Оглавление

[1. Модели взаимодействия сетевых приложений (OSI/ISO, TCP/IP). Архитектура распределенного приложения (клиент/сервер). Основные технологии создания распределенных программных систем. Спецификация NDIS. 2](#_Toc188176033)

[2. Стек протоколов TCP/IP. Публичные и частные пространства адресов, типы портов. Параметры настройки TCP/IP. 8](#_Toc188176034)

[3. Основные характеристики протоколов Ethernet, SLIP, PPP, IPv4, IPv6, ICMP, ARP, RARP, TCP, UDP. Понятия: надежный и ненадежный протоколы, протоколы с установкой соединения или без установки соединения, протоколы ориентированные на поток или на сообщения. 17](#_Toc188176035)

[4. Понятие сокета. Основные параметры сокета. 22](#_Toc188176036)

[5. Интерфейс Nimed Pipe. 25](#_Toc188176037)

[6. Интерфейс MailSlot. 41](#_Toc188176038)

[7. Структура программы TCP-сервера. 50](#_Toc188176039)

[8. Структура программы TCP-клиента. 56](#_Toc188176040)

[9. Структура программы UDP-сервера. 60](#_Toc188176041)

[10. Структура программы UDP-клиента. 64](#_Toc188176042)

[11. Структура параллельного сервера. 64](#_Toc188176043)

[12. Структура параллельного сервера. Accept Server. GarbageCleaner. 68](#_Toc188176044)

[13. Широковещание. Обнаружение сервера с помощью широковещания. 68](#_Toc188176045)

[14. Применение символического адреса хоста. 70](#_Toc188176046)

[15. Основные сетевые утилиты и их назначение. 72](#_Toc188176047)

[16. Служба DNS. 77](#_Toc188176048)

[17. Служба DHCP. 85](#_Toc188176049)

[18. Стандарты сообщений Internet. 92](#_Toc188176050)

[**Лексический анализ сообщений** 92](#_Toc188176051)

[**Синтаксис** 95](#_Toc188176052)

[**Вопросы безопасности** 110](#_Toc188176053)

[Приложение A. Примеры сообщений 111](#_Toc188176054)

[19. Протокол HTTP. 117](#_Toc188176055)

[20. Служба RPC. 123](#_Toc188176056)

[21. NAT, proxy-серверы, межсетевые экраны, ремайлеры. 131](#_Toc188176057)

[22. Web-сервисы: SOAP, XML, WSDL, UDDI. 141](#_Toc188176058)

[23. Национальная инфраструктура информационной безопасности. 158](#_Toc188176059)

[24. Безопасность в сетях: конфиденциальность, аутентификация, обеспечение строгого выполнения обязательств, авторизация, обеспечение целостности, криптография, криптоанализ, криптология, шифр, код, ключ шифра, IPsec, SSL/TSL, HTTPS, DNSsec. 160](#_Toc188176060)

[**1.4. Оценка IPSec** 164](#_Toc188176061)

# Модели взаимодействия сетевых приложений (OSI/ISO, TCP/IP). Архитектура распределенного приложения (клиент/сервер). Основные технологии создания распределенных программных систем. Спецификация NDIS.

Когда рассматривают принципы взаимодействия различных частей (процессов) распределенного приложения, то говорят о ***модели взаимодействия***, а когда рассматривают распределение ролей между различными частями (процессами) распределенного приложения, то говорят об ***архитектуре распределенного приложения***. Или простыми словами:

 **Модель взаимодействия** — это правила общения между частями приложения.

 **Архитектура приложения** — это распределение ролей и задач между этими частями.

Для обсуждения принципов взаимодействия процессов распределенного приложения, как правило, применяется модель ***ISO/OSI*** (International Standards Organization/Open System Interconnection reference model), которая была разработана в 1980-х годах и регулируется стандартом ISO 7498. Официальное название модели ISO/OSI – ***сетевая эталонная модель взаимодействия открытых систем Международной организации по стандартизации***. Спецификации ISO/OSI используются производителями аппаратного и программного обеспечений.

Наиболее популярной архитектурой для распределенного программного приложения является ***архитектура клиент-сервер***. Будем говорить, что распределенное приложение имеет архитектуру клиент-сервер, если все процессы распределенного приложения можно условно разбить на две группы. Одна группа процессов называется серверами другая – клиентами. Обмен данными осуществляется только между процессами-клиентами и процессами-серверами. Основное отличие процесса-клиента от процесса-сервера в том, что инициатором обмена данными всегда является процесс-клиент. Другими словами процесс-клиент обращается за услугой (сервисом) к процессу-серверу. Такая архитектура лежит в основе большинства современных информационных систем.

**Модель взаимодействия открытых систем**

Функции, обеспечивающие взаимодействие открытых систем в модели ISO/OSI распределены по следующим семи уровням: 1) физический; 2) канальный; 3) сетевой; 4) транспортный; 5) сеансовый; 6) представительский; 7) прикладной. Задача каждого уровня – предоставление услуг вышестоящему уровню таким образом, чтобы детали реализации этих услуг были скрыты. ***Протоколы*** - наборы правил и соглашений, описывающих процедуры взаимодействия каждого уровня модели с соседними уровням.

Опишем кратко назначение всех уровней модели OSI.

**Физический уровень**. Физический уровень определяет свойства среды передачи данных (коаксиальный кабель, витая пара, оптоволоконный канал и т.п.) и способы ее соединения с сетевыми адаптерами: технические характеристики кабелей (сопротивление, емкость, изоляция и т.д.), перечень допустимых разъемов, способы обработки сигнала и т.п.

**Канальный уровень**. На канальном уровне модели рассматривается два подуровня: подуровень управления доступом к среде передачи данных и подуровень управления логическим каналом. Управление доступом к среде передачи данных определяет методы совместного использования сетевыми адаптерами среды передачи данных. Подуровень управления логической связью определяет понятия канала между двумя сетевыми адаптерами, а также способы обнаружения и исправления ошибок передачи данных. Основное назначение процедур канального уровня подготовить блок данных (обычно называемый кадром) для следующего сетевого уровня.

Здесь следует отметить два момента: 1) начиная с подуровня управления логической связью и выше протоколы никак не зависят от среды передачи данных; 2) для организации локальной сети достаточно только физического и канального уровней, но такая сеть не будет масштабируемой (не сможет расширяться), т.к. имеет ограниченные возможности адресации и не имеет функций маршрутизации.

**Сетевой уровень**. Сетевой уровень определяет методы адресации и маршрутизации компьютеров в сети. В отличие от канального сетевой уровень определяет единый метод адресации для всех компьютеров в сети не зависимого от способа передачи данных. На этом уровне определяются способы соединения компьютерных сетей. Результатом процедур сетевого уровня является пакет, который обрабатывается процедурами транспортного уровня.

**Транспортный уровень**. Основным назначением процедур транспортного уровня является подготовка и доставка пакетов данных между конечными точками без ошибок и в правильной последовательности. Процедуры транспортного уровня формируют файлы для сеансового уровня из пакетов, полученных от сетевого уровня.

**Сеансовый уровень**. Сеансовый уровень определят способы установки и разрыва соединений (называемых сеансами) двух приложений, работающих в сети.

Следует отметить, что сеансовый уровень - это точка взаимодействия программ и компьютерной сети.

**Представительский уровень**. На представительский уровне определяется формат данных, используемых приложениями. Процедуры этого уровня описывают способы шифрования, сжатия и преобразования наборов символов данных.

**Прикладной уровень**. Основное назначения уровня: определить способы взаимодействия пользователей с системой (определить интерфейс). На рис. 1.2.1 изображена схема взаимодействия двух систем с точки зрения модели OSI. Толстой линией со стрелками на концах обозначается движение данных между системами. Данные проходят от прикладного уровня одной системы до прикладного уровня другой через все нижние уровни системы. Причем по мере своего движения от отправителя к получателю (из одной системы в другую) на каждом уровне данные подвергаются необходимому преобразованию (в соответствии с протоколами модели): при движении от прикладного уровня к физическому данные преобразовываются в формат, позволяющий передать данные по физическому каналу; при движении от физического уровня до прикладного происходит обратное преобразование данных. При такой организации обмена данными фактически взаимодействие осуществляется между одноименными уровнями (на рисунке это взаимодействие обозначено линиями со стрелками между уровнями двух систем).

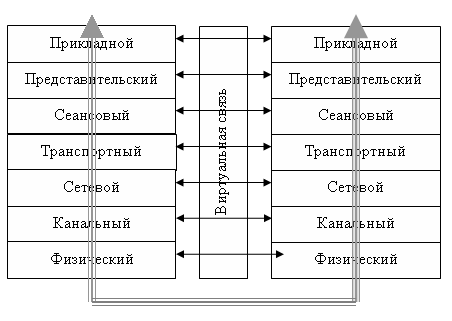


Рисунок 1.2.1. Схема взаимодействия открытых систем

Выше уже отмечалось, что точкой взаимодействия программ и компьютерной сети является сеансовый уровень. Рассмотрим этот момент более подробно для распределенного приложения состоящего из двух взаимодействующих процессов.

Уровни модели OSI/ISO:

* Физический уровень - обеспечивает физическую передачу данных через канал связи. Физический уровень определяет свойства среды передачи данных (коаксиальный кабель, витая пара, оптоволоконный канал и т.п.) и способы ее соединения с сетевыми адаптерами: технические характеристx ики кабелей (сопротивление, емкость, изоляция и т.д.), перечень допустимых разъемов, способы обработки сигнала и т.п.
* Канальный уровень - обеспечивает обработку ошибок и управление доступом к среде передачи данных. Формирует кадры из пакетов и готовит их к передаче на физический уровень. Проверяет доступность физической среды передачи и исправляет ошибки. Для обеспечения корректности передачи помещает в начало и конец каждого кадра специальную последовательность бит и высчитывает контрольную сумму. Определяет правила использования физического канала. При большом размере блока данных делит его на кадры меньшего размера. Функции этого уровня реализуются сетевыми адаптерами и их драйверами. Делится на два подуровня: LLC (logical link control)-подуровень - логический контроль связи

MAC (media access control)-подуровень — контролирует доступ к физической среде

LLC находится выше MAC.

* Сетевой уровень - обеспечивает маршрутизацию, адресацию и передачу пакетов данных между различными сетями.
* Транспортный уровень - обеспечивает установление и контроль соединения, разбиение и сборку данных. Основным назначением процедур транспортного уровня является подготовка и доставка пакетов данных между конечными точками без ошибок и в правильной последовательности. Восстанавливает передачу после ошибок, разбивает сообщение на удобные блоки, распределяет приоритеты передачи и подтверждает передачу. Контроль передачи данных, разбиение данных на блоки, индексация блоков данных.
* Сеансовый уровень - обеспечивает управление соединением между устройствами и синхронизацию передачи данных, которые работают в сети. Следует отметить, что ***сеансовый уровень - это точка взаимодействия программ и компьютерной сети.***

Обеспечивает управление диалогом, определяя, какой из сторон является активной. Координирует сеанс (диалог), может устанавливать контрольные точки

Определяет, какой будет передача между двумя прикладными процессами: (Дуплексный\полудуплексный). ***Установка, поддержание и разрыв сеанса.***

* Представительский уровень - обеспечивает преобразование данных в удобный для передачи формат. Процедуры этого уровня описывают способы шифрования, сжатия и преобразования наборов символов данных.
* Прикладной уровень - обеспечивает работу прикладных программ. Основное назначения уровня: определить способы взаимодействия пользователей с системой (определить интерфейс).

**Структура TCP/IP**

Так как архитектура TCP/IP была разработана задолго до модели ISO/OSI, то неудивительно, что конструкция TCP/IP несколько отличается от эталонной модели. На рисунке 2.2.1. изображены уровни обеих моделей и устанавливается их соответствие.

Модель TCP/IP состоит из 4 уровней:

1. Уровень доступа к сети (Network Access Layer): Определяет физические способы передачи данных по сети. Примеры протоколов: Ethernet, Wi-Fi, Token Ring.
2. Интернет-уровень (межсетевой) (Internet Layer): Обеспечивает маршрутизацию пакетов между сетями. Примеры протоколов: IP (Internet Protocol), ICMP (Internet Control Message Protocol), ARP (Address Resolution Protocol), RARP (Reverse ARP).
3. Транспортный уровень (Transport Layer): Обеспечивает передачу данных между приложениями на разных узлах сети. Примеры протоколов: TCP, UDP.
4. Прикладной уровень (Application Layer): Обеспечивает взаимодействие между приложениями и пользователями. Примеры протоколов: HTTP, FTP, SMTP, DNS, DHCP.



Рисунок 2.2.1. Уровни моделей ISO/OSI и TCP/IP

**Архитектура клиент-сервер**

Распределенное приложение, имеющее архитектуру клиент-сервер, подразумевает наличие в своем составе два вида процессов: процессы-серверы и процессы-клиенты. Далее эти процессы будем называть просто серверами и клиентами.

Инициатором обмена данными между клиентом и сервером всегда является клиент. Для этого клиент должен обладать информацией о месте нахождения сервера или иметь механизмы для его обнаружения. Способы связи между клиентом и сервером могут быть различными и в общем случае зависят от интерфейсов, поддерживаемых операционной средой, в которой работает распределенное приложение. Клиент должен быть тоже распознан сервером, для того, чтобы сервер, во-первых, мог его отличить от других клиентов, а во-вторых, чтобы смог обмениваться с клиентом данными. Если вся основная работа выполняется на сервере, а клиент просто предоставляет интерфейс для пользователя, такой клиент называется **тонким.**

По методу облуживания серверы подразделяются на ***итеративные***  и ***параллельные***  серверы ***(iterative and concurrent servers)***. Принципиальная разница заключается в том, что параллельный сервер предназначен для обслуживания нескольких клиентов одновременно и поэтому использует специальные средства операционной системы позволяющие распараллеливать обработку нескольких клиентских запросов. Итеративный сервер, как правило, обслуживает запросы клиентов поочередно, заставляя клиентов ожидать своей очереди на обслуживание, или просто отказывает клиенту в обслуживании. По всей видимости, можно говорить о итеративно-параллельных серверах, когда сервер имеет ограниченные возможности по распараллеливанию своей работы. В этом случае только часть клиентских запросов будет обслуживаться параллельно.

**Основные технологии создания распределенных программных систем:**

* Сетевые протоколы - правила, которые определяют, как компьютеры общаются друг с другом через сеть. Например, TCP/IP, HTTP, REST, SOAP, MQTT и другие.
* RPC (удалённый вызов процедур): Эта технология позволяет вызывать удаленные процедуры или функции на удаленных компьютерах через сеть. Примеры включают технологии, такие как gRPC, XML-RPC, JSON-RPC.
* Межпроцессное взаимодействие (IPC): Позволяет взаимодействовать между процессами или потоками на разных узлах системы. Примеры включают сокеты, разделяемую память, очереди сообщений и т.д.
* Сервисно-ориентированная архитектура (SOA): Подход, при котором программа делится на независимые сервисы, которые взаимодействуют через стандартизированные протоколы. Примеры включают SOAP, REST, WSDL.
* Микросервисная архитектура: Разбиение приложения на небольшие, автономные и независимо масштабируемые сервисы. Каждый сервис выполняет определенные функции и взаимодействует с другими сервисами через API.

**Спецификация NDIS (Network Driver Interface Specification):**

NDIS является стандартным интерфейсом, определяющим способ взаимодействия между сетевыми адаптерами (сетевыми картами) и драйверами сетевых адаптеров в операционных системах Windows.

**Как это работает?**

* **Связь между устройствами**: NDIS определяет, как драйверы сетевых карт должны взаимодействовать, чтобы обеспечивать работу сети. Это как набор правил, по которым работают участники команды.
* **Упрощение для разработчиков**: Благодаря NDIS разработчики приложений могут работать с сетевыми адаптерами, не беспокоясь о том, как они устроены внутри. Это похоже на то, как мы можем пользоваться автомобилем, не зная, как устроен двигатель.
* **Посредник**: NDIS действует как посредник (или "проводник") между низкоуровневыми драйверами и самим оборудованием. Он "обвивает" эти драйверы, чтобы они могли общаться друг с другом и с компьютером.

# Стек протоколов TCP/IP. Публичные и частные пространства адресов, типы портов. Параметры настройки TCP/IP.

Семейство протоколов TCP/IP, часто именуемое ***стеком TCP/IP***, стало промышленным стандартом де-факто для обмена данными между процессами распределенного приложения и поддерживается всеми без исключения операционными системами общего назначения. Обширная коллекция сетевых протоколов и служб, называемая стеком TCP/IP намного больше, чем сочетание двух основных протоколов давшее ей имя. Тем не менее, эти протоколы являются основой стека TCP/IP: ***TCP*** ***(Transmission Control Protocol)*** обеспечивает надежную доставку данных в сети; ***IP* *(Internet Protocol)*** организует маршрутизацию сетевых передач от отправителя к получателю и отвечает за адресацию сетей и компьютеров.

Существует шесть ключевых организаций, которые занимаются стандартизацией технологий для работы в Интернете:

1. **ISOC (Internet Society)**: Обеспечивает поддержку и развитие Интернета.
2. **IAB (Internet Architecture Board)**: Следит за техническим развитием Интернета и его архитектурой.
3. **IETF (Internet Engineering Task Force)**: Основная организация, которая разрабатывает и поддерживает правила и форматы для сетевых протоколов. Они создают документы, называемые RFC (Request for Comments), в которых описаны все важные детали работы Интернета.
4. **IRTF (Internet Research Task Force)**: Занимается долгосрочными исследованиями технологий Интернета.
5. **ISTF (Internet Societal Task Force)**: Фокусируется на социальных аспектах Интернета.
6. **ICANN (Internet Corporation for Assigned Names and Numbers)**: Управляет доменными именами и адресами в Интернете.

Из этих организаций **IETF** является самой важной, потому что именно она отвечает за создание правил и стандартов, которые позволяют всем устройствам и программам правильно работать в сети.

**Структура TCP/IP**

Так как архитектура TCP/IP была разработана задолго до модели ISO/OSI, то неудивительно, что конструкция TCP/IP несколько отличается от эталонной модели. На рисунке 2.2.1. изображены уровни обеих моделей и устанавливается их соответствие. Уровни моделей похожи, но не идентичны. В общем и целом, Транспортные уровни обеих моделей соответствуют друг другу, но есть и некоторые различия. Например, некоторые функции Сеансового уровня модели ISO/OSI берет на себя Транспортный уровень TCP/IP. Содержимое Сетевого уровня модели ISO/OSI тоже примерно соответствует Межсетевому уровню TCP/IP. В большей или меньшей степени Прикладной уровень TCP/IP соответствует трем уровням Сеансовому, Представительскому и Прикладному модели ISO/OSI, а Уровень доступа к сети – совокупности Физического и Канального уровней.



Рисунок 2.2.1. Уровни моделей ISO/OSI и TCP/IP

В дальнейшем при описании протоколов стека TCP/IP будут использоваться и названия уровней модели ISO/OSI, если это помогает определить более точное место протокола в иерархии.

**Протоколы Уровня доступа к сети**

На Уровне доступа к сети задействованы протоколы для создания локальных сетей (***LAN***, Local-Area Networks) и для соединения с глобальными сетями (***WAN***, Wide-Area Networks). Одной из основных характеристик протоколов канального уровня является максимальная единица передачи данных MTU (Maximum Transmission Unit), которая определяет максимальную длину в байтах данных передаваемых в одном кадре. От значения MTU зависит скорость передачи по каналу. Если IP-модулю требуется отправить данные (дейтаграмму) имеющие длину большую MTU, то он производит фрагментацию, разбивая дейтаграмму на части имеющие длину меньшую, чем MTU.

**Протоколы Межсетевого уровня**

Важнейшими протоколами Межсетевого уровня TCP/IP являются: ***IP*** (Internet Protocol), ***ICMP*** (Internet Control Message Protocol), ***ARP*** (Address Resolution Protocol), ***RARP*** (Reverse ARP).

**Протокол IP**. В семействе протоколов TCP/IP протоколу IP отведена центральная роль. Его основной задачей является доставка ***дейтограмм*** (так называется единица передачи данных в терминологии IP). При этом протокол по определению является ***ненадежным*** и ***не поддерживающим соединения***.

Ненадежность протокола IP обусловлена тем, что нет гарантии, что посланная узлом сети дейтаграмма дойдет до места назначения. Сбой, произошедший на любом промежуточном узле сети, может привести к уничтожению дейтаграмм. Предполагается, что необходимая степень надежности должна обеспечиваться протоколами верхних уровней.

IP не ведет никакого учета очередности доставки дейтаграмм: каждая дейтаграмма обрабатывается независимо от остальных. Поэтому очередность доставки может нарушаться. Предполагается, что учетом очередности дейтаграмм должен заниматься протокол верхнего уровня.

Когда данные передаются по сети, на первом уровне (Уровень доступа к сети) используются MAC-адреса. Это уникальные адреса, которые помогают устройствам находить друг друга в локальной сети.

На следующем уровне, Межсетевом уровне, используются IP-адреса. Основное отличие IP-адресов в том, что они не зависят от конкретного устройства. Это значит, что данные можно отправлять и получать через разные устройства, такие как модемы или сетевые карты.

Все устройства с IP-адресом называются хостами. **Хост** — это любое устройство, которое подключено к сети и может отправлять или получать данные.

IP-адрес представляет собой последовательность из 32 битов. Причем старшие (левые) биты этой последовательности отводятся для адреса сети, а младшие (правые) – для адреса хоста в этой сети. Для записи IP-адреса, как правило, используются четыре десятичных числа, разделенных точкой. Каждое десятичное число является десятичным представлением 8 битов (***октет*** в терминологии TCP/IP) адреса. На рисунке 2.4.1 разобран пример перевода IP-адреса из двоичного формата в десятичный.

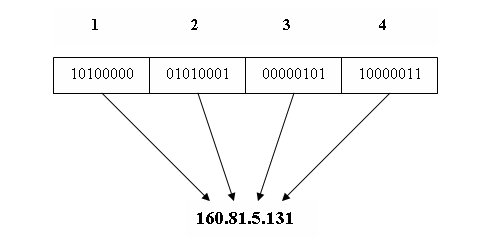


Рисунок 2.4.1. Представление IP-адреса в десятичном формате

Количество бит отведенных для адреса сети и адреса хоста определяется ***моделью адресации***. Существует две модели адресации: ***классовая*** и ***бесклассовая***. В классовой модели адресации все адреса подразделяются на пять классов: A, B, C, D, E. Принадлежность к классу определяется старшими битами адреса. На рисунке 2.4.2 приведены форматы адресов для всех классов.

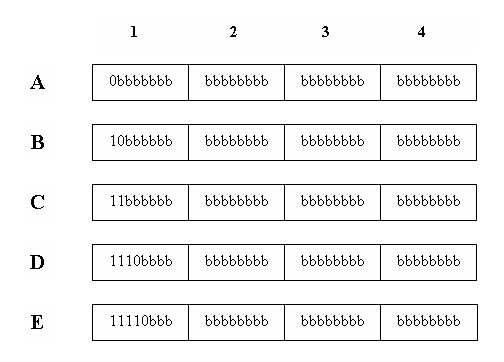


Рисунок 2.4.2 Форматы IP-адресов в классовой модели адресации

Классы D и E имеют специальное назначение: D – предназначен для использования групповых адресов, позволяющих отправлять сообщения группе хостов; E – исключительно для экспериментального применения. Распределение октетов IP-адреса на адрес сети и адрес хоста для классов A, B и C приводится на рисунке 2.4.3. Закрашенные октеты обозначают часть IP-адреса отведенную для адреса сети, не закрашенные – часть адреса, используемую для адресов хостов. Правый столбец показывает формат адресов в десятичном виде: символом n обозначается сетевая часть адреса, символом h – часть адреса для идентификации хоста.

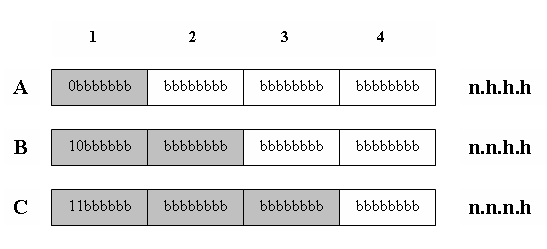


Рисунок 2.4.3. Распределение разрядов IP-адреса на адрес сети и адрес хоста

для классовой модели IP-адресов

Таблица 2.4.1

|  |  |  |
| --- | --- | --- |
| **Класс** | **Диапазон адресов** | **Диапазон частных адресов** |
| А | 0.0.0.0 – 127.255.255.255 | 10.0.0.0 – 10.255.255.255 |
| B | 128.0.0.0 – 191.255.255.255 | 172.16.0.0 – 172.31.255.255 |
| C | 192.0.0.0 – 223.255.255.255 | 192.168.0.0 – 192.168.255.255 |
| D | 224.0.0.0 – 239.255.255.255 | не предусмотрен |
| E | 240.0.0.0 – 247.255.255.255 | не предусмотрен |



Для каждого класса IP-адресов есть специальные диапазоны, которые можно использовать внутри организаций. Эти адреса называются частными адресами и описаны в документе RFC 1918. Их можно использовать без контроля, но они не уникальны в Интернете, поэтому их нельзя использовать для связи с другими устройствами в глобальной сети. Кроме того, адреса вида 127.n.n.n предназначены для выполнения возвратного тестирования (***loopback testing***). Существует некоторая путаница с применением адресов состоящих из всех нулей или единиц. До выхода документа RFC 1878 (1995 г.) не разрешалось использование в качестве адреса хоста нулевую последовательность битов (этот адрес считался адресом самой сети) и последовательность битов, состоящую из одних единиц (этот адрес использовался для широковещательных сообщений в сети). RFC 1878 разрешает использование таких адресов. Диапазоны IP- адресов для каждого класса сведены в таблице 2.4.1.

Применение классовой модели адресации не всегда удобно. Альтернативой классовой модели является ***бесклассовая междоменная маршрутизация*** – ***CIDR*** (Classless Inter-Domain Routing). CIDR позволяет произвольным образом назначать границу сетевой и хостовой части IP-адреса. Для этого каждому IP-адресу прилагается 32-битовая маска, которую часто называют ***маской сети*** (net mask) или ***маской подсети*** (subnet mask). Сетевая маска конструируется по следующему правилу: на позициях, соответствующих адресу сети, биты установлены; на позициях, соответствующих адресу хоста, биты сброшены. Установка маски подсети осуществляется при настройке протоколов TCP/IP на компьютере. Вычисление адреса сети выполняется с помощью операции конъюнкции между IP-адресом и маской подсети. На рисунке 2.4.4. разобран пример вычисления адреса сети с помощью маски подсети.

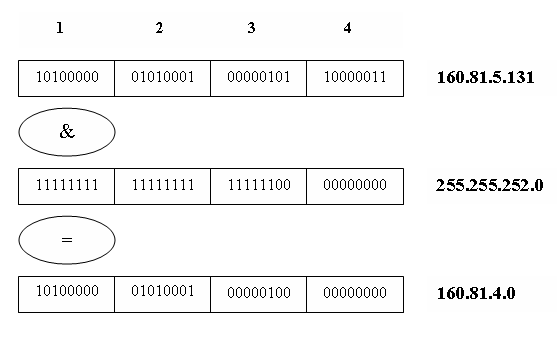


Рисунок 2.4.4. Вычисление адреса сети с помощью маски сети

Протокол IP отвечает за доставку данных (называемых **дейтаграммами**) в большой сети, которая состоит из нескольких меньших сетей. Эти меньшие сети соединяются с помощью специальных устройств, которые называются шлюзами.

**Шлюз** — это как компьютер с несколькими подключениями, который помогает передавать данные между сетями. Например, один шлюз может соединять сеть 172.16.5.0 и сеть 172.16.12.0.

Чтобы данные могли пройти из одной сети в другую, шлюз использует **таблицу маршрутов**. Эта таблица хранит информацию о том, какие адреса сетей существуют и как к ним добраться.

Когда данные отправляются, процесс нахождения следующего устройства, куда должны быть отправлены данные, называется **маршрутизацией**. Это как GPS для данных, который указывает, куда им нужно идти дальше.

**Протоколы Транспортного уровня**

На транспортном уровне TCP/IP есть два основных протокола для передачи данных:

TCP (Transmission Control Protocol): Это протокол, который устанавливает соединение перед отправкой данных. Он гарантирует, что данные будут доставлены в правильном порядке и без ошибок.

UDP (User Datagram Protocol): Это протокол, который отправляет данные без предварительного соединения. Он быстрее, но не гарантирует успешную доставку или порядок данных.

Когда данные отправляются или принимаются, они используют номер порта. Это как адрес для конкретного приложения на устройстве. Чтобы полностью определить, куда отправляются данные в сети TCP/IP, нужно знать три вещи: IP-адрес, номер порта и тип протокола (TCP или UDP).

Когда данные перемещаются вниз по уровням сети, они упаковываются (как матрешка), а когда идут обратно — распаковываются. Каждый уровень добавляет или убирает информацию, чтобы данные могли быть правильно доставлены и использованы на конечном устройстве.

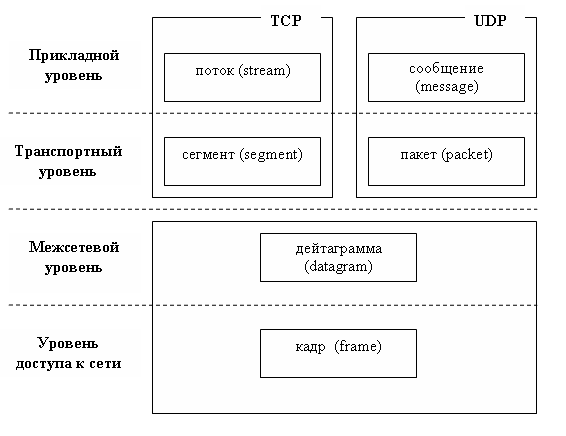


Рисунок 2.5.1. Протоколы TCP и UDP в стеке TCP/IP

Номера портов, используемые для идентификации прикладных процессов (в соответствии с документами IANA), делятся на три диапазона: ***хорошо известные номера портов (well-known port number)***, ***зарегистрированные номера портов (registered port number)***, ***динамически номера портов (dynamic port number)***. Распределение номеров портов по диапазонам приведено в таблице 2.5.1.

Таблица 2.5.1

|  |  |
| --- | --- |
| Хорошо известные номера портов | 0 – 1023 |
| Зарегистрированные номера портов | 1024 – 49151 |
| Динамические номера портов | 49152– 65535 |

В сети существуют разные номера портов, которые помогают различным программам и службам общаться друг с другом.

* **Хорошо известные номера портов**: Эти номера используются основными службами, такими как веб-серверы или почтовые службы. У них есть специальные привилегии.
* **Зарегистрированные номера портов**: Эти номера присваиваются промышленным приложениям и программам, чтобы они могли работать корректно.
* **Динамические номера портов**: Их иногда называют **эфемерными портами**. Эти номера временно выделяются приложениям, когда они начинают работать. Обычно операционная система сама назначает эти номера. В некоторых системах диапазон для таких портов — от 1024 до 5000.

Таблица 2.5.2

|  |  |  |
| --- | --- | --- |
| **Номер порта** | **Протокол** | **Описание** |
| 20 | TCP | File Transport Protocol (FTP) |
| 21 | TCP | File Transport Protocol (FTP) |
| 22 | TCP | Secure Shell (SSH) |
| 23 | TCP | Telnet |
| 25 | TCP | Simple Mail Transfer Protocol (SMTP) |
| 53 | TCP | Domain Name Server (DNS) |
| 53 | UDP | Domain Name Server (DNS) |
| 66 | TCP | Oracle SQL\*NET |
| 67 | UDP | Dynamic Host Configuration Protocol (DHCP) Server |
| 68 | UDP | Dynamic Host Configuration Protocol (DHCP) Client |
| 80 | TCP | World Wide Web (WWW) |
| 110 | TCP | Post Office Protocol, Version 3 (POP3) |
| 111 | TCP | Remote Procedure Call (RPC) |
| 111 | UDP | Remote Procedure Call (RPC) |
| 143 | TCP | Internet Message Access Protocol (IMAP4) |
| 1352 | TCP | Lotus Notes |
| 1433 | TCP | Microsoft SQL Server |
| 1522 | TCP | Oracle SQL Sever |

* **Публичный IP-адрес**: Это адрес, который позволяет вашему устройству общаться с другими устройствами в Интернете. Он уникален в глобальной сети, и именно его видят другие устройства, когда вы заходите в Интернет.
* **Частный IP-адрес**: Это адрес, который используется только внутри вашей локальной сети, например, в доме или офисе. Он не виден в Интернете и помогает устройствам общаться друг с другом внутри сети.

# Основные характеристики протоколов Ethernet, SLIP, PPP, IPv4, IPv6, ICMP, ARP, RARP, TCP, UDP. Понятия: надежный и ненадежный протоколы, протоколы с установкой соединения или без установки соединения, протоколы ориентированные на поток или на сообщения.

**Протоколы Уровня доступа к сети**

**Протокол Ethernet**

* Ethernet — это способ подключения устройств в локальной сети, например, в офисе или дома.
* Он использует метод CSMA/CD, который помогает устройствам не мешать друг другу при передаче данных.
* Устройства в сети Ethernet имеют 48-битные адреса.
* Скорость передачи данных может достигать 1 гигабита в секунду.
* Максимальный размер отправляемых данных составляет 1518 байт, причем сами данные могут занимать от 46 до 1500 байт.
* Вся эта работа происходит с помощью сетевой карты (NIC), которая подключает ваш компьютер к сети.

**Протокол SLIP**

* SLIP (Serial Line IP) — это старый способ подключения компьютеров к Интернету через последовательные порты, например, RS-232.
* Он использует очень простую упаковку данных.
* Главный недостаток SLIP в том, что оба компьютера должны знать друг о друге свои IP-адреса, так как SLIP не может обмениваться этой информацией сам.
* Если вы используете SLIP, то линия не может быть использована другими протоколами, и нет контроля за передачей данных, что делает его менее надежным.

**Протокол PPP**

* PPP (Point-to-Point Protocol) — это более универсальный протокол для подключения двух устройств, который поддерживает разные сети, включая TCP/IP.
* Он может использоваться с различными технологиями, такими как ISDN и SONET.
* PPP позволяет объединять несколько каналов связи, чтобы увеличить скорость передачи данных.
* Он автоматически определяет адреса и контролирует качество передачи данных.
* В настоящее время PPP является самым популярным протоколом для подключения компьютеров к Интернету и заменил SLIP.

**Протоколы Межсетевого уровня**

**Протокол IP**. Его основной задачей является доставка ***дейтограмм*** (так называется единица передачи данных в терминологии IP). При этом протокол по определению является ***ненадежным*** и ***не поддерживающим соединения***.

Ненадежность протокола IP обусловлена тем, что нет гарантии, что посланная узлом сети дейтаграмма дойдет до места назначения. Сбой, произошедший на любом промежуточном узле сети, может привести к уничтожению дейтаграмм.

IP не ведет никакого учета очередности доставки дейтаграмм: каждая дейтаграмма обрабатывается независимо от остальных.

Главной особенностью IP-адреса является его независимость от физической устройства, подключенного к сети. Все устройства, имеющие IP-адрес, в терминологии протокола IP называются  ***хостами*** (host).

**Протокол ICMP.** (Протокол контроля сообщений в Internet) предназначен для транспортировки информации о сетевой деятельности и маршрутизации. IPCM сообщения представляют собой специально отформатированные IP-дейтаграммы, которым соответствуют определенные типы (15 типов) и коды сообщений. С помощью протокола ICMP осуществляется деятельность утилит достижимости (***ping***, ***traceroute***); регулируется частота отправки IP-дейтаграмм, оптимизируется MTU для маршрута передачи IP-дейтаграмм; доставляется хостам, маршрутизаторам и шлюзам всевозможная служебная информация; осуществляется поиск и переадресация маршрутизаторов; оптимизируются маршруты; диагностируются ошибки и оповещаются узлы IP-сети.

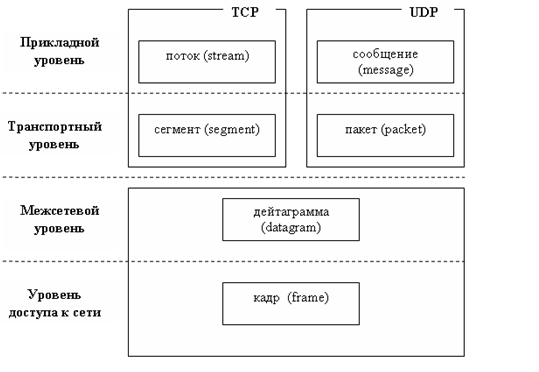
**Протокол ARP** Для установки соответствия между 32-разрядными IP-адресами и теми или иными MAC-адресами, действующими на канальном уровне, применяется механизм привязки адресов по протоколу ARP. Основной задачей ARP является динамическая (без вмешательства администратора, пользователя, прикладной программы) проекция IP-адресов в соответствующие МАС-адреса аппаратных средств. Эффективность работы ARP обеспечивается тем, что каждый хост кэширует специальную ARP-таблицу. Время существования записи в этой таблице составляет обычно 10- 20 минут с момента ее создания и может быть изменено с помощью параметров реестра [6]. Просмотреть текущее состояние ARP-таблицы можно с помощью команда ***arp***. Кроме того, протокол ARP используется для проверки существования в сети дублированного IP-адреса и разрешения запроса о собственном MAC-адресе хоста во время начальной загрузки.

**Протокол RARP**. Протокол RARP, (Reverse ARP), по своей функции противоположен протоколу ARP. RARP применятся для получения IP-адреса по MAC-адресу. В настоящее время протокол заменен на протокол Прикладного уровня DHCP, предлагающий более гибкий метод присвоение адресов.

**Протокол IPv6.** Наиболее распространенной на настоящий момент версией протокола IP является IPv4 – именно об этой версии говорилось выше. Этот протокол оказался самым удачным сетевым протоколом из всех, когда-либо созданных. Поэтому IPv4 быстро превратился в стандарт. Можно сказать, что протокол IPv4 стал жертвой собственной популярности, т.к. предлагаемое полезное пространство **адресов практически исчерпано**. решение этой проблемы появился протокол **IPv6**. Главным отличительным признаком протокола IPv6 является **128-битный адрес**, позволяющий увеличить адресное пространство боле чем на 20 порядков.

**Основная концепция IPv6**: каждый отдельный узел должен иметь собственный уникальный идентификатор интерфейса. Кроме того, протокол IPv6 требует соответствие идентификаторов интерфейсов формату IEEE EUI-64, позволяющему применять фиксированные (“зашитые” при изготовлении в специальную память сетевой платы) MAC-адреса сетевых плат. Например, 48-битный МАС-адрес платы Ethernet изначально предназначен для глобальной идентификации. Первые 24 бита этого адреса обозначают производителя платы (в соответствии с кодировкой ICANN) и индивидуальную партию изделия, а остальные 24 бита определяются производителем, с таким расчетом, чтобы каждый номер был уникален в пределах всей его продукции. Таким образом уникальный идентификатор интерфейса IPv6 на основе Ethernet cодержит в младших 64-х разрядах 128-битного адреса MAC-адрес платы Ethernet. Причем дополнение 48-бит адреса MAC-адреса до 64 бит осуществляется добавлением 16 бит (0xFFFF) между двумя его половинами. На первоначальном этапе внедрения IPv6 предполагается совместное использование обеих версий IP-протокола. При этом предполагается использование, так называемых, IPv4-совместимых и IPv4-преобразованных адресов. Другой интересной особенностью IPv6 является возможность ***автоконфигурации****.* Автокофигурация – это процесс, позволяющий хосту находить информацию для настройки собственных IP-параметров. В версии IPv6 основным средством позволяющим выполнять подобную настройку является протокол DHCP. Пересмотр процесса автоконфигурации вызван сложность администрирования сетей с большим количеством хостов и в связи с необходимостью поддерживать мобильных (перемещающихся) пользователей. Большое внимание в новой версии протокола уделяется вопросам безопасности

**Протоколы Транспортного уровня**

****

**Протокол UDP.** Протокол UDP является протоколом без установления соединения. Основными свойствами протокола являются:

1. отсутствие механизмов обеспечения надежности: пакеты не упорядочиваются, и их прием не подтверждается;
2. отсутствие гарантий доставки: пакеты оправляются без гарантии доставки, поэтому процесс Прикладного уровня (программа пользователя) должен сам отслеживать и обеспечивать (если это необходимо повторную передачу);
3. отсутствие обработки соединений: каждый оправляемый или получаемый пакет является независимой единицей работы; UDP не имеет методов установления, управления и завершения соединения между отправителем и получателем данных;
4. UDP может по требованию вычислять контрольную сумму для пакета данных, но проверка соответствия контрольной суммы ложится на процесс Прикладного уровня;
5. отсутствие буферизации: UDP оперирует только одним пакетом и вся работа по буферизации ложится на процесс Прикладного уровня;
6. UDP не содержит средств, позволяющих разбивать сообщение на несколько пакетов (фрагментировать) – вся эта работа возложена на процесс Прикладного уровня.

**Протокол TCP**. Протокол TCP является ***надежным*** байт-ориентированным протоколом с ***установлением соединения***. При получении дейтаграммы, в поле Protocol которой указан код 6 (код протокола TCP) IP-протокол извлекает из дейтаграммы данные, предназначенные для Транспортного уровня, и переправляет их модулю протокола TCP. Модуль TCP анализирует служебную информацию заголовка сегмента (структура TCP-сегмента приведена в [5,6]), проверяет целостность (по контрольной сумме) и порядок прихода данных, а также подтверждает их прием отправляющей стороне. По мере получения правильной последовательности неискаженных данных процесса отправителя, используя поле Destination Port Number заголовка сегмента, модуль TCP переправляет эти данные процессу получателя.

Протокол TCP рассматривает данные отправителя как непрерывный не интерпретируемый (не содержащий управляющих для TCP команд) поток октетов. При этом TCP при отправке разделяет (если это необходимо) этот поток на части (TCP-сегменты) и объединяет полученные от протокола IP-дейтаграммы при приеме данных. Немедленную отправку данных может быть затребовано процессом с помощью специальной функции PUCH, иначе TCP сам решает, когда отправлять данные отправителя и когда их передавать получателю.

Модуль TCP обеспечивает защиту от повреждения, потери, дублирования и нарушения очередности получения данных. Для выполнения этих задач все октеты в потоке данных пронумерованы в возрастающем порядке. Заголовок каждого сегмента содержит число октетов и порядковый номер первого октета данных в данном сегменте. Каждый сегмент данных сопровождается контрольной суммой, позволяющей обнаружить повреждение данных. При отправлении некоторого числа последовательных октетов данных, отправитель ожидает подтверждение приема. Если подтверждения не приходит, то предполагается, что группа октетов не дошла по назначению или была повреждена – в этом случае предпринимается повторная попытка переслать данные.

**Надежный и ненадежный протоколы**:

**Надежный** протокол обеспечивает доставку данных без потерь, дублирования или изменения порядка передачи. Он гарантирует, что данные будут успешно доставлены получателю и подтверждает получение каждого пакета данных. Примером надежного протокола является TCP (Transmission Control Protocol), который используется в Интернете для передачи данных с гарантией доставки.

**Ненадежный** протокол, напротив, не гарантирует доставку данных. Он может потерять, дублировать или изменить порядок пакетов данных без уведомления об этом. Примером ненадежного протокола является UDP (User Datagram Protocol), который широко используется для стриминга видео, аудио и игр, где небольшая задержка важнее, чем гарантированная доставка данных.

**Протоколы с установкой соединени**я требуют предварительного установления логического канала связи между отправителем и получателем перед передачей данных. Этот процесс называется "установкой соединения" и обычно включает в себя обмен определенными управляющими сообщениями. Примером протокола с установкой соединения является TCP, который устанавливает виртуальное соединение между клиентом и сервером перед передачей данных.

**Протоколы без установки соединения**, наоборот, не требуют предварительной установки логического канала связи. Они передают данные независимо друг от друга. Примером протокола без установки соединения является UDP, который просто отправляет пакеты данных без необходимости установки соединения.

Протоколы, **ориентированные на поток**, предоставляют непрерывный поток данных между отправителем и получателем. Они гарантируют, что данные будут доставлены в том же порядке, в котором были отправлены, без разделения на отдельные сообщения. TCP является примером протокола, ориентированного на поток.

**Протоколы, ориентированные на сообщения**, передают данные в виде отдельных сообщений или пакетов данных. Каждое сообщение может быть обработано независимо от других сообщений. UDP является

***1) отсутствие механизмов обеспечения надежности;***

***2) отсутствие гарантий доставки;***

***3) отсутствие обработки соединений;***

***4) отсутствие буферизации;***

***5) отсутствие фрагментации;***

***6) отсутствие пересчёта контрольной суммы.***

# Понятие сокета. Основные параметры сокета.

***Сокет*** – совокупность IP-адреса и номера порта. Сокет однозначно идентифицирует прикладной процесс в сети TCP/IP. Следует помнить, что одни и те же номера портов могут быть использованы как для протокола UDP, так и для протокола TCP.

Для создания сокета используется функция socket. Описание функции приводится на рисунке 3.7.1.

// -- **создать сокет**

// **Назначение**: функция позволяет создать сокет (точнее

// дескриптор сокета) и задать его характеристики

//

**SOCKET socket(**

**int af,** //[in] address family формат адреса

**int type,** //[in] socket type тип сокета

**int prot** //[in] protocol протокол

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает дескриптор сокета, в другом

// случае возвращается **INVALID\_SOCKET**

// **Примечания:** - параметр **af** для стека TCP/IP принимает

// значение **AF\_INET,** означает работу с IPv4-адресами;

// - параметр **type** может принимать два значения:

// **SOCK\_DGRAM** – сокет, ориентированный на

// сообщения(UDP); **SOCK\_STREEM** – сокет

// ориентированный на поток (TCP);

// старший номер версии;

// - параметр **prot** определяет протокол

// транспортного уровня: для TCP/IP можно

// указать 0 или NULL

Рисунок 3.7.1. Функция socket

После завершения работы с сокетом, обычно, его закрывают (освобождают ресурс). Для закрытия сокета применяется функция closesocket. Описание этой функции приводится на рисунке 3.7.2.

// -- **закрыть существующий сокет**

// **Назначение**: переводит сокет в неработоспособное состояние и

// освобождает все ресурсы связанные с ним

//

**SOCKET closesocket(**

**SOCKET s,** //[in] дескриптор сокета

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает нуль, в другом случае

// возвращается **SOCKET\_ERROR**

Рисунок 3.7.2. Функция closesocket

Для установки параметров существующего сокета используется функция bind. Описание функции приводится на рисунке 3.8.1.

// -- **связать сокет с параметрами**

// **Назначение**: функция связывает существующий сокет с

// с параметрами, находящимися в структуре

// SOCKADDR\_IN

//

**int bind(**

**SOCKET s,** //[in] сокет

**const struct sockaddr\_in\* a,** //[in]указатель на **SOCKADDR\_IN**

**int la** //[in] длина **SOCKADDR\_IN** в байтах

**)**

// **Код возврата:** в случае успешного завершения функция

// возвращает нуль, в случае ошибки

// возвращается **SOCKET\_ERROR**

Рисунок 3.8.1. Функция bind

Функция связывает дескриптор сокета и структуру SOCKADDR\_IN, которая предназначена для хранения параметров сокета. Шаблон структуры SOCKADDR\_IN содержится в файле Winsock2.h. Рис 3.8.2

**#define INADDR\_ANY (u\_long)0x00000000** //любой адрес

**#define INADDR\_LOOPBACK 0x7f000001** // внутренняя петля

**#define INADDR\_BROADCAST (u\_long)0xffffffff** // широковещание

**#define INADDR\_NONE 0xffffffff** // нет адреса

**#define ADDR\_ANY INADDR\_ANY** // любой адрес

**struct in\_addr**

**{** // IP-адрес

**union {**

**struct { u\_char s\_b1,s\_b2,s\_b3,s\_b4; } S\_un\_b;**

**struct { u\_short s\_w1,s\_w2; } S\_un\_w;**

**u\_long S\_addr;**

**} S\_un;**

**#define s\_addr S\_un.S\_addr** // 32-битный IP-адрес

**#define s\_host S\_un.S\_un\_b.s\_b2**

**#define s\_net S\_un.S\_un\_b.s\_b1**

**#define s\_imp S\_un.S\_un\_w.s\_w2**

**#define s\_impno S\_un.S\_un\_b.s\_b4**

**#define s\_lh S\_un.S\_un\_b.s\_b3**

**}**

**struct sockaddr\_in {**

**short sin\_family;** //тип сетевого адреса

**u\_short sin\_port;** // номер порта

**struct in\_addr sin\_addr;** // IP-адрес

**char sin\_zero[8];** // резерв

**};**

**typedef struct sockaddr\_in SOCKADDR\_IN;** //

**typedef struct sockaddr\_in \*PSOCKADDR\_IN;**

**typedef struct sockaddr\_in FAR \*LPSOCKADDR\_IN;**

Рисунок 3.8.2. Структура SOCKADDR\_IN

IP-адрес и номер порта в структуре SOCKADDR\_IN хранятся в специальном сетевом формате. Этот формат отличается, от формата компьютеров с архитектурой Intel. В составе Winsock2 имеются функции, позволяющие преобразовывать форматы данных.

Для преобразования номера порта в формат TCP/IP следует использовать функцию htons. Функция ntohs является обратной функцией, предназначена для преобразования двух байтов в формате TCP/IP в формат u\_short.

Полезной является функция inet\_addr, предназначенная для преобразования символьного представления IPv4-адреса в формат TCP/IP. Функция inet\_ntoa предназначена для обратного преобразования из сетевого представления в символьный формат.

Функция bind связывает сокет с параметрами, заданными в структуре SOCKADDR\_IN. Структура содержит три значения (параметры сокета): тип используемого адреса (константа AF\_INET используется для обозначения семейства IP-адресов); номер порта (устанавливается значение 2000 с помощью функции htons) и адрес интерфейса . Последний параметр определяет собственный IP-адрес сервера. При этом предполагается, что хост, в общем случае, может иметь несколько IP-интерфейсов. Если требуется использовать определенный IP-интерфейс хоста, то необходимо его здесь указать. Если выбор IP-адреса не является важным или IP-интерфейс один на хосте, то следует указать значение INADDR\_ANY (как это сделано в примере). Программа клиента для пересылки сообщений (обратите внимание, что при создании сокета использовался параметр со значением SOCKET\_DGRAM), должна их отправлять именно этому сокету (т.е. указывать его IP-адрес и его номер порта).

//................................................................

**SOCKET sS;** // серверный сокет

**if ((sS = socket(AF\_INET, SOCK\_DGRAM, NULL))== INVALID\_SOCKET)**

**throw SetErrorMsgText("socket:",WSAGetLastError());**

**SOCKADDR\_IN serv;** // параметры сокета sS

**serv.sin\_family = AF\_INET;** // используется IP-адресация

**serv.sin\_port = htons(2000);** // порт 2000

**serv.sin\_addr.s\_addr = INADDR\_ANY;** // любой собственный IP-адрес

**if (bind(sS,(LPSOCKADDR)&serv, sizeof(serv))== SOCKET\_ERROR)**

**throw SetErrorMsgText("bind:",WSAGetLastError());**

//................................................................

# Интерфейс Nimed Pipe.

Именованным каналом называется объект ядра операционной системы, который обеспечивает обмен данными между процессами, выполняющимися на компьютерах в одной локальной сети. Процесс, создающий именованный канал, называется ***сервером именованного канала***. Процессы, которые связываются с именованным каналом, называются ***клиентами именованного канала***. Любой именованный канал идентифицируется своим именем, которое задается при создании канала.

Именованные каналы бывают: ***дуплексные*** (позволяющие передавать данные в обе стороны) и ***полудуплексны***е (позволяющие передавать данные только в одну сторону). Передача данных в именованном канале может осуществляться как потоком, так и сообщениями. Обмен данными в канале может быть ***синхронным*** и ***асинхронным***.

Для использования функций интерфейса Named Pipe в программе на языке C++ необходимо включить в ее текст заголовочный файл Windows.h. Сами функции интерфейса располагаются в библиотеке KERNEL32.DLL ядра операционной системы.

В таблице 4.2.1 перечислены основные функции интерфейса Named Pipe. Следует отметить, что функции CreateFile, ReadFile, WriteFile, которые тоже перечислены в таблице 4.2.1, применяются не только для работы с именованными каналами, но и для работы с файловой системой, с сокетами и т.д. Поэтому эти универсальные функции часто не указывают в составе Named Pipe API.

Все функции Named Pipe API можно разбить на три группы: функции управления каналом (создать канал, соединить сервер с каналом, открыть канал, получить информацию об именованном канале, получить состояние канала, изменить характеристики канала); функции обмена данными (писать в канал, читать из канала, копировать данные канала) и функции для работы с транзакциями.

Таблица 4.2.1

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **CallNamedPipe** | Выполнить одну транзакцию |
| **ConnectNamedPipe** | Соединить сервер с каналом |
| **CreateFile** | Открыть канал |
| **CreateNamedPipe** | Создать именованный канал |
| **DisconnectNamedPipe** | Закончить обмен данными |
| **GetNamedPipeHandleState** | Получить состояние канала |
| **GetNamedPipeInfo** | Получить информацию об именованном канале |
| **PeekNamedPipe** | Копировать данные канала |
| **ReadFile** | Читать данные из канала |
| **SetNamedPipeHandleState** | Изменить характеристики канала |
| **TrasactNamedPipe** | Писать и читать данные канала |
| **WaitNamedPipe** | Определить доступность канала |
| **WriteFile** | Писать данные в канал |

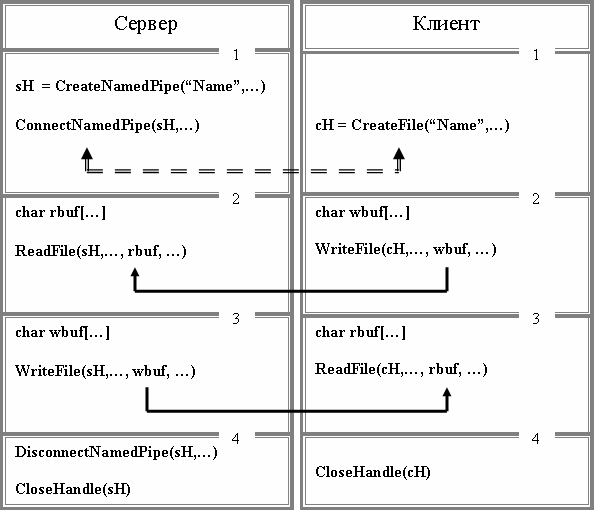


Рисунок 4.2.1. Схема взаимодействия процессов, использующих Named Pipe API

Следует сразу отметить, что при создании именованного канала в программе на языке С++, одновременно создается дескриптор (HANDEL), который потом используется другими функциями Named Pipe API для работы с данным экземпляром именованного канала. По окончании работы с каналом необходимо эти дескрипторы закрыть с помощью функции CloseHandle Win32 API.

На рисунке 4.2.1 изображены две программы, реализующие два процесса распределенного приложения. Каждая из программ разбита на четыре блока. Сплошными направленными линиями обозначается движение данных от одного процесса к другому. Прерывистой линией обозначается синхронизация процессов.

В первом блоке программы сервера выполняются две функции: CreateNamedPipe (создать именованный канал) и ConnectNamedPipe (подсоединить сервер к каналу). Одним из параметров функции CreateNamedPipe является имя канала (строка), а результатом ее работы (возвращаемым значением) является дескриптор (HANDEL) канала. Функция ConnectNamedPipe приостанавливает выполнение программы клиента до момента, пока программа клиента не выполнит функцию CreateFile.

Во втором и третьем блоках программы сервера осуществляется ввод и вывод данных (функции ReadFile и WriteFile) в именованном канале. Следует обратить внимание, что функции осуществляющие ввод и вывод используют в качестве одного из своих параметров дескриптор именованного канала.

В последнем четвертом блоке программы сервер разрывает соединение с помощью функции DisconnectNamedPipe и закрывает дескриптор именованного канала.

Для программы клиента остается пояснить только первый блок, т.к. всем остальным блокам, есть аналогичные в программе сервера. В первом блоке программы клиента выполняется функция CreateFile, одним из параметров которой является строка с именем канала. Если к моменту выполнения функции канал уже создан и сервер подсоединился к каналу, то функция CreateFile возвращает дескриптор именованного канала, который потом используется в других функциях программы клиента. Иногда перед выполнением функции CreateFile, выполняют функцию WaitNamedPipe, позволяющую определить доступность экземпляра канала.

Как уже отмечалось, передача данных может осуществляться как потоком, так и сообщениями. Передача данных потоком возможна в том случае, если сервер и клиент работают на одном компьютере и использует локальные имена канала, в других случаях передача данных осуществляется сообщениями. Схема изображенная на рисунке 4.2.1 является общей для этих двух случаев.

**Создание именованного канала**

**С**ервером именного канала является процесс, создающий именованный канал. Именованный канал создается с помощью функции CreateNamedPipe (рисунок 4.3.1).

// **-- создать именованный канал**

// **Назначение:** функция предназначена для создания именованного канала

**HANDLE CreateNamedPipe**   **(**

**LPCTSTR pname,** // [in] символическое имя канала

**DWORD omode,** // [in] атрибуты канала; передача данных (чтение/запись)

**DWORD pmode,** // [in] режимы передачи данных (смс/потоки)

**DWORD pimax,** // [in] макс. к-во экземпляров канала

**DWORD osize,** // [in] размер выходного буфера

**DWORD isize,** // [in] размер входного буфера

**DWORD timeo,** // [in] время ожидания связи с клиентом

**LPSECURITY\_ATTRIBUTES sattr** // [in] атрибуты безопасности

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает дескриптор именованного канала, иначе

// возможны следующие значения:

// **INVALID\_HANDLE\_VALUE** – неудачное завершение;

// **ERROR\_INVALID\_PARAMETER** - значение параметра **pimax**

// превосходит величину **PIPE\_UNLIMITED\_INSTANCES**

// **Примечание: pname** - указывает на строку именем канала в

// локальном формате;

// **omode** - задает флаги направления передачи, напримерфлаг **FILE\_ACCESS\_DUPLEX** разрешает чтение и запись в канал; помимо направления здесь могут быть заданы флаги асинхронной передачи, режимы буферизации и безопасности;

// **pmode** – задает флаги способов передачи данных, например, флаг **PIPE\_TYPE\_MESSAGE|PIPE\_WAIT** разрешает запись данных сообщениями в синхронном режиме, а флаг

//  **PIPE\_READTYPE\_MESSAGE|PIPE\_WAIT** разрешает чтение сообщений в синхронном режиме;

// **pimax –** максимальное количество экземпляров канала, значение должно находиться в пределах от 1 до **PIPE\_UNLIMITED\_INSTANCES**;

// **osize, isize –** значения рассматриваются Windows только как пожелания пользователя (рекомендуется **0**);

// **timeo** – параметр устанавливает время ожидания связи c сервером в миллисекундах для функции **WаitNamedPipe** с параметром **NMWAIT\_USE\_DEFAULT\_WAIT;** может быть установлено значение **INFINITE** (ждать бесконечно);

// **sattr –** для установки атрибутов безопасности по умолчанию, следует установить значение **NULL**

Рисунок 4.3.1. Функция CreateNamedPipe

Первый параметр функции CreateNamedPipe **–** указатель на строку имени канала. В зависимости от контекста в функциях используется два формата имени канала: локальный формат (рисунок 4.3.2) и сетевой формат (рисунок 4.3.3).

**\\.\pipe\xxxxx**

где: **точка (.) -** обозначает локальный компьютер;

**pipe -** фиксированное слово;

**xxxxx -** имя канала

,...

Рисунок 4.3.2. Локальный формат имени канала

**\\servname\pipe\xxxxx**

где:  **servname -** имя компьютера – сервера именованного канала;

**pipe -** фиксированное слово;

**xxxxx -** имя канала

,...

Рисунок 4.3.3. Сетевой формат имени канала

Для связи сервера по одному именованному каналу с несколькими клиентами, сервер должен создать несколько экземпляров этого канала. Каждый экземпляр создается функцией CreateNamedPipe, которая возвращает дескриптор экземпляра именованного канала. Отметим, что поток создающий экземпляр именованного канала должен иметь право доступа FILE\_CREATE\_PIPE\_INSTANCE. Этим правом по умолчанию обладает поток создавший канал. Подробнее о применении нескольких экземпляров одного канала можно ознакомиться в [4] .

// **-- соединить сервер с именованным каналом**

// **Назначение:** функция предназначена для ожидания сервером

// подсоединения к экземпляру именованного канала

// клиента

**BOOL ConnectNamedPipe**

**(**

**HANDLE hP,** // [in] дескриптор именованного канала

**LPOVERLAPPED ol** // [in,out] используется для асинхр. связи

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

// **Примечание:** параметр **ol** используется только в том случае,

// если используется асинхронная связь, в случае

// синхронной связи можно установить значение **NULL**

Рисунок 4.3.4. Функция ConnectNamedPipe

После того, как сервер создал именованный канал, он должен дождаться соединения клиента с этим сервером. Для этого сервер должен вызвать функцию ConnectNamedPipe (рисунок 4.3.4).

**Соединение клиентов с именованным каналом**

Прежде чем соединиться с именованным каналом, клиент может определить: доступен ли какой либо экземпляр этого канала. С этой целью клиент может вызывать функцию WaitNamedPipe (рисунок 4.3.1).

После обнаружения свободного канала, клиент может установить связь с каналом помощью функции CreateFile (рисунок 4.3.2). После успешного выполнения функции клиент и сервер могут обмениваться данными.

Здесь не рассматриваются атрибуты безопасности, которые могут быть определены при вызовах функций CreateNamedPipe и CreateFile. Однако поясним, что для организации обмена необходимо, чтобы атрибуты безопасности в этих функциях были согласованными.

// **-- определить доступность канала**

// **Назначение:** функция предназначена для ожидания клиентом

// доступного именованного канала

**BOOL WаitNamedPipe**

**(**

**LPCTSTR pn,**  // [in] символическое имя канала

**DWORD to** // [in] интервал ожидания (мс)

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

// **Примечание:-** если используется локальный канал, то имя

// канала задается в локальном формате, если же канал

// создан на другом компьютере, то имя канала следует

// задавать в сетевом формате;

// -параметр **to** определяет интервал времени

// ожидания (в миллисекундах) освобождения экземпляра

// канала; если для параметра **to** установлено значение

// **NMWAIT\_USE\_DEFAULT\_WAIT**, то интервал определяется

// параметром **timeo** функции **CreateNamedPipe**; если

// установлено значение **NMWAIT\_WAIT\_FOREVER**, то время

// ожидания бесконечно

Рисунок 4.4.1. Функция WaitNamedPipe

При установке в функциях Named Pipe API атрибутов безопасности по умолчанию, как это сделано во всех приведенных здесь примерах, подсоединиться к каналу удаленный клиент сможет только в том случае, если он запущен от того же имени пользователя и с тем же паролем, что и сервер.

Кроме того, следует обратить внимание на правильное использование имени канала. На рисунке 4.3.5 при создании канала с помощью функции CreateNamedPipe использовалось имя канала [\\.\pipe\ConsolePipe](file:///\\.\pipe\ConsolePipe). При записи строки с именем канала в программе на языке C++, символ обратного слеша, в соответствии с правилами языка, удваиваются.

При использовании форматов имени канала, необходимо помнить, что:

1. при создании канала всегда используется локальный формат имени;
2. если клиент удаленный (на другом компьютере), то он всегда должен использовать сетевой формат имени; при этом обмен данными между клиентом и сервером осуществляется сообщениями;
3. если клиент локальный и использует сетевой формат имени при подсоединении к каналу (функция CreateFile), то обмен данными осуществляется сообщениями;
4. если клиент локальный и использует локальный формат имени канала, то обмен данными осуществляется потоком.

// **-- открыть канал**

// **Назначение:** функция предназначена для подключения клиента

// к именованному каналу

**HANDLE CreateFile**

**(**

**LPCTSTR pname,** // [in] символическое имя канала

**DWORD accss,** // [in] чтение или запись в канал

**DWORD share,** // [in] режим совместного использования

**LPSECURITY\_ATTRIBUTES sattr** // [in] атрибуты безопасности

**DWORD oflag,** // [in] флаг открытия канала

**DWORD aflag,** // [in] флаги и атрибуты

**HANDLE exten,** // [in] дополнительные атрибуты

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает дескриптор именованного канала, иначе

// **INVALID\_HANDLE\_VALUE** – неудачное завершение

// **Примечание:-** параметр **pname** указывается в локальном или

// сетевом формате: в зависимости от способа применения

// - параметр **accss** может принимать значения **GENERIC\_READ**

// (чтение), **GENERIC\_WRITE** (запись) или

// **GENERIC\_READ|** **GENERIC\_WRITE** (запись,чтение)

// - параметр **share** можетпринимать значения

// **FILE\_SHARE\_READ** (совместное чтение),

// **FILE\_SHARE\_WRITE** (совместная запись),

// **FILE\_SHARE\_READ| FILE\_SHARE\_WRITE** (чтение и запись);

// - параметр **sattr** для установки атрибутов безопасности

// по умолчанию, следует установить значение **NULL;**

// - значение параметра **oflag** всегдаустанавливаетсяв

// **OPEN\_EXISTING**(открытие существующего канала);

// - значение параметров **aflag** и **exten** можно установить в

// **NULL**, что соответствует значениям по умолчанию

Рисунок 4.4.2. Функция CreateFile

Если не существует экземпляров именованного канала с тем именем, которое указано в параметре функции WaitNamedPipe, то эта функция немедленно заканчивается неудачей (FALSE), независимо от установленного в параметре функции значения интервала ожидания. Если же канал создан, но сервер не выполнил функцию ConnectNamedPipe, то функция WaitNamedPipe на стороне клиента все равно вернет FALSE и сформирует диагностический код (функции GetLastError) ERROR\_PIPE\_CONNECTED. Даже в том случае, если функция WaitNamedPipe обнаружит свободный экземпляр канала (и вернет TRUE), то все равно нет гарантии, что до выполнения функции CreateFile этот канал не будет занят другим клиентом. Все эти замечания, делают применение функции WaitNamedPipe в большинстве случаев нецелесообразным.

**Обмен данными по именованному каналу**

Для обмена данными по именованному каналу используются три функции: ReadFile, WriteFile и PeekNamedPipe (рисунки 4.5.1, 4.5.2, 4.5.3).

// **-- читать данные из канала**

// **Назначение:** функция предназначена чтения данных из

// именованного канала

**BOOL ReadFile**

**(**

**HANDLE hP,**  // [in] дескриптор канала

**LPVOID pb,** // [out] указатель на буфер ввода

**DWORD sb,** //[in] количество читаемых байт

**LPDWORD ps,** // [out] количество прочитанных байт

**LPOVERLAPPED ol** // [in,out] для асинхронной обработки

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

// **Примечание** если не используется асинхронная обработка

// параметр **ol** рекомендуется установить в **NULL**

Рисунок 4.5.1. Функция ReadFile

// **-- писать данные в канал**

// **Назначение:** функция предназначена записи данных в

// именованный канал

**BOOL WriteFile**

**(**

**HANDLE hP,**  // [in] дескриптор канала

**LPVOID pb,** // [in] указатель на буфер вывода

**DWORD sb,** //[in] количество записываемых байт

**LPDWORD ps,** // [out] количество записанных байт

**LPOVERLAPPED ol** // [in,out] для асинхронной обработки

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

// **Примечание** если не используется асинхронная обработка

// параметр **ol** рекомендуется установить в **NULL**

Рисунок 4.5.2. Функция WriteFile

Параметры функций ReadFile и WriteFile достаточно просты и не требуют дополнительного пояснения. Функция PeekNamedPipe копирует данные из канала в буфер. При этом данные не извлекаются и их еще можно считать (извлечь) с помощью функции ReadFile.

// **-- копировать данные канала**

// **Назначение:** функция предназначена для получения данных

// из канала без извлечения

**BOOL PeekNamedPipe**

**(**

**HANDLE hP,**  // [in] дескриптор канала

**LPVOID pb,** // [out] указатель на буфер

**DWORD sb,** //[in] размер буфера

**LPDWORD pi,** // [out] количество прочитанных байт

**LPDWORD pa,** // [out] количество доступных байт

**LPDWORD pr,** // [out] количество непрочитанных байт

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

Рисунок 4.5.3. Функция PeekNamedPipe

**Передача транзакций по именованному каналу**

Для обмена сообщениями по сети может использоваться функция TansactNamedPipe (рисунок 4.6.1), которая объединяет операции чтения и записи в одну операцию. Такую объединенную операцию называют ***транзакцией***  именованного канала. Функция TansactNamedPipe может быть использована только в том случае, если сервер именованного канала установил флаг PIPE\_TYPE\_MESSAGE.

Применение TransactNamedPipe целесообразно, если другая сторона канала может обеспечить достаточно быструю реакцию и отправить ответ на пришедшее сообщение.

// **-- писать и читать данные канала**

// **Назначение:** функция предназначена для выполнения записи в

// канал и чтения из канала за одну операцию

**BOOL TransactNamedPipe**

**(**

**HANDLE hP,**  // [in] дескриптор канала

**LPVOID pw,** // [in] указатель на буфер для записи

**DWORD sw,** //[in] размер буфера для записи

**LPVOID pr,** // [out] указатель на буфер для чтения

**DWORD sr,** //[in] размер буфера для чтения

**LPDWORD pi,** // [out] количество прочитанных байт

**LPOVERLAPPED ol** // [in,out] для асинхронного доступа

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

// Примечание: параметр **ol** используется для асинхронного

// доступа к каналу, если асинхронный доступ не

// не предполагается, то следует указать **NULL**

Рисунок 4.6.1. Функция TransactNamedPipe

Часто взаимодействие сервера и клиента сводится к простому запросу клиента к серверу для получения некоторого сервиса. После выполнения запрошенной клиентом сервисной услуги, сервер информирует клиента о результате своей работы. Т.е. речь идет об одиночных эпизодических транзакциях.

Если требуется передать только одну транзакцию, то используют функцию CallNamedPipe (рисунок 4.6.2), которая работает следующим образом.

Сначала осуществляется установка связи с именованным каналом, имя которого указывается в параметрах функции. При этом именованный канал должен быть открыт в режиме данных сообщениями. После установки связи функция пересылает в канал единственное сообщение и получает одно сообщение в ответ. После обмена данными осуществляется разрыв связи с именованным каналом.

// **-- выполнить одну транзакцию**

// **Назначение:** функция предназначена для установки связи с

// именованным каналом, выполнения одной транзакции

// и разрыва связи

**BOOL CallNamedPipe**

**(**

**LPCTSTR nP,**  // [in] указатель на имя канала

**LPVOID pw,** // [in] указатель на буфер для записи

**DWORD sw,** //[in] размер буфера для записи

**LPVOID pr,** // [out] указатель на буфер для чтения

**DWORD sr,** //[in] размер буфера для чтения

**LPDWORD pi,** // [out] количество прочитанных байт

**DWORD to** // [in] интервал ожидания

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

// **Примечание:** параметр **to** устанавливает интервал времени в

// в миллисекундах; кроме того, здесь могут быть

// установлены те же значения, что и в функции

// **WaitNamedPipe**

Рисунок 4.6.2. Функция CallNamedPipe

**Определение состояния и изменение характеристик именованного канала**

Для получения информации о созданном именованном канале можно использовать две функции GetNamedPipeInfo и GetNamedPipeHandleState (рисунки 4.7.1 и 4.7.2).

// **-- получить информацию об именованном канале**

// **Назначение:** функция предназначена для получения

// статических характеристик именованного канала

**BOOL GetNamedPipeInfo**

**(**

**HANDLE hP,**  // [in] дескриптор именованного канала

**LPDWORD pfg,** // [in] указатель на флаг-тип канала

**LPDWORD psw,** //[out] указатель на размер выходного буфера

**LPDWORD psr,** //[out] указатель на размер входного буфера

**LPDWORD pmi,** //[out] указатель на макс. к-во экземпляров канала

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

// **Примечание:** параметр **pfg** указывает на переменную типа

// **DWORD**, в которой установлен тип именованного канала,

// атрибуты которого запрашиваются; для установки

// этой переменной должны использоваться константы:

// **PIPE\_CLIENT\_END**, **PIPE\_SERVER\_END** - для обозначения

// типа используемого в функции дескриптора;

// **PIPE\_TYPE\_BYTE**, **PIPE\_TYPE\_MESSAGE** – для установки

// типа передачи (поток и сообщения)

Рисунок 4.7.1. Функция GetNamedPipeInfo

// **-- получить состояния именованного канала**

// **Назначение:** функция предназначена для получения

// динамических характеристик именованного канала

**BOOL GetNamedPipeHandleState**

**(**

**HANDLE hP,**  // [in] дескриптор именованного канала

**LPDWORD pst,** // [out] указатель на состояние канала

**LPDWORD pci,** //[out] указатель на к-во экземпляров каналов

**LPDWORD pcc,** //[out] указатель на макс. к-во байт

**LPDWORD pto,** //[out] указатель на интервал задержки

**LPTSTR pun,** // [out] указатель на имя владельца канала

**DWORD lun** //[in] длина буфера для имени владельца канала

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

// **Примечание: -** параметр **pst** указывает на переменную типа

// **DWORD**, в которой установлена комбинация значений:

// **PIPE\_NOWAIT** - канал не блокирован;

// **PIPE\_READMODE\_MESSAGE** – канал открыт в режиме

// передачи сообщениями;

// - параметр **pcc** - указывает на максимальное

// количество байтов, которые клиент именного канала

// должен записать в канал перед передачей серверу;

// - параметр **pto** - указывает на количество миллисекунд

// которые должно пройти прежде, чем данные будут

// переданы

Рисунок 4.7.2. Функция GetNamedPipeHandleState

Чаще всего функция GetNamedPipeInfo используется на стороне клиента после открытия канала для выяснения размеров буферов, установленных операционной системой при создании канала.

Функция GetNametPipeHandleState используется для получения динамических параметров (которые могут быть изменены) именованного канала. Для изменения некоторых параметров может быть использована функция SetNamedHandleState (рисунок 4.7.3).

// **-- изменить характеристики канала**

// **Назначение:** функция предназначена для изменения

// динамических характеристик именованного канала

**BOOL SetNamedPipeHandleState**

**(**

**HANDLE hP,**  // [in] дескриптор именованного канала

**LPDWORD pst,** // [in] указатель на новое состояние канала

**LPDWORD pcc,** //[in] указатель на макс. к-во байтов

**LPDWORD pto** //[in] указатель на интервал задержки

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

Рисунок 4.7.2. Функция SetNamedPipeHandleState

# Интерфейс MailSlot.

В этой главе рассматривается еще один IPC – механизм, поддерживаемый операционной системой Windows и имеющий название **Mailslots** (почтовый ящик). Также как и Named Pipe механизм Mailslots может быть использован для обмена данными между распределенными в локальной сети процессами.

**Назначение и состав интерфейса Mailslot**

Почтовым ящиком (Mailslot) называется объект ядра операционной системы, который обеспечивает передачу данных от процессов-клиентов к процессам-серверам, выполняющимся на компьютерах в одной локальной сети. Процесс, создающий почтовый ящик называется ***сервером почтового ящика***. Процессы, которые связываются с почтовым ящиком, называются ***клиентами почтового*** ящика.

Каждый почтовый ящик имеет имя, которое определяется сервером при создании и используется клиентами для доступа. Передача может осуществляться только сообщениями и в одном направлении – от клиента к серверу. Обмен данными может происходить в синхронном и асинхронном режимах. Допускается создание нескольких серверов с одинаковым именем почтового ящика – в этом случае все отправляемые клиентом сообщения будут поступать во все почтовые ящики, имеющие имя, указанное клиентом. Однако, следует сказать, что такая рассылка сообщений возможна только в том случае, когда длина отправляемых сообщений не превышает 425 байт.

В том случае, если клиент отправляет сообщение размером меньше, чем 425 байт, то пересылка осуществляется без гарантии доставки. Пересылка сообщения размером более 425 байт возможна только от одного клиента к одному серверу.

Перечень функций интерфейса Mailslot API приводится в таблице 5.2.1. Функции CreateFile, ReadFile, WriteFile являются универсальными и используются также для работы с именованными каналами, файловой системой, сокетами и т.д.

Таблица 5.2.1

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **CreateFile** | Открыть почтовый ящик |
| **CreateMailslot** | Создать почтовый ящик |
| **GetMailslotInfo** | Получить информацию о почтовом ящике |
| **ReadFile** | Читать данные из почтового ящика |
| **SetMailslotInfo** | Изменить время ожидания сообщения |
| **WriteFile** | Писать данные в почтовый ящик |

Как и в случае с именованными каналами, для использования функций Mailslot API в программе на языке С++ достаточно включить в ее текст заголовочный файл Widows.h.

На рисунке 5.2.1 изображена схема взаимодействия процесса-сервера и процесса-клиента в простейшем случае. Каждая программа разбита на три блока. Сплошной направленной линией обозначается движение данных от одного процесса к другому.

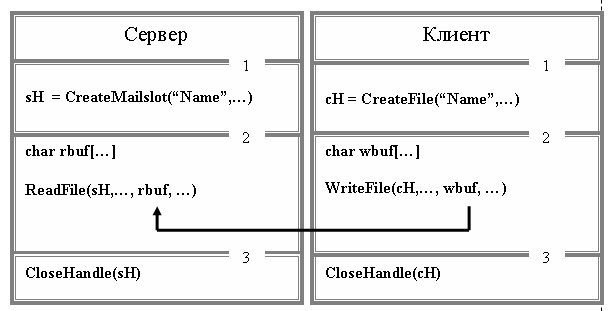


Рисунок 5.2.1. Схема взаимодействия процессов использующих Mailslot API

В первом блоке программы сервера выполняется функция CreateMailslot, создающая почтовый ящик. В случае успешного завершения функция возвращает дескриптор почтового ящика, который будет использоваться дальше. Кроме того, один из параметров функции CreateMailslot определяет время ожидания функцией ReadFile, очередного сообщения от клиента. В простейшем случае можно установить бесконечное время ожидания. Во втором блоке сервера осуществляется считывание данных из почтового ящика. В последнем третьем блоке сервера закрывается дескриптор почтового ящика, что приводит к его уничтожению.

В первом блоке программы клиента осуществляется подсоединение клиента к почтовому ящику с помощью функции CreateFile (открыть почтовый ящик). В случае успешного выполнения, функции формирует дескриптор почтового ящика, который потом используется функцией WriteFile (второй блок клиента) для записи данных в почтовый сервер. При завершении программы, следует закрыть дескриптор почтового ящика с помощью функции CloseHandle.

В принципе, между процессами обмен данными можно организовать в обе стороны. Для этого необходимо в рамках каждого процесса создать свой почтовый ящик, который бы использовался для приема сообщений.

**Создание почтового ящика**

Для создания почтового ящика используется функция CreateMailslot (рисунок 5.3.1).

// **-- создать почтовый ящик**

// **Назначение:** функция предназначена для создания почтового

// ящика

**HANDLE CreateMailslot**

**(**

**LPCTSTR pname,**  // [in] символическое имя ящика

**DWORD maxms,** // [in] максимальная длина сообщения

**DWORD timeo,** //[in] интервал ожидания

**LPSECURITY\_ATTRIBUTES sattr** // [in] атрибуты безопасности

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает дескриптор почтового ящика, иначе

// значение **INVALID\_HANDLE\_VALUE**

// **Примечание: pname** - указывает на строку именем канала в

// **локальном формате**;

// **timeo** – параметр устанавливает время ожидания

// cобщения функцией **ReadFile;** для задания бесконечного

// ожидания, следует установить значение

// **MAILSLOT\_WAIT\_FOREVER;**

// **sattr –** для установки атрибутов безопасности

// по умолчанию следует установить значение **NULL**

// возвращает дескриптор почтового ящика, иначе

// значение **INVALID\_HANDLE\_VALUE**

Рисунок 5.3.1. Функция CreateMailslot

Параметр функции CreateMailslot, задающий имя почтового ящика, подразумевает, что это имя задано в локальном формате. На рисунках 5.3.2 – 5.3.4 указаны три возможных формата имени почтового ящика.

|  |
| --- |
| **\\.\mailslot\xxxxx**  где: **точка (.) -** обозначает локальный компьютер;  **mailslot -** фиксированное слово;  **xxxxx -** имя почтового ящика |

,...Рисунок 5.3.2. Локальный формат имени почтового ящика

Локальный формат имени почтового ящика используется при создании почтового ящика (ящик всегда создается на локальном для сервера компьютере), а также программой клиентом при открытии ящика, если предполагается использовать для записи все ящики с заданным именем на одном локальном компьютере.

Сетевой формат имени почтового ящика используется программой клиента, для записи сообщений в группу одноименных почтовых ящиков, которые находятся на компьютере, указанном в имени.

**\\servname\mailslot\xxxxx**

где: **servname -** имя компьютера-сервера почтового ящика;

**mailslot -** фиксированное слово;

**xxxxx -** имя почтового ящика

,...

Рисунок 5.3.3. Сетевой формат имени почтового ящика

**\\domain\mailslot\xxxxx**

где: **domain -** имя домена компьютеров или \*;

**mailslot -** фиксированное слово;

**xxxxx -** имя почтового ящика

,...

Рисунок 5.3.4. Доменный формат имени почтового ящика

Доменный формат имени почтового ящика используется программой клиента для записи сообщений в группу одноименных почтовых ящиков, которые находятся на всех компьютерах указанного домена. Если необходимо записать в сообщение в группу почтовых ящиков, которые находятся на компьютерах первичного домена, то вместо имени домена можно указать символ \*.

**Соединение клиентов с почтовым ящиком**

Для установки связи с почтовым ящиком программа клиента использует функцию CreateFile (рисунок 5.4.1).

Как уже отмечалось, функция CreateFile является универсальной и значения ее параметров практически ничем не отличаются от значений, применяющихся для связи клиента именованного канала с сервером именованного канала. В описании функции и приведенных примерах не рассматриваются никакие параметры, определяющие атрибуты безопасности. Использование атрибутов безопасности установленных по умолчанию, приводит к тому, что связь может быть установлена только между процессами, которые запущены от одного имени и с одним общим паролем. Для знакомства с возможностями интерфейса Mailslots, связанными с системой безопасности операционной системы Windows рекомендуется обратиться к источникам [4] или http://msdn2.miscrosoft.com

Следует обратить внимание на формат имени открываемого почтового ящика. Этот формат определяет пространство поиска почтовых ящиков, с которыми будет установлена связь.

// **-- открыть почтовый ящик**

// **Назначение:** функция предназначена для подключения клиента

// к почтовому ящику

**HANDLE CreateFile**

**(**

**LPCTSTR mname,** // [in] символическое имя почтового ящика

**DWORD accss,** // [in] чтение или запись

**DWORD share,** // [in] режим совместного использования

**LPSECURITY\_ATTRIBUTES sattr** // [in] атрибуты безопасности

**DWORD oflag,** // [in] флаг открытия почтового ящика

**DWORD aflag,** // [in] флаги и атрибуты

**HANDLE exten,** // [in] дополнительные атрибуты

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает дескриптор именованного канала, иначе

// **INVALID\_HANDLE\_VALUE** – неудачное завершение

// **Примечание:-** параметр **mname** указывается в локальном,

// сетевом или доменном формате: в зависимости от

// способа применения;

// - параметр **accss** должен принимать значение

// **GENERIC\_WRITE**

// - параметр **share** можетпринимать значения

// **FILE\_SHARE\_READ** (совместное чтение),

// **FILE\_SHARE\_WRITE** (совместная запись),

// **FILE\_SHARE\_READ| FILE\_SHARE\_WRITE** (чтение и запись);

// - параметр **sattr** для установки атрибутов безопасности

// по умолчанию, следует установить значение **NULL;**

// - значение параметра **oflag** всегдаустанавливаетсяв

// **OPEN\_EXISTING**(открытие существующего ящика);

// - значение параметра **aflag** можно установить в **NULL**,

// что определяет значения флагов и атрибутов по

// умолчанию или установить **FILE\_ATTRIBUTE\_NORMAL**;

// - значение параметра **exten** следует становить в **NULL**

Рисунок 5.4.1. Функция CreateFile

**Обмен данными через почтовый ящик**

Для записи данных в почтовый ящик используется функция WriteFile, а для чтения данных из почтового ящика функция ReadFile. Значения параметров, используемые в этих универсальных функциях при работе с почтовыми ящиками, практически ничем не отличаются от значений, применяемых при работе с именованными каналами. Разница заключается лишь в том, что в одном случае функции используют дескрипторы каналов, а другом - дескрипторы почтовых ящиков. Кроме того, следует помнить, что функцию ReadFile может выполнять только программа сервера, а WriteFle могут выполнять и сервер (сервер может записывать в свой собственный ящик) и клиент.

//..................................................................

**HANDLE hM;** // дескриптор почтового ящика

**DWORD rb;** // длина почитанного сообщения

**char rbuf[100];** // буфер ввода

**try**

**{**

**if ((hM = CreateMailslot("\\\\.\\mailslot\\myslot",**

**NULL,**

**MAILSLOT\_WAIT\_FOREVER,** // ждать вечно

**NULL)) == INVALID\_HANDLE\_VALUE)**

**throw "CreateMailslotError";**

**if(!ReadFile(hM,**

**rbuf,** // буфер

**sizeof(rbuf),** // размер буфера

**&rb,** // прочитано

**NULL))**

**throw "ReadFileError";**

//..................................................................

**}**

//..................................................................

Рисунок 5.5.1. Создание почтового ящика

//..................................................................

**HANDLE hM;** // дескриптор почтового ящика

**DWORD wb;** // длина записанного сообщения

**char wbuf[] = "Hello Mailslot”;** //буфер вывода

**try**

**{**

**if ((hM = CreateFile("\\\\isit301\\mailslot\\myslot",**

**GENERIC\_WRITE,** // будем писать в ящик

**FILE\_SHARE\_READ,** // разрешаем одновременно читать

**NULL,**

**OPEN\_EXISTING,** // ящик уже есть

**NULL, NULL)) == INVALID\_HANDLE\_VALUE)**

**throw "CreateFileError";**

**if(!WriteFile(hM,**

**wbuf,** // буфер

**sizeof(wbuf),** // размер буфера

**&wb,** // записано

**NULL))**

**throw "ReadFileError";**

//..................................................................

**}**

//..................................................................

Рисунок 5.5.2. Соединение клиента с почтовым ящиком

На рисунках 5.5.1 и 5.5.2 представлены фрагменты программ сервера и клиента. В программе сервера создается почтовый ящик и читается сообщение из него. В программе клиента осуществляется подключение к почтовому ящику и записывается в него сообщение. Следует обратить внимание, что программы клиента и сервера находятся на разных компьютерах, т.к. символическое имя почтового ящика в функции CreateFile указано в сетевом формате.

**Получение информации о почтовом ящике**

Получить информацию о характеристиках почтового ящика можно с помощью функции GetMailslotInfo (рисунок 5.6.1).

// **-- получить информацию о почтовом ящике**

// **Назначение:** функция предназначена для получения

// характеристик созданного почтового ящика

**BOOL GetMailslotInfo**

**(**

**HANDLE hM,** // [in] дескриптор почтового ящика

**LPDWORD ml,** // [out] максимальная длина сообщения

**LPDWORD nl,** // [out] длина следующего сообщения

**LPDWORD nm** // [out] количество сообщений

**LPDWORD to** // [out] интервал ожидания сообщения

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

Рисунок 5.6.1. Функция GetMailslotInfo

Функция GetMailslotInfo может быть использована только на стороне сервера почтового ящика и параметр hM должен быть получен в результате выполнения функции CreateMailslot. Чаще всего функция применяется для выяснения количества непрочитанных сообщений накопившихся в почтовом ящике.

**Изменение интервала ожидания сообщения**

Время ожидания функцией ReadFile поступления сообщения в почтовый ящик первоначально устанавливается при создании почтового ящика с помощью функции CreateMailslot. В процессе работы, может оказаться необходимым изменить значение этого интервала или вообще сделать его нулевым. Для этого применяется функция SetMailslotInfo (рисунок 5.6.1). Функция может быть выполнена только в программе сервера и использует в качестве аргумента дескриптор почтового сервера, который был получен при выполнении функции CreateMailslot.

// **-- изменить время ожидания сообщения**

// **Назначение:** функция предназначена для изменения

// одной характеристики почтового ящика –

// интервала времени ожидания

**BOOL SetMailslotInfo**

**(**

**HANDLE hM,** // [in] дескриптор почтового ящика

**PDWORD to** // [in] новое значение интервала

**);**

// **Код возврата:** в случае успешного завершения функция

// возвращает **TRUE**, иначе **FALSE**

Рисунок 5.7.1. Функция SetMailslotInfo

**Итоги главы**

1. Механизм Mailslots (почтовый ящик) является одним из IPC-механизмов операционной системы Windows, позволяющий создавать распределенные приложения архитектуры клиент-сервер в локальной сети TCP/IP.

2. Почтовый ящик представляет собой объект операционной системы, предоставляющий возможность пересылать данные в одном направлении: от клиента к серверу.

3. Почтовый ящик идентифицируется своим именем. Сервером называется процесс создающий почтовый ящик. Клиентом – процесс, который подключается к почтовому ящику и записывает в него данные.

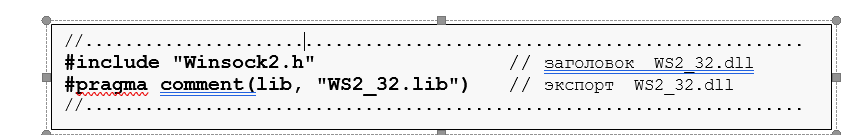
4. Обмен данными осуществляется сообщениями и может происходить в синхронном и асинхронном режимах. Если клиент и сервер находятся на разных компьютерах, доставка сообщений не гарантируется.

5. Допускается создание нескольких ящиков с одним и тем же именем. Если пересылаемые сообщения не превышают 425 байт, то возможна передача данных одновременно нескольким почтовым ящикам.

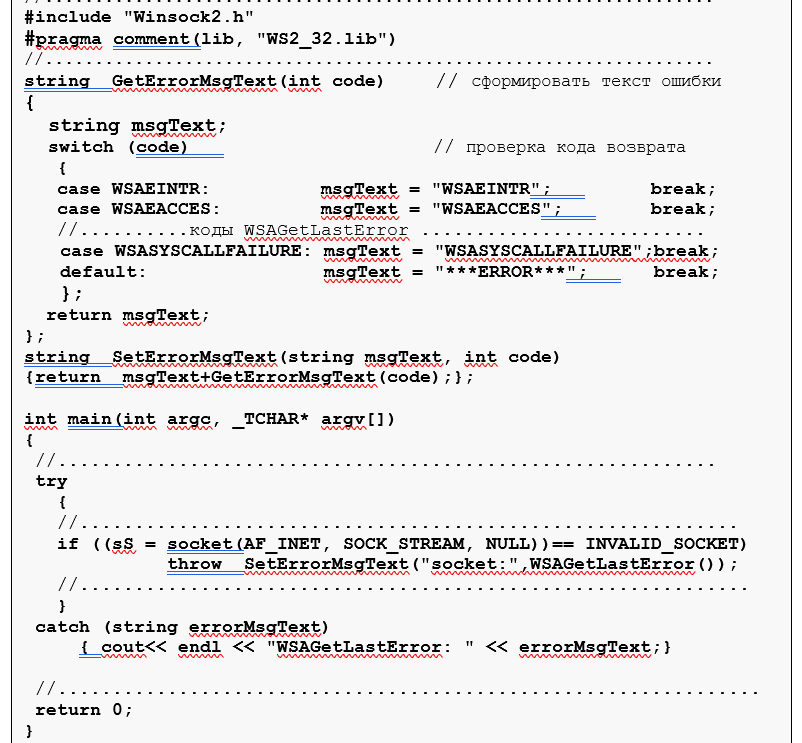
6. В состав Mailslots API входят функции создания почтового ящика, подсоединения клиента к почтовому ящику, функции записи и чтения сообщений, а также функции для получения и установки характеристик почтового ящика.

# Структура программы TCP-сервера.

Для использования интерфейса Winsock2 в исходный текст программы следует включить следующую последовательность директив компилятора C++.

Динамическая библиотека WS2\_32.DLL (которая содержит все функции Winsock2), входит в стандартную поставку Windows, а библиотека экспорта WS2\_32.LIB и заголовочный файл Winsock2.h в стандартную поставку Visual C++. 

Обработка ошибок: Диагностирующий код может быть получен с помощью функции **WSAGetLastError**. Функция WSAGetLastError вызывается, непосредственно сразу после функции Winsock2, завершившейся с ошибкой

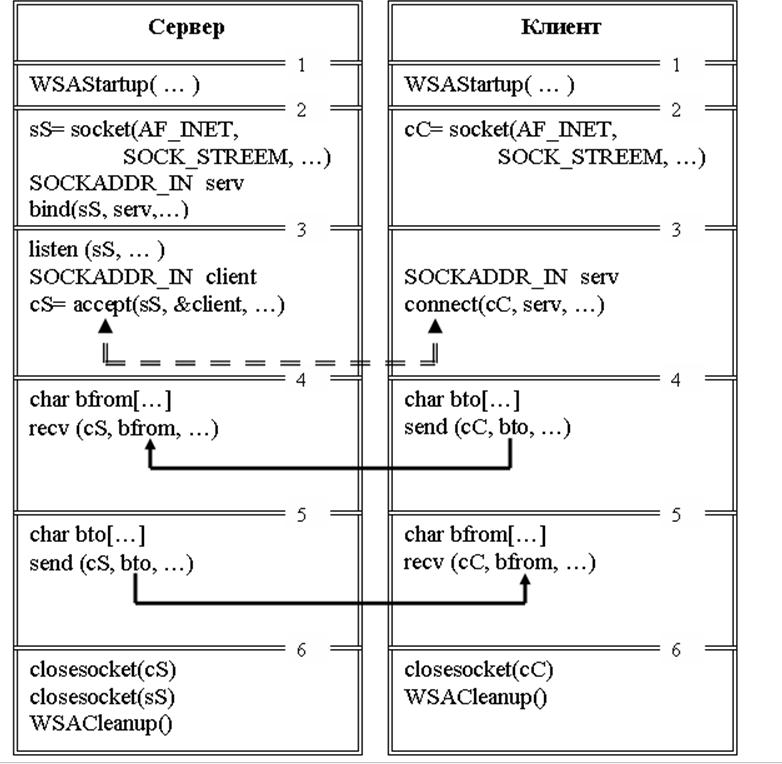


В приведенном примере для обработки ошибок используется функция **SetErrorMsgText**, которая в качестве параметра получает префикс формируемого сообщения об ошибке, код функции WSAGetLastError, а возвращает текст сообщения (используя функцию GetErrorMsgText)

**Схема взаимодействия процессов в распределенном приложении**

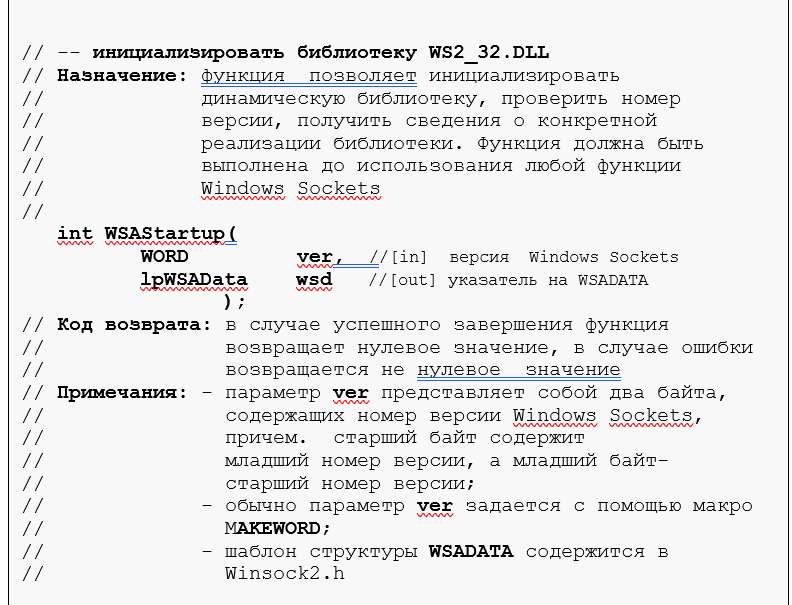
Между сокетами устанавливается TCP-соединение и весь обмен данных осуществляется в рамках этого соединения. Передача по каналу является надежной и данные поступают в порядке их отправления

В распределенных приложениях архитектуры клиент-сервер, клиенту и серверу отводится разная роль: инициатором обмена является клиент, а сервер ждет запросы клиента и обслуживает их. Таким образом, предполагается, что к моменту выдачи запроса клиентом, сервер должен быть уже активным, а клиент должен “знать” параметры сокета сервера.

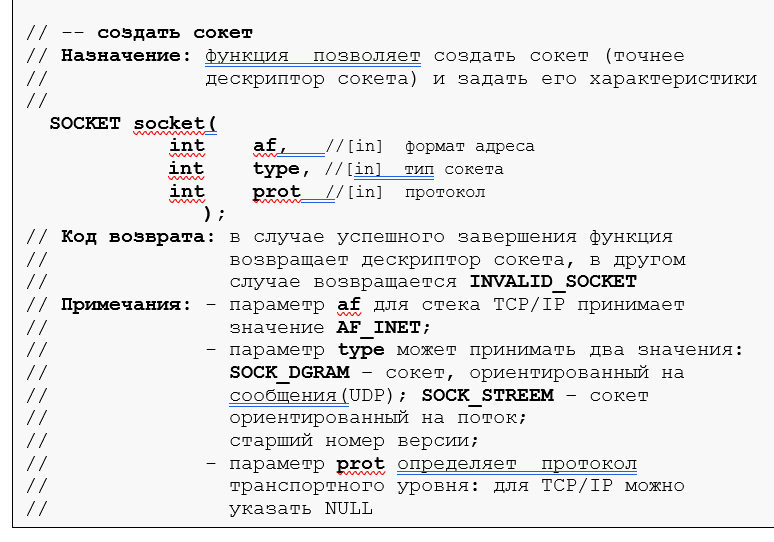


Сплошными направленными линиями обозначается движение данных по сети TCP/IP, прерывистой – синхронизация (ожидание) процессов.

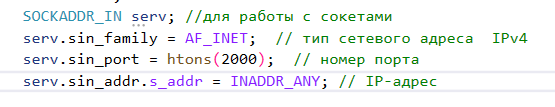
Первые блоки обеих программ идентичны и предназначены для инициализации библиотеки WS2\_32.DLL.



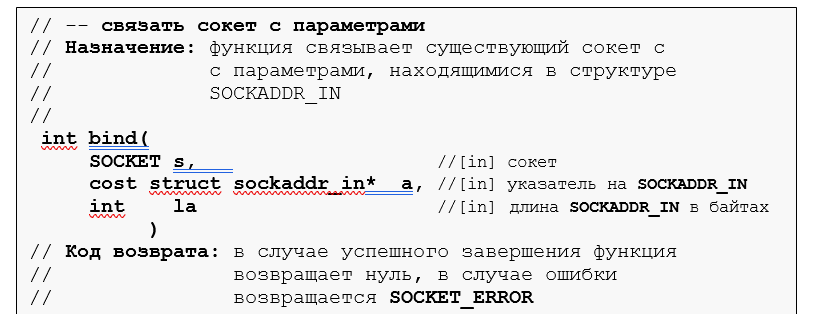
**Второй блок программы** сервера создает сокет (функция soсket) и устанавливает параметры этого сокета. Следует обратить внимание на параметр SOCK\_STREEM параметра функции socket, указывающий, что сокет будет использоваться для соединения (сокет ориентированный на поток). Для установки параметров сокета, используется функция bind. При этом говорят, что сокет ***связывают*** с параметрами. Для хранения параметров сокета в Winsock2 предусмотрена специальная структура SOCKADDR\_IN (она тоже присутствует на рисунке). Перед выполнением функции bind, которая использует эту структуру в качестве параметра, необходимо ее заполнить данными. Пока скажем только, что в SOCKADDR\_IN хранится IP-адрес и номер порта сервера.



Инициализация Sockaddr

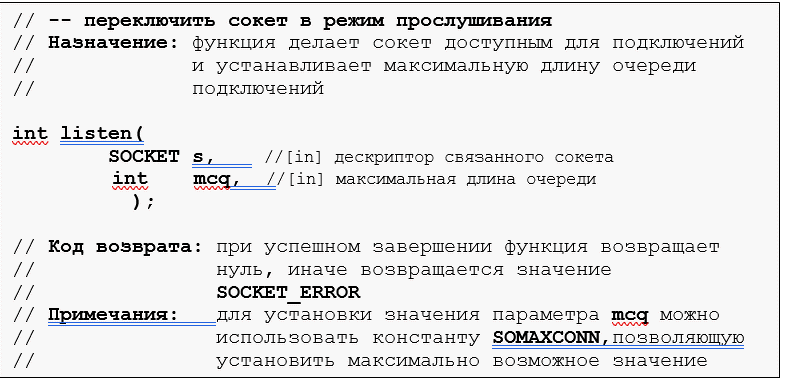


Для преобразования номера порта в формат TCP/IP следует использовать функцию htons

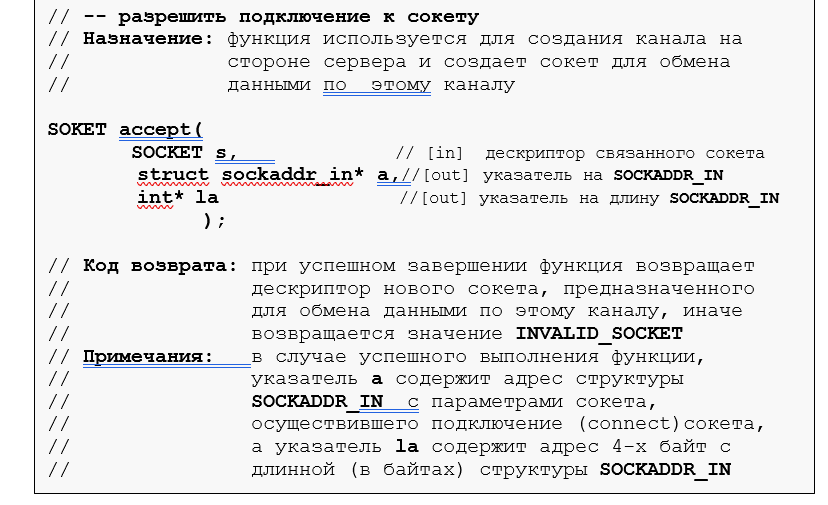


В третьем блоке программы сервера выполняются две функции Winsock2**: listen и accept**.

**Функция listen** переводит сокет, ориентированный на поток, в состояния прослушивания (открывает доступ к сокету) и задает некоторые параметры очереди соединений.

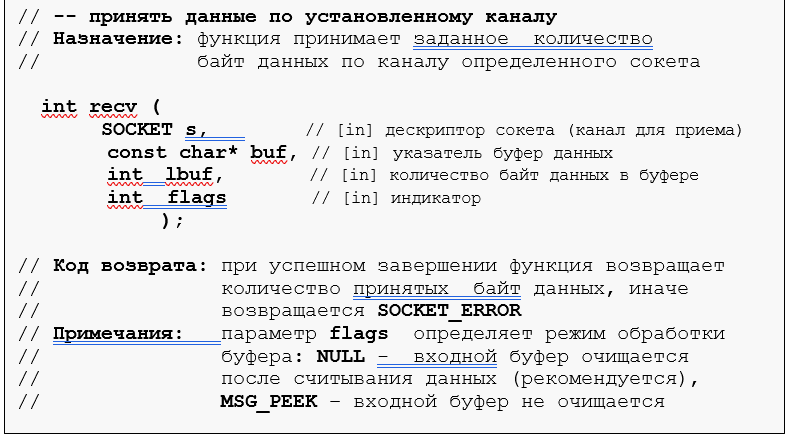


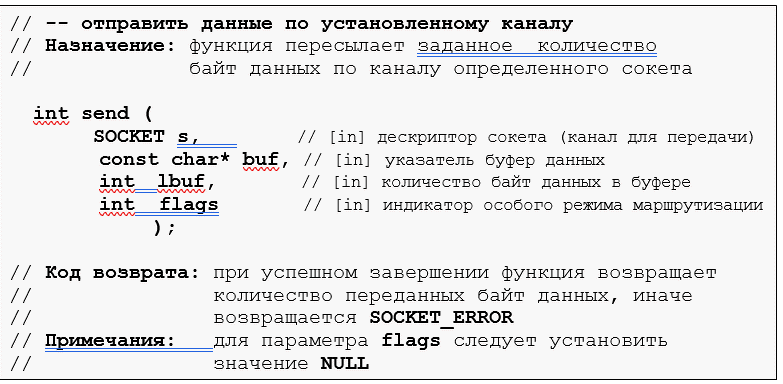
**Функция accep**t переводит процесс сервера в состояние ожидания, до момента пока программа клиента не выполнит функцию connect (подключится к сокету). Если на стороне клиента корректно выполнена функция connect, то функция accept **возвращает новый сокет** (с эфемерным портом), который предназначен для обмена данными с подключившимся клиентом. Кроме того, автоматически заполняется структура SOCKADDR\_IN параметрами сокета клиента.



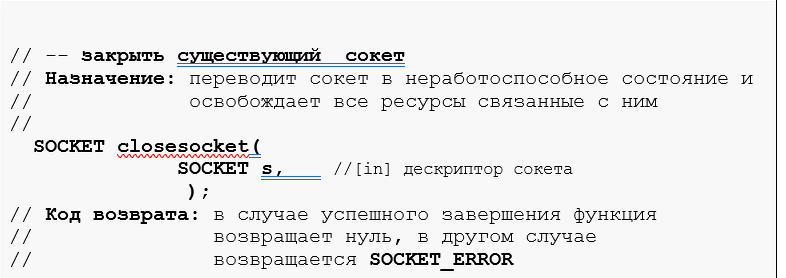
Четвертый и пятый блоки программы сервера предназначены для обмена данными по созданному соединению. Следует обратить внимание, что, во-первых, используются функции send и recv, а во-вторых, в качестве параметра эти функции используют сокет, созданный командой accept.

Работа функций send и recv является синхронной, т.е. до тех пор, пока не будет выполнена пересылка или прием данных выполнение программы приостанавливается

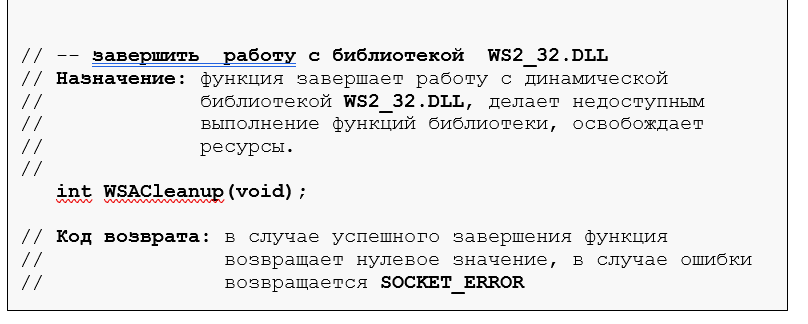




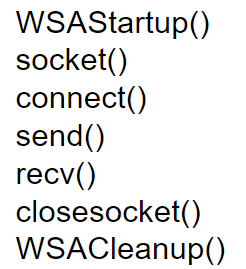
Закрытие сокета



Завершение работы с библиотекой



# Структура программы TCP-клиента.



Канал связи (или соединение) создается между двумя сокетами, ориентированными на поток. На стороне сервера это должен быть связанный (функция bind) и переключенный в режим прослушивания (функция listen) сокет. На стороне клиента должен быть создан дескриптор ориентированного на поток сокета (функция socket).

Канал связи создается в результате взаимодействия функций accept (на стороне сервера) и connect (на стороне клиента). Алгоритм взаимодействия этих функций зависит от установленного режима ввода-вывода для участвующих в создании канала сокетов.

Функция accept (описание на рисунке 3.10.1) приостанавливает выполнение программы сервера до момента срабатывания в программе клиента функции connect (описание на рисунке 3.10.2). В результате работы функции accept создается новый сокет, предназначенный для обмена данными с клиентом.

// **-- разрешить подключение к сокету**

// **Назначение:** функция используется для создания канала на

//стороне сервера и создает сокет для обмена

//данными по этому каналу

**SOKET accept(**

**SOCKET s,** // [in] дескриптор связанного сокета

**struct sockaddr\_in\* a,**//[out] указатель на **SOCKADDR\_IN**

**int\* la** //[out] указатель на длину **SOCKADDR\_IN**

**);**

// **Код возврата:** при успешном завершении функция возвращает

// дескриптор нового сокета, предназначенного

// для обмена данными по этому каналу, иначе

// возвращается значение **INVALID\_SOCKET**

// **Примечания**: в случае успешного выполнения функции,

// указатель **a** содержит адрес структуры

// **SOCKADDR\_IN** с параметрами сокета,

// осуществившего подключение (connect)сокета,

// a указатель **la** содержит адрес 4-х байт с

// длинной (в байтах) структуры **SOCKADDR\_IN**

Рисунок 3.10.1 Функция accept

// **-- установить соединение с сокетом**

// **Назначение:** функция используется клиентом для создания

// канала с определенным сокетом сервера

**int connect (**

**SOCKET s,** // [in] дескриптор связанного сокета

**struct sockaddr\_in\* a,** // [in] указатель на **SOCKADDR\_IN**

**int la** // [in] длина **SOCKADDR\_IN** в байтах

**);**

// **Код возврата:** при успешном завершении функция возвращает

// нуль, иначе возвращается значение

// **SOCKET\_ERROR**

// **Примечания**: - параметр **a** является указателем на структуру

// **SOCKADDR\_IN;** структура должна быть

// инициализирована параметрами серверного

// сокета (тип адреса, IP-адрес, порт);

// - параметр **la**, должен содержать длину

// (в байтах) структуры **SOCKADDR\_IN**

Рисунок 3.10.3 Функция connect

На стороне клиента создание канала осуществляется с помощью функции connect. Для того, чтобы выполнить функцию connect, достаточно просто предварительно создать сокет (функция socket), ориентированный на поток. Функция connect указывает модулю TCP сокет клиента, который будет использоваться для соединения с сокетом сервера (его параметры указываются через параметры connect). При этом предполагается, что серверный сокет создан (функции socket и bind) и для него уже выполнена функция listen. На рисунке 3.10.4 приведен фрагмент текста программы клиента, в котором используется функция connect.

//................................................................

**try**

**{**

//....**WSAStartup(...)**...........................................

**SOCKET cC;** // серверный сокет

**if ((cC = socket(AF\_INET, SOCK\_STREAM, NULL))== INVALID\_SOCKET)**

**throw SetErrorMsgText("socket:",WSAGetLastError());**

**SOCKADDR\_IN serv;** // параметры сокета сервера

**serv.sin\_family = AF\_INET;** // используется IP-адресация

**serv.sin\_port = htons(2000);** // TCP-порт 2000

**serv.sin\_addr.s\_addr = inet\_addr("80.1.1.7");** // адрес сервера

**if ((connect(cC,(sockaddr\*)&serv, sizeof(serv))) == SOCKET\_ERROR)**

**throw SetErrorMsgText("connect:",WSAGetLastError());**

//..............................................................

**}**

**catch (string errorMsgText)**

**{ cout << endl << errorMsgText;}**

//..............................................................

Обмен данными по каналу связи осуществляется между двумя сокетами и возможен сразу после того, как этот канал создан (выполнена функция accept на стороне сервера и функция connect на стороне клиента). Для пересылки данных по каналу Winsock2 предоставляет функции send и recv (рисунки 3.11.1 и 3.11.2). Функция send пересылает по каналу, указанного сокета, определенное количество байт данных. Функция recv принимает по каналу, указанного сокета, определенное количество байт данных. Для работы обеих функций в программе необходимо выделить память (буфер) для приема или оправления данных. Размер буфера для приема данных и для отправления данных указывается в параметрах функций. Реальное количество пересланных или принятых байт данных возвращается функциями send и recv в виде кода возврата.

Следует иметь в виду, что не всегда количество переданных или оправленных байт совпадает с размерами выходного или входного буферов. Более того, разрешается выполнять пересылку с нулевым количеством байт. Обычно такую пересылку используют для обозначения конца передачи. Для принимающей стороны операционная система буферизирует принимаемые данные. Если при очередном приеме данных размеры буфера будут исчерпаны, отправляющей стороне будет выдано соответствующий код ошибки.

// **-- отправить данные по установленному каналу**

// **Назначение:** функция пересылает заданное количество

// байт данных по каналу определенного сокета

**int send (**

**SOCKET s,** // [in] дескриптор сокета (канал для передачи)

**const char\* buf,** // [in] указатель буфер данных

**int lbuf,** // [in] количество байт данных в буфере

**int flags** // [in] индикатор особого режима маршрутизации

**);**

// **Код возврата:** при успешном завершении функция возвращает

// количество переданных байт данных, иначе

// возвращается **SOCKET\_ERROR**

// **Примечания**: для параметра **flags** следует установить

// значение **NULL**

Рисунок 3.11.1 Функция send

Работа функций send и recv является синхронной, т.е. до тех пор, пока не будет выполнена пересылка или прием данных выполнение программы приостанавливается. Поэтому, если одной из сторон распределенного будет выдана функция recv для которой нет данных для приема и при этом соединение не разорвано, то это приведет к зависанию программы на некоторое время (называемоe time-out) и завершению функции recv c соответствующим кодом ошибки.

// **-- принять данные по установленному каналу**

// **Назначение:** функция принимает заданное количество

// байт данных по каналу определенного сокета

**int recv (**

**SOCKET s,** // [in] дескриптор сокета (канал для приема)

**const char\* buf,** // [in] указатель буфер данных

**int lbuf,** // [in] количество байт данных в буфере

**int flags** // [in] индикатор

**);**

// **Код возврата:** при успешном завершении функция возвращает

// количество принятых байт данных, иначе

// возвращается **SOCKET\_ERROR**

// **Примечания**: параметр **flags** определяет режим обработки

// буфера: **NULL** - входной буфер очищается

// после считывания данных (рекомендуется),

// **MSG\_PEEK** – входной буфер не очищается

Рисунок 3.11.2 Функция recv

# Структура программы UDP-сервера.

Существование двух различных протоколов на транспортном уровне TCP/IP, определяет две схемы взаимодействия процессов распределенного приложения: схема, ориентированная на сообщения, и схема, ориентированная на поток.

Принципиальное различие этих схем, заключается в следующем.

В первом между сокетами курсируют UDP-пакеты, и поэтому вся работа, связанная с обеспечением надежности и установкой правильной последовательности передаваемых пакетов возлагается на само приложение. В общем случае, получатель узнает адрес отравителя вместе с пакетом данных.

Во втором случае между сокетами устанавливается TCP-соединение и весь обмен данных осуществляется в рамках этого соединения. Передача по каналу является надежной и данные поступают в порядке их отправления.

В распределенных приложениях архитектуры клиент-сервер, клиенту и серверу отводится разная роль: инициатором обмена является клиент, а сервер ждет запросы клиента и обслуживает их. Таким образом, предполагается, что к моменту выдачи запроса клиентом, сервер должен быть уже активным, а клиент должен “знать” параметры сокета сервера. На рисунках 3.4.1 и 3.4.2. изображены схемы взаимодействия клиента и сервера, для первого и второго случаев. **Второй случай в следующем вопросе!**



Рисунок 3.4.1. Схема взаимодействия процессов без установки соединения

На рисунке 3.4.1 схематично изображены две программы, реализующие два процесса распределенного приложения. Рассматриваемое приложение имеет архитектуру клиент-сервер (на рисунке сделаны соответствующие обозначения). Обе программы разбиты на пять блоков, а стрелками обозначается движение информации по сети TCP/IP.

Первые блоки обеих программ одинаковые и предназначены для инициализации библиотеки WS2\_32.DLL.

Второй блок программы сервера создает сокет (функция soсket) и устанавливает параметры этого сокета. Следует обратить внимание на параметр SOCK\_DGRAM функции socket, указывающий на тип сокета (в данном случае – сокет, ориентированный на сообщения). Для установки параметров сокета, используется функция bind. При этом говорят, что сокет ***связывают*** с параметрами. Для хранения параметров сокета в Winsock2 предусмотрена специальная структура SOCKADDR\_IN (она тоже присутствует на рисунке). Перед выполнением функции bind, которая использует эту структуру в качестве параметра, необходимо ее заполнить данными. Пока скажем только, что в SOCKADDR\_IN хранится IP-адрес и номер порта сервера.

В третьем блоке программы сервера выполняется функция recvfrom, которая переводит программу сервера в состояние ожидания, до поступления сообщения от программы клиента (функция sendto). Функция recvfrom тоже использует структуру SOCKADDR\_IN – в нее автоматически помещаются параметры сокета клиента, после приема от него сообщения. Данные поступают в буфер, который обеспечивает принимающая сторона (на рисунке символьный массив bfrom). Следует отметить, что в качестве параметра функции recvfrom используется связанный сокет – именно через него осуществляется передача данных.

Четвертый блок программы сервера предназначен для пересылки данных клиенту. Пересылка данных осуществляется с помощью функции sendto. В качестве параметров sendto использует структуру SOCKADDR\_IN с параметрами сокета принимающей стороны (в данном случае клиента) и заполненный буфер с данными.

Пятые блоки программ сервера и клиента одинаковые и предназначены для закрытия сокета и завершения работы с библиотекой WS2\_32.DLL.

Всем блокам программы клиента, кроме второго, есть аналог в программе сервера. Второй блок, в сравнении с сервером, не использует команду bind. Здесь проявляется основное отличие между сервером и клиентом. Если сервер, должен использовать однозначно определенные параметры (IP-адрес и номер порта), то для клиента это не обязательно – ему Windows выделяет эфемерный порт. Т.к. инициатором связи является клиент, то он должен точно “знать” параметры сокета сервера, а свои параметры клиент получит от Windows и сообщит их вместе с переданным пакетом серверу.

**Обмен данными без соединения**

Если для передачи данных на транспортном уровне используется протокол UDP, то говорят, об обмене данными без соединения или об обмене данными с помощью сообщений. Для отправки и приема сообщений в Winsock2 используются функции sendto и recvfrom (рисунки 3.12.1 и 3.12.2). При этом предполагается, что сообщения будут курсировать между сокетами ориентированными на сообщения.

Особенностью использования этих функций заключается в том, что протоколом UDP не гарантируется доставка и правильная последовательность приема отправленных сообщений. Весь контроль надежности доставки и правильной последовательностью поступления сообщений возлагается на разработчика приложения. В связи с этим, обмен данными с помощью сообщений используется, в основном, для широковещательных сообщений или для пересылки коротких сообщений, последовательность получения которых не имеет значения.

// **-- отправить сообщение**

// **Назначение:** функция предназначена для отправки сообщения

// без установления соединения

//

**int sendto(**

**SOCKET s,** // [in] дескриптор сокета

**const char\* buf,** // [in] буфер для пересылаемых данных

**int len,** // [in] размер буфера **buf**

**int flags,** // [in] индикатор режима маршрутизации

**const struct sockaddr\* to,** // [in] указатель на **SOCKADDR\_IN**

**int tolen** // [in] длина структуры to

**);**

// **Код возврата:** при успешном завершении функция возвращает

// количество пересланных байт данных, иначе

// возвращается **SOCKET\_ERROR**

// **Примечания:** - функция может применяться только для

// сокетов, ориентированных на сообщения

// (**SOCK\_DGRAM**))

// - параметр **to** указывает на структуру

// **SOCKADDR\_IN** с параметрами сокета получателя;

// - для параметра **flags** рекомендуется установить

// значение **NULL**

Рисунок 3.12.1. Функция sendto

Также как и функции send и recv, функции sendto и recvfrom работают в синхронном режиме, т.е. вызвав, например, функцию recvfrom вызывающая программа не получит управления до момента завершения приема данных.

Следует также обратить внимание, что обе функции используют в качестве параметров структуру SOCKADDR\_IN. В случае выполнения функции send, структура должна содержать параметры сокета получателя. У функция recv структура SOCKADDR\_IN, наоборот, предназначена для получения параметров сокета отправителя.

Часто протокол UDP (и соответственно функции sendto и recvfrom) используется для пересылки сообщений предназначенных для рассылки одного сообщения всем компьютерам сети (широковещательные сообщения). Для этого в параметрах сокета отправляющей стороны используются специальные широковещательные и групповые IP-адреса. Использование и групповых адресов функцией sendto по умолчанию запрещено. Разрешение на использование широковещательных устанавливается функцией setsockopt.

На рисунках 3.12.3 и 3.12.4 приведены примеры применения функций recvfrom и sendto, которые используются в программах сервера и клиента соответственно.

// **-- принять сообщение**

// **Назначение:** функция предназначена для получения сообщения

// без установления соединения

**int recvfrom(**

**SOCKET s,** // [in] дескриптор сокета

**char\* buf,** // [out] буфер для получаемых данных

**int len,** // [in] размер буфера **buf**

**int flags,** // [in] индикатор режима маршрутизации

**struct sockaddr\* to,** // [out] указатель на **SOCKADDR\_IN**

**int\* tolen** //[out] указатель на размер to

**);**

// **Код возврата:** при успешном завершении функция возвращает

// количество принятых байт данных, иначе

// возвращается **SOCKET\_ERROR**

// **Примечания:** - функция может применяться только для

// сокетов, ориентированных на сообщения

// (**SOCK\_DGRAM**));

// - параметр **to** указывает на структуру

// **SOCKADDR\_IN** с параметрами сокета отправителя;

// - параметр **tolen** содержит адрес четырех байт, с

// размером структуры **SOCADDR\_IN**

// - для параметра **flags** рекомендуется установить

// значение **NULL**

Рисунок 3.12.2 Функция recvfrom

//........................................................................

**try**

**{** //...**WSAStartup(...),sS = socket(...,SOCKET\_DGRAM,...)**

**SOCKADDR\_IN serv**; // параметры сокета sS

**serv.sin\_family = AF\_INET;** // используется IP-адресация

**serv.sin\_port = htons(2000);** // порт 2000

**serv.sin\_addr.s\_addr = INADDR\_ANY;** // адрес сервера

**if(bind(sS,(LPSOCKADDR)&serv, sizeof(serv))== SOCKET\_ERROR)**

**throw SetErrorMsgText("bind:", WSAGetLastError());**

**SOCKADDR\_IN clnt;** // параметры сокета клиента

**memset(&clnt,0,sizeof(clnt));** // обнулить память

**int lc = sizeof(clnt);**

**char ibuf[50];** //буфер ввода

**int lb = 0;** //количество принятых байт

**if (lb = recvfrom(sS, ibuf, sizeof(ibuf), NULL,**

**(sockaddr\*)&clnt, &lc)) == SOCKET\_ERROR)**

**throw SetErrorMsgText("recv:",WSAGetLastError());**

//................................................................

**}**

**catch (string errorMsgText)**

**{cout << endl << errorMsgText;}**

//........................................................................

Рисунок 3.12.3 Пример использования функции recvfrom в программе сервера

//........................................................................

**try**

**{**//...**WSAStartup(...),сС = socket(...,SOCKET\_DGRAM,...)**

**SOCKADDR\_IN serv;** // параметры сокета сервера

**serv.sin\_family = AF\_INET;** // используется ip-адресация

**serv.sin\_port = htons(2000);** // порт 2000

**serv.sin\_addr.s\_addr = inet\_addr("127.0.0.1");** // адрес сервера

**char obuf[50]= "client: I here";** //буфер вывода

**int lobuf = 0;** //количество отправленных

**if ((lobuf = sendto(cC,obuf,strlen(obuf)+1,NULL,**

**(sockaddr\*)&serv, sizeof(serv))) == SOCKET\_ERROR)**

**throw SetErrorMsgText("recv:",WSAGetLastError());**

//............................................................

**}**

**catch (string errorMsgText)**

**{cout << endl << errorMsgText;}**

//........................................................................

Рисунок 3.12.3 Пример использования функции sendto в программе клиента

# Структура программы UDP-клиента.

== вопросу 9

# Структура параллельного сервера.

Распределенное приложение, предполагает наличие одной программы сервера, которая одновременно обслуживает несколько программ клиентов. Управление сервером осуществляется с помощью специальной программы, которую будем называть консолью управления.

Серверы, одновременно обслуживающие несколько клиентов, по методу облуживания подразделяются на ***итеративные*** и ***параллельные***  серверы ***(iterative and concurrent servers)***.

**Работа итеративного сервера** описывается циклом из четырех шагов: 1) ожидание запроса от клиента; 2) обработка запроса; 3) отправка результата запроса; 4) возврат в ждущее состояние 1. Очевидно, что сервер этого класса может применяться в том случае, если предполагаются короткие запросы от клиентов, не требующие больших затрат на обработку и длинных ответов сервера. Как правило, подобные серверы работают над UDP, когда нет необходимости создавать отдельный канал связи для каждого клиента. Консоль управления в этом случае может быть выполнена в виде специального клиента, запросы которого и есть команды управления сервером.

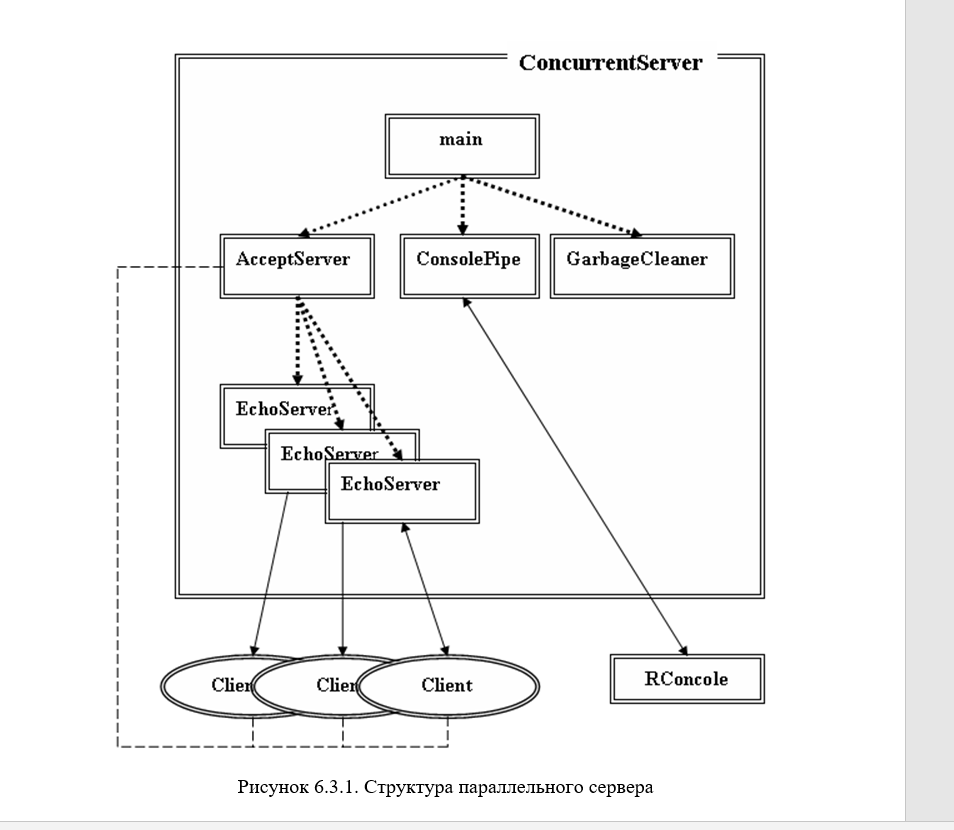
**Параллельные серверы** имеют другой цикл работы: 1) ожидание запроса от клиента; 2) запуск нового сервера для обработки текущего запроса; 3) возврат в ждущее состояние 1. Преимущество параллельных серверов заключается в том, что он лишь порождает новые серверы, которые и занимаются обработкой запросов клиентов. Очевидно, что для создания параллельных серверов необходимым условием является мультизадачность операционной среды сервера. По всей видимости, параллельные серверы целесообразно использовать, если предполагается наличие относительно длительного сеанса связи между клиентом и сервером. Как правило, параллельные серверы работают над TCP. Консоль управления может быть создана, как отдельный процесс или поток (в зависимости от возможностей операционной системы) в рамках сервера или так же, как предлагалось для итеративного сервера, выполнить в виде специализированного клиента.

**Сложности в разработке**: Критическим по времени для параллельного сервера является момент подключения клиента. Сервер не должен тратить много времени на подключение клиента

С другой стороны, может потребоваться управлять процессом подключения.

После подключения клиента к параллельному серверу запускается отдельный процесс, обслуживающий запрос клиента. Для управления обслуживающими процессами и для сбора статистики требуется динамическая структура данных (позволяющая добавлять и удалять элементы структуры), предназначенная для хранения информации о работающих в настоящий момент обслуживающих процессах. Как правило, для хранения подобной информации используют связный список.

Отдельные процессы, работающие в рамках параллельного сервера, могут использовать общие ресурсы. Некоторые ресурсы, могут быть разрушены при совместном использовании несколькими параллельными процессами (например, связный список). Для разрешения этой проблемы следует использовать механизмы синхронизации, позволяющие последовательно использовать такие критические ресурсы.



структура параллельного сервера, назначением которого является, одновременное обслуживание нескольких клиентских программ. Обслуживание заключается в получении от клиента по установленному TCP-соединению последовательности символов и в возврате (пересылке) этой последовательности обратно. Кроме того, предполагается, что сервер может выполнять команды, введенные с консоли управления, с которой поддерживается связь через именованный канал (Named Pipe).

Все процессы, работающие в рамках сервера, изображены на рисунке прямоугольниками. Пунктирными направленными линиями обозначается создание (запуск) одного процесса другим.

Процесс c именем **main**, является главным процессом сервера, который получает управление от операционной системы. Этот процесс создает и запускает новые процессы **AcceptServer, ConsolePipe и GarbageCleaner. Процесс AcceptServer**, в свою очередь создает несколько процессов с именем EchoServer.

Сплошными двунаправленными линиями обозначается перемещение данных. Данные перемещаются между EchoServer-процессами сервера и клиентскими программами (с именем Client), обозначенными на рисунке овалами, а также между программой с именем RConsole, реализующей клиентскую часть консоли управления, и процессом ConsolePipe, который реализует серверную часть консоли управления. Штриховой линией, соединяющей изображение клиентских программ и изображение процесса AcceptServer, обозначается процедура создания соединения между клиентом и сервером.

Опишем назначение компонентов изображенного на рисунке 6.3.1 распределенного приложения.

**Процесс main**. Основное назначение – запуск, инициализация и завершение работы сервера. Как уже отмечалось, именно этот процесс первым получает управление от операционной системы. Процесс main запускает основные процессы: AcceptServer, ConsolePipe и RConsole.

**Процесс AcceptServer**. AcceptServer создается процессом main и предназначен для выполнения процедуры подключения клиентов к серверу, для исполнения команд консоли управления, а также для запуска процессов EchoServer, обслуживающих запросы клиентских программ по созданным соединениям. Кроме того, AcceptServer создает список подключений, который далее будем называть ***ListContanct***. При подключении очередного клиента, процесс AcceptServer добавляет в ListContact элемент, предназначенный для хранения информации о состоянии данного подключения.

**Процесс ConsolePipe**.ConsolePipe создается процессом main и является сервером именованного канала, по которому осуществляется связь между программой RConsole (консоль управления сервером) и параллельным сервером.

**Процесс** **GarbageCleaner**. Основным назначением процесса GarbageCleaner является удаление элемента списка подключений ListContact, после отключения программы клиента. Следует сразу отметить, что ListContact является ресурсом, требующим последовательного использование. Одновременная запись и (или) удаление элементов списка может привести к разрушению списка ListContact.

**Процесс EchoServer**. Процессы EchoServer создаются процессом AcceptServer по одному для каждого успешного подключения программы клиента. Основным назначением процесса EchoServer является прием данных по созданному процессом AcceptServer подключению и отправка этих же данных без изменения обратно программе клиента. Условием окончания работы сервера является получение от клиента пустого сегмента данных (имеющего нулевую длину).

**Программа Client**. Программа Client предназначена для пересылки данных серверу и получения ответа от сервера. Программа может работать, как на одном компьютере с сервером (будет использоваться интерфейс внутренней петли), так и на другом компьютере, соединенным с компьютером сервера сетью TCP/IP. Для окончания работы с сервером программа формирует и отправляет сегмент данных нулевой длины.

**Программа RConsole**. Программа RConsole предназначена для ввода команд управления сервером и для вывода диагностических сообщений полученных от сервера. RConsole является клиентом именованного канала.

**Список подключений ListContact**. Список ListContact (не изображен на рисунке) создается на основе стандартного класса list и предназначен для хранения информации о каждом подключении. Список создается пустым при инициализации процесса AcceptServer. В рамках этого же процесса осуществляется добавление элементов списка, по одному для каждого подключения. При отключении программы клиента от сервера, соответствующий элемент списка помечается, как неиспользуемый. Удаление неиспользуемого элемента осуществляется процессом GarbageCleaner, который работает в фоновом режиме.

# Структура параллельного сервера. Accept Server. GarbageCleaner.

==11

# Широковещание. Обнаружение сервера с помощью широковещания.

До сих пор при разработке распределенного приложения предполагались известными сетевой адрес компьютера, на котором находится программа-сервер и номер порта, прослушиваемый этой программой. В реальности распределенное приложение не должно быть привязано к конкретным параметрам сокетов, т.к. это делает ограниченным его применение.

Чтобы сделать приложение более гибким, можно хранить номер порта в конфигурационных файлах, которые сервер считывает при запуске. Иногда номер порта передаётся в командной строке. Так, например, большинство серверов баз данных в качестве одного из параметров инициализации используют номер порта, а при конфигурации (или инсталляции) клиентских приложений указывается сетевой адрес и порт сервера.

В некоторых случаях удобно, чтобы клиент сам находил адрес сервера, если известен только номер порта. Для этого используются широковещательные сообщения, которые отправляются всем компьютерам в сети. Предполагается, что сервер (или несколько серверов) должен находиться в состоянии ожидания (прослушивания) на доступном в сети компьютере. При получении сообщения от клиента, сервер определяет параметры сокета клиента и передает клиенту необходимые данные для установки канала связи. В общем случае в сети может находиться несколько серверов, которые откликнутся на запрос клиента. В этом случае алгоритм работы клиента должен предполагать процедуру обработки откликов и выбора подходящего сервера. Важно отметить, что этот метод работает только в пределах локальной сети, так как широковещательные сообщения обычно не проходят через маршрутизаторы и шлюзы.

Использование широковещательных адресов возможно только в протоколе UDP. Поэтомупри создании дескрипторов сокетов (в программах клиентов и серверов) при вызове функции socket значение параметраtype должно быть SOCK\_DGRAM**,** а для обмена данными этом случае используются функции sendto и recvfrom**.**

// **-- установить опции сокета**

// **Назначение:** функция предназначена для установки режимов

// использования сокета

**int setsockopt (**

**SOCKET s,** // [in] дескриптор сокета

**int level,** // [in] уровень действия режима

**int optname,** // [in] режим сокета для установки

**const char\* optval,** // [in] значение режима сокета

**int fromlen** // [in] длина буфера **optval**

**);**

// **Код возврата:** в случае успешного завершения возвращается

// нуль, иначе функция возвращает значение

// **SOCKET\_ERROR**

// **Примечания:** - поддерживаются два значения параметра **level**:

// **SOL\_SOCKET** и **IPPROTO\_TCP;**

// - для уровня **SOL\_SOCKET** параметр **optval** может принимать более десяти различных значений; например, **SO\_BROADCAST** - для разрешения использования широковещательного адреса**;**

**//** - для уровня **IPPROTO\_TCP** поддерживается одно значение параметра **level**:**TCP\_NODELAY**,которое позволяет устанавливать или отменять использование алгоритма Нейгла (см. TCP/IP);

// - значение **fromlen** всегда **sizeof(int)**;

// - если необходимо установить указанный параметр**(optname)** в состояние Enabled**,** то в поле **optval** должно быть не нулевоезначение (например, **0x00000001**), еслижепараметр устанавливается в состояние Disabled, то поле **optval** должно содержать **0x00000000**

Рисунок 3.15.1. Функция setsockopt

Стандартный широковещательный адрес в формате TCP/IP задается с помощью константы INADDR\_BROADCAST, которая определена в Winsock2.h. По умолчанию использование стандартного широковещательного адреса не допускается и для его применения необходимо установить специальный режим использования сокета SO\_BROADCASTс помощью функции setsockopt (рисунок 3.15.1). Проверить установленные для сокета режимы можно с помощью функции getsockopt (описание здесь не приводится).

//...........................................................

**SOCKET cC;**

**if ((cC = socket(AF\_INET, SOCK\_DGRAM, NULL))== INVALID\_SOCKET)**

**throw SetErrorMsgText("socket:",WSAGetLastError());**

**int optval = 1;**

**if (setsockopt(cC,SOL\_SOCKET,SO\_BROADCAST,**

**(char\*)&optval,sizeof(int)) == SOCKET\_ERROR)**

**throw SetErrorMsgText("opt:",WSAGetLastError());**

**SOCKADDR\_IN all;** // параметры сокета sS

**all.sin\_family = AF\_INET**; // используется IP-адресация

**all.sin\_port = htons(2000);** //порт 2000

**all.sin\_addr.s\_addr = INADDR\_BROADCAST;** // всем

**char buf[] = "answer anyone!";**

**if ((sendlen = sendto(cC, sendbuf, sizeof(buf), NULL,**

**(sockaddr\*)&all, sizeof(all)))== SOCKET\_ERROR)**

**throw SetErrorMsgText("sendto:",WSAGetLastError());**

//...........................................................

Рисунок 3.15.1. Пример применения setsockopе

На рисунке 3.15.1 приводится фрагмент программы, использующей стандартный широковещательный адрес. Функция setsockopt используется в этом примере для установки опции сокета SO\_BROADCAST, позволяющей использовать адрес INADDR\_BROADCAST.

# Применение символического адреса хоста.

**Применение символических имен компьютеров**

В предыдущем разделе разбирался механизм поиска серверного компьютера с помощью использования широковещательных адресов. При наличии специальной службы в сети способной разрешить адрес компьютера по его символическому имени (например, DNS или некоторые протоколы, работающие поверх TCP/IP) проблему можно решить с помощью функции gethostbyname (рисунок 3.16.1). При этом предполагается, что известно символическое имя компьютера, на котором находится программа сервера.

Такое решение достаточно часто применяется разработчиками распределенных систем. Связав набор программ-серверов с определенными стандартными именами компьютеров, распределенное приложение становится не зависимым от адресации в сети. Естественно при этом необходимо позаботиться, чтобы существовала служба, разрешающая адреса компьютеров по имени. Установка таких служб, как правило, возлагается на системного администратора сети.

Помимо функции gethostbyname в составе Winsock2имеется функция gethostbyaddr (рисунок 3.16.2)**,**  назначение которой противоположно**:** получение символического имени компьютера по сетевому адресу. Обе функции используют структуру hosten (рисунок 3.16.3), содержащуюся в Winsock2.h.

// -- получить адрес хоста по его имени

// Назначение: функция для получения информации о хосте по

// его символическому имени

hostent\* gethostbyname

(

const char\* name, // [in] символическое имя хоста

);

// Код возврата: в случае успешного завершения функция

// возвращает указатель на структуру hosten,

// иначе значение NULL

// Примечание: допускается в качестве символического имени,

// указать символическое обозначение адреса

// хоста в виде n.n.n.n

Рисунок 3.16.1. Функция gethostbyname

// -- получить имя хоста по его адресу

// Назначение: функция для получения информации о хосте по

// его символическому имени

hostent\* gethostbyaddr

(

const char\* addr, // [in] адрес в сетевом формате

int la, // [in] длина адреса в байтах

int ta // [in] тип адреса: для TCP/IP AF\_INET

);

// Код возврата: в случае успешного завершения функция

// возвращает указатель на структуру hosten,

// иначе возвращается значение NULL

Рисунок 3.16.2 Функция gethostbyaddr

**typedef struct hostent {** // структура hostent

**char FAR\* h\_name;** // имя хоста

**char FAR FAR\*\* h\_aliases;** // список алиасов

**short h\_addrtype**; // тип адресации

**short h\_length;** // длина адреса

**char FAR FAR\*\* h\_addr\_list;** // список адресов

**} hostent;**

Рисунок 3.16.3 Структура hostent

Следует отметить, что символическое имя ***localhost***  является зарезервированным именем и предназначено для обозначения собственного имени компьютера. Если с помощью функции gethostbyname получить адрес компьютера с именем localhost, то в будет собственный получен IP-адрес компьютера или адрес INADDR\_LOOPBACK.

Кроме того, для получения действительного собственного имени компьютера (NetBIOS-имени или DNS-имени) можно использовать функцию gethostname (рисунок 3.16.4).

// -- получить имя хоста

// Назначение: функция для получения собственного имени хоста

int gethostname

(

char\* name , // [out] имя хоста

int ln // [in] длина буфера name

);

// Код возврата: в случае успешного завершения функция

// возвращает нуль, иначе возвращается значение

// SOCKET\_ERROR

Рисунок 3.16.4 Функция gethostname

Символический адрес хоста - это читаемое для человека имя или метка, используемая для идентификации узла (хоста) в компьютерной сети. Вместо использования числового IP-адреса, символический адрес хоста предоставляет более удобное и запоминающееся имя для обращения к узлу в сети.

Примеры символических адресов хостов:

Доменные имена: В интернете доменные имена представляют собой символические адреса хостов. Например, "google.com", "facebook.com" и "example.com" - это доменные имена, которые соответствуют IP-адресам соответствующих серверов.

Имена компьютеров в локальной сети: В локальной сети, каждый компьютер может иметь символический адрес хоста, например, "laptop", "printer", "server", которые используются для обращения к соответствующему компьютеру в сети.

Для преобразования символического адреса хоста в соответствующий IP-адрес используется процесс, называемый DNS-разрешением (Domain Name System). DNS-серверы отвечают за преобразование символических адресов хостов в соответствующие IP-адреса, позволяя установить сетевое соединение с нужным узлом.

# Основные сетевые утилиты и их назначение.

Утилиты представляют собой внешние команды операционной системы и предназначаются для диагностики сети. При этом утилиты, наименования которых выделено жирным шрифтом считаются стандартными для протокола TCP/IP и присутствуют в большинстве операционных систем.

Таблица 2.9.1

|  |  |
| --- | --- |
| **Наименование**  **утилиты** | **Назначение утилит** |
| **ping** | Проверка соединения с одним или более хостами в сети |
| **tracert** | Определение маршрута до пункта назначения |
| **route** | Просмотр и модификация таблицы сетевых маршрутов |
| **netstat** | Просмотр статистики текущих сетевых TCP/IP-соединений |
| **arp** | Просмотр и модификация ARP-таблицы |
| **nslookup** | Диагностика DNS-серверов |
| hostname | Просмотр имени хоста |
| ipconfig | Просмотр текущей конфигурации сети TCP/IP |
| nbtstat | Просмотр статистики текущих сетевых NBT-соединений |
| net | Управление сетью |

**Утилита ping.** Как уже отмечалось раньше, ping в своей работе использует протокол ICMP и предназначена для проверки соединения с удаленным хостом. Проверка соединения осуществляется путем посылки в адрес хоста специальных ICMP-пакетов, которые в соответствии с протоколом должны быть возвращены, отправляющему хосту (эхо-пакеты и эхо-ответы).

Для получения справки о параметрах утилиты pingследует выполнить команду ping без параметров.

**ping** hostname

где hostnаme – NetBIOS или DNS - имя хоста или его IP-адрес.

**Утилит tracert.** Как и утилита ping, tracert использует ICMP протокол для определения маршрута до пункта назначения. В результате работы утилиты на консоль выводятся все промежуточные узлы маршрута от исходного хоста до пункта назначения и время их прохождения.

Для получения справки о параметрах утилиты tracert следует выполнить команду tracert без параметров.

**tracert** hostname

где hostnаme – NetBIOS или DNS - имя хоста или его IP-адрес.

**Утилита route**. Утилита route позволяет манипулировать таблицей сетевых маршрутов, которая имеется на каждом компьютере с TCP/IP-интерфейсом. Утилита обеспечивает выполнение четырех команд: print (распечатка таблицы сетевых маршрутов), add (добавить маршрут в таблицу), change (изменение существующего маршрута), delete (удаление маршрута).

**route** print

где параметр (команда) print, без уточняющих операндов, указывает на необходимость распечатки всей таблицы.

**Утилита netstat.** Утилита отражает состояние текущих TCP/IP-соединений хоста, а также статистику работы протоколов. С помощью утилиты netstat можно распечатать номера ожидающих портов всех соединений TCP/IP, имена исполняемых файлов, участвующих в подключениях, идентификаторы соответствующих Windows-процессов и т.д.

Для получения справки о параметрах утилиты netstat, следует выполнить следующую команду.

**netstat** -?

Активные соединения TCP/IP на компьютере можно просмотреть, набрав на консоли команду nestat с параметром -a.

**netstat** -a

**Утилита arp**. Утилита используется для просмотра и модификации ARP-таблицы, используемой для трансляции IP-адресов в адреса протоколов канального уровня (MAC-адреса). С помощью параметров команды можно распечатывать таблицу, удалять и добавлять данные ARP-таблицы. Корректировку ARP-таблицы может осуществлять только пользователь с правами администратора.

Получить текущее состояние ARP-таблицы можно с помощью следующей команды.

**arp** -a

**Утилита nslookup.** Утилита nslookup предназначена для проверки правильности работы DNS-серверов. С помощью утилиты, пользователь может выполнять запросы к DNS-серверам на получение адреса хоста по его DNS-имени, на получение адресов и имен почтовых серверов, ответственных за доставку почты для отдельных доменов DNS, на получение почтового адреса администратора DNS-сервера и т.д. и т.п. Утилита работает в двух режимах: в режиме однократного выполнения (при запуске в командной строке задается полный набор параметров) и в интерактивном режиме (команды и параметры задаются в режиме диалога).

Запуск утилиты в интерактивном режиме осуществляется запуском команды nslookup без параметров.

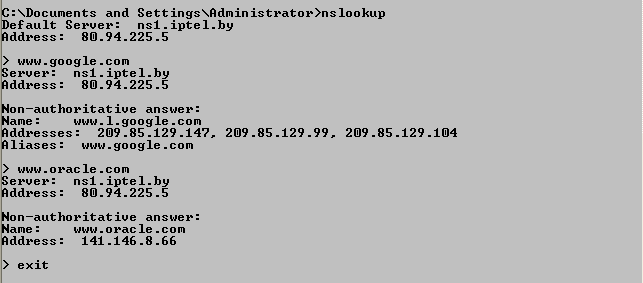


Рисунок 2.9.1. Пример выполнения команды nslookup

На рисунке 2.9.1 демонстрируется работа с утилитой nslookup. Сразу после запуска команды на консоль выводится имя хоста и IP-адрес активного DNS-сервера. После этого в диалоговом режиме были получены IP-адреса хостов с именами [www.google.com](http://www.google.com), [www.oracle.com](http://www.oracle.com) и для завершения работы утилиты была введена команда exit.

**Утилита hostname**. Утилита предназначена для вывода на консоль имени хоста, на котором выполняется данная команда. Команда hostname не имеет никаких параметров.

**Утилита ipconfig.** Утилита ipconfig является наиболее востребованной сетевой утилитой. С ее помощью можно определить конфигурацию IP-интерфейса и значения всех сетевых параметров. Особенно эта утилита полезна на компьютерах, работающих с протоколом DHCP: команда позволяет проверить параметры IP-интерфейсов установленные в автоматическом режиме.

Для получения справки о параметрах утилиты следует ввести следующую команду.

**ipconfig**  /?

Короткий отчет о конфигурации TCP/IP можно получить выдав команду ipconfig без параметров. Для получения полного отчета, можно использовать ключ /all, как это сделано на рисунке 2.9.2.

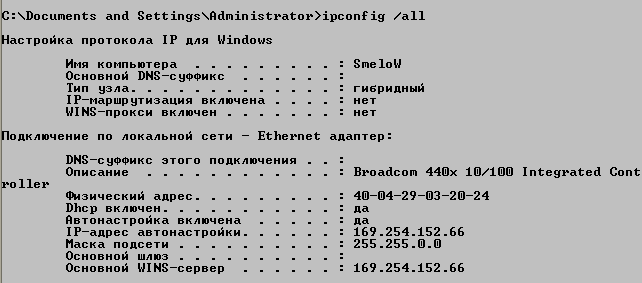


Рисунок 2.9.2. Пример выполнения команды ipconfig

# Служба DNS.

**DNS** (система доменных имен) преобразует доменные имена, удобные для человеческого восприятия (например, www.amazon.com), в IP-адреса, понимаемые машиной (например, 192.0.2.44).

**Протокол DNS** относится к протоколам стека TCP/IP (прикладной уровень).Описан в RFC 1034 (Concepts and Facilities), 1035(implementation and specification) или STD 13.

Служба DNS (домены, зоны; зоны прямого и обратного просмотра; основные и дополнительные зоны; рекурсивный и итеративный запросы на разрешение имен).

С помощью DNS стало возможным реализовать масштабируемый распределенный механизм, устанавливающий соответствие между иерархическими именами сайтов и числовыми IP-адресами.

**Необходимость отображения имен сетевых узлов в IP-адреса**

Компьютеры и другие сетевые устройства, отправляя друг другу пакеты по сети, используют IP-адреса.. Однако пользователю (человеку) гораздо проще и удобнее запомнить некоторое символические имена сетевых узлов, чем четыре бессодержательных для него числа. Однако, если люди в своих операциях с сетевыми ресурсами будут использовать имена узлов, а не IP-адреса, тогда должен существовать механизм, сопоставляющий именам узлов их IP-адреса.

Есть два таких механизма — локальный для каждого компьютера файл *hosts* и централизованная иерархическая служба имен *DNS*.

**ICANN** (Internet Corporation for Assign Names and Numbers) – организация, обеспечивающая управление всем адресным пространством Internet.

**Использование локального файла hosts и системы доменных имен DNS для разрешения имен сетевых узлов**

На начальном этапе развития сетей, когда количество узлов в каждой сети было небольшое, достаточно было на каждом компьютере хранить и поддерживать актуальное состояние простого текстового файла, в котором содержался список сетевых узлов данной сети. Список устроен очень просто — в каждой строке текстового файла содержится пара «IP-адрес — имя сетевого узла». В системах семейства Windows данный файл расположен в папке “*%system root%\system32\drivers\etc*” (где *%system root%* обозначает папку, в которой установлена операционная система). Сразу после установки системы Windows создается файл hosts с одной записью «127.0.0.1 localhost».

С ростом сетей поддерживать актуальность и точность информации в файл hosts становится все труднее. Для этого надо постоянно обновляя содержимое этого файла на всех узлах сети. Кроме того, такая простая технология не позволяет организовать пространство имен в какую-либо структуру. Поэтому появилась необходимость в централизованной базе данных имен, позволяющей производить преобразование имен в IP-адреса без хранения списка соответствия на каждом компьютере. Такой базой стала DNS (Domain Name System) — система именования доменов, которая начала массовую работу в 1987 году.

Заметим, что с появлением службы DNS актуальность использования файла host совсем не исчезла, в ряде случаев использование этого файла оказывается очень эффективным.

**Служба DNS: пространство имен, домены**

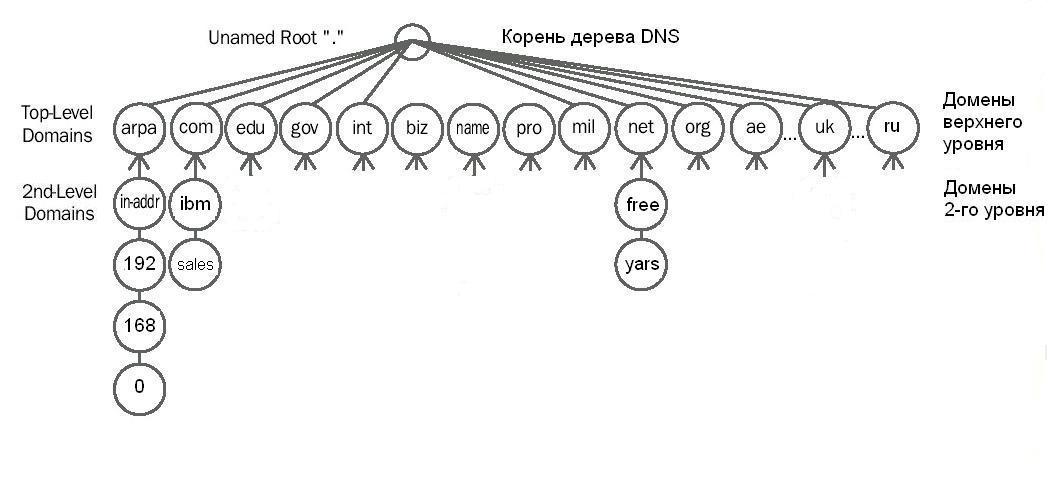
DNS — это иерархическая база данных, сопоставляющая имена сетевых узлов и их сетевых служб IP-адресам узлов. Содержимое этой базы, с одной стороны, распределено по большому количеству серверов службы DNS, а с другой стороны, является централизованно управляемым. В основе иерархической структуры базы данных DNS лежит *доменное пространство имен* (*domain namespace*), основной структурной единицей которого является домен, объединяющий сетевые узлы (хосты), а также поддомены. Процесс поиска в БД службы DNS имени некого сетевого узла и сопоставления этому имени IP-адреса называется «разрешением имени узла в пространстве имен DNS».

Служба DNS состоит из трех основных компонент:

* **Пространство имен DNS и соответствующие ресурсные записи (RR, resource record)** — это сама распределенная база данных DNS;
* **Серверы имен DNS** — компьютеры, хранящие базу данных DNS и отвечающие на запросы DNS-клиентов;
* **DNS-клиенты (DNS-clients, DNS-resolvers)** —компьютеры, посылающие запросы серверам DNS для получения ресурсных записей.

**Пространство имен.**

Пространство имен DNS — иерархическая древовидная структура, начинающаяся с корня, не имеющего имени и обозначаемого точкой “**.**”. Схему построения пространства имен DNS лучше всего проиллюстрировать на примере сети Интернет (Рис. 3.8).



Для доменов 1-го уровня различают 3 категории имен:

* **ARPA** — специальное имя, используемое для обратного разрешения DNS (из IP-адреса в полное имя узла);
* **Общие (generic) имена 1-го уровня** — 16 (на данный момент) имен, назначение которых приведено в таблице 3.4;
* **Двухбуквенные имена для стран** — имена для доменов, зарегистрированных в соответствующих странах (например, *ru* — для России, *ua* — для Украины, *uk* — для Великобритании и т.д.).

|  |  |
| --- | --- |
| **Имя домена** | **Назначение** |
| aero | Сообщества авиаторов |
| biz | Компании (без привязки к стране) |
| com | Коммерческие организации, преимущественно в США (например, домен *microsoft.com* для корпорации Microsoft) |
| coop | Кооперативы |
| edu | Образовательные учреждения в США |
| gov | Правительственные учреждения США |
| info | Домен для организаций, предоставляющих любую информацию для потребителей |
| int | международные организации (например, домен *nato.int* для НАТО) |
| mil | Военные ведомства США |
| museum | Музеи |
| name | Глобальный домен для частных лиц |
| net | Домен для Интернет-провайдеров и других организаций, управляющих структурой сети Интернет |
| org | Некоммерческие и неправительственные организации, преимущественно в США |
| pro | Домен для профессиональных объединений (врачей, юристов, бухгалтеров и др.) |
| job | Кадровые агентства |
| travel | Туроператоры |

Корневые серверы DNS — DNS-серверы, содержащие информацию о доменах верхнего уровня, указывающую на DNS-серверы, поддерживающие работу каждого из этих доменов. Основные корневые серверы DNS размещены в домене root-servers.net обозначаются латинскими буквами от A до М. Они управляются различными организациями, действующими по согласованию с ICANN. Количество серверов ограничено в связи с максимальным объёмом UDP-пакета (большее количество серверов потребовало бы перехода на TCP-протокол для получения ответа, что существенно бы увеличило нагрузку).

Хотя из-за ограничения на размеры DNS-пакета (512 байт) в DNS-ответ может быть помещено всего 13 серверов, на самом деле сейчас за этими 13 «виртуальными» серверами стоят 190 серверов (зеркала).(13 корневых dns servera)

Домен верхнего (первого) уровня (англ. top-level domain — TLD) — в иерархии системы доменнных имён (DNS) самый высокий уровень после корневого домена (англ. root domain). Является начальной точкой отсчёта (справа налево), с которой начинается доменное имя в Интернете.

Полное имя узла (FQDN, fully qualified domain name) состоит из нескольких имен, называемых метками (label) и разделенных точкой. Самая левая метка относится непосредственно к узлу, остальные метки — список доменов от домена первого уровня до того домена, в котором находится узел (данный список просматривается справа налево).

FQDN (сокр. от англ. Fully Qualified Domain Name, полностью определённое имя домена, иногда сокращается до «полное имя») — имя домена, не имеющее неоднозначностей в определении. Включает в себя имена всех родительских доменов иерархии DNS.

В DNS и, что особенно существенно, в файлах зоны(англ.), FQDN завершаются точкой (например, example.com.), то есть включают корневое доменное имя, которое является безымянным.

Максимальный размер FQDN — 255 байт, с ограничением в 63 байта на каждое имя домена.

Серверы имен DNS.

Серверы имен DNS (или DNS-серверы) — это компьютеры, на которых хранятся те части БД пространства имен DNS, за которые данные серверы отвечают, и функционирует программное обеспечение, которое обрабатывает запросы DNS-клиентов на разрешение имен и выдает ответы на полученные запросы.

DNS-клиенты.

DNS-клиент — это любой сетевой узел, который обратился к DNS-серверу для разрешения имени узла в IP-адрес или, обратно, IP-адреса в имя узла.

**Служба DNS: домены и зоны**

Как уже говорилось выше, каждый DNS-сервер отвечает за обслуживание определенной части пространства имен DNS. Информация о доменах, хранящаяся в БД сервера DNS, организуется в особые единицы, называемые *зонами* (*zones*).

Зона — основная единица репликации данных между серверами DNS. Каждая зона содержит определенное количество ресурсных записей для соответствующего домена и, быть может, его поддоменов.

Системы семейства Windows Server поддерживают следующие типы зон:

* **Стандартная основная** (**standard primary**) — главная копия стандартной зоны; только в данном экземпляре зоны допускается производить какие-либо изменения, которые затем реплицируются на серверы, хранящие дополнительные зоны;
* **Стандартная дополнительная** (**standard secondary**) — копия основной зоны, доступная в режиме «только-чтение», предназначена для повышения отказоустойчивости и распределения нагрузки между серверами, отвечающими за определенную зону; процесс репликации изменений в записях зон называется «*передачей зоны*» (*zone transfer*)  
  (информация в стандартных зонах хранится в тестовых файлах, файлы создаются в папке «%system root%\system32\dns», имя файла, как правило, образуется из имени зоны с добавлением расширения файла «.dns»; термин «стандартная» используется только в системах семейства Windows);
* **Интегрированная в Active Directory** (**Active Directory–integrated**) — вся информация о зоне хранится в виде одной записи в базе данных Active Directory (такие типы зон могут существовать только на серверах Windows, являющихся контроллерами доменов Active Directory; в интегрированных зонах можно более жестко управлять правами доступа к записям зоны; изменения в записях зоны между разными экземплярами интегрированной зоны производятся не по технологии передачи зоны службой DNS, а механизмами репликации службы Active Directory);
* **Зона-заглушка** (**stub**; только в Windows 2003) — особый тип зоны, которая для данной части пространства имен DNS содержит самый минимальный набор ресурсных записей (начальная запись зоны SOA, список серверов имен, отвечающих за данную зону, и несколько записей типа A для ссылок на серверы имен для данной зоны).

**Первичный DNS-сервер**: сервер, который всегда может прочитать свой зонный файл. сервер, имеющий право на внесение изменений в данные зоны. Для каждой зоны может быть только один первичный сервер.

**Вторичный DNS**-**сервер:** сервер, получающий свои данные о первичного сервера. Он поддерживает свой зонный файл, но постоянно его обновляет. Для обновления использует запись SOA. Для каждой зоны, должен существовать вторичный сервер. Большинство реализаций обязывают вторичный сервер обновлять только изменения зоны первичного сервера. Вторичные сервера поддерживаю актуальную копию зонного файла.

**Кэширующий сервер:**  используется редко, в больших разветвленных организациях.

**Зоны прямого и обратного просмотра**

Зоны, рассмотренные в предыдущем примере, являются *зонами прямого просмотра* (*forward lookup zones*). Данные зоны служат для разрешения имен узлов в IP-адреса. Наиболее часто используемые для этого типы записей: A, CNAME, SRV.

Для определения имени узла по его IP-адресу служат *зоны обратного просмотра* (*reverse lookup zones*), основной тип записи в «обратных» зонах — PTR. Для решения данной задачи создан специальный домен. с именем «*in-addr.arpa*». Для каждой IP-сети в таком домене создаются соответствующие поддомены, образованные из идентификатора сети, записанного в обратном порядке. Записи в такой зоне будут сопоставлять идентификатору узла полное FQDN-имя данного узла. Например, для IP-сети *192.168.0.0/24* необходимо создать зону с именем «*0.168.192.in-addr.arpa*». Для узла с IP-адресом 192.168.0.10 и именем *host.company.ru* в данной зоне должна быть создана запись «10 PTR host.company.ru».

**Алгоритмы работы итеративных и рекурсивных запросов DNS**

Все запросы, отправляемые DNS-клиентом DNS-серверу для разрешения имен, делятся на два типа:

* итеративные запросы (клиент посылает серверу DNS запрос, в котором требует дать наилучший ответ без обращений к другим DNS-серверам);
* рекурсивные запросы (клиент посылает серверу DNS запрос, в котором требует дать окончательный ответ даже если DNS-серверу придется отправить запросы другим DNS-серверам; посылаемые в этом случае другим DNS-серверам запросы будут итеративными).

В основном DNS-клиентами используются рекурсивные запросы. На рис. 5.3 проиллюстрирован процесс разрешения доменного имени с помощью рекурсивного запроса.

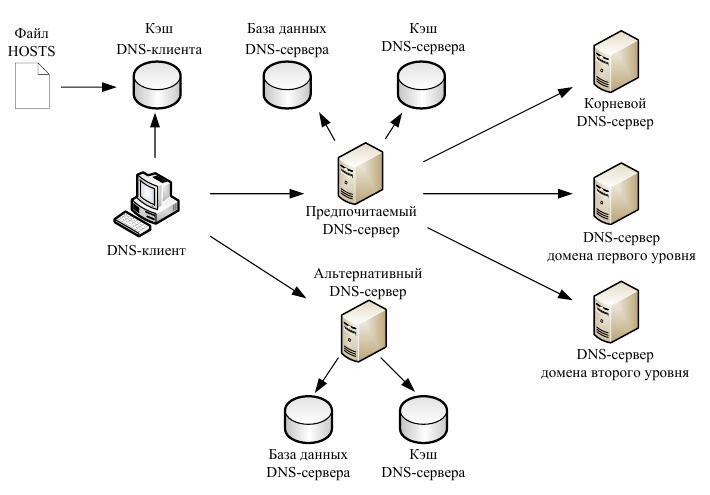


Рис. 5.3. Процесс обработки рекурсивного DNS-запроса

Сначала DNS-клиент осуществляет поиск в собственном локальном кэше DNS-имен. Это память для временного хранения ранее разрешенных запросов. В эту же память переносится содержимое файла HOSTS (каталог windows/system32/drivers/etc). Утилита IPconfig с ключом /displaydns отображает содержимое DNS-кэша. Если кэш не содержит требуемой информации, DNS-клиент обращается с рекурсивным запросом к предпочитаемому DNS-серверу (Preferred DNS server), адрес которого указывается при настройке стека TCP/IP. DNS-сервер просматривает собственную базу данных, а также кэш-память, в которой хранятся ответы на предыдущие запросы, отсутствующие в базе данных. В том случае, если запрашиваемое доменное имя не найдено, DNS-сервер осуществляет итеративные запросы к DNS-серверам верхних уровней, начиная с корневого DNS-сервера.

**Реализация службы DNS в системах семейства Windows Server**

Главная особенность службы DNS в системах семейства Windows Server заключается в том, что служба DNS разрабатывалась для поддержки службы каталогов Active Directory. Для выполнения этой функции требуются обеспечение двух условий:

* поддержка службой DNS динамической регистрации (dynamic updates);
* поддержка службой DNS записей типа SRV.

Для непосредственного отображения пространства имен в пространство IP-адресов служат т.н. ресурсные записи (RR, resource record). Каждый сервер DNS содержит ресурсные записи для той части пространства имен, за которую он несет ответственность (authoritative). Таблица 3.5 содержит описание наиболее часто используемых типов ресурсных записей.

|  |  |  |
| --- | --- | --- |
| **Тип ресурсной записи** | **Функция записи** | **Описание использования** |
| A | Host Address  Адрес хоста, или узла | Отображает имя узла на IP-адрес  (например, для домена *microsoft.com* узлу с именем *www.microsoft.com* сопоставляется IP-адрес с помощью такой записи:  www A 207.46.199.60) |
| CNAME | Canonical Name (alias)  Какноническое имя (псевдоним) | Отображает одно имя на другое |
| MX | Mail Exchanger  Обмен почтой | Управляет маршрутизация почтовых сообщений для протокола SMTP |
| NS | Name Server  Сервер имен | Указывает на серверы DNS, ответственные за конкретный домен и его поддомены |
| PTR | Pointer  Указатель | Используется для обратного разрешения IP-адресов в имена узлов в домене *in-addr.arpa* |
| SOA | Start of Authority  Начальная запись зоны | Используется для указания основного сервера для данной зоны и описания свойств зоны |
| SRV | Service Locator  Указатель на службу | Используется для поиска серверов, на которых функционируют определенные службы (например, контроллеры доменов Active Directory или серверы глобального каталога) |

**Реализации DNS-серверов**: BIND (Berkeley Internet Name Domain), Microsoft DNS Server, OpenDNS.

**BIND (Berkeley Internet Name Domain, до этого: Berkeley Internet Name Daemon)** — открытая и наиболее распространённая реализация DNS-сервера, обеспечивающая выполнение преобразования DNS-имени в IP-адрес и наоборот.

BIND поддерживается организацией Internet Systems Consortium. BIND был создан студентами и впервые был выпущен в BSD 4.3.

В Unix этот сервер является стандартом де-факто, но имеются и альтернативы:

* PowerDNS — www.powerdns.com;
* MyDNS — DNS-сервер, использующий в качестве БД MySQL;
* Microsoft DNS Server — входит в состав серверных версий Windows.

Утилита **nslookup** используется для проверки способности DNS-серверов выполнять разрешение имен.

# Служба DHCP.

DHCP (англ. Dynamic Host Configuration Protocol — протокол динамической конфигурации узла) — это сетевой протокол, позволяющий компьютерам автоматически получать IP-адрес и другие параметры, необходимые для работы в сети TCP/IP. Данный протокол работает по модели «клиент-сервер». Для автоматической конфигурации компьютер-клиент на этапе конфигурации сетевого устройства обращается к так называемому серверу DHCP, и получает от него нужные параметры. Сетевой администратор может задать диапазон адресов, распределяемых сервером среди компьютеров. Это позволяет избежать ручной настройки компьютеров сети и уменьшает количество ошибок. Протокол DHCP используется в большинстве крупных (и не очень) сетей TCP/IP.

Работа протокола DHCP базируется на классической схеме клиент-сервер. В роли клиентов выступают компьютеры сети, стремящиеся получить IP-адреса в так называемую аренду (lease), а DHCP-серверы выполняют функции диспетчеров, которые выдают адреса, контролируют их использование и сообщают клиентам требуемые параметры конфигурации. Сервер поддерживает пул свободных адресов и, кроме того, ведет собственную регистрационную базу данных. Взаимодействие DHCP-серверов со станциями-клиентами осуществляется путем обмена сообщениями.

Стандарт протокола DHCP был принят в октябре 1993 года. Действующая версия протокола (март 1997 года) описана в RFC 2131. Новая версия DHCP, предназначенная для использования в среде IPv6, носит название DHCPv6 и определена в RFC 3315 (июль 2003 года).

DHCP появился не на пустом месте - различные схемы управления IP-адресами в сетевой среде предлагались и раньше. Однако эти схемы имеют по крайней мере один из двух недостатков - не допускают динамического назначения IP-адресов либо позволяют передавать от сервера на станцию-клиент лишь небольшое число параметров конфигурации.

При разработке протокола DHCP преследовалась цель устранить оба ограничения. Требовался механизм, который позволил бы ликвидировать стадию ручного конфигурирования компьютеров, поддерживал многосегментные сети, не требуя наличия DHCP-сервера в каждой подсети, не конфликтовал с существующими сетевыми протоколами и компьютерами, имеющими статичную конфигурацию, был способен взаимодействовать с ретранслирующими агентами протокола BOOTP и обслуживать BOOTP-клиентов, наконец, допускал управление передаваемыми параметрами конфигурации. BOOTP протокол, позволяющий бездисковым клиентам во время старта конфигурировать установки TCP/IP. Что касается более узких задач, то DHCP должен был обеспечивать уникальность сетевых адресов, используемых разными компьютерами сети в данный момент, сохранение прежней конфигурации клиентской станции после перезагрузки клиента или сервера, автоматическое присвоение параметров конфигурации вновь подключенным машинам.

Упомянутое выше требование поддержки базовых элементов протокола BOOTP возникло не случайно. DHCP разрабатывался как непосредственное расширение BOOTP и именно в таком качестве воспринимается BOOTP-клиентами. Этому обстоятельству в первую очередь способствует формат сообщений DHCP, во многом совпадающий с форматом, который применяется протоколом-предшественником и определен в документе RFC 951.

**Устройство протокола**

Протокол DHCP является клиент-серверным, то есть в его работе участвуют клиент DHCP и сервер DHCP. Передача данных производится при помощи протокола UDP, при этом сервер принимает сообщения от клиентов на порт 67 и отправляет сообщения клиентам на порт 68.

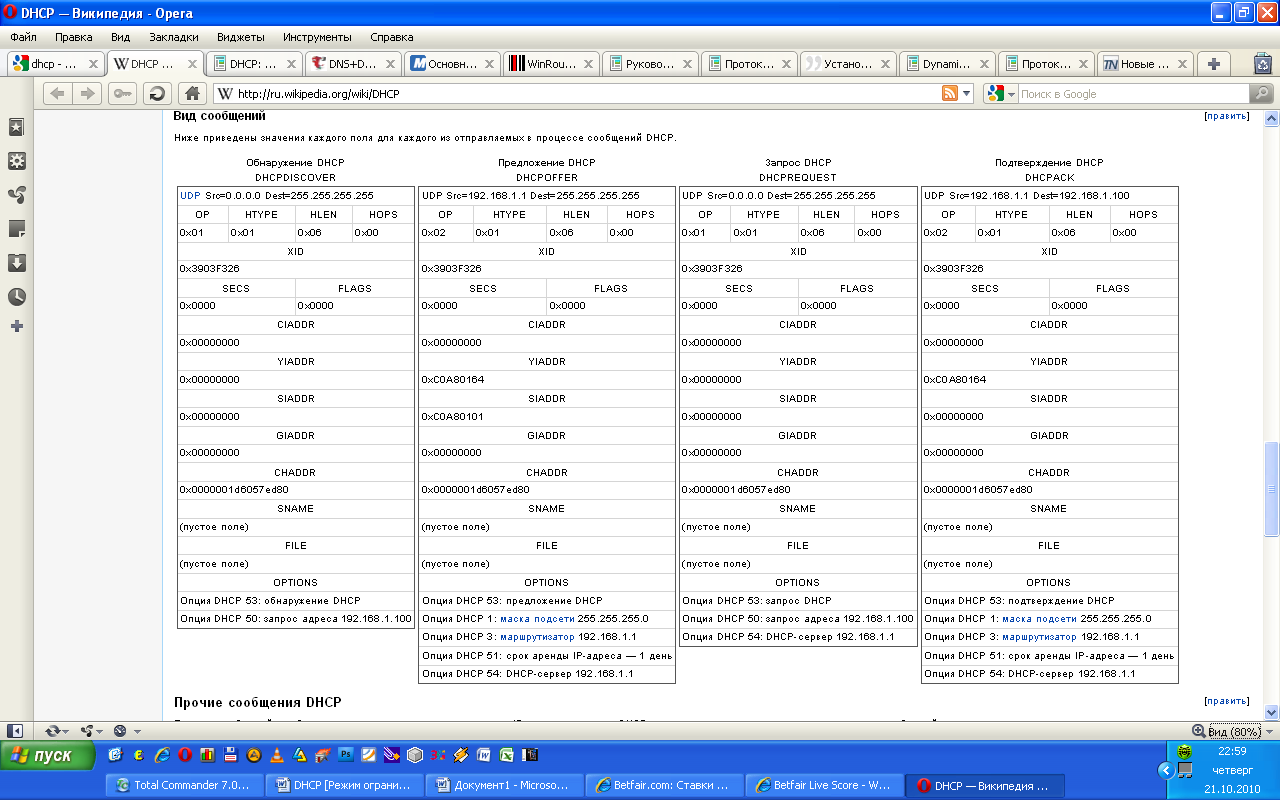
**Структура сообщений DHCP**

Все сообщения протокола DHCP разбиваются на поля, каждое из которых содержит определённую информацию. Все поля, кроме последнего (поля опций DHCP), имеют фиксированную длину.

|  |  |  |
| --- | --- | --- |
| Поле | Описание | Длина (в байтах) |
| op | Тип сообщения. Может принимать два значения: BOOTREQUEST (1, запрос от клиента к серверу) и BOOTREPLY (2, ответ от сервера к клиенту). | 1 |
| htype | Тип аппаратного адреса. Допустимые значения этого поля определены в RFC «Assigned Numbers». Например, для MAC-адреса Ethernet 10 Мбит/с это поле принимает значение 1. | 1 |
| hlen | Длина аппаратного адреса в байтах. Для MAC-адреса Ethernet — 6. | 1 |
| hops | Количество промежуточных маршрутизаторов (так называемых агентов ретрансляции DHCP), через которые прошло сообщение. Клиент устанавливает это поле в 0. | 1 |
| xid | Уникальный идентификатор транзакции, генерируемый клиентом в начале процесса получения адреса. | 4 |
| secs | Время в секундах с момента начала процесса получения адреса. Может не использоваться (в этом случае оно устанавливается в 0). | 2 |
| flags | Поле для флагов — специальных параметров протокола DHCP. | 2 |
| ciaddr | IP-адрес клиента. Заполняется только в том случае, если клиент уже имеет собственный IP-адрес и способен отвечать на запросы ARP (это возможно, если клиент выполняет процедуру обновления адреса по истечении срока аренды). | 4 |
| yiaddr | 'your' (client) IP address (RFC 2131) | 4 |
| siaddr | IP-адрес сервера. Возвращается в предложении DHCP (см. ниже). | 4 |
| giaddr | IP-адрес агента ретрансляции, если таковой участвовал в процессе доставки сообщения DHCP до сервера. | 4 |
| chaddr | Аппаратный адрес (обычно MAC-адрес) клиента. | 16 |
| sname | Необязательное имя сервера в виде нуль-терминированной строки. | 64 |
| file | Необязательное имя файла на сервере, используемое бездисковыми рабочими станциями при удалённой загрузке. Как и sname, представлено в виде нуль-терминированной строки. | 128 |
| options | Поле опций DHCP. Здесь указываются различные дополнительные параметры конфигурации. В начале этого поля указываются четыре особых байта со значениями 99, 130, 83, 99 («волшебные числа»), позволяющие серверу определить наличие этого поля. | переменная |

**Вид сообщений**

Отправляемых в процессе сообщений DHCP.Обнаружение DHCP



**Распределение IP-адресов**

Протокол DHCP поддерживает три механизма выделения адресов: автоматический, динамический и ручной.

**Ручное распределение**. При этом способе сетевой администратор сопоставляет аппаратному адресу (обычно MAC-адресу) каждого клиентского компьютера определённый IP-адрес. Фактически, данный способ распределения адресов отличается от ручной настройки каждого компьютера лишь тем, что сведения об адресах хранятся централизованно (на сервере DHCP), и потому их проще изменять при необходимости.

**Автоматическое распределение**. При данном способе каждому компьютеру на постоянное использование выделяется произвольный свободный IP-адрес из определённого администратором диапазона.

**Динамическое распределение**. Этот способ аналогичен автоматическому распределению, за исключением того, что адрес выдаётся компьютеру не на постоянное пользование, а на определённый срок. Это называется арендой адреса. По истечении срока аренды IP-адрес вновь считается свободным, и клиент обязан запросить новый (он, впрочем, может оказаться тем же самым).

Рассмотрим пример процесса получения IP-адреса клиентом от сервера DHCP. Предположим, клиент ещё не имеет собственного IP-адреса, но ему известен его предыдущий адрес — 192.168.1.100.

1. Клиент посылает в собственную физическую подсеть широковещательное сообщение DHCPDISCOVER, в котором могут указываться устраивающие клиента IP-адрес и срок его аренды.

при этом в качестве IP-адреса источника указывается 0.0.0.0 (так как компьютер ещё не имеет собственного IP-адреса), а в качестве адреса назначения — широковещательный адрес 255.255.255.255.

Если в данной подсети DHCP-сервер отсутствует, сообщение будет передано в другие подсети ретранслирующими агентами протокола BOOTP (они же вернут клиенту ответные сообщения сервера).

Клиент заполняет несколько полей сообщения начальными значениями:

В поле xid помещается уникальный идентификатор транзакции, который позволяет отличать данный процесс получения IP-адреса от других, протекающих в то же время.

В поле chaddr помещается аппаратный адрес (MAC-адрес) клиента.

В поле опций указывается последний известный клиенту IP-адрес. В данном примере это 192.168.1.100. Это необязательно и может быть проигнорировано сервером.

2. Любой из DHCP-серверов может ответить на поступившее сообщение DHCPDISCOVER сообщением DHCPOFFER, включив в него доступный IP-адрес (yiaddr) и, если требуется, параметры конфигурации клиента. На этой стадии сервер не обязан резервировать указанный адрес. В принципе, он имеет право предложить его другому клиенту, также отправившему запрос DHCPDISCOVER. Тем не менее спецификации RFC 2131 рекомендуют серверу без необходимости не применять подобную тактику, а кроме того, убедиться (например, выдав эхо-запрос ICMP) в том, что предложенный адрес в текущий момент не используется каким-либо из компьютеров сети.

3. Клиент не обязан реагировать на первое же поступившее предложение. Допускается, чтобы он дождался откликов от нескольких серверов и, остановившись на одном из предложений, отправил в сеть широковещательное сообщение DHCPREQUEST. В нем содержатся идентификатор выбранного сервера и, возможно, желательные значения запрашиваемых параметров конфигурации.

Не исключено, что клиента не устроит ни одно из серверных предложений. Тогда вместо DHCPREQUEST он снова выдаст в сеть запрос DHCPDISCOVER, а серверы так и не узнают, что их предложения отклонены. Именно по этой причине сервер не обязан резервировать помещенный в DHCPOFFER адрес.

Если в процессе ожидания серверных откликов на DHCPDISCOVER достигнут тайм-аут, клиент выдает данное сообщение повторно.

4. Присутствующий в сообщении DHCPREQUEST идентификатор позволяет соответствующему DHCP-серверу убедиться в том, что клиент принял именно его предложение. В ответ сервер отправляет подтверждение DHCPACK, содержащее значения требуемых параметров конфигурации, и производит соответствующую запись в базу данных.

Если к моменту поступления сообщения DHCPREQUEST предложенный адрес уже <ушел> к другому клиенту (например, первая станция слишком долго <размышляла> над поступившими предложениями), сервер отвечает сообщением DHCPNACK.

5. Получив сообщение DHCPACK, клиент обязан убедиться в уникальности IP-адреса (средствами протокола ARP) и зафиксировать суммарный срок его аренды. Последний рассчитывается как время, прошедшее между отправкой сообщения DHCPREQUEST и приемом ответного сообщения DHCPACK, плюс срок аренды, указанный в DHCPACK.

Обнаружив, что адрес уже используется другой станцией, клиент обязан отправить серверу сообщение DHCPDECLINE и не ранее чем через 10 с начать всю процедуру снова. Процесс конфигурирования возобновляется и при получении серверного сообщения DHCPNACK.

При достижении тайм-аута в процессе ожидания серверных откликов на сообщение DHCPREQUEST клиент выдает его повторно.

6. Для досрочного прекращения аренды адреса клиент отправляет серверу сообщение DHCPRELEASE.

Приведенная последовательность действий заметно упрощается, если станция-клиент желает повторно работать с IP-адресом, который когда-то уже был ей выделен. В этом случае первым отправляемым сообщением является DHCPREQUEST, в котором клиент указывает прежде использовавшийся адрес. В ответ он может получить сообщение DHCPACK или DHCPNACK (если адрес занят либо клиентский запрос является некорректным, например из-за перемещения клиента в другую подсеть). Обязанность проверить уникальность IP-адреса опять-таки возлагается на клиента.

**Недостатки DHCP**

Освобождая сетевых администраторов от множества рутинных операций, DHCP оставляет нерешенными ряд проблем, которые рано или поздно могут возникнуть в реальной сетевой среде.

К недостаткам этого протокола прежде всего следует отнести крайне низкий уровень информационной безопасности, что обусловлено непосредственным использованием протоколов UDP и IP. В настоящее время не существует практически никакой защиты от появления в сети несанкционированных DHCP-серверов, способных рассылать клиентам ошибочную или потенциально опасную информацию - некорректные или уже задействованные IP-адреса, неверные сведения о маршрутизации и т.д. И наоборот, клиенты, запущенные с неблаговидными целями, могут извлекать конфигурационные сведения, предназначенные для <законных> компьютеров сети, и тем самым оттягивать на себя значительную часть имеющихся ресурсов. Понятно, что возможности административного ограничения доступа, о которых говорилось выше, не способны закрыть эту брешь в системе информационной безопасности.

По мнению некоторых экспертов, в настоящее время DHCP недостаточно отказоустойчив. Протоколу явно недостает механизма активного уведомления клиентов об экстремальных ситуациях (например, о систематической нехватке адресов) и серверного подтверждения об освобождении адреса, иногда в сети наблюдаются всплески числа запросов на повторное использование адресов и т.д. Впрочем, работа над протоколом еще не завершена, и не исключено, что некоторые недостатки будут устранены в последующих редакциях.

**Преимущества DHCP**

Вот некоторые преимущества использования DHCP:

Автоматическое присвоение IP-адресов: DHCP позволяет автоматически назначать IP-адреса устройствам в сети. Это упрощает процесс настройки и управления IP-адресами, особенно в больших сетях. Вместо ручной настройки каждого устройства администратор может настроить DHCP-сервер, который будет автоматически присваивать адреса устройствам при их подключении.

Управление сетевыми параметрами: DHCP также может автоматически назначать другие сетевые параметры, такие как маска подсети, шлюз по умолчанию и DNS-серверы. Это упрощает задачу настройки этих параметров для всех устройств в сети и обеспечивает их единообразие и согласованность.

Изменение и обновление параметров: DHCP позволяет централизованно изменять и обновлять сетевые параметры. Если, например, изменяется IP-адрес шлюза по умолчанию или адрес DNS-сервера, администратор может внести изменения только на DHCP-сервере, и все подключенные устройства получат обновленные параметры при следующем запросе DHCP.

Предотвращение конфликтов адресов: DHCP-сервер отслеживает присвоенные адреса и не позволяет повторное использование одного и того же адреса в сети. Это помогает предотвратить возникновение конфликтов адресов, которые могут возникать при ручной настройке IP-адресов.

Централизованное управление: DHCP позволяет централизованно управлять сетевыми адресами и параметрами. Администратор может настроить DHCP-сервер и контролировать процесс присвоения адресов и параметров в сети. Это облегчает задачу администрирования сети и уменьшает вероятность ошибок.

Динамическое перераспределение адресов: DHCP позволяет освобождать и повторно использовать IP-адреса, когда устройство отключается от сети или больше не нуждается в адресе. Это позволяет более эффективно использовать доступные адреса

**Опции DHCP**

Помимо IP-адреса, DHCP также может сообщать клиенту дополнительные параметры, необходимые для нормальной работы в сети. Эти параметры называются опциями DHCP. Список стандартных опций можно найти в RFC 2132.

Некоторыми из наиболее часто используемых опций являются:

IP-адрес маршрутизатора по умолчанию;

маска подсети;

адреса серверов DNS;

имя домена DNS.

Некоторые поставщики программного обеспечения могут определять собственные, дополнительные опции DHCP.

**Информация DHCP**

Сообщение информации DHCP (DHCPINFORM) предназначено для определения дополнительных параметров TCP/IP (например, адреса маршрутизатора по умолчанию, DNS-серверов и т. п.) теми клиентами, которым не нужен динамический IP-адрес (то есть адрес которых настроен вручную). Серверы отвечают на такой запрос сообщением подтверждения (DHCPACK) без выделения IP-адреса.

**Реализации**

Компания Microsoft впервые включила сервер DHCP в поставку серверной версии Windows NT 3.5, выпущенной в 1994 году. Начиная с Windows 2000 Server реализация DHCP-сервера от Microsoft позволяет динамически обновлять записи DNS, что используется в Active Directory.

Internet Systems Consortium выпустил первую версию ISC DHCP Server (для Unix-подобных систем) 6 декабря 1997 года. 22 июня 1999 года вышла версия 2.0, более точно соответствующая стандарту.

Компания Cisco включила сервер DHCP в Cisco IOS 12.0 в феврале 1999 года. Sun добавила DHCP-сервер в Solaris 8 в июле 2001 года.

Тем не менее, как частенько бывает в сетевой индустрии, механизмы DHCP уже реализованы в продуктах ряда производителей. К счастью, любые изменения в алгоритмах его работы легко учесть на программном уровне, так что, приобретая серверное или клиентское программное обеспечение определенной компании, можно не опасаться заточения в неприступную крепость патентованных решений. Скорее, следует обратить внимание на то, насколько удачно конкретная реализация DHCP вписывается в имеющуюся вычислительную среду и взаимодействует с другими сетевыми службами, в частности с DNS. Публикуемые в этом номере результаты сравнительных испытаний DNS- и DHCP-серверов (см. статью <Игра в имена>) способны стать неплохим подспорьем для пользователя.

# Стандарты сообщений Internet.

RFC 5322 — Формат сообщений Internet

Этот документ задает формат сообщений Internet (IMF) — синтаксис текстовых сообщений, передаваемых между пользователями компьютеров через систему электронной почты.

В этом документе задан синтаксис только для текстовых сообщений.

В контексте электронной почты сообщения представляются состоящими из конверта и тела (содержимого) письма. Конверт содержит информацию, требуемую для [передачи и доставки сообщения](http://rfc2.ru/5321.rfc). Тело письма является объектом, который доставляется адресату. Данная спецификация применима только к формату и части семантики содержимого писем. Документ не включает спецификации данных, включаемых в конверт сообщения.

Однако некоторые системы обработки сообщений могут использовать информацию из тела письма для создания конверта. Одной из задач настоящей спецификации является упрощение извлечения такой информации программными средствами.

Назначением этой спецификации является задание формата содержимого писем, передаваемых между системами.

**Лексический анализ сообщений**

**Общее описание**

На базовом уровне сообщение представляет собой последовательность символов. Сообщения, соответствующие данной спецификации, включают символы с десятичными кодами от 1 до 127, интерпретируемые в соответствии с кодировкой US-ASCII. Для краткости в этом документе такие символы будут называться просто «символами US-ASCII».

Символы сообщения делятся на строки. Строка представляет собой последовательность символов, завершающуюся кодами возврата каретки и перевода строки — в конце строки помещается символ возврата каретки (CR, десятичный код ASCII — 13), непосредственно за которым следует символ перевода строки (LF, десятичный код ASCII - 10). В данном документе такая последовательность обозначается «CRLF».

Сообщение состоит из полей заголовков (совокупность этих полей называют разделом заголовков сообщения), за которыми может следовать тело сообщения. Раздел заголовков представляет собой последовательность символьных строк, синтаксис которых описан в данной спецификации. Тело сообщения представляет собой последовательность символов, которая следует после раздела заголовков и отделена от него пустой строкой (строкой, содержащей только CRLF).

**Предельные размеры строк**

Данная спецификация вносит два ограничения на число символов в строке. Строка должна содержать не более 998 символов; следует использовать строки размером не более 78, без учета CRLF.

Ограничение в 998 обусловлено возможностями множества реализаций в части передачи, приема и хранения, которые не позволяют работать с сообщениями IMF, содержащими строки размером более 998 символов. Системы приема могут из соображений отказоустойчивости отказываться от ограничения на размер строк. Однако существует множество реализаций, которые не принимают сообщения со строками размером более 1000 (включая CR и LF в конце строки), — разработчикам важно принимать этот факт во внимание.

Рекомендация использовать строки размером не более 78 обусловлена параметрами пользовательского интерфейса многих реализаций, которые могут отсекать лишние символы или неаккуратно переносить слова при наличии в строке более 78 символов, несмотря на то, что такие реализации не соответствуют требованиям данной спецификации и приводят к потере символов. Однако наличие такого ограничения не запрещает реализациям корректно отображать сообщения со строками произвольной длины для повышения уровня отказоустойчивости.

**Поля заголовков**

Поля заголовков представляют собой строки, начинающиеся с имени поля, за которым следует двоеточие (":"), содержимое поля и знак завершения строки CRLF. Имя поля должно состоять только из печатаемых символов US-ASCII (т. е., символов с кодами от 33 до 126, включительно), исключая двоеточие. Значение поля может включать печатаемые символы US-ASCII, символы пробела (SP, код ASCII - 32) и горизонтальной табуляции (HTAB, код ASCII — 9), которые вместе называют также пробельными символами. В значение поля недопустимо включать символы CR и LF, за исключением их использования в «фальцованных» и «нефальцованных» полях.

**Бесструктурные поля заголовков**

Некоторые поля заголовков в этой спецификации определены просто как неструктурированные (эти поля содержат произвольный набор печатаемых символов US-ASCII и пробельных символов) без дополнительных ограничений. Такие поля будем называть бесструктурными. Семантически бесструктурное поле трактуется просто как строка символов без дополнительной обработки.

**Структурированные поля заголовков**

Синтаксис некоторых полей в данной спецификации вносит дополнительные ограничения по сравнению с описанными выше бесструктурными полями. Такие поля называются структурированными. Структурированное поле представляет собой последовательность лексем. Многие из таких лексем (в соответствии с их синтаксисом) допускают включение комментариев в начале или в конце лексемы, а также пробельных символов, которые могут использоваться для фальцовки.

**Длинные поля заголовков**

Каждое поле заголовка логически представляет собой строку символов, состоящую из имени поля, двоеточия и тела (значения) поля. Однако для удобства и с учетом ограничения размеров строки (998/78 символов), значение поля может быть разбито на несколько строк; это называется «фальцовкой» (folding). Общим правило заключается в том, что данная спецификация разрешает включение последовательности CRLF (новая строка) перед любыми пробельными символами.

Например, поле заголовка

Subject: This is a test

можно записать в форме

Subject: This

is a test

Процесс преобразования фальцованного многострочного представления поля в обычное однострочное называется расфальцовкой (unfolding) и выполняется путем простого удаления всех последовательностей CRLF, непосредственно за которыми следуют пробельные символы (WSP). Каждое поле заголовка для дальнейшего синтаксического и семантического анализа следует трактовать в его нефальцованном представлении. На нефальцованные поля заголовков не накладываются ограничения по размеру и они, следовательно, могут иметь любую длину.

**Тело письма**

Тело сообщения представляет собой простые строки символов US-ASCII. Для содержимого сообщения существует только два типа ограничений:

* символы CR и LF должны использоваться только совместно, как CRLF; недопустимо использование этих символов в теле сообщения по-отдельности;
* строки символов должны быть не длиннее 998 символов, следует ограничивать размер строк 78 символами (без учета CRLF).

Символы CR и LF обозначают управляющие символы, используемые для управления переводом строк и перемещения курсора в текстовых файлах и протоколах.

CR (Carriage Return) - это управляющий символ, обозначаемый кодом ASCII 13 или шестнадцатеричным значением 0x0D. Символ CR указывает на перенос каретки (возврат курсора) в начало строки. Когда встречается символ CR, курсор перемещается в начало текущей строки, заменяя любой текст, который может быть уже напечатан на этой строке.

LF (Line Feed) - это управляющий символ, обозначаемый кодом ASCII 10 или шестнадцатеричным значением 0x0A. Символ LF указывает на перевод строки, перемещение курсора на новую строку. Когда встречается символ LF, курсор перемещается на следующую строку, обычно в позицию, непосредственно под предыдущей строкой.

**Синтаксис**

**Введение**

Описанный в этом разделе синтаксис задает корректный формат почтовых сообщений Internet. Сообщения, соответствующие данной спецификации.

Для определяемых выражений дается краткое описание синтаксиса и применения, после чего приводится синтаксис в формате ABNF и семантический анализ. Часть примитивов, используемых в документе, не определена здесь и заимствована из основных правил, приведенных в документе. К таким примитивам относятся: CR, LF, CRLF, HTAB, SP, WSP, DQUOTE, DIGIT, ALPHA и VCHAR.

В некоторых определениях будут встречаться элементы, имена которых начинаются с префикса "obs-". Такие элементы обозначают лексемы, относящиеся к устаревшему синтаксису. Во всех случаях такие конструкции при генерации корректных почтовых сообщений Internet игнорируются и их недопустимо использовать в качестве компонент сообщения. Однако при интерпретации сообщений такие лексемы должны рассматриваться как синтаксически корректные.

**Лексемы**

Разберем правила, используемые для определения базового лексического анализатора, который представляет лексемы анализаторам верхнего уровня. Разберем лексемы, используемые в структурированных полях заголовков.

**Квотирование символов**

Некоторые символы имеют специальное значение (например, используются в качестве границ лексем). Для использования таких символов в общепринятом смысле служит механизм квотирования (добавления «кавычек»).

quoted-pair = ("\" (VCHAR / WSP)) / obs-qp

При появлении любой пары с квотированием (quoted-pair) она интерпретируется как отдельный символ. Т. е., символ \, являющийся частью пары с квотированием, становится семантически «невидимым».

**Пробелы для фальцовки и комментарии**

Пробельные символы, включая и те, которые служат для фальцовки, могут появляться между разными элементами в теле полей заголовков. Строки символов, трактуемые, как комментарии, также могут включаться в тело структурированных полей заголовков с заключением их в скобки.

Строки символов, заключенные в скобки, рассматриваются, как комментарии, если они не являются частью строки с квотированием. Комментарии могут быть вложенными.

В этой спецификации есть несколько мест, где комментарии и FWS могут вставляться свободно. Для согласования с таким синтаксисом определена дополнительная лексема CFWS, показывающая возможность включения комментариев и/или FWS. Однако в тех случаях, где в данной спецификации разрешается CFWS, недопустимо использовать их так, чтобы фальцованное поле заголовка включало только символы WSP и ничего другого.

FWS = ([\*WSP CRLF] 1\*WSP) / obs-FWS

; пробельные символы для фальцовки

ctext = %d33-39 / ; печатаемые символы US-ASCII,

%d42-91 / ; не включая

%d93-126 / ; "(", ")" и "\"

obs-ctext

ccontent = ctext / quoted-pair / comment

comment = "(" \*([FWS] ccontent) [FWS] ")"

CFWS = (1\*([FWS] comment) [FWS]) / FWS

Комментарии обычно используются в теле структурированных полей для обеспечения некой дополнительной информации для человека. Поскольку в комментариях могут содержаться символы FWS, это позволяет выполнять фальцовку внутри комментария. Отметим также, что возможность использования в комментариях пар с квотированием, позволяет включать в комментарии скобки и символы обратной дробной черты (\), если они задаются в форме пары с квотированием. Семантически внешние скобки не являются частью комментария — комментарием является то, что заключено в эти скобки. Как было отмечено выше, символы «\» в парах с квотированием и последовательности CRLF внутри FWS внутри комментариев являются семантически невидимыми и, следовательно, не являются частью комментария.

FWS в качестве комментария (CFWS) между лексемами в структурированном поле заголовка семантически интерпретируется как один символ пробела.

**Атом**

Некоторые конструкции в теле структурированных полей заголовков представляют собой просто строки некоторых базовых символов. Такие конструкции называют атомами.

В некоторых структурированных полях заголовков допускается включение точки («.», код ASCII - 46) в atext. Для таких конструкций определена дополнительная лексема «атом с точкой» (dot-atom).

atext = ALPHA / DIGIT / ; Печатаемые символы US-ASCII,

"!" / "#" / ; не включая специальных символов.

"$" / "%" / ; Используются для атомов.

"&" / "'" /

"\*" / "+" /

"-" / "/" /

"=" / "?" /

"^" / "\_" /

"`" / "{" /

"|" / "}" /

"~"

atom = [CFWS] 1\*atext [CFWS]

dot-atom-text = 1\*atext \*("." 1\*atext)

dot-atom = [CFWS] dot-atom-text [CFWS]

specials = "(" / ")" / ; Специальные символы, которые не

"<" / ">" / ; появляются в atext

"[" / "]" /

":" / ";" /

"@" / "\" /

"," / "." /

DQUOTE

Лексемы atom и dot-atom интерпретируются, как единый элемент, включающий строку символов. Семантически дополнительные комментарии и FWS, окружающие остальные символы, не являются частью атома — атом представляет собой только символы atext (или atext и «.» для dot-atom).

**Строки в кавычках**

Строки, включающие символы, недопустимые для использования в атомах, могут быть представлены с использованием двойных кавычек (DQUOTE, код ASCII - 34), окружающих такие символы.

qtext = %d33 / ; Печатаемые символы US-ASCII,

%d35-91 / ; не включая "\"

%d93-126 / ; и символа кавычек

obs-qtext

qcontent = qtext / quoted-pair

quoted-string = [CFWS]

DQUOTE \*([FWS] qcontent) [FWS] DQUOTE

[CFWS]

Строка в кавычках трактуется как единый элемент. Т. е., семантически строка в кавычках эквивалентна атому. Поскольку строки в кавычках могут содержать FWS, в них возможна фальцовка. Отметим также, что благодаря возможности использования внутри кавычек пар с квотированием, строки в кавычках могут содержать также символы кавычек и обратной дробной черты, если они представлены в форме пар с квотированием.

Семантически, ни опциональные CFWS за пределами кавычек, ни сами символы кавычек не являются частью строки в кавычках — к такой строке относятся только символы, расположенные между кавычками. Как было отмечено выше, символ «\» в паре с квотированием или CRLF в FWS/CFWS, включенные в строку в кавычках, семантически невидимы и, следовательно, не являются частью строки в кавычках.

**Прочие лексемы**

Определены и дополнительных лексемы: слово (word) и фраза (phrase) для комбинаций атомов и/или строк в кавычках и неструктурированный текст (unstructured) для использования в бесструктурных полях заголовков и в некоторых местах структурированных полей.

word = atom / quoted-string

phrase = 1\*word / obs-phrase

unstructured = (\*([FWS] VCHAR) \*WSP) / obs-unstruct

**Задание даты и времен**

Значения даты и времени появляются в нескольких полях заголовка. В этом параграфе определяет синтаксис для задания даты и времени. Хотя спецификация даты/времени допускает включение пробельных символов фальцовки, рекомендуется использовать один пробел в каждом случае включения FWS (так, где это требуется или допускается); некоторые старые реализации неспособны корректно интерпретировать пробельные символы фальцовки.

date-time = [ day-of-week "," ] date time [CFWS]

day-of-week = ([FWS] day-name) / obs-day-of-week

day-name = "Mon" / "Tue" / "Wed" / "Thu" /

"Fri" / "Sat" / "Sun"

date = day month year

day = ([FWS] 1\*2DIGIT FWS) / obs-day

month = "Jan" / "Feb" / "Mar" / "Apr" /

"May" / "Jun" / "Jul" / "Aug" /

"Sep" / "Oct" / "Nov" / "Dec"

year = (FWS 4\*DIGIT FWS) / obs-year

time = time-of-day zone

time-of-day = hour ":" minute [ ":" second ]

hour = 2DIGIT / obs-hour

minute = 2DIGIT / obs-minute

second = 2DIGIT / obs-second

zone = (FWS ( "+" / "-" ) 4DIGIT) / obs-zone

Дата (day) показывает порядковый номер дня в месяце. Год (year) может принимать любое значение, не ранее 1900.

Время суток (time-of-day) показывает число часов, минут и (опционально) секунд, прошедщих с полуночи указанной даты.

Значения date и time-of-day следует указывать по местному времени.

Часовой пояс (zone) показывает смещение относительно универсального времени (UTC, ранее использовался термин GMT) значений местного времени, указанного в date и time-of-day. Знаки «+» и «-» показывают направление смещения — вперед (т. е., к востоку) или назад (к западу) от универсального времени. Первые две цифры показывают разницу с универсальным временем в часах, а две последних — дополнителюную разницу в минутах. Следовательно, +hhmm означает смещение на +(hh \* 60 + mm) минут от универсального времени, а -hhmm — на -(hh \* 60 + mm) минут. Для индикации часового пояса, время которого совпадает с универсальным, следует использовать форму +0000.

Значение date-time должно быть семантически корректным. Т. е., значение day-of-week (при его наличии) должно содержать день недели, числовое значение day-of-month должно находиться в диапазоне от 1 до числа дней в соответствующем месяце (указанного года), значение time-of-day должно находиться в диапазоне от 00:00:00 до 23:59:60 (число секунд, допустимое для смены суток; см. [RFC1305]), а две последних цифры поля zone должны иметь значение от 00 до 59.

**Задание адреса**

Адреса появляются в нескольких полях заголовка для индикации отправителей и получателей сообщения. Адрес может задавать персональный почтовый ящик или группу почтовых ящиков.

address = mailbox / group

mailbox = name-addr / addr-spec

name-addr = [display-name] angle-addr

angle-addr = [CFWS] "<" addr-spec ">" [CFWS] /

obs-angle-addr

group = display-name ":" [group-list] ";" [CFWS]

display-name = phrase

mailbox-list = (mailbox \*("," mailbox)) / obs-mbox-list

address-list = (address \*("," address)) / obs-addr-list

group-list = mailbox-list / CFWS / obs-group-list

Почтовый ящик принимает почту. Это концептуальный объект, который не обязательно связан с файловым хранилищем. Например, некоторые сайты могут выводить почтовые сообщения на принтер или передавать их на специальные адресуемые устройства вывода.

Обычно почтовый ящик состоит из двух частей: (1) необязательное отображаемое имя, которое идентифицирует имя получателя (человека или системы) и может выводиться пользователю в почтовых программах и (2) поле addr-spec, заключенное в угловые скобки («<» и «>»). Существует дополнительная форма указания почтового ящика, в которой имя получателя отсутствует, а addr-spec указывается без угловых скобок.

Когда желательно трактовать несколько почтовых ящиков, как один объект (например, список рассылки), может использоваться групповая конструкция. Такая конструкция позволяет отправителю указать именованную группу получателей. Это обеспечивается путем создания для группы отображаемого имени, за которым следует список разделенных запятыми почтовых ящиков произвольного (включая 0) размера, завершающийся точкой с запятой (;). Поскольку список почтовых ящиков может быть пустым, использование групповой конструкции также обеспечивает простой способ взаимодействия с адресатами, когда сообщение передается одной или множеству именованных групп адресатов без указания иднивидуальных почтовых ящиков этих адресатов.

**Задание addr-spec**

Поле addr-spec представляет собой специфический для Internet идентификатор, содержащий локально интерпретируемую строку, за которой следует символ @ (код ASCII - 64) и доменное имя Internet. Эта локально интерпретируемая строка представляет собой строку в кавычках или атом с точкой. Если строка может быть представлена атомом с точкной (т. е., не содержит символов, кроме atext и завершающей точки или, за которой следуют символы atext), следует использовать форму dot-atom и не следует применять форму quoted-string. Комментарии и пробельные символы фальцовки не следует помещать рядом с @ в поле addr-spec.

addr-spec = local-part "@" domain

local-part = dot-atom / quoted-string / obs-local-part

domain = dot-atom / domain-literal / obs-domain

domain-literal = [CFWS] "[" \*([FWS] dtext) [FWS] "]" [CFWS]

dtext = %d33-90 / ; Печатаемые символы US-ASCII,

%d94-126 / ; не включая

obs-dtext ; "[", "]", or "\"

Доменная часть идентифицирует точку, в которую доставляется почта. В форме dot-atom она интерпретируется, как доменное имя Internet. В форме domain-literal доменная часть интерпретируется, как «дословный» Internet-адрес конкретного хоста. В обоих случаях использование адресации и транспортировка почты на конкретный хост определяются отдельными документами.

Локальная часть (local-part) зависит от домена. В адресах она просто интерпретируется на конкретном хосте, как имя почтового ящика.

**Общий синтаксис сообщений**

Сообщение состоит из полей заголовка, за которыми может следовать тело сообщения. Строки сообщения должны иметь размер не более 998 символов, не считая CRLF, но рекомендуется использовать строки не длиннее 78 без учета CRLF. Хотя в теле сообщения могут появляться любые символы, перечисленные в правиле text, использование управляющих символов US-ASCII (коды 1-8, 11, 12, 14-31) не является хорошим тоном, поскольку их интерпретация при отображении на приемной стороне не гарантируется.

message = (fields / obs-fields)

[CRLF body]

body = (\*(\*998text CRLF) \*998text) / obs-body

text = %d1-9 / ; Символы, за исключением

%d11 / ; CR и LF

%d12 /

%d14-127

Тело сообщения представляет собой просто набор строк текста, интерпретируемых в соответствии с настоящей спецификацией.

**Определения полей**

Все поля заголовка имеют одинаковую синтаксическую структуру: имя поля, за которым следует двоеточие (:) и тело (значение) поля.

Важно подчеркнуть, что порядок следования полей заголовка не гарантируется. Поля заголовков могут появляться в произвольном порядке. Более того, известно, что порядок полей заголовков может изменяться при передаче сообщений через Internet. Однако в соответствиии с данной спецификацией порядок полей заголовка не следует менять при передаче или преобразовании сообщений. Более важно отметить, что порядок трассировочных полей и полей resent изменять недопустимо и следует сохранять эти поля в блоках, добавляемых в начало сообщения (prepend).

Обязательными полями заголовка являются только поле даты и поле адреса отправителя сообщения. Все остальные поля являются синтаксически опциональными.

fields = \*(trace

\*optional-field /

\*(resent-date /

resent-from /

resent-sender /

resent-to /

resent-cc /

resent-bcc /

resent-msg-id))

\*(orig-date /

from /

sender /

reply-to /

to /

cc /

bcc /

message-id /

in-reply-to /

references /

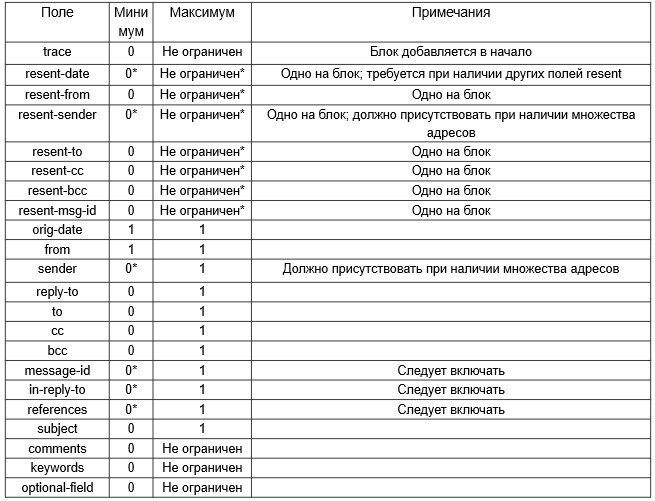
subject /

comments /

keywords /

optional-field)

Приведенная ниже таблица показывает минимальное и максимальное число полей каждого типа в разделе заголовков сообщения, а также ограничения на использование полей. Звездочка (\*) после в колонке минимального или максимального числа полей говорит о наличии дополнительных ограничений, указанных в колонке «Примечания».



Точная интерпретация каждого поля рассмотрена в последующих параграфах.

**Поле даты создания**

Поле даты создания состоит из имени поля «Date», за которым следуют значения даты и времени.

orig-date = "Date:" date-time CRLF

Поле указывает дату и время, когда отправитель, указанный в сообщении, завершил подготовку сообщения к передаче системе доставки. Например, это может быть момент нажатия пользователем кнопки «send» или «submit» в почтовой программе. В любом случае, это поле не предназначено для указания времени реальной передачи сообщения, а содержит значение времени, когда человек или другой создатель сообщения завершил подготовку сообщения к передаче (например, пользователь портативного компьютера, не подключенного к сети, мог просто поместить сообщение в очередь для доставки; поле даты создания предназначено для хранения даты и времени в момент постановки письма в очередь, а не момента подключения пользователя к сети).

**Поля источника**

В число полей источника сообщения входят from и sender (когда применим), а также опционально — reply-to. Поле from содержит имя «From» и разделенный запятыми список из одного или нескольких имен почтовых ящиков. Эсли это поле содержит более одного адреса почтового ящика в списке, в сообщение должно включаться поле «Sender» с указанием одного почтового ящика. Дополнительно может также включаться поле «Reply-To», содержащее список разделенных запятыми почтовых ящиков (не менее одного).

from = "From:" mailbox-list CRLF

sender = "Sender:" mailbox CRLF

reply-to = "Reply-To:" address-list CRLF

Поле источника показывает почтовый ящик(и) источника сообщения. Поле «From:» задает автора (авторов) сообщения, т. е., почтовый ящик(и) человека или системы, ответственных за создание данного сообщения. Поле «Sender:» указывает почтовый ящик агента, ответственного за реальную передачу сообщения. Например, если секретарь отправил письмо от имени своего руководителя, почтовый ящик секретаря будет указан в поле «Sender:», а адрес действительного автора сообщения — в поле «From:». Если источник сообщения может быть задан единственным почтовым ящиком (автор и отправитель совпадают), поле «Sender:» использовать не следует. В остальных случаях это поле следует включать в заголовок.

Поля источника также обеспечивают информацию, требуемую для ответа на сообщение. При наличии поля «Reply-To:» оно указывает адрес(а), по которому отправитель сообщения предлагает направлять ответ. В отсутствие поля «Reply-To:» ответ следует по умолчанию отправлять по адресу (адресам), указанному в поле «From:», если автором ответа явно не указан иной адрес.

В любом случае в поле «From:» не следует указывать адрес, не имеющий отношения к автору письма. Формирование адресов для отправки ответов на сообщения дополнительно рассматривается в параграфе 3.6.3.

**Поля адресов получателей**

К полям получателей относятся три однотипных поля, , каждое из которых содержит имя («To», «Cc» или «Bcc») и список из одного или множества разделенных запятыми адресов (почтовых ящиков или групп).

to = "To:" address-list CRLF

cc = "Cc:" address-list CRLF

bcc = "Bcc:" [address-list / CFWS] CRLF

Эти поля задают получателей сообщения. Каждое из полей может содержать один или множество адресов, идентифицирующих получателей сообщения. Различие между полями заключается только в способе их использования.

Поле «To:» содержит адрес(а) основного получателя (получателей) сообщения.

Поле «Cc:» (от «Carbon Copy» — копия по аналогии с печатью под копирку на пишущей машинке) содержит адреса других лиц, которым направляется это сообщение, хотя содержимое сообщения может быть не адресовано им напрямую.

Поле «Bcc:» (от «Blind Carbon Copy» — «слепая копия» по аналогии с последним экземпляром при печати под копирку) содержит адреса получателей, которые не будут показаны другим получателям этого сообщения. Существует три варианта использования поля «Bcc:». В первом случае, когда сообщение с полем «Bcc:» готовится к отправке, строка «Bcc:» удаляется из него, хотя все адресаты (включая указанных в поле «Bcc:») получают копию этого письма. Во втором варианте получатели, указанные в полях «To:» и «Cc:», получат сообщение с удаленной строкой «Bcc:», но адресаты, указанные в поле «Bcc:» получат копию письма с сохраненной строкой «Bcc:» (при указании в поле «Bcc:» множества адресатов некоторые реализации на самом деле шлют каждому адресату из поля «Bcc:» отдельную копию письма, содержащую в этом поле только адрес данного получателя). И, наконец, когда поле «Bcc:» не содержит адресов, это поле может включаться во все копии сообщения для адресатов, заданных другими полями. Выбор конкретного варианта использования поля «Bcc:» определяется реализацией, но при этом следует принимать во внимание информацию, приведенную ниже в разделе «Вопросы безопасности».

Когда сообщение является ответом на другое письмо, почтовые ящики авторов исходного письма (значение поля «From:») или почтовые ящики, указанные в поле «Reply-To:» (если оно есть), могут появляться в поле «To:» ответного письма, поскольку эти адресаты являются явными отправителями исходного письма. Если сообщение передается в ответ на письмо, имеющее поля получателей, зачастую бывает полезно отправить копию ответа всем получателям сообщения в дополнение к отправке письма автору. При создании такого отклика адреса из полей «To:» и «Cc:» исходного сообщения могут появляться в поле «Cc:» ответного письма, поскольку эти адресаты были открытоо указаны в числе получателей копии исходного письма. Если в исходном письме присутствует поле «Bcc:», адреса из этого поля могут появляться в поле «Bcc:» ответа на письмо, но их не следует включать в поле «To:» или «Cc:».

**Поля идентификации**

Хотя в таблице поле «Message-ID:» указано, как необязательное, это поле следует включать в каждое сообщение. Более того, в ответные сообщения следует включать поля «In-Reply-To:» и «References:» в соответствии с приведенным ниже описанием.

Поле «Message-ID:» содержит один уникальный идентификатор сообщения. Каждое из полей «References:» и «In-Reply-To:» содержит один или множество уникальных идентификаторов сообщений, опционально разделенных символами CFWS.

Синтаксис идентификатора сообщения (msg-id) является ограниченной версией конструкции addr-spec, заключенной в угловые скобки «<» и «>». В отличие от addr-spec, этот синтаксис разрешает форму dot-atom-text слева от символа @ и не имеет сиволов CFWS где-либо внутри идентификатора.

message-id = "Message-ID:" msg-id CRLF

in-reply-to = "In-Reply-To:" 1\*msg-id CRLF

references = "References:" 1\*msg-id CRLF

msg-id = [CFWS] "<" id-left "@" id-right ">" [CFWS]

id-left = dot-atom-text / obs-id-left

id-right = dot-atom-text / no-fold-literal / obs-id-right

no-fold-literal = "[" \*dtext "]"

Поле «Message-ID:» обеспечивает уникальный идентификатор сообщения, который указывает на конкретный вариант конкретного письма. Уникальность идентификатора гарантируется хостом, создающим сообщение. Этот идентификатор предназначен для машинной обработки и не имеет большого смысла для человека. Идентификатор сообщения относится только к одному варианту конкретного письма — для следующего экземпляра сообщения будет создан другой идентификатор.

Поля «In-Reply-To:» и «References:» используются при создании ответных сообщений. Они содержат идентификатор исходного сообщения и идентификаторы других сообщений (например, в случае ответа на письмо, которое само является ответом на другое письмо). Поле «In-Reply-To:» может использоваться для идентификации сообщения (сообщений), на которое отвечает данное сообщение, тогда как поле «References:» может служить для идентификации «ветви» разговора.

При создании ответа на сообщение поля «In-Reply-To:» и «References:» в ответном письме строятся в соответствии с приведенным ниже описанием.

Поле «In-Reply-To:» ответного письма будет содержать информацию из поля «Message-ID:» исходного («родительского») сообщения. Если ответ дается на несколько писем сразу, поле «In-Reply-To:» будет содержать информацию из полей «Message-ID:» всех родительских сообщений. Если в родительских сообщениях нет полей «Message-ID:», в ответе не будет содержаться поля «In-Reply-To:».

Поле «References:» будет включать содержимое поля «References:» из родительского письма (если там это поле присутствует), вслед за которым будет включено родительское поле «Message-ID:» (при его наличии). Если в родительском сообщении нет поля «References:», но имеется поле «In-Reply-To:» с единственным идентификатором, в ответе поле «References:» будет включать содержимое родительского поля «In-Reply-To:», за которым будет следовать содержимое родительского поля «Message-ID:» (при его наличии). Если в родительском сообщении нет полей «References:», «In-Reply-To:» и «Message-ID:», в ответе не будет поля «References:».

Идентификатор сообщения (msg-id) должен быть уникальным в глобальном масштабе. Эту уникальность должен обеспечивать генератор идентификаторов сообщений. Существует несколько алгоритмов, обеспечивающих решение этой задачи. Поскольку синтаксис msg-id подобен сиснтаксису addr-spec, хорошим методом является включение в идентификатор доменного имени (или полного адреса IP) хоста, на котором создается идентификатор сообщения, справа от знака @ (доменные имена и адреса IP обычно уникальны) и включение текущего абсолютного значения даты и времени в комбинации с неким другим уникальным в данный момент (возможно последовательным) идентификатором, доступным в системе (например, идентификатором процесса), слева от @. Хотя будут работать и другие алгоритмы, рекомендуется использовать в правой части некий идентификатор домена (имя хоста или нечто иное), чтобы генератор идентификатора сообщения мог гарантировать уникальность левой части идентификатора в масштабе данного домена.

Семантически угловые скобки не являются частью msg-id; идентификатором является строка символов между скобками.

**Информационные поля**

Информационные поля являются необязательными. Поля «Subject:» и «Comments:» являются бесструктурными и, следовательно, могут содержать текст или пробельные символы для фальцовки. Поле «Keywords:» содержит список из одного или более разделенных запятыми слов или строк в кавычках.

subject = "Subject:" unstructured CRLF

comments = "Comments:" unstructured CRLF

keywords = "Keywords:" phrase \*("," phrase) CRLF

Эти три поля предназначены только для включения понятной человеку информации о сообщении. Поле «Subject:» используется наиболее широко и содержит короткую строку, описывающую тему сообщения. При использовании в ответах это поле может начинаться с префикса «Re: » (сокращение от латинского «in re», означающего «по вопросу ..»), за которым следует содержимое поля «Subject:» исходного письма. В таких случаях следует использовать префикс «Re: » только один раз, поскольку использование другого текста или включение нескольких префиксов может приводить к нежелательным последствиям. Поле «Comments:» содержит произвольную информацию, дополняющую текст в теле письма. Поле «Keywords:» содержит список разделенных запятыми слов и фраз " field contains a comma-separated list of important words and phrases that might be useful for the recipient.

**Поля пересылки**

Поля пересылки следует добавлять во все сообщения, которые повторно вводятся пользователем в транспортную систему. Каждый раз, когда это происходит, следует добавлять отдельный набор полей пересылки. Все поля пересылки, соответствующие отдельному факту пересылки, следует группировать. Каждый набор полей пересылки помещается в начало сообщения; т. е., самый свежий набор таких полей появляется в сообщении первым. При добавлении полей пересылки никакие другие поля сообщения не меняются.

Каждое поле пересылки синтаксически соответствует определенному полю при обычной передаче. Например, поле «Resent-Date:» соответствует полю «Date:», а «Resent-To:» — полю «To:».

При использовании полей пересылки должны передаваться поля «Resent-From:» и «Resent-Date:». Следует передавать также поле «Resent-Message-ID:». Поле «Resent-Sender:» не следует использовать, если это поле будет идентично полю «Resent-From:».

resent-date = "Resent-Date:" date-time CRLF

resent-from = "Resent-From:" mailbox-list CRLF

resent-sender = "Resent-Sender:" mailbox CRLF

resent-to = "Resent-To:" address-list CRLF

resent-cc = "Resent-Cc:" address-list CRLF

resent-bcc = "Resent-Bcc:" [address-list / CFWS] CRLF

resent-msg-id = "Resent-Message-ID:" msg-id CRLF

Поля пересылки служат для идентификации сообщений, как повторно вводимых пользователем в транспортную систему. Цель использования этих полей заключается в предоставлении конечному адресату сообщения в таком виде, как будто оно было получено непосредственно от исходного отправителя с сохранением всех исходных полей. Каждый набор полей пересылки соответствует одному факту пересылки. Т. е., при многократной пересылке сообщения каждый из таких наборов описывает отдельный факт пересылки. Поля пересылки являются информационными. Эти поля недопустимо использовать в процессах автоматической генерации ответов и других операциях автоматизированной обработки сообщений.

Поля инициатора пересылки указывают почтовый ящик лиц(а) или систем(ы), переславших это сообщение. Как и для обычных полей инициатора существует две формы — простой вариант «Resent-From:», содержащий почтовый ящик выполняющего пересылку лица, и более сложный вариант, когда одно лицо (идентифицируется полем «Resent-Sender:») пересылает сообщение от имени одного или множества других лиц (идентифицируются полем «Resent-From:»).

Примечание. При ответе на пересланное сообщение исходные поля «From:», «Reply-To:», «Message-ID:» и др. используются в обычном порядке. Поля пересылки являются только информационными и недопустимо использовать их при ответе на сообщение.

Поле «Resent-Date:» указывает дату и время диспетчеризации сообщения при его пересылке. Подобно полю «Date:», оно содержит не дату и время реальной отправки сообщения, а дату и время постановки в очередь.

Функционально поля «Resent-To:», «Resent-Cc:» и «Resent-Bcc:» идентичны полям «To:», «Cc:» и «Bcc:», соответственно, за исключением того, что они указывают получателей пересланного сообщения, а не исходного.

Поле «Resent-Message-ID:» содержит уникальный идентификатор пересланного сообщения.

**Поля трассировки**

Поля трассировки представляют собой группу полей заголовка, включающую опциональное поле «Return-Path:» и одно или множество полей «Received:». Поле заголовка «Return-Path:содержит пару угловых скобок, в которые заключено необязательное значение addr-spec. Поле «Received:» содержит (возможно пустой) список маркеров, за которым следует точка с запятой (;) и значение даты и времени. Каждый маркер должен представлять собой элемент типа word, angle-addr, addr-spec или domain. Спецификации, описывающие использование полей трассировки (такие, как [RFC5321]), могут вносить дополнительные ограничения.

trace = [return]

1\*received

return = "Return-Path:" path CRLF

path = angle-addr / ([CFWS] "<" [CFWS] ">" [CFWS])

received = "Received:" \*received-token ";" date-time CRLF

received-token = word / angle-addr / addr-spec / domain

Полное описание использования полей трассировки в почте Internet содержится в [RFC5321]. В данной спецификации трассировочные поля рассматриваются исключительно в качестве информационных и любая формальная интерпретация этих полей выходит за рамки документа.

**Дополнительные поля**

В сообщениях могут появляться поля, не рассмотренные в этом документе. Такие поля должны соответствовать синтаксису optional-field. Этот синтаксис включает имя поля, состоящее из печатаемых символов US-ASCII без двоеточий и пробелов (SP), за которым следует двоеточие и произвольный текст, соответствующий синтаксису бесструктурных полей.

Недопустимо совпадение имен дополнительных полей с именами полей, указанными в данном документе.

optional-field = field-name ":" unstructured CRLF

field-name = 1\*ftext

ftext = %d33-57 / ; Печатаемые символы US-ASCII,

%d59-126 ; за исключением «:».

В настоящей спецификации дополнительные поля считаются неинтерпретируемыми.

**Вопросы безопасности**

Требуются определенные меры предосторожности при отображении сообщений на терминале или в программе эмуляции терминала. Многофункциональные терминалы могут реагировать на escape-последовательности и другие комбинации управляющих символов US-ASCII, что может вызывать неожиданные эффекты. В число таких эффектов может входить изменение клавиатурной раскладки и другие эффекты, способные приводить к нарушению работы и даже к повреждению данных. Управляющие символы могут вызывать (иногда программируемо) генерацию ответов на сообщения, что позволяет вводить команды от имени пользователя. Возможно также воздействие на подключенные к терминалу устройства (например, принтеры). Программы просмотра сообщений могут по своему усмотрению удалить опасные escape-последовательности из сообщения перед его выводом. Однако некоторые escape-последовательности могут быть нужны в сообщениях и поэтому не следует удалять escape-последовательности без разбора.

Передача отличных от текста объектов в сообщениях может приводить к возникновению дополнительных опасностей.

Многие реализации используют поле «Bcc:» (последний экземпляр), для доставки сообщений некоторым получателям без ведома других адресатов этого сообщения. Некорректная обработка полей «Bcc:» может привоодить к утечке конфиденциальной информации, что, в свою очередь, может снижать уровень безопасности за счет распространения информации о существовании определенных почтовых адресов. Например, при использовании первого метода, когда строка «Bcc:» удаляется из сообщения, скрытые получатели не имеют явного указания на то, что они были скрыты от других адресатов (за исключением того факта, что их адреса отсутствуют в заголовке сообщения). По этой причине кто-либо из скрытых получателей сообщения может отправить свой ответ всем показанным в заголовке получателям и непреднамеренно показать, что часть адресатов сообщения была скрыта. При использовании второго метода скрытые адресаты указываются в поле «Bcc:» отдельной копии сообщения. Если поле «Bcc:» содержит все скрытые адреса, получатели узнают о других скрытых адресатах. Даже при создании отдельного поля «Bcc:» для каждого скрытого адресата, реализациям следует с осторожностью относиться к обработке ответов на такие сообщения, во избежание непреднамеренного распространения информации о скрытых получателях сообщения другим адресатам.

Приложение A. Примеры сообщений

представлены различные примеры сообщений. Примеры предназначены для оказания помощи разработчикам при реализации данной спецификации, но их не следует воспринимать как нормативы. Хотя примеры в этом разделе подобраны аккуратно, при возникновении противоречий между примером и текстом предпочтение должно отдаваться спецификации, а не примеру.

Примеры сообщений отделены от текста строками «----», которые не являются частью сообщений.

Приложение A.1. Примеры адресации

В этом параграфе приведены примеры сообщений, которыми могут обмениваться два индивидуальных адресата.

**Приложение. Сообщение от одного лица другому с простой адресацией**

Этот пример можно назвать каноническим сообщением. Письмо имеет одного автора (John Doe), одного получателя (Mary Smith), тему, дату, идентификатор сообщения и текст в теле письма.

----

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

Если у John есть секретарь Michael, который на самом деле отправляет сообщение, автором которого является по-прежнему John и ответы на письмо должны направляться секретарю, следует использовать поле sender:

----

From: John Doe <jdoe@machine.example>

Sender: Michael Jones <mjones@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

**Приложение. Различные типы почтовых ящиков**

Это сообщение содержит множество адресов в полях получателей и использует иной формат адресов.

----

From: "Joe Q. Public" <john.q.public@example.com>

To: Mary Smith <mary@x.test>, jdoe@example.org, Who? <one@y.test>

Cc: <boss@nil.test>, "Giant; \"Big\" Box" <sysservices@example.net>

Date: Tue, 1 Jul 2003 10:52:37 +0200

Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.

----

Отметим, что отображаемые имена Joe Q. Public и Giant; "Big" Box требуется заключать в двойные кавычки, поскольку первое имя включает точку, а во втором содержится точка с запятой и символы двойных кавычек (в форме пар с квотированием). Отображаемое имя Who?, напротив, может использоваться без кавычек, поскольку в атомах допускается использование вопросительного знака. Отметим также, что в адресах jdoe@example.org и boss@nil.test не указаны связанные с ними отображаемые имена, а для адреса jdoe@example.org используется упрощенная форма без угловых скобок.

**Приложение. Групповые адреса**

----

From: Pete <pete@silly.example>

To: A Group:Ed Jones <c@a.test>,joe@where.test,John <jdoe@one.test>;

Cc: Undisclosed recipients:;

Date: Thu, 13 Feb 1969 23:32:54 -0330

Message-ID: <testabcd.1234@silly.example>

Testing.

----

В этом примере поле «To:» включает группу «A Group», в состав которой входят 3 адреса, а в поле «Cc:» указана пустая группы получателей Undisclosed recipients (нераскрытые получатели).

Приложение A.2. Ответные сообщения

Следующая группа из трех примеров показывает поток обмена сообщениями между John и Mary. Сначала John отправляет сообщение Mary, на которое Mary отвечает и John в ответ отправляет письмо Mary.

Отметим специально поля «Message-ID:», «References:» и «In-Reply-To:» в каждом сообщении.

----

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

При отправке ответов поле Subject зачастую сохраняется с добавлением префикса «Re: », как описано в параграфе 3.6.5.

----

From: Mary Smith <mary@example.net>

To: John Doe <jdoe@machine.example>

Reply-To: "Mary Smith: Personal Account" <smith@home.example>

Subject: Re: Saying Hello

Date: Fri, 21 Nov 1997 10:01:10 -0600

Message-ID: <3456@example.net>

In-Reply-To: <1234@local.machine.example>

References: <1234@local.machine.example>

This is a reply to your hello.

----

Отметим поле «Reply-To:» в приведенном выше сообщении. Когда John отвечает на приведенное выше письмо Mary, ответ следует направлять по адресу из поля «Reply-To:», а не по адресу из поля «From:».

----

To: "Mary Smith: Personal Account" <smith@home.example>

From: John Doe <jdoe@machine.example>

Subject: Re: Saying Hello

Date: Fri, 21 Nov 1997 11:00:00 -0600

Message-ID: <abcd.1234@local.machine.test>

In-Reply-To: <3456@example.net>

References: <1234@local.machine.example> <3456@example.net>

This is a reply to your reply.

----

Приложение A.3. Пересылка сообщений

Начнем с сообщения, которое будет использоваться в качестве примера несколько раз:

----

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

Mary, получив это сообщение, хочет отправить копию Jane так, что (a) сообщение выглядит, как отправленное John; (b) если Jane ответит на это сообщение, ответ должен быть отправлен John; (c) вся исходная информация, включая дату письма, изначално посланного Mary, идентификатор сообщения и исходный адрес сохраняется. В этом случае в начало сообщения добавляются поля пересылки:

----

Resent-From: Mary Smith <mary@example.net>

Resent-To: Jane Brown <j-brown@other.example>

Resent-Date: Mon, 24 Nov 1997 14:22:01 -0800

Resent-Message-ID: <78910@example.net>

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

Если Jane, в свою очередь, решит переслать сообщение еще кому-либо, она будет добавлять свои поля пересылки перед сообщением, показанным выше (для краткости этот вариант примера опущен).

Приложение A.4. Сообщения с полями трассировки

При пересылке сообщения через транспортную систему, как описано в [RFC5321], в начало сообщения добавляются трассировочные поля. Ниже приведен пример, показывающий, как могут выглядеть эти поля. Отметим, что первая строка «Received:» оказалась слишком длинной и в нее включены фальцовочные пробелы.

----

Received: from x.y.test

by example.net

via TCP

with ESMTP

id ABC12345

for <mary@example.net>; 21 Nov 1997 10:05:43 -0600

Received: from node.example by x.y.test; 21 Nov 1997 10:01:22 -0600

From: John Doe <jdoe@node.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.node.example>

This is a message just to say hello.

So, "Hello".

----

Приложение A.5. Пробелы, комментарии и другие странности

Пробеля, включая фальцовочные, и комментарии могут вставляться между разными элементами поля. Возьмем пример из параграфа A.1.3 и добавим в него пробелы и комментарии в полях заголовка.

----

From: Pete(A nice \) chap) <pete(his account)@silly.test(his host)>

To:A Group(Some people)

:Chris Jones <c@(Chris's host.)public.example>,

joe@example.org,

John <jdoe@one.test> (my dear friend); (the end of the group)

Cc:(Empty list)(start)Hidden recipients :(nobody(that I know)) ;

Date: Thu,

13

Feb

1969

23:32

-0330 (Newfoundland Time)

Message-ID: <testabcd.1234@silly.test>

Testing.

----

Приведенный пример выглядит странновато, но является совершенно допустимым. Отметим специально (1) комментарий в поле «From:» (включая скобку в паре с квотированием); (2) отсутствие пробела после двоеточия в поле «To:», а также комментарий и фальцовочные пробелы после имени группы, специальный символ (.) в комментарии поля с адресом Chris Jones и фальцовочные пробелы перед и после «joe@example.org,»; (3) множество вложенных комментариев в поле «Cc:», а также комментарий, следующий непосредственно за двоеточием после «Cc»; (4) фальцовочный пробел (но не комментарии, за исключением комментария в конце поля даты), а также отсутствие значения секунд; (5) пробелы после (но не внутри) идентификатора в поле «Message-ID:».

Приложение A.6. Устаревшие формы

Здесь приведены примеры устаревших синтаксических элементов (которые недопустимо создавать), описанных в разделе 4 данного документа.

**Приложение A.6.1. Устаревшая адресация**

Отметим в приведенном ниже примере отсутствие кавычек вокруг Joe Q. Public, маршрут в адресе Mary Smith, две запятых в поле «To:» и пробелы рябом с точкой в адресе jdoe.

----

From: Joe Q. Public <john.q.public@example.com>

To: Mary Smith <@node.test:mary@example.net>, , jdoe@test . example

Date: Tue, 1 Jul 2003 10:52:37 +0200

Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.

----

**Приложение A.6.2. Устаревшие даты**

В приведенном ниже сообщении используется устаревший формат даты, включая символьное представление часового пояса и 2-значное обозначение года. Отметим, что отсутствие дня недели не является спецификой устаревшего синтаксиса — это поле является опциональным и в современном синтаксисе.

----

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: 21 Nov 97 09:55:06 GMT

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

**Приложение A.6.3. Устаревшие пробелы и комментарии**

Наличие пробелов и комментариев в устаревшем синтаксисе разрешается в большем числе мест, нежели позволяет современный синтаксис. Однако фальцовочные строки, содержащие только пробелы, по прежнему допускаются.

----

From : John Doe <jdoe@machine(comment). example>

To : Mary Smith

\_\_

<mary@example.net>

Subject : Saying Hello

Date : Fri, 21 Nov 1997 09(comment): 55 : 06 -0600

Message-ID : <1234 @ local(blah) .machine .example>

This is a message just to say hello.

So, "Hello".

----

Особо отметим вторую строку поля «To:», начинающуюся с двух пробельных символов. (отметим, «\_\_» представляет пробелы). Следовательно, это является рассматриваемой частью фальцовки, как описано в параграфе 4.2. Комментарии и пробелы в адресах, датах и идентификаторах сообщений являются частью устаревшего синтаксиса.

# Протокол HTTP.

* **HTTP** (англ. *HyperText Transfer Protocol* — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов).

Основой HTTP является технология «клиент-сервер», то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

HTTP в настоящее время повсеместно используется во Всемирной паутине для получения информации с веб-сайтов.

Особенностью протокола **HTTP** является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым

Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов **HTTP** использует глобальные **URI**. HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами. Браузер, посылающий запросы, может отслеживать задержки ответов. Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

**Плюсы**

***Простота***

Протокол настолько прост в реализации, что позволяет с лёгкостью создавать клиентские приложения.

***Расширяемость***

Возможности протокола легко расширяются благодаря внедрению своих собственных заголовков, сохраняя совместимость с другими клиентами и серверами.

***Распространённость***

Как следствие, это обилие различной документации по протоколу на многих языках мира, включение удобных в использовании средств разработки в популярные IDE, поддержка протокола в качестве клиента многими программами и обширный выбор среди хостинговых компаний с серверами HTTP.

**Минусы**

* **Большой размер сообщений**

Использование текстового формата в протоколе порождает соответствующий недостаток: большой размер сообщений по сравнению с передачей двоичных данных. Для решения данной проблемы в протокол встроены средства для обеспечения кэширования на стороне клиента, а также средства компрессии передаваемого контента.

* **Отсутствие «навигации»**

Например, клиент не может явным образом запросить список доступных файлов, как в протоколе FTP. Предполагалось, что конечный пользователь уже знает URI необходимого ему документа, закачав который, он будет производить навигацию благодаря гиперссылкам. Это вполне нормально и удобно для человека, но затруднительно, когда стоят задачи автоматической обработки и анализа всех ресурсов сервера без участия человека. Решение этой проблемы лежит полностью на плечах разработчиков приложений, использующих данный протокол.

* **Нет поддержки распределённости**

В 1998 году W3C предложил альтернативный протокол HTTP-NG (англ. HTTP Next Generation) для полной замены устаревшего с акцентