме иногда необходима чтобы выполнить вручную **fsck**, прежде, чем что-нибудь смонтируется или как-то иначе коснется поврежденного раздела (любое действие на испорченной файловой системе может испортить ее еще больше, так что **fsck** должен быть выполнен как можно скорее).

Стартовый скрипт **init** автоматически запускает систему в однопользовательском режиме, если **fsck** при загрузке выявил ошибки на дисках. Такая мера предосторожности защищает те файловые системы, которые **fsck** не смог исправить самостоятельно.

Как мера защиты, правильно сконфигурированная система будет спрашивать root-пароль перед запуском оболочки в однопользовательском режиме. Иначе, было бы просто только ввести подходящую строку в LILO, чтобы войти как root. С другой стороны, если файл /etc/passwd поврежден в результате дисковых сбоев, такой подход принесет немало неприятностей.

2. DNS

В Интернете существует два основных способа адресации компьютеров. Первый - численный (или IP-адрес; например, 193.233.88.39), второй - символьный (alice.stup.ac.ru). DNS была создана для того, чтобы поставить в соответствие один способ другому.

Чтобы облегчить упорядочивание наименований, вся структура компьютерных имен устроена таким образом: есть отдельные уровни (домены), которые могут включать в себя как другие

поддомены, так и имена компьютеров. Все названия должны состоять только из латинских букв, цифр и знака "минус". Отдельные уровни доменов разделяются точкой.

Типичное полное доменное имя компьютера может выглядеть так: [computer3.otdel-5.firma.penza.ru](#). В этом примере такой адрес мы присвоили компьютеру номер три, который стоит в отделе 5 фирмы с названием "firma", которая находится в Пензе ("penza"), в России ("ru"). Локальным именем компьютера (hostname) здесь является "computer3", а ".ru" обычно называется доменом верхнего уровня. Домен [penza.ru](#), соответственно, является доменом второго уровня; [firma.penza.ru](#) третьего...

В пределах домена каждого уровня есть группа людей, которые отвечают за этот домен. Они могут добавлять имена вновь появившихся компьютеров, менять их или удалять. И по сути дела, то, как будет называться та машина, на которой работаете вы в своей фирме, зависит от того, что им подскажет фантазия написать в конфигурационном файле DNS.

Тот, кто имеет право администрировать домен, может делать изменения только в пределах этого домена. Например, системный администратор отдела №5 может, скажем, изменить имя "computer3" на "computer4" или на что-то более человеческое, например, назвать этот компьютер "julia" (Тогда полный его адрес станет [julia.otdel-5.firma.penza.ru](#)). Но для того, чтобы изменить имя домена 4-го уровня "otdel-5", администратору придется просить об этом у администратора фирмы (если, конечно, это не одно и то же лицо).

Процедура получения имени, например, в зоне .ru или .com называется регистрацией домена. Конечно, каждая компания, подключающаяся к Интернет, стремится зарегистрировать как можно более естественное и легкое для запоминания имя. Так, для "Microsoft inc." логично зарезервировать домен [microsoft.com](#)

Доменов верхнего уровня очень немного - всего около 250. Большая часть из них - так называемые, географические домены. Например, .de (Deutschland, Германия), .ru (Russia, Россия), .iq (Iraq, Ирак). Оставшиеся негеографические домены верхнего уровня - .com (для коммерческих компаний), .net (для сетевых ресурсов), .edu (образовательные учреждения), .mil (военные организации), .org (некоммерческие организации), .gov (правительственные ведомства), .int (интернациональные корпорации). Также в последнее время были введены дополнительные домены верхнего уровня: .name, .biz, .info

Если вам бы захотелось зарегистрировать еще один домен верхнего уровня, то потребовалось для этого предоставить такие серьезные обоснования, что гораздо проще было бы организовать свое маленькое государство и для него уже получить географический домен.

По статистическим данным, на начало 1998 года во всем Интернете было зарегистрировано около 30 миллионов хостов. Основная часть приходится на домен .com, далее в порядке убывания идут: .net (Networks), .edu (Educational), .jp (Japan), .mil (US Military), .us (United States), .de (Germany), .uk (United Kingdom), .ca (Canada), .au (Australia), .org (Organizations), .gov (Government)

Существует несколько вариантов настройки DNS на компьютере:

- Настройка через файл /etc/hosts
- Настройка локального кэширующего DNS сервера
- Настройка удаленного DNS сервера

Также может применяться смешанный вариант настройки. Например, при отсутствии записи в файле /etc/hosts будет выполняться поиск через DNS сервер. За настройку порядка поиска отвечает файл /etc/host.conf

```
order hosts, bind
```

Для указания, какие DNS сервера будут использоваться и в каком порядке выполнять поиск используется файл /etc/resolv.conf

```
nameserver 192.168.10.1
nameserver 193.233.88.3
```

search vt stup.ac.ru

Рассмотрим варианты настройки преобразования имен.

/etc/hosts

Файл hosts содержит по одной записи на строку, состоящую из IP адреса, имени хоста и необязательного списка псевдонимов имени. Поля отделяются пробелами или табуляцией, и поле адреса должно начинаться в первой колонке. Все, что следует после символа (#), расценивается как комментарий и игнорируется.

Имя хоста может быть также полностью квалифицированным или относительно локальной области.

Пример файла /etc/hosts

```
# IP          local      fully qualified domain name
#
127.0.0.1     localhost
#
193.233.88.39 alice      alice.stup.ac.ru
```

DNS сервер

Прежде всего, для полноценной работы DNS вам необходимо два или больше компьютеров, так называемых, name-серверов, которые независимо друг от друга подключены к Интернет (лучше, если они будут находиться в разных сетях или даже разных странах). Такая структура обеспечит неизменную работу системы преобразования символьного адреса в числовой и обратно, даже если какое-то время некоторые из этих компьютеров будут недоступны по сети. На таких компьютерах запускается специальная программа-демон **named**, которая обрабатывает запросы на преобразование адресов и отвечает на них. Настроить DNS - означает корректно написать конфигурационные файлы named.

Name-сервера бывают **primary** и **secondary**. Иногда их называют первичными и вторичными, а также **master** и **slave**. Primary name-сервер может быть только один. На нем хранится вся информация о доменах, и если происходят изменения, то конфигурация правится только на нем. Secondary name-серверов может быть несколько, но обычная практика - один secondary name-сервер. Дополнительные вторичные name-сервера служат для повышения скорости расшифровывания вашего адреса и для повышения устойчивости такого преобразования. Для небольших сетей три и больше вторичных name-сервера - это уже излишество. Secondary name-сервера с заданной периодичностью в автоматическом режиме считывают текущую конфигурацию с primary-сервера. Заметим, что один и тот же компьютер может одновременно являться primary-сервером для одних доменов и secondary name-сервером для нескольких других.

Рассмотрим различные директивы конфигурационного файла:

Все настройки разделены на группы.

Группа глобальных настроек начинается с определения `options` и все, что находится в пределах фигурных скобок, относится к этой группе.

Следует заметить, что каждый параметр должен заканчиваться знаком “;”. Если какой либо параметр содержит несколько значений, то они также объединяются фигурными скобками и после каждого следует разделитель.

Рассмотрим параметры группы `options`.

Параметр `directory` определяет рабочий каталог, в котором будут находиться файлы зон (конфигурационные файлы для каждого из доменов, которые этот name-сервер будет поддерживать).

Пример:

```
directory "/var/named";
```

Далее идет описание серверов, на которые будут перенаправляться запросы поиска соответствия между DNS именем и IP адресом (и наоборот). В списке может содержаться не более трех адресов.

Пример:

```
forwarders { 193.233.88.3; 193.233.88.2; };
```

Если самостоятельный поиск соответствия не дал положительного результата, то запрос будет перенаправлен DNS серверам, указанным в списке `forwarders`.

Может быть указан параметр, `forwarders first`; который изменяет порядок взаимодействия с серверами из списка `forwarders`. Если этот параметр присутствует, то запрос сначала будет передан DNS серверам из этого списка, и только если от них не пришло ответа, будет обрабатываться самостоятельно. В случае если указан параметр `forwarders only`; DNS сервер сам не будет обрабатывать запросы, а будет только использовать ответы от серверов, указанных в списке `forwarders`.

Если вы хотите получить эффект от использования кэша провайдера, то адрес его DNS сервера необходимо включить раздел `forwarders`. Если соответствующий сервер имён провайдера работает быстро и имеет хороший канал связи, то в результате такой настройки можно получить хороший результат и сэкономить трафик.

Параметр `listen-on` определяет, на каком порту и на каких адресах DNS сервер будет ожидать запросов.

Пример:

```
listen-on port 53 { 127.0.0.1; 192.168.0.1; };
```

Значение `port` может не указываться, в этом случае будет использован номер порта по умолчанию (для DNS это 53). В фигурных скобках перечисляются локальные IP адреса, на которых будут ожидать входящие запросы. В качестве адреса может быть указано ключевое слово `none` или адрес `0.0.0.0`. В первом случае, запросы не будут ожидать ни на каком из адресов, во втором случае на всех имеющихся адресах.

Также есть параметр `listen-on-v6`. Синтаксис его аналогичен параметру `listen-on` и определяет он то же самое, но для протокола IPv6. Ключевое слово может `none` в списке адресов может быть использовано только в одном из этих двух параметров, или ни в каком из них. Поскольку протокол IPv6 в данный момент практически не распространен, то обычно используется запись `listen-on-v6 { none; };`, которая отключает его использование.

Для того, чтобы ограничить источники запросов только от определенных адресов существует параметр `allow-query`. Этот параметр задает список подсетей или адресов, от которых запросы будут приниматься и обрабатываться. Все запросы от любых других источников будут игнорироваться.

Пример:

```
allow-query { 127.0.0.1; 192.168.0.0; };
```

Для описания зон необходимо использовать параметры, вынесенные в группу `zone`. Для каждой описываемой зоны должна присутствовать своя группа в конфигурационном файле. После ключевого слова `zone` в кавычках идет имя зоны и открывающая фигурная скобка.

Как говорилось ранее, зоны могут быть первичными или вторичными (**master** и **slave**). Для указания типа зоны существует параметр `type`, который может принимать значения `master` — для первичной зоны и `slave` — для вторичной.

Необходимо заметить, что для первичной зоны обязательно должен присутствовать дополнительный параметр `file`, который указывает, в каком файле находится описание зоны.

Пример:

```
zone "vt" {
    type master;
    file "vt.zone";
};
```

Для вторичной зоны обязательным является параметр `masters`, который определяет адрес сервера, с которого будет забираться информация о данной зоне. В этом списке может присутствовать как адрес первичного сервера, так и адреса других вторичных серверов.

Пример:

```
zone "stup.ac.ru" {
    type slave;
    masters { 193.233.88.3; };
};
```

Если для вторичной зоны не указан параметр `file`, то имя файла, в котором будет храниться описание зоны, выбирается DNS сервером самостоятельно.

```
zone "." {
    type hint;
    file "root.hint";
};
```

Это описание специальной корневой зоны, файл которой содержит IP-адреса компьютеров, знающих "все" о доменных именах, то есть, содержат домен "точка". В этих серверах хранится информация обо всех доменах верхнего уровня, и если на DNS-запрос ответ не был дан на более раннем этапе, то, добравшись до одного из таких top-level серверов, запрос пойдет по необходимой ветке вниз.

Файл, указанный в описании этой зоны нужно периодически забирать с FTP.RS.INTERNIC.NET, так как IP-адреса top-level серверов по прошествии несколько месяцев могут поменяться.

Рассмотрим файл настройки зон, который необходимо правильно сформировать, чтобы обеспечить функционирование первичного сервера имен.

Каждая строка этого файла имеет следующий формат:

`[domain] [opt_ttl] [opt_class] [type] [resource_record_data]`

где `domain` есть "." для описания домена верхнего уровня, "@" для текущего домена или обычное доменное имя (в частности, просто имя машины).

`opt_ttl` - необязательное поле, целое число, которое означает время жизни (time-to-live) этой записи в секундах. По истечении этого срока содержимое записи должно автоматически обновиться.

`opt_class` - тип адреса объекта. Такой тип существует только один, который, собственно, и указывается ("IN").

`type` - тип записи (рассматриваются ниже)

`resource_record_data` - данные этого типа

Рассмотрим пример primary name-сервера и, соответствующего ему, файла зоны.

```
@                IN SOA  ns.vt   root (
                28800 ; Refresh    8 hours
                7200  ; Retry      2 hours
                604800 ; Expire     7 days
                86400 ; Minimum TTL 1 day
                )
octopus          IN A              192.168.0.1
mail             IN CNAME         octopus
```

Заметим, что все строки, начинающиеся с ";" являются комментариями и используются для улучшения читабельности текста.

Самая первая запись в любом таком файле выглядит следующим образом:

```
@           IN           SOA  ns.vt.  root (
```

Символ "@" означает, что дальнейшие директивы относятся к текущему домену. Поле "IN" можно считать несущественным, а после него идет описание типа записи:

SOA - (Start of authorizing) - зона ответственности. Дальнейшие параметры определяют, кто отвечает за этот домен, как часто обновлять информацию об этой зоне на secondary-серверах, а также другую служебную информацию. Первое поле параметров этой записи - имя primary name-сервера, поддерживающего данную зону. После него следует адрес технического контактного лица, отвечающего за домен. **Обратите особое внимание на точки в конце этих адресов!** Точка в конце означает, что этот адрес является обычным доменным именем, и его не нужно дополнять справа доменом, которому соответствует наш конфигурационный файл. Точку в конце символьных имен нужно не забывать ставить и в других полях зоны. Это одна из самых распространенных ошибок.

Оставшиеся в круглых скобках после контактного лица пять параметров записи SOA - целые числа, определяющие временной интервал обмена информацией об этом домене между primary и secondary.

Первое число – серийный номер (**serial number**) записи. По изменению этого номера secondary name-сервера определяют, было ли изменено содержимое домена или нет, и соответственно, нужно ли считывать весь домен с primary-сервера. Серийный номер должен состоять из десяти цифр и обязательно должен быть заменен вручную на какой-либо больший при любом изменении любой записи в файле домена. Для этих целей идеально подходит такой вариант: первые четыре цифры serial'a - год, затем две на месяц и число. Последние две - порядковый номер (начиная с 00) редакции в течение дня. Если мы меняем конфигурацию домена, то заодно мы должны увеличить и серийный номер.

После серийного номера идет поле **Refresh**, которое указывает время в секундах, по истечении которого secondary name-сервер проверяет, не изменилась ли данная зона на primary-сервере, и если изменения были, то происходит передача файла с зоной.

Если при этом у него не получилось по каким-либо причинам соединиться с сервером, то следующую попытку secondary сделает по истечении **Retry** секунд (третий параметр).

В случае если и последующие попытки подключиться к primary и узнать информацию о зоне оканчиваются неудачей, вторичный name-сервер по прошествии **Expire** секунд забудет всю информацию по этой зоне.

Последнее поле (**'Minimum TTL'**) указывает на минимальное время жизни (Time To Live) записей в файле зоны, если только в какой-то записи не будет указано другое значение в необязательном поле opt_ttl.

Рекомендуемые значения для этих величин таковы:

```
28800 ; Refresh      8 hours
7200  ; Retry        2 hours
604800 ; Expire      7 days
86400 ; Minimum TTL  1 day
```

Однако вам никто не вправе помешать поставить свои значения. Только не ставьте слишком маленькие величины (меньше часа) - иначе вы просто забудете сеть бесполезной информацией, непрерывно пересылаемой от primary к secondary.

Теперь рассмотрим записи следующего типа:

NS - (Name Server) - перечисляет name-сервера (и primary, и secondary), которые поддерживают эту зону. **Не забывайте про точку в конце имени!**

IN	NS	ns.pri.vt.
	NS	ns.sec.vt.

Здесь мы указали два name-сервера (первый из них - primary, второй - secondary), которые содержат всю информацию о домене.

Далее идет одна из наиболее часто встречающихся записей DNS:

A - (Address) - адрес хоста. На первом месте в такой строке будет стоять символьное имя компьютера в текущем домене, после этого "IN A", а затем - числовой IP-адрес, соответствующий этой машине. Рекомендуется внести, например, такую строку в файл зоны:

localhost	IN	A	127.0.0.1
-----------	----	---	-----------

Это позволит обращаться с любого компьютера в сети к самому себе, используя зарезервированное имя localhost. Для того чтобы как-нибудь назвать новый компьютер в сети, вам нужно просто добавить такую строчку в файл зоны.

