**Лекция за 18.04.2012, часть 2**

***Настройка определителя.***

Не будем касаться настройки определителя в Windows: там это делается в настройках TCP/IP, есть закладка DNS, где можно установить все необходимые параметры. Смотреть будем на настройку в UNIX-подобных системах.

Когда говорят об именах в DNS, выделяют специальный термин — ***Full qualified domain name*** — ***Квалифицированное доменное имя***, либо ***Полностью специфицированное доменное имя***. Смысл этого термина заключается в следующем: это имя, которое гарантированно показывает полную иерархию размещения данного домена или хоста в пространстве имен DNS. Для того, что имя считалось полностью квалифицированным, оно должно терминироваться точкой, т. е. фактически он показывает полный путь в графе пространства имен, начиная от пустого корневого домена. В качестве примера: [www.math.spbu.ru](http://www.math.spbu.ru/) — в такой форме не является полностью квалифицированным, а в такой форме: [www.math.spbu.ru](http://www.math.spbu.ru/). - является. На самом деле все запросы между серверами (или вообще все запросы в DNS) касаются всегда полностью квалифицированных имен, т. е. Даже если не поставили в конце имени точку, предварительно происходит его (имени) квалификация. Каким образом это осуществляется? В настройках resolver'а есть параметр, который позволяет в случае использования неквалифицированного имени дополнять его различными DNS-суффиксами (т. е. добавлять имя текущего домена или какого-нибудь по нашему выбору. Суффиксы должны быть прописаны где-то в настройках). Это позволяет сократить количество набираемого текста. На самом деле, с появлением браузеров, которые обладают поиском по собственной истории, необходимость в этом резко упала, но, тем не менее, возможность сохраняется. После квалификации происходит по нему поиск. Если у нас перечисляется несколько квалификационных суффиксов, то поиск осуществляется последовательным перебором, добавлением к неквалифицированному имен суффиксов, пока что-нибудь не найдется (выдается первый ответ). Соответственно, если перечисляется достаточно большое количество суффиксов, то в таком случае есть шанс либо на очень долгий поиск, либо найдем не то, что хотели найти («Компьютер делает то, что от него просят, а не то, что вы от него хотите»), т. е. если в строке браузера набрать www, то он подставит первый квалификационный суффикс, а т. к. www является самым часто употребляемым именем среди имен хостов, то он скорее всего его найдет, а если вы имели в виду вовсе не веб-сервер того домена, который первым перечислен, то придется набирать полное имя. Вопрос: набираем в строке браузера, например, [www.microsoft.com](http://www.microsoft.com/), однако людей, которые в конце ставят точку — немного. Как действовать в этом случае? То есть, почему это имя не расширяет суффиксами (точкой в конце, например)? На самом деле существует ещё одно условие квалификации: для того, чтобы имя не считалось квалифицированным, оно должно содержать небольшое число точек. Сейчас по умолчанию считается, что если в имени две точки или более, то оно является квалифицированным. Это заслуга компании Microsoft, которая впервые в своем resolver'е пошла на такое упрощение жизни для пользователей.

В UNIX-подобных операционных системах существует специальная настройка, которая говорит о том, сколько точек должно быть в имени для того, чтобы оно являлось полностью квалифицированным. В большинстве UNIX-подобных операционных системах определитель настраивается через файл ***etc/resolv.conf***; исключения есть, но их мало. Любые изменения в этом файле вступают в действие немедленно, т.е. если захотим изменить настройки определителя, то достаточно что-то поменять в этом файле. Этот файл считывается каждый раз при выполнении функции getHostByName или getNameByHost (это, скорее всего, означает, что данный файл будет постоянно закэширован). Это текстовый файл, который состоит из строк вида «параметр\_значение», параметр и значение разделяются пробелами, либо другими space'овыми символами (Tab, например). Что там есть:

***domain******math.spbu.ru*** /\* эта строка описывает текущий домен \*/

***search******math.spbu.ru****,* ***spbu.ru*** /\* через запятую перечисляются все доменные суффиксы, которые могут использоваться для квалификации имен, если они не являются полностью квалиф-ми \*/

/\* Если имя не будет полностью квалифицированным, то ему будут добавляться эти суффиксы и будет производиться поиск по соответствующим квалифицированным именам. В конце math.spbu.ru и spbu.ru точки ставить не надо, т. к. resolver при квалификации добавляет суффикс и ставит в конце точку. Ничего от двух точек в конце не изменится \*/

***dots 2***  /\* сколько точек должно быть в имени, чтобы оно считалось полностью квалиф-ым \*/

/\* 2 — по умолчанию. Но выбрано не случайно: когда пользователь работает с интернетом, он, как правило, работает с хостами, а не с доменами. Впервые хосты появляются на втором уровне. А в именах хостов, которые будут размещаться на втором уровне, будет 2 точки, не считая терминальной. \*/

***nameserver******192.168.0.1*  /\*** здесь последовательно перечисляются

***nameserver 176.16.14.32*** сервера имен для текущего домена \*/

***nameserver 10.0.116.239*** /\* Это единственная часть в файле, где важен порядок. Сервера в нем перечисляются последовательно, поэтому имеет смысл первым ставить в некотором смысле ближайший сервер, то есть до которого или пинг короче или который наиболее доступен с точки зрения загруженности и т. п. Во всем остальном порядок не важен. Второй сервер будет опрошен, только если первый сервер будет недоступен. \*/

Resolver — штука не очень умная и на это есть причины: во-первых, все это в общих чертах было создано году в 83-м, когда объемы памяти были очень небольшими и поэтому раздувать сверх меры стандартную библиотеку C никто не хотел. Во-вторых, компьютеры были довольно слабыми, поэтому если в библиотечные функции (особенно в библиотечные функции, работающие с сетью) вставлять излишнюю логику, то это может на порядки увеличить время работы функции. Поэтому, фактически, большая часть работы по resolving'у была переведена с клиента на сервер. Но там-то это не единственная причина, это ещё способ снизить нагрузки на сеть.

Это не единственный файл, который определяет логику работы resolver'а. Имеется ещё один файл, чаще всего это ***etc/nsswitch.conf***. Этот файл начинает очень сильно зависеть от операционной системы. Даже в разных версиях одной и той же ОС он может быть разным (имя тоже может быть другим). Это тоже текстовый файл, он содержит перечисление источников, которые должны нам выдавать информацию об именах. Типичный вид файла:

***hosts******dns*** /\* имя сначала будет искаться в локальном файле hosts, а уже потом функция getHostByName полезет в интернет, если она в локальном файле hosts не найдет. Первоначально это было сделано для совместимости с теми огромными хост-файлами системы ARPANET (см. начало лекции), а потом долгое время файл hosts использовался как некий кэширующий файл (причем вручную кэширующийся, вручную редактировался и вручную создавался) для хранения наиболее часто используемых IP-адресов. Причем resolver может работать с очень разными источниками имен, не только с dns; есть такой источник nis (**n**etwork **i**nformation **s**ystem) — это аналог домен microsoft в UNIX. Есть ещё nisplus; chaos, hesiod (chaos сейчас ещё хоть как-то используется) — это альтернативные пространства имен, о которых говорилось в начале лекции. Они отличаются логикой работы, но не принципом построения. То есть это тоже иерархические пространства имен, т. е. там тоже можно выделить некие аналоги доменов и хостов (хосты то вообще везде одинаковы; и термин зона как раз, наверное, пришел из chaos, он там используется как синоним домена). Chaos фактически позволял создавать виртуальные домены, т.е. там были конструкты, которые были контейнерами для произвольных типов записей. Но это оказалось никому не нужно.

Рассмотрим, из чего состоит база данных DNS, а именно из каких записей. Первоначально БД DNS (и до сих пор на большинстве серверов) состоит из записей, которые находятся в обыкновенных текстовых файлах. Притом на каждую зону заводится собственный файл, он называется ***зональным***. В принципе, можно сделать так, чтобы в одном файле хранились данные нескольких зон, можно разделить данные зоны по нескольким файлам. Если у вас такая большая зона, что её удобнее хранить в нескольких файлах, то это значит, что надо переходить на систему, которая хранит просто данные в какой-то БД, а не в текстовых файлах.

Зональные файлы тоже состоят из строк, которые называются записи. Общий вид записи:

***[name][ttl][class] type data***

***name*** — имя записи, может быть опущено;

***ttl*** — время жизни этой записи на кэширующих серверах, тоже может быть опущено;

***class*** — наименование пространства имен, для которого данная зона описана;

***type*** — тип записи;

***data*** — данные записи;

Пространство имен, про которое будем всё это время говорить — это IN (INTERNET).

NIS, NISP (Network Information Service Plus), CH, GS (соответствующие классы для nis, nisp, chaos, hesiod).

Какой алфавит у записей? Общий алфавит — это алфавит ASCII (опять же большие и маленькие символы не различаем). В имени применимы буквы, цифры, дефис; для поля данных — зависит от типа записи; ttl — число; class — уже перечислено; типы записей тоже имеют ограниченное количество вариантов, часть из них будет рассмотрена.

Теперь о специальных символах:

**1)** «;» - начало комментария (всё, что правее «;» не обрабатывается)

**2)** «( - пара открывающая и закрывающая скобки, расположенные на разных строках

)» - между скобками игнорируется символ переноса строки (т. е. всё, что между скобками интерпретируется как одна строка). Скобки в одной строке игнорируются.

**3)** «@» - этот метасимвол не очень удачный, с ним связано много проблем. Это метасимвол, соответствующий имени текущей зоны. Но прикол в том, что стандарт на DNS появился раньше, чем появился стандарт на протокол SNTP и синтаксис электронных адресов, т. е. когда создавался этот стандарт, ещё никто не знал, что «@» будет столь популярным символом.

Поле ttl — чисто цифровое, поэтому имя не должно состоять из одних цифр. Множества значений type и class — ограничены и имя записи не может совпадать ни с одним из этих символов. Правда существует одно ограничение - type и class всегда записываются в верхнем регистре. В принципе можно использовать в качестве имени IN, но тогда надо обязательно указать class, чтоб можно было однозначно интерпретировать запись. Поэтому хорошо бы писать имя класса IN, потому что имя такое всё же бывает (например, для описания внешних и внутренних интерфейсов какого-нибудь маршрутизатора имена IN и OUT — вполне уместны). Поэтому можно считать, что имя класса почти не опционально. Есть ещё одно очень важное соглашение: если не использовать имя в записи, то считается, что используется имя предыдущей записи. То есть отсутствующее имя выступает как некий скрытый метасимвол.

Разберем синтаксис нескольких типов записей и поймем по ним логику работы DNS:

Записи, относящиеся к конкретной зоне, т. е. в первую очередь это первая запись зонального файла, должна начинаться с записи вполне определенного типа, а именно с ***записи типа SOA (Start Of Authority)*** — начало ответственности, т. е. если мы используем один файл для хранения записей нескольких зон, то, соответственно, зоны считаются от одного SOA до другого. Запись выглядит так:

***@ IN SOA ns.cs.colorado.edu. admin.cs.colorado.edu.(***

***1999121501 ; serial***

***21600 ; Refresh, 6h***

***1800 ; retry, 30m***

***1209600 ;***

***7200) ; ttl, 2h***

Запись типа SOA это вообще чуть ли не образец для всех записей, поскольку он содержит все элементы синтаксиса. Как видно из «@» - имя для записи SOA является именем текущей зоны. Откуда берется имя зоны? Из файла настроек сервера имен, т. е. в настройках сервера имен прописывается соответствие между именем зоны и файлом, в котором эта зона описывается. Если хотим в одном файле несколько зон, тогда в этом файле пишем имя зоны явно. Обычно так не делают (чревато ошибками). IN — INTERNET. SOA — тип записи. Далее у нас идут два полностью квалифицированных имени. Первое — имя master сервера имени этой зоны, второе — не имя, а почтовый адрес. Здесь и проявляется отсутствие «свободной собаки». Это имя интерпретируется в SNTP как почтовый адрес следующим образом: первая точка в имени заменяется на символ «@». Если хотим связаться с администратором данной зоны, то должны получить запись SOA, посмотреть вот на этот адрес и заменить первую точку на «@». По этому адресу высылаются автоматические отчеты о каких-либо проблемах, если такие настройки есть. Есть такой формат почтового адреса, он не придуман специально для этого. Этот формат появился до SNTP с «@». К сожалению, нельзя использовать почтовые адреса, которые содержат точку в пользовательской части адреса.

За полной квалификацией имен здесь надо следить, потому что если у нас будет адрес без терминальной точки, то он будет автоматически квалифицироваться при обработке этого файла сервером. Правила квалификации для зональных файлов очень просты: имя квалифицируется добавлением имени текущей зоны. Поэтому мы если случайно забудем поставить точку в конце, то её дополнят, причем не так, как мы хотели. Далее идет «(». В принципе, скобка может стоять в любом месте, но, чтобы не было путаницы, лучше её ставить в конце.

А дальше идет набор чисел. Смысл этих пяти чисел таков:

***serial*** — т. н. порядковый номер зоны(или порядковый номер файла). В логике работы DNS существует очень простой индикатор того, что зональный файл изменился. Время от времени slave'ы запрашивают у master'а запись типа SOA и сравнивают в ней serial с тем, который у них сохранен. Если serial увеличился, то это индикатор того, что надо закачать всю зону целиком. На самом деле это уже немножко устаревшая логика, в современных версиях DNS серверов используется активная синхронизация, т. е. master server извещает своих slave'ов об изменении зоны. Но, с целью совместимости, старая логика продолжает поддерживаться и, видимо, будет поддерживаться ещё долго. Значение serial должно только увеличиваться: если serial уменьшился, то slave ничего не будет делать (посмотрит, что его serial больше, чем serial на master, и никаких изменений в своей зоне не произведет), но, на самом деле, при взаимодействии с кэширующими серверами может возникнуть неприятная ситуация, связанная с тем, что может сохраниться несколько различных версий зоны, поскольку master и slave начнут отдавать разные данные. Когда DNS ведётся в ручную (если размер домена не превышает 200-300 машин), то serial записывается в некой мнемонической форме, а именно: год-месяц-число-версия изменения зоны в текущий день(1999 = год, 12 = месяц, 15 = число, 01 — версия (вряд ли в один будем вносить больше 99 изменений в зону)). Соответственно это гарантирует общую инкрементацию serial'а.

Следующее число — ***refresh*** — это время обновления (измеряется в секундах). В нашем случае это 6 часов. Это время, через которое slave'ы должны обращаться к master за вопросом об изменении файла зоны, т. е. каждые 6 часов они запрашивают у master текущее содержание записи SOA. В принципе, в нынешних версиях всех DNS-серверов можно указывать не количество секунд, можно писать, например, 6h — шесть часов. Но опять же, с целью совместимости со старыми версиями серверов, очень рекомендуется писать всё-таки в секундах (достоверно известно, что кое-какие большие кэширующие сервера до сих пор используют старые версии ПО).

***retry*** — в данном случае это 30 минут — это интервал, через который slave'ы будут стучаться к master, если master оказывается недоступным, т. е. сначала мы к нему через 6 часов постучались, и он не ответил, тогда мы начинаем ломиться к нему каждые 30 минут.

Четвертое число — у нас это 2 недели - показывает, в течение какого срока slave'ы должны стучаться к master, если он всё время не отвечает (если через 6 часов не ответил, то начинаем стучаться каждые 30 минут в течение 14 дней). Если нам так и не ответили, то делаем вывод, что сервер сдох.

Как определять эти имена? Указанные времена не являются временами по умолчанию. Для каждой зоны они должны быть описаны явно. Если мы редко вносим изменения в зону (т. е. у нас устоявшийся домен), то время refresh можно увеличить примерно до суток. Если же у нас достаточно хорошая сеть, то время retry можно, наоборот, уменьшить. Если мы знаем, что не вносим критических изменений в DNS, то это время тоже может быть увеличено до 2-3-х часов. Четвертое число в хороших сетях рекомендуется уменьшать до нескольких суток.

Последнее из приведенных чисел — 7200 — это ***ttl*** (время жизни записей для этого сервера по умолчанию для кэширующих серверов). Логика определения ttl: время жизни на кэширующих серверах вычисляется по трем числам: берётся ttl текущей записи SOA, берётся ttl самой записи (если оно там прописано) и берется настройка кэширующего сервера. После этого логика следующая: если минимальным из трёх является время в настройках кэширующего сервера, то используется именно оно (это означает, что нагрузка на этот сервер очень велика и он часто сбрасывает кэш). Если есть ttl в записи, то берется максимальное из двух ttl (записи и SOA), т. е. рекомендуется в записи SOA ставить минимальное время ttl, а если нам требуется его увеличить, то надо отдельно писать ttl для записи.

В сущности, запись SOA является одной из самых главных и любые повреждения в ней приводят к тому, что зона не будет загружаться. Во всех остальных случаях это уже зависит от логики работы сервера.

***Запись типа NS (Name Server)***

***@ IN NS ns.cs.colorado.edu.***

***IN NS anchor.cs.colorado.edu.***

***IN NS nc.cs.utah.edu.***

«@» - работаем с текущей зоной (cs.colorado.edu.). При работе с майкрософтовскими DNS-северами не рекомендуется использовать отсутствие имен, т. к. там есть, например, очень хитрый сервер, который любит перемешивать строки. Он хранит данные в registry, но при экспорте данных он строки переупорядочивает в алфавитном порядке. После NS должно идти полностью квалифицированное имя сервера, причём это должно быть имя, которое где-то в интернете описано через левую часть записи типа A (address), т. е. это не должен быть не IP-адрес,не синоним, это должно быть имя. Для работающей зоны записи типа SOA и NS — обязательны. В принципе, других записей в зоне может и не быть. В записи типа NS перечисляются все авторитетные для данной зоны сервера, master server берется из SOA.

***Запись типа A (Address)***

***anchor IN A 128.138.243.100***

Сначала идет имя хоста, потом IN — class, A — тип, потом — IP-адрес. Имя в данном случае используется неквалифицированное, следовательно, оно будет квалифицироваться путем добавления текущего имени зоны (т. е. cs.colorado.edu.). В принципе, точно так же вместо просто anchor можно написать anchor.@ - эффект будет точно такой же (вместо @ вставиться имя текущей зоны).

***Запись типа CNAME***

***ftp IN CNAME anchor***

***kb IN CNAME kibblesnbits***

***ns3 IN CNAME nc.cs.utah.edu.***

Первая запись — это ftp, которая ссылается на уже известный нам anchor. Слева и справа используются неквалифицированные имена, и, значит, они квалифицируются путем добавления имени текущей зоны. Вторая запись ссылается на kibblesnbits. CNAME вообще используются, чтобы сделать названия более читаемыми. Последняя — ns3 — ссылается на имя в другом домене, такое тоже бывает. И ссылается у нас как раз на третий nameserver для данного домена. Соответственно, если мы ссылаемся на имя в другом домене, то мы должны написать его полностью квалифицированно, т. е. с терминальной точкой.

Особенности использования CNAME: можно делать CNAME на CNAME. Уровень вложения может быть произвольным, однако надо понимать, как работает resolving CNAME'ов: getHostByName передает серверу имя, но не знает, что это — левая часть записи типа A или типа CNAME. Сервер отправляет запросы по всей этой цепочке (которую разбирали в части 1). Когда он получает ответ, то если это CNAME, то ему возвращается тоже символическое имя (понятно, что если пришел не IP-адрес, а имя, то, значит, в исходном случае был CNAME). Соответственно, он запускает на него новый resolving, он ещё пока ничего не передает адресату. Если это CNAME из того же самого домена, то не нужно всё цепочку заново перебирать и он сразу обратится к известному ему nameserver'у. В любом случае на каждый CNAME нужно начинать новое обращение, т. е. CNAME'ы с уровнем вложенности 3-4 обрабатываются очень долго.

Как избежать гнилой рекурсии? В теории можно написать CNAME хоть на самого себя. Сервер это «съест» (синтаксис не нарушен). Но при обработке ответов CNAME'ов в настоящее время nameserver'а кэшируют эти ответы и, если они встречают зацикливание, т. е. в цепочке ответов попадается уже имеющиеся, то они сразу выдают сообщение об ошибке.

(В своё время была распространена следующая форма кибератак: на кэширующих серверах подменялся кэш, в результате зацикливали CNAME'ы и т. п.)

***PTR запись***

Запись используется для обратного resolving'а, а именно для того, чтобы по IP-адресу определить имя хоста. Обратные записи (записи типа PTR) требуют описания специальных зон. Эти зоны являются архаизмом домена .arpa и архаизмом классовой системы адресации, потому что эти зоны строятся для сетей классов A, B, C. Каким образом это происходит: есть IP-адрес сети класса B: 128.138.243.100; Как строится обратная зона: берем сетевую часть адреса (128.138), записываем в обратном порядке и добавляем в конец in-addr.arpa.:

***138.128.in-addr.arpa.*** -эта зона, как и любая другая, должна иметь запись SOA и NS, но вместо записи типа A в ней запись типа PTR и запись выглядит так:

***100.243 IN PTR anchor.cs.colorado.edu.***- хостовая часть записи тоже в обратном порядке (без точки в конце); IN — INTERNET; PTR — pointer; дальше обязательно пишем полностью квалифицированное имя хоста (квалифицировать нужно в любом случае). Когда запускаем обратный resolving, у нас определитель перестраивает адрес в обратном порядке (т. е. у нас будет 100.243.138.128 в нашем случае), выделяет из них адрес сети (138.128) и отправляет вопрос в зону 138.128.in-addr.arpa. о записи PTR для хоста 100.243 и ему соответственно возвращается anchor.cs.colorado.edu..

А теперь о недостатках подобной системы: это единственное место, где до сих пор эксплуатируется классовая система имен, то фактически этой означает, что для того, чтобы ввести полноценную обратную зону, мы должны обладать полным контролем над всей сетью класса B, а это не всегда получается. Существует методика, которая помогает это обойти: например, пусть провайдер раздает несколько сетей, которые вместе составляют сеть класса C. При этом обратную зону для этой сети класса C будет вести провайдер, но провайдер будет делать не записи типа PTR, он будет делать в соответствующий обратной зоне записи типа CNAME, а каждый конкретный администратор сети, который входит в эту общую сеть класса C заведет у себя некие фиктивные зоны с фиктивными именами, в которых уже будут непосредственно содержаться записи типа PTR, т. е. resolving будет происходить в две стадии: сначала у нас «отрезолвится» запись типа CNAME в настоящей обратной зоне у провайдера, а уже потом пройдет второе обращение к nameserver'у, к соответствующей организации, которая имеет в этой сети подсеть, которая выдаст на этот CNAME настоящий pointer.