В том случае, когда необходимо для одного IP адреса задать несколько имен используется директива **CNAME** (Canonical name) Вы как бы добавляете компьютеру еще одно имя, которое при разрешении превратится в тот же IP-адрес, что и основное имя.

www	IN	CNAME	ns.vt
-----	----	-------	-------

Допускается использовать в качестве канонического имени и alias, лишь бы такая цепочка псевдонимов не замыкалась. В качестве **CNAME** может выступать любой именной адрес в сети, не обязательно из текущего домена, например:

other	IN	CNAME	www.sun.com.
-------	----	-------	--------------

Заметим еще один момент. Второе имя компьютеру можно дать и так:

mail	IN	A	193.233.88.39
relay	IN	A	193.233.88.39

Следующий важный тип записей - записи типа MX.

MX - (Mail eXchange) - пересылка почтовых сообщений. Они обычно следуют за записями типа 'A' или 'SOA'. Используются они обычно так:

[domain] IN MX [pref_value] [mail_server]

где domain - необязательное поле. Если оно есть, то запись относится к этому домену, если нет - то к предыдущему с типом 'A'. В том случае, если на месте [domain] стоит символ '@' или это поле отсутствует, но сама запись находится в самом начале файла зоны (сразу же после SOA), то такое поле MX будет относиться к домену, которому соответствует текущий файл описания.

'IN' также можно указывать, а можно опускать. mail_server - доменное имя почтового сервера, которому достанется вся почта, приходящая на этот домен. На этом сервере должны стоять программы, поддерживающие почтовый протокол SMTP. Для повышения надежности пересылки почты, вы можете указать подряд несколько почтовых серверов. В том случае, если один из них перестанет работать, вся почта на домен будет отправляться на эти "запасные" mail-сервера, которые при первой же возможности отправят все накопившиеся у них электронные письма на основной почтовый сервер. Чтобы регулировать такую систему, и вводится параметр pref_value (приоритет соответствующего почтового сервера), который может меняться от 0 (самый большой приоритет) до 32767 (минимальный приоритет). Если у вас запись MX единственная для какого-то домена, то значение этого поля не играет роли. Но при использовании только одного mail-сервера в те моменты, когда он по каким-либо причинам недоступен, почта будет теряться. Поэтому обыкновенно ставят две-три MX-записи, а приоритеты у них устанавливают круглыми числами от 10 до 100. Пример:

alice	IN	A	193.233.88.39
	MX	10	mail

Следующие типы записей используются реже, тем не менее, приведем их тут:

NULL - пустая запись. Скорее всего, должна использоваться для резервирования доменного имени, но на практике применяется только в том случае, если необходимо убрать на некоторое время из DNS'a запись о какой-то машине. Хотя проще такие строчки просто закомментировать.

RP - (Responsible Person) - ответственное за этот домен лицо:

```
xerox          IN      RP      Ivanov Ivan
```

HINFO - (Host Information) - информация о компьютере (тип процессора, операционная система и т.д.). Используется крайне редко.

Нам осталось рассмотреть последний особый тип записи '**PTR**' (**domain name pointer**), эта запись используется в файлах так называемой "обратной зоны" (**reverse-dns**).

В рассматриваемых до этого примерах мы видели, как установить взаимосвязь между символьными именами и числовыми IP-адресами, но только в одну сторону: от символьных имен к числовым IP-адресам. Доменная система имен должна также обеспечивать обратное преобразование - из числового адреса в символьное имя. Для этих целей, собственно и служат файлы обратной зоны (Reverse-DNS), структура которых во многом напоминает файлы зон, которые мы рассматривали выше. Для обратных зон также необходимы primary и secondary сервера. В конфигурационном файле для установки primary name-сервера может быть написана примерно такая строка:

```
primary 88.233.193.in-addr.arpa rev.stup.zone
```

Домен '**in-addr.arpa**' - служебный. Он используется для обозначения числовых IP-адресов.

При таком способе записи от вашего полного IP адреса из 4-х чисел остаются только первые три, общие для компьютеров внутри вашей локальной сети. Причем, эти числа меняются местами.

Secondary name-сервера обратных зон устанавливаются аналогично прямым зонам.

```
secondary 100.250.158.in-addr.arpa 158.250.100.1 158.250.100.zone
```

В обратной зоне в основном используются записи типа PTR. Структура записи такова:

```
[ip_address #4] IN PTR [имя.домена]
```

где **ip_address #4** - последнее из 4-х чисел IP-адреса (его значения могут быть от 0 до 255). Первые три компонента задаются для данного файла обратной зоны в файле конфигурации. Запись PTR задает соответствие между IP-адресом и именем домена. Для правильной работы необходимо соответствие числовых и символьных адресов в прямых и обратных зонах, иначе результат будет непредсказуем.

```
1          IN      PTR      www.firm.ru.
```

В файле обратной зоны первой записью стоит запись типа SOA, полностью аналогичная SOA прямой зоны. После этого идет перечисление name-серверов (записи типа NS). И особняком стоит строка, которая обозначает весь Ethernet-участок сети. Прописывать нулевой адрес необязательно, но это считается правилом хорошего тона.

3. Менеджеры сетевых служб

Существуют различные варианты запуска сетевых сервисов. Наиболее распространенный вариант – когда программа, ответственная за функционирование сервиса, постоянно находится в памяти и обрабатывает все поступающие запросы. Такая реализация позволяет оперативно реагировать на входящие соединения. Второй вариант реализации – использование единой

программы, которая отслеживает входящие соединения на нескольких портах, и при появлении такого запускает приложение, обеспечивающее соответствующий сервис.

В первом случае для каждого поддерживаемого сервиса в памяти постоянно присутствует соответствующее приложение. Это оправдано в том случае, когда необходима максимальная оперативность при обработке входящих соединений или когда поток этих соединений достаточно велик (например HTTP или FTP).

Реализация второго подхода также дает свои преимущества в случаях, когда входящие соединения редки и 1-2 секунды задержки не играют существенной роли. При этом в памяти находится только одно небольшое приложение, которое устанавливает соединение и только затем загружает соответствующее программное обеспечение для работы с ним. При этом такая программа одновременно может контролировать множество портов.

inetd

Рассмотрим наиболее распространенное приложение, обеспечивающее выполнение данных функций (***inetd***).

Основным файлом конфигурации для сервиса ***inetd*** является файл */etc/inetd.conf*. Рассмотрим его структуру подробнее.

Каждая строка этого файла представляет собой описание одного из контролируемых им сервисов. Если в строке присутствует знак «#», то весь текст, следующий за ним, считается комментарием. Формат строки этого конфигурационного файла следующий:

<service_name> <sock_type> <proto> <flags> <user> <server_path> <args>

<service_name> – символьное имя сервиса;

<sock_type> – тип сокета, который использует данный сервис (stream, dgram, raw, rdm, seqpacket);

<proto> – название протокола (tcp, udp, tcp4, udp4, tcp6, udp6, tcp46, udp46) остальные протоколы описаны в файле */etc/protocols* (гpc сервисы не рассматриваем);

<flags> – флаги (wait и nowait) применимы только к пакетным сервисам (с типом dgram), потоковые сервисы всегда должны использовать флаг nowait;

<user> – имя пользователя, с правами которого будет выполняться серверное приложение;

<server_path> – полный путь до серверного приложения;

<args> – аргументы серверного приложения, перечисляются через пробел до конца строки или знака комментария.

Имя сервиса также определяет и номер порта, на котором ***inetd*** будет ожидать входящих соединений для данного сервиса. Список всех сервисов с указанием, какие порты для них определены, находится в файле */etc/services*.

Поле «протокол» определяет сетевой протокол для данного сервиса. Типы протоколов tcp и udp соответствуют tcp4 и udp4 для обратной совместимости. При указании их в качестве протокола входящие запросы будут обрабатываться только от клиентов в сетях IPv4. При необходимости обрабатывать запросы от клиентов в сетях IPv6 необходимо указывать tcp6 и udp6. В случае, если необходимо обрабатывать входящие запросы от всех клиентов следует указать tcp46 и udp46.

Флаг wait используется только для пакетных протоколов и задает необходимость передавать все запросы в одно приложение, которое будет их обрабатывать. Для всех остальных типов протоколов следует указывать флаг nowait. Также можно указать максимальное количество запускаемых в течении 1 минуты приложений для каждого протокола. Для этого после флага через символ «.» необходимо указать число, которое и будет определять это количество. Если число не указано, по умолчанию используется значение 40.

В поле имя пользователя необходимо указать имя реально существующего в системе пользователя. Также, дополнительно, можно указать и имя группы. Оно добавляется после имени пользователя, а разделителем является символ «.». Соответствующее серверное приложение будет выполняться с правами указанного пользователя и группы.

В качестве пути до файла серверного приложения может быть указано ключевое слово `internal`, в этом случае обработка входящих соединений будет производиться самим сервисом `inetd`.

Пример:

<code>echo</code>	<code>stream</code>	<code>tcp</code>	<code>nowait</code>	<code>root</code>	<code>internal</code>
<code>echo</code>	<code>dgram</code>	<code>udp</code>	<code>wait</code>	<code>root</code>	<code>internal</code>
<code>time</code>	<code>stream</code>	<code>tcp</code>	<code>nowait</code>	<code>root</code>	<code>internal</code>
<code>time</code>	<code>dgram</code>	<code>udp</code>	<code>wait</code>	<code>root</code>	<code>internal</code>
<code>ftp</code>	<code>stream</code>	<code>tcp</code>	<code>nowait</code>	<code>root</code>	<code>/usr/sbin/tcpd</code> <code>vsftpd</code>

Следует отметить, что в качестве серверного приложения может быть использован `wrapperr`. Это промежуточное приложение, которое позволяет ввести дополнительный контроль, например, на доступ к сервису с определенных адресов и т.д.

Управление стандартным `wrapperr` осуществляется при помощи двух файлов:

```
/etc/hosts.allow  
/etc/hosts.deny
```

Записи в файле `hosts.allow` проверяются первым, и его правила проверяются сверху вниз. В случае нахождения правила с разрешением (то есть, соответствие хоста, домена, маски подсети или других сетевых параметров), доступ к сервису открывается. Если ничего подходящего не найдено, проверяются записи в файле `hosts.deny` в поисках таких же правил. Этот файл ничем не отличается от предыдущего, кроме того, что все данные в нем интерпретируются, как запреты. Первое же найденное правило приведет к тому, что в доступе будет отказано. Если ничего не найдено, доступ будет предоставлен.

С помощью этих двух файлов можно организовать два варианта политики безопасности: разрешено все, что не запрещено и запрещено все, что не разрешено. Для реализации первого варианта политики безопасности необходимо в файле `hosts.deny` добавлять запрещающие записи, а файл `hosts.allow` оставить пустым. Для реализации второго варианта достаточно в файле `hosts.deny` сделать одну запись:

```
ALL: 0.0.0.0/0.0.0.0
```

А в файл `hosts.allow` вписывать разрешающие правила.

Формат файлов `hosts.allow` и `hosts.deny` следующий:

```
<service_name>: <ip addr>/<mask>
```

В качестве `<service_name>` указывается имя сервиса аналогичное имени в файле `inetd.conf`. `<ip addr>/<mask>` определяют подсеть, для которой будет выполняться правило.

Пример:

```
in.telnetd: 10.0.0.0/255.255.255.0  
in.ftpd: 0.0.0.0/0.0.0.0
```

Также в качестве `<service_name>` может быть указано ключевое слово `ALL`, которое определяет, что правило будет действовать на все сервисы.

xinetd

На данный момент существует более новая реализация сервиса, в качестве которой выступает программа **`xinetd`**.

Конфигурационным файлом для этой программы является файл `/etc/xinetd.conf`.

Для описания сервиса в этом файле используется следующий формат:

```
service <service_name>  
{
```

```

<attribute> <assign_op> <value> <value> ...
...
}

```

Оператор присвоения, *assign_op*, может быть один из '=', '+=', '-='. Большинство атрибутов поддерживают только один оператор присваивания, '='. Атрибуты являющиеся наборами значений поддерживают все операторы присваивания. Для любых атрибутов, '+=' означает добавление значения к набору и '-=' означает удаление значения из набора. Список этих атрибутов должен быть указан после того, как все атрибуты описаны.

id

Этот атрибут используется для однозначного определения сервиса. Он используется потому, что есть сервисы, которые могут использовать различные протоколы и их необходимо описать в различных секциях файла конфигурации. По умолчанию, значение поля *id* совпадает с именем сервиса.

type

Может быть использована любая комбинация из следующих значений:

RPC – RPC сервис

INTERNAL – сервис предоставляемый *xinetd*.

UNLISTED – сервис не указан в стандартном (системном) файле (например */etc/rpc* для RPC служб, или */etc/services* для не RPC сервисов).

flags

Используется любая комбинация следующих флагов:

REUSE – Устанавливает флаг *SO_REUSEADDR* для сокета сервиса (этот порт может быть использован более чем одним приложением).

INTERCEPT – перехватывает пакеты или установленные соединения в порядке их поступления из разрешенных мест.

NORETRY – Избегать повторных попыток в случае неудачи запуска программы, обрабатывающей запросы данного сервиса.

IDONLY – Разрешать соединения только в случае идентификации удаленного пользователя на удаленной стороне (т.е. на удаленном хосте должен быть запущен сервер идентификации *ident*). Этот флаг применяется только к сервисам, поддерживающим соединение. Этот флаг не эффективен, если *USERID* параметр настройки логов не используется.

NAMEINARGS – Заставляет рассматривать первый аргумент в "server_args" как argv[0] когда выполняется программа-сервер, указанная в "server". Это позволяет использовать tcpd путем помещения tcpd в "server" и имя сервера в "server_args" как в обычном inetd.

NODELAY – Если служба является tcp сервисом и флаг *NODELAY* установлен, тогда флаг *TCP_NODELAY* устанавливается для сокета. Актуальна только для tcp сервисов.

socket_type (аналогичен соответствующему параметру inetd)

Возможные значения для этого атрибута:

stream – потоковый сервис

dgram – пакетный сервис

raw – сервис требующий прямого доступа к IP

seqpacket – сервис требующий reliable sequential datagram transmission

protocol (аналогичен соответствующему параметру inetd)

определяет протокол который используется сервисом. Протокол должен присутствовать в */etc/protocols*. Если этот параметр не определен, сервис будет использовать протокол по умолчанию.

wait (аналогичен соответствующему параметру inetd)

Этот атрибут определяет, является ли сервис однопоточным (значение *yes*) или многопоточным (значение *no*).

user

определяет *uid*, под которым будет выполняться процесс. В файле */etc/passwd* должна существовать запись соответствующая указанному идентификатору пользователя. Этот

параметр не действует, если эффективный **uid xinetd** не соответствует идентификатору супер-пользователя.

group

аналогично **uid**, но только для группы пользователей.

instances

определяет число серверов, которое может быть одновременно активировано для одного сервиса (по умолчанию ограничений нет). Значение этого параметра должно быть числом или ключевым словом **UNLIMITED**

nice

определяет приоритет, с которым выполняются процессы сервера. Значением является число (возможно отрицательное) См. описание команды **nice** для дополнительной информации.

server

определяет программу, обеспечивающую данный сервис.

server_args

определяет аргументы, переданные серверу. В отличие от **inetd**, имя сервера *не* включается в **server_args**.

only_from

определяет удаленный хост, для которого данный сервис разрешен. Значением является список IP адресов, который может быть указан как комбинация следующих вариантов:

а) в числовой форме **%d.%d.%d.%d**. Если правые компоненты указаны как "0", то они трактуются как знаки подстановки. Например, 128.138.12.0 соответствует всем хостам в сети 128.138.12.0/24, значение 0.0.0.0 соответствует любому Интернет адресу.

б) перечисление адресов в форме **%d.%d.%d.{%d,%d,...}**. Производится подстановка адресов из списка. Таким образом, можно указать только несколько адресов, а не всю подсеть.

в) указать имя сети описанной в файле **/etc/networks**

г) имя хоста. Когда соединение с **xinetd** устанавливается, он выполняет обратное преобразование и каноническое имя, полученное в результате этого преобразования, сравнивается с указанным именем хоста. Также можно использовать имена доменов в форме **.domain.com**. Необходимо чтобы в результате обратного преобразования адреса попадал в **.domain.com**. Для обратного преобразования используется запрос к DNS серверу и если такое преобразование невозможно для текущего соединения (соответствие отсутствует), будет большая задержка перед началом обработки каждого соединения.

д) указать **ip** адрес/маску подсети в формате 1.2.3.4/32.

no_access

определяет удаленный хост, для которого данный сервис недоступен. Значение может быть указано так же как для параметра **only_from**. Эти два атрибута определяют методы управления доступом реализованные в **xinetd**. Если эти параметры не указаны для сервиса, то он считается доступным для всех. Если оба параметра указаны, срабатывает тот который наиболее точно совпадает с удаленным хостом. Например:

only_from содержит 128.138.209.0 и

no_access содержит 128.138.209.10 в этом случае хост с адресом 128.138.209.10 не получит доступ к сервису.

access_times

определяет промежутки времени, в которые сервис доступен. Интервал указывается в форме **ЧЧ:ММ-ЧЧ:ММ** (соединение *должно* быть доступно в том числе и на границах указанного временного интервала). Часы указываются в диапазоне с 0 до 23 и минуты с 0 до 59.

log_type

определяет куда направляется вывод логов. Существует два формата:

syslog "facility [level]" - Регистрация событий происходит с помощью указанной **facility** сервиса syslog. Возможные **facility**: *daemon, auth, user, local0-7*. Возможные **level**: *emerg, alert, crit, err, warning, notice, info, debug*. Если уровень не указан, по умолчанию принимается *info*.

FILE "file [soft_limit [hard_limit]]" – Все регистрируемые события добавляются в *файл* который создается если еще не существует. Дополнительно можно указать два ограничения на размер файла. Первое ограничение – "мягкое", если **xinetd** достигает указанного значения, то это событие регистрируется используя syslog с уровнем *alert*. Второе ограничение – жесткое: **xinetd** прекращает запись событий в лог-файл когда его размер превышает это ограничение и извещает о происшедшем через систему syslog с уровнем *alert*. Если жесткое ограничение не указано, по умолчанию он принимается на 1% более, чем мягкое ограничение, но при этом приращение должно находиться в диапазоне **LOG_EXTRA_MIN** и **LOG_EXTRA_MAX**, которые по умолчанию определены как 5Кбайт и 20Кбайт.

log_on_success

определяет, какая информация записывается, когда сервер стартует и когда завершает работу (id сервиса всегда включается в запись). Любая комбинация следующих значений может быть указана:

PID – записывает PID сервера (если сервис предоставлен самим **xinetd** без запуска нового процесса записанный PID=0)

HOST – записывает адрес удаленного хоста

USERID – записывает id пользователя на удаленном хосте используя протокол идентификации RFC 1413. Эта опция доступна только для много-нитевых сервисов, ориентированных на работу с поточными соединениями.

EXIT – фиксирует факт завершения работы сервера, код завершения или сигнал по которому завершился процесс.

DURATION – регистрирует длительность сессии

log_on_failure

определяет, какая информация регистрируется, когда сервер не может быть запущен (даже по причине недостатка ресурсов или ограничений системы контроля доступа). ID сервиса всегда включается в запись вместе с причиной неудачи запуска. Любая комбинация следующих значений может быть указана:

HOST – регистрирует IP адрес удаленного хоста.

USERID – регистрирует id пользователя на удаленном хосте используя протокол идентификации (RFC 1413) Эта опция доступна только для много-нитевых сервисов, ориентированных на работу с поточными соединениями.

ATTEMPT – регистрирует неудачные попытки соединения.

RECORD – записывает информацию с удаленной стороны в случае если сервер не может быть запущен. Это позволяет отслеживать попытки использования сервиса. Например, login сервис фиксирует имя локального пользователя, удаленного пользователя и тип терминала. В настоящий момент сервисы, поддерживающие эту опцию: *login, shell, exec, finger*.

rpc_version

определяет версию RPC для RPC сервиса. Версия может быть просто номером или диапазоном в форме *number-number*.

env

Значение этого атрибута является списком строк в виде 'name=value'. Эти строки добавляются к существующему окружению перед стартом сервера.

passenv

Значение этого атрибута – список переменных окружения **xinetd** которые должны быть переданы в окружение сервера. Пустой список означает, что переменные не передаются за исключением тех что явно определены с использованием параметра *env*.

port

определяет порт, используемый сервисом. Если этот атрибут указывается для сервиса присутствующего в */etc/services*, он должен совпадать со значением указанным в этом файле.

redirect

Позволяет перенаправлять tcp соединения на другой компьютер. Когда приходит запрос на соединение на соответствующем порту **xinetd** порождает процесс, который устанавливает соединение с указанными хостом и портом, и переправляет все данные между двумя

хостами. Эта опция используется в том случае, когда внутренние машины не видны снаружи. Синтаксис следующий:

redirect = (ip address) (port)

Вместо IP адреса можно использовать имя хоста. Разрешение имени производится только однажды, когда **xinetd** стартует и полученный IP адрес используется до тех пор пока **xinetd** будет перегружен. Параметр "server" не требуется, когда эта опция указана и если "server" и "redirect" указаны совместно то "redirect" имеет приоритет.

bind

Позволяет привязать сервис к указанному интерфейсу. Это означает, что можно настроить телнет-сервер чтобы он слушал только на защищенном внутреннем интерфейсе. Или один порт на одном интерфейсе может выполнять одну функцию, а на другом интерфейсе совершенно другую. Синтаксис:

bind = (ip адрес интерфейса).

interface

Синоним **bind**.

banner

Имя файла содержимое которого отправляется/отображается на удаленный хост когда соединение с сервисом установлено. Этот баннер печатается вне зависимости от контроля доступа. Он всегда печатается при установлении соединения. Актуально для протоколов аналогичных *telnet*.

banner_success

Имя файла, содержимое которого будет отображено на удаленном хосте когда соединение с сервисом разрешено. Актуально для протоколов аналогичных *telnet*.

banner_fail

Имя файла, содержимое которого будет отображено на удаленном хосте когда в соединении с соответствующим сервисом будет отказано. Актуально для протоколов аналогичных *telnet*.

per_source

Может быть целым числом или ключевым словом "**UNLIMITED**". Параметр определяет максимальное число экземпляров серверных процессов указанного сервиса на IP адрес источника. Значение также может быть указано в секции значений по умолчанию.

cps

Ограничивает скорость обработки входящих соединений. Параметр имеет два аргумента. Первый аргумент – число обрабатываемых входящих соединений в секунду. Если число соединений превышает указанное значение, то сервис становится временно недоступным. Второй аргумент – временной интервал в секундах, который необходимо выдержать перед восстановлением доступности сервиса.

max_load

Принимает вещественное значение, показывающее при каком значении загрузки системы сервис должен прекратить обработку запросов и установление соединений. Например: 2 или 2.5 Сервис должен прекратить обрабатывать запросы при указанном значении загрузки. В настоящее время поддерживается только в Linux и Solaris.

Нет необходимости указывать все вышеперечисленные параметры для каждого сервиса. Необходимыми являются следующие:

socket_type

user – только для внешних сервисов

server – только для внешних сервисов

wait

protocol – только для *RPC* и *unlisted* сервисов

rpc_version – только для *RPC* сервисов

rpc_number – только для *RPC* сервисов

port – только для *RPC* сервисов (не *RPC*)

Следующие атрибуты поддерживают все операторы присваивания:

```
only_from
no_access
log_on_success
log_on_failure
passenv
env – (не поддерживает оператор '==')
```

Также эти атрибуты могут появляться более чем однажды в записи описывающей сервис. Остальные атрибуты поддерживают только оператор '=' и могут появляться в секции описывающей сервис лишь однажды.

Файл конфигурации может содержать секцию для задания значений по умолчанию, она имеет вид:

```
defaults
{
<атрибут> = <значение> <значение> ...
...
}
```

Эта секция задает значения атрибутов для всех секций сервисов где явно не описаны эти атрибуты. Возможные атрибуты:

```
log_type
log_on_success – (кумулятивный эффект)
log_on_failure – (кумулятивный эффект)
only_from – (кумулятивный эффект)
no_access – (кумулятивный эффект)
passenv – (кумулятивный эффект)
instances
disabled – (кумулятивный эффект)
enabled – (кумулятивный эффект)
```

Атрибуты с кумулятивным эффектом можно указывать много раз, причем значения будут накапливаться (то есть '=' делает тоже самое, что и '+='). За исключением **disabled** они имеют то же значение, как если бы были указаны в секции сервиса.

disabled определяет сервисы, которые не доступны, даже если они имеют собственные секции в файле конфигурации. Это позволяет быстро переконфигурировать **xinetd**, указав нежелательные сервисы как аргументы для **disabled** вместо того чтобы комментировать соответствующие секции. Значением этого атрибута является список разделенных пробелами идентификаторов сервиса (id)

enabled имеет те же свойства что и **disabled**. Разница в том, что **enabled** список разрешенных сервисов. Если атрибут **enabled** указан, то только разрешенные сервисы будут доступны. Если **enabled** не указан, то подразумевается что все сервисы разрешены за исключением перечисленных в **disabled**.

xinetd предоставляет следующие сервисы собственными силами (как потоковые, так и пакетные): *echo*, *time*, *daytime*, *chargen* и *discard*. На эти сервисы действуют те же правила и настройки, как и на все остальные за исключением того, что в этом случае **xinetd** не порождает новых процессов. По этому для этих сервисов нет ограничений на количество запускаемых процессов.

Существует так же административный интерфейс, выполненный в виде внутреннего сервиса. Резервированным именем сервиса является "**xadmin**" и он всегда является внутренним сервисом. Для этого сервиса нужно указать номер порта и задать правила доступа, потому что никакой парольной защиты не существует. Для доступа к данному интерфейсу необходимо использовать программу telnet.

4. Apache

Сервер Apache имеет три файла конфигурации, они находятся в каталоге `/etc/httpd/`. Эти файлы позволяют настроить все аспекты функционирования сервера. В данном каталоге присутствуют следующие файлы: `httpd.conf`, `srm.conf`, `access.conf`. Основные настройки находятся в файле `httpd.conf`.

Рассмотрим основные изменения в файлах конфигурации, которые необходимо сделать, прежде чем запускать сервер.

Файл **`httpd.conf`** содержит конфигурацию сервера.

ServerType

Для этой директивы значением по умолчанию является *standalone*. Серверы, работающие в автономном режиме (*standalone*), запускаются из загрузочных сценариев при запуске системы. В качестве альтернативы можно использовать значение *inetd*.

Использование *inetd* идеально для сравнительно редко используемых служб, к которым протокол HTTP не относится. Поскольку для каждой HTML-страницы, для каждого встроенного изображения и любого другого объекта, запрашиваемого пользователем, требуется отдельное соединение и, следовательно, отдельная копия сервера, то каждую минуту, если не каждую секунду, будет осуществляться несколько новых соединений. Кроме того, сервер Apache должен считывать информацию из файла конфигурации и производить ее синтаксический разбор при каждом запуске своей копии демоном *inetd*. Такие накладные расходы неприемлемы. Практически для каждого HTTP-сервера следует использовать в этой директиве режим *standalone*.

Port

В этой директиве задается номер сетевого порта, на котором будет работать сервер, если он запущен в автономном режиме (если используется *inetd*, то номер порта следует задать в файле `/etc/services`).

Значением по умолчанию для этой директивы является *Port 80*. Если необходимо запустить сервер на этом порту или на любом другом, номер которого меньше 1024, то потребуются привилегии суперпользователя (*root*).

HostnameLookups

В целях наблюдения за деятельностью пользователей сервер Apache ведет серию журнальных файлов. В одном из них ведется учет компьютеров, осуществивших доступ к серверу. Директива *HostnameLookups* указывает, записывается ли в журнальный файл имя компьютера (например, **`alice.stup.ac.ru`**) или только его IP-адрес (например, `193.233.88.39`). По умолчанию сервер сохраняет имя компьютера, но если ожидаемый объем трафика очень велик, отключение этой опции позволит уменьшить нагрузку на сервер. Также не все адреса имеют соответствие с DNS именем, что может вызывать существенные задержки в ответах. Отключение производится изменением строки на *HostnameLookups off*.

User и Group

Эти параметры задают действительные идентификаторы пользователя и группы, которые присваиваются серверу, работающему в автономном режиме. Можно задавать как имена, так и их числовые эквиваленты, которые можно найти в файлах `/etc/passwd` и `/etc/group` соответственно.

По умолчанию в качестве имени пользователя принимается `wwwrun` (в дистрибутиве SuSE Linux). Ни в коем случае нельзя запускать сервер с привилегиями суперпользователя (*root*).

В качестве идентификатора группы следует использовать идентификатор какой-нибудь нейтральной группы, имеющейся в системе (например *nobody*).

ServerAdmin

Это официальный электронный адрес вебмастера Web-узла. Обычно используется адрес в форме `WebMaster@server_name`.

ServerRoot

В этой директиве задается базовый каталог HTTP-сервера Apache.

BindAddress

Эта директива используется только для компьютеров, имеющих более одного IP-адреса. С ее помощью можно устанавливать, на каком из IP-адресов компьютера сервер будет ожидать входящих соединений.

ErrorLog и TransferLog

При помощи этих двух директив задается местоположение журнальных файлов, в которых регистрируются ошибки и попытки доступа к серверу соответственно. Сервер подразумевает, что имена, начинающиеся не с косой черты (/), отсчитываются от каталога *ServerRoot*.

В файле, указанном в директиве *ErrorLog*, сервер сохраняет сообщения диагностики, включая сообщения об ошибках, выдаваемые сценариями CGI. В файле, указанном в директиве *TransferLog*, сервер сохраняет все запросы клиентов. Если включена описанная выше опция *HostnameLookups*, то вместе с запросами регистрируются имена компьютеров. Если опция выключена, то регистрируются только IP-адреса компьютеров клиентов.

PidFile и ScoreBoardFile

Если сервер работает в автономном режиме, то в директиве *PidFile* задается имя файла, в котором исходный процесс-сервер регистрируется, указывая свой идентификационный номер. Эту информацию можно использовать для завершения или перезапуска сервера.

Директива *ScoreBoardFile* позволяет серверу Apache следить за собственной производительностью.

ServerName

В этой строке должно стоять официальное имя сервера в том виде, в котором оно появляется в строке URL (то есть *http://www.ИМЯ_вашего_сервера/*). Это должно быть имя компьютера, зарегистрированное в DNS организации или провайдера.

Timeout

Это промежуток времени в секундах, в течение которого сервер ждет продолжения недополученного запроса или продолжает попытки возобновления приостановленной передачи ответа. Если сервер расположен на медленном соединении с Интернетом или требуется передавать большие файлы, следует увеличить это значение.

KeepAlive и KeepAliveTimeout

Характеристики *KeepAlive* являются свойствами протокола HTTP 1.1, позволяющими ускорить обработку запросов. HTTP 1.0 при каждом запросе на передачу объекта создается новое соединение между клиентом и сервером. В сервере Apache реализована свойственная протоколу HTTP 1.1 возможность *KeepAlive*, что означает возможность запрашивать несколько объектов в рамках одного соединения. Например, раньше передача Web-страницы с четырьмя встроенными изображениями потребовала бы пять отдельных соединений, а с использованием *KeepAlive* все последовательные запросы производятся в рамках одного соединения. Это сокращает время обработки запросов сервером.

Значением по умолчанию для *KeepAlive* является 5 – максимальное число запросов в рамках одного соединения. Когда значение этого параметра равно 5, наблюдается увеличение производительности. Чтобы отключить эту возможность, установите параметр *KeepAlive* равным нулю. Параметр *KeepAliveTimeout* отражает промежуток времени между последовательными запросами. Другими словами, если клиент в рамках одного соединения производит три запроса, а затем в течение нескольких секунд запросов от него не поступает, сервер считает, что третий запрос был последним. По умолчанию принимается значение 15 секунд.

StartServers

Когда выбирается автономный (*standalone*) режим работы сервера (*ServerType*) и для параметра *StartServers* задается значение, большее 1 (по умолчанию принимается значение 5), происходит коренное изменение работы сервера. Вместо запуска одного экземпляра сервер при старте создает несколько собственных копий, при этом образуется *пул серверов*.

Исходная копия сервера выступает в качестве планировщика для всех серверов в пуле, принимая соединения и передавая их свободным копиям. Теоретически, такая стратегия ускоряет процесс обслуживания запросов, снижая накладные расходы.

MaxSpareServers и MinSpareServers

Если число поступающих запросов превышает число серверов в пуле, заданное параметром *StartServers*, буфер серверов увеличивается для обслуживания запросов. Эти дополнительные процессы-серверы не завершаются после обработки запроса, ради которого они были запущены; они остаются в памяти. Директива *MaxSpareServers* позволяет настраивать число свободных серверов, находящихся в пуле. Если их больше, чем указано в директиве *MaxSpareServers*, то лишние процессы завершаются. Аналогично, если свободных серверов в пуле меньше, чем допускает директива *MinSpareServers*, то в преддверии наплыва запросов создаются дополнительные копии сервера.

MaxClients

Поскольку Web-серверы обрабатывают большое количество запросов от многочисленных клиентов, это может привести к запуску такого количества серверов, что компьютер перестанет справляться с нагрузкой. Директива *MaxClients* устанавливает максимальное число копий сервера, которые могут выполняться одновременно. Когда достигается этот предел (по умолчанию 150), новые запросы получают отказ.

MaxRequestsPerChild

В этой директиве задается время жизни любого отдельного сервера в пуле серверов. Обработав установленное здесь количество запросов (по умолчанию 30), копия сервера завершается, а вместо нее запускается новая. В большинстве систем изменение этого параметра не дает заметного эффекта.

Listen

Использование виртуальных серверов позволяет придать адресу URL такой вид, будто он указывает на отдельный узел, даже если в действительности за ним стоит только часть большого сервера.

DocumentRoot

В этой директиве задается каталог, из которого берутся передаваемые клиентам документы.

UserDir

В этой директиве задается название подкаталога в домашнем каталоге пользователя, из которого берутся документы.

DirectoryIndex

Эта директива позволяет задать название документа, возвращаемого по запросу, который не содержит в строке URL названия документа.

В директиве *DirectoryIndex* можно задать несколько имен файлов. Если первый документ, указанный в строке, не найден в каталоге, то сервер ищет следующий и, в случае успеха, передает его клиенту.

DirectoryIndex index.html index.htm

В этом примере, если в одном и том же каталоге присутствуют файлы **index.html** и **index.htm**, в качестве индексной страницы будет передан файл **index.html**, поскольку он стоит в списке первым.

Redirect

Директива *Redirect* оказывается полезной, когда возникает неизбежная необходимость в переносе документов в другой каталог на сервере или даже на другой сервер. Например:

```
Redirect /Linux http://www.linux.ru/
```

В результате, все запросы на документы, касающиеся Linux и имеющие старый адрес, будут перенаправлены на указанный сервер.

Alias

Директива *Alias* дает возможность предоставлять доступ к документам, находящимся не только в каталоге, указанном в директиве *DocumentRoot*, и его подкаталогах, но и в других каталогах.

ScriptAlias

В директиве *ScriptAlias* указывается, в каких каталогах разрешен запуск сценариев CGI. Например:

```
ScriptAlias /cgi-bin/ /srv/www/cgi-bin/
```

В первом аргументе указывается виртуальный путь, который клиенты могут использовать для запуска сценариев CGI. Во втором аргументе указывается каталог, к которому осуществляется доступ в действительности.

Разрешается добавлять неограниченное число директив *ScriptAlias*.

ErrorDocument

Эта директива позволяет поставить в соответствие кодам ошибок HTTP-сервера адреса URL на том же сервере.

Для управления доступом к файлам и каталогам используются директивы `<File></File>` и `<Directory></Directory>`

Например, с помощью этих директив можно ограничить доступ к определенным файлам и каталогам. А также настроить и другие их параметры.

Имя каталога должно быть абсолютным (разрешено использование шаблонов). В основном, используется следующий синтаксис:

```
<Directory directory_name>  
    access control directives  
</Directory>
```

Эти директивы управления доступом (access control directives) управляют типом доступа, предоставляемого сервером к указанному каталогу и всем расположенным в нем файлам и каталогам.

Allow/Override

Эта директива регулирует отношения между глобальным файлом доступа и пользовательскими файлами **.htaccess**. В директиве указываются разделенные пробелами опции управления доступом, переопределение которых в файле **.htaccess**, находящемся в дереве каталогов ниже этого каталога, разрешено. Возможные значения директивы:

None – Сервер игнорирует файлы **.htaccess** в этом каталоге.

ALL – По умолчанию пользователи имеют право переопределять в файлах **.htaccess** все глобальные установки доступа, для которых в этих файлах имеются эквивалентные команды.

Options – Разрешает использование директивы *Options*

AuthConfig – Разрешает использование директив *AuthName*, *AuthType*, *AuthUserFile* и *AuthGroupFile*, необходимых для защиты каталогов паролями.

Options

Директива *Options* позволяет управлять тем, какие функции сервера доступны для использования в каталоге, указанном в секции *<Directory>* или в файле **.htaccess**. В директиве *Options* можно использовать следующие аргументы:

None – В указанном каталоге не разрешается использование каких-либо функций.

All – В указанном каталоге разрешается использование всех возможностей (принимается по умолчанию).

FollowSymLinks – Сервер следует символическим ссылкам, имеющимся в указанном каталоге.

SymLinksIfOwnerMatch – Сервер следует символическим ссылкам, только если каталог назначения принадлежит тому же пользователю, что и сама ссылка.

ExecCGI – В указанном каталоге разрешается выполнение сценариев CGI.

Includes – В указанном каталоге разрешено использование серверных включений. Использование серверных включений требует, чтобы сервер производил синтаксический разбор всех HTML-файлов перед отправкой их клиентам.

Indexes – Сервер разрешает пользователям заказывать индексную страницу указанного каталога. Выключение этой опции запрещает передачу только списка файлов в каталоге, но не файла **index.html**.

IncludesNoExec – Эта директива разрешает использование в указанном каталоге серверных включений, но запрещает запуск из них внешних программ.

7. Firewall

Следующие возможности можно реализовать используя только ядро ОС Linux:

- Управлять полосой пропускания для разных компьютеров;
- Успешно разделять полосу пропускания между разными типами трафика;
- Приоритезировать трафик в полосе, согласно нашим нуждам (Quality of Service);
- Защищаться от DoS атак;
- Фильтровать трафик (firewall);
- Использовать один реальный адрес, для работы всей локальной сети в Интернет (SNAT);
- Объединять несколько каналов в один, и балансировать нагрузку;
- Ограничивать доступ к сервисам;
- Перебрасывать порты с одной машины на другую (DNAT);
- Ограничивать полосу пропускания, как входящего, так и исходящего трафика;
- Маршрутизировать по портам, адресам, MAC-адресу, TOS, времени дня, и даже по имени пользователя;
- Туннелирование, multicast routing, IPSec, и многое другое.

Все эти вещи обычно не нужны для рабочей станции, но жизненно необходимы маршрутизатору. Во многом возможности ядра Линукс на уровне, или даже опережают возможности Cisco IOS, но, к сожалению, отстают в этой области по документированности.

Все вышеперечисленные свойства ядра уже по умолчанию включены в него, и ими можно пользоваться. Но работают они в конфигурации «по умолчанию», то есть, ничего не делают. Все эти возможности присутствуют в ядрах с версии 2.2.x. Для того, чтобы задействовать, настроить, или посмотреть дополнительные возможности ядра, необходимо установить пакет *iproute2*. Центральная утилита пакета – *"ip"* помогает реализовать возможности многотабличной маршрутизации (*advanced routing*), туннелирования и *multicast routing*. Кроме того, Linux имеет гибкую систему управления

трафиком, называемую Traffic Control. Эта система поддерживает множество методов для классификации, приоритизации, разделения, и ограничения (shaping) обоих типов трафика – входящего и исходящего. Управлять всем этим позволяет вторая утилита пакета – "tc". Обе утилиты являются интерпретаторами команд, а все функции выполняет ядро.

Сразу после установки пакета iproute2 уже можно просматривать некоторую информацию системе. Например:

```
ip link list – покажет конфигурацию интерфейсов;  
ip address show – покажет IP адреса на интерфейсах;  
ip route show – покажет таблицу маршрутов main;  
ip neigh show – покажет таблицу ARP;  
ip rule list – покажет список правил, согласно которым принимается решение о маршрутизации.
```

По умолчанию, у ядра имеется три таблицы маршрутов – local, main и default. Старая утилита route показывает содержимое только таблицы main. Таблицы local и default – новые. Можно самостоятельно создать дополнительные таблицы маршрутов, и с помощью правил (ip rules) указывать, какой трафик маршрутизировать, согласно какой таблице. Классический механизм маршрутизации основан на destination address пакета. Кроме этого, можно управлять маршрутизацией, используя другие поля пакета. В этом и заключается концепция Advanced Routing.

Для начала рассмотрим вариант простейшей маршрутизации по адресу источника. Предположим, у нас есть высокоскоростной и дорогой канал передачи данных (xDSL) и медленная, но дешёвая коммутируемая линия (dial-up). Маршрут по умолчанию в основной таблице установлен на xDSL, но мы хотим одну из машин внутренней сети направить в нашу медленную связь, и, таким образом, освободить основной канал. Для реализации этого необходимо создать для этой машины отдельную таблицу маршрутов, которую назовем Manager:

```
echo 100 Manager >> /etc/iproute2/rt_tables
```

Далее создаем правило по адресу нашей выделенной машины, чтобы маршрутизация для нее переходила в новую таблицу:

```
ip rule add from 192.168.0.6 table Manager
```

Добавляем маршрут по умолчанию в таблицу Manager:

```
ip route add default via 192.168.0.1 dev ppp2 table Manager
```

Сбрасываем кэш маршрутов:

```
ip route flush cache
```

Здесь ppp2 – наше медленное соединение.

Рассмотрим ситуацию с разделением каналов. Предположим у нас есть два канала до разных провайдеров. Создаем для каждого свою таблицу маршрутов, и правила привязывающие соответствующие сети провайдеров к нужным интерфейсам. Это гарантирует нам возврат пакета, отправленного через одного из провайдеров от того же провайдера. Однако если маршрут по умолчанию будет указывать только на одно соединение, мы получим разбалансированную нагрузку по каналам. Для решения этой проблемы маршрут по умолчанию укажем на два устройства сразу:

```
ip route add default scope global nexthop via $ip_prov1 dev ppp0 weight 1 nexthop  
via $ip_prov2 dev ppp1 weight 1
```

Это сбалансировать маршруты через оба канала. Параметр weight может быть настроен для перевеса нагрузки в ту или иную сторону. Однако следует отметить, что в общем случае невозможно определить, по какому каналу приходит тот или иной пакет (не входящий, а исходящий).

Для управления трафиком в составе `iproute2` присутствует утилита `"tc"`.

Управление трафиком дает следующие возможности:

SHAPING – техника ограничения трафика в ту или иную сторону. Может применяться не только для разделения канала, но и для сглаживания бросков при пиковых нагрузках.

SCHEDULING – упорядочивание типов трафика в канале, позволяет избегать задержек для критичных типов трафика. Другими словами, это приоритезация (QoS).

POLICING – политика входящего трафика позволяет определить, когда трафик пропускать, а когда уничтожить (drop).

Процесс управления трафиком осуществляется с помощью трех взаимосвязанных частей – дисциплин (Queue Disciplines -> **qdiscs**), классов (**classes**) и фильтров (**filters**).

qdisc – алгоритм, который управляет очередью пакетов на интерфейсе, входящей (**ingress**), или исходящей (**egress**).

Classless Qdisc – **qdisc**, который не может иметь подклассов. Следовательно, отсутствует и разветвленная система обработки пакетов.

Classfull Qdisc – дисциплина, которая может содержать подклассы, которые, в свою очередь, могут содержать вложенные дисциплины и т.д.

Classes – логические контейнеры. Интерфейс с развитой системой классов, организует их в виде дерева. В нем конечные классы становятся листьями этой структуры (**leaves**), и всегда содержат **qdisc**.

Classifier – каждая, полная классов дисциплина, определяет с помощью классификации в какой класс отсылать пакет.

Filter - классификация происходит с использованием фильтров. Если значения фильтра совпали (например IP адрес), пакет следует согласно этому фильтру в нужный класс.

Scheduling – как уже было указано выше, это механизм, позволяющий определить, какому пакету в какой последовательности покидать **qdisc**.

Shaping – процесс задержки пакетов в **qdisc** или их отбрасывания. Shaping с постановкой в очередь (т.е. с задержкой) может выполняться только в **egress qdiscs**.

Важно понимать, что реально ограничивать мы можем только исходящий трафик.

Рассмотрим непосредственно алгоритмы управления трафиком.

Даже если пакет `iproute2` не установлен, ядро уже использует по умолчанию **classless qdisc**. Он называется **pfifo_fast**, и представляет собой трех-полосный буфер, приоритезация в котором может производиться с помощью поля Type Of Service (TOS), т.е. с помощью `iptables`. Однако это работает не очень хорошо, особенно в условиях высокой загрузки канала. Кроме того, бесклассовая архитектура этой дисциплины работает только для всего интерфейса в целом.

Наиболее часто используется **Token Bucket Filter qdisc**. Это **shaper**, который можно использовать, если необходимо просто ограничить полосу пропускания по заданному критерию. Механизм действия TBF такой: существует корзина (**bucket**), в которую собираются жетоны (**tokens**). Как только корзина наполняется, **qdisc** отдает трафик. Если скорость трафика превышает размер корзины (скорости ее наполнения), то трафик задерживается, или даже теряется.

Другой, часто используемый бесклассовый **qdisc** – **SFQ (Stochastic Fairness Queueing)**. Принцип его действия таков, что поступающий в **qdisc** трафик разбивается на множество FIFO очередей, каждой из которых в случайном порядке отдается приоритет. Таким образом, каждая TCP сессия находится в равных условиях, по сравнению с остальными, и не может заполнить канал целиком.

Следует заметить, что:

1. Для простого ограничения трафика лучше использовать TBF;
2. Если канал реально загружен, лучшим решением будет SFQ;
3. Для ограничения входящего трафика, который не будет перенаправляться дальше, следует использовать Ingress qdisk.

До сих пор, не было отмечено преимуществ классового подхода для управления трафиком. Бесклассовый **qdisk**, установленный сразу на устройство, действует целиком на все это устройство. Бесклассовые дисциплины используются как листья в дереве классовой дисциплины. Это позволяет реализовывать наследование и приоритезацию сразу по нескольким классам. Классовость дает гибкость в построении нашего дерева правил, в которых в качестве классификаторов может быть множество параметров. Например адреса или типы трафика по портам. Таким образом, в классовой структуре, классы расположены в root qdisk, который в свою очередь привязан к интерфейсу; и классы тоже могут содержать подклассы и qdisk в качестве листьев.

Существует два широко используемых Classfull Qdisks. Первый, весьма сложный в своей семантике, но поэтому гибкий - CBQ (Class Based Queue). Эта дисциплина была разработана одной из первых, и на ее основе строятся все наиболее популярные схемы распределения трафика. Она позволяет выполнять все возможные манипуляции по построению классов, приоритезации трафика и его ограничению. При разделении полосы с помощью шейперов, например TBF или SFQ, возможно наследование свободного канала как вниз по дереву классов, так и вверх. CBQ позволяет приоритезировать трафик. При разделении канала на полосы, дисциплина работает в целом относительно точно, так как использует его временные характеристики. Но не всегда точно в рамках отдельного класса. Основной рекомендацией при использовании CBQ является то, что надо следить, чтобы сумма полос нижележащих классов была меньше или равна каналу родительского класса. Из этого правила могут быть исключения, например, если канал никогда полностью не загружен по всем классам (когда трафик минимален). Но в любом случае описание структуры CBQ начинается с указания точной максимальной полосы пропускания вашего канала. Что не всегда известно. Сложность при использовании CBQ заключается в его не простом синтаксисе, а также довольно не интуитивной системе построения классов в целом.

Второй Classfull Qdisk – HTB (Hierarchical Token Bucket). Эта дисциплина появилась в официальном ядре начиная с версии 2.4.20. Алгоритм ее работы во многом схож с TBF и CBQ одновременно. HTB отличается такая же гибкость, как у CBQ, но в отличие от последнего, HTB гораздо более интуитивно понятен и имеет упрощенную конфигурацию. Более точен в механизмах ограничения канала. Кроме того, для HTB не нужно описывать пропускную способность всего канала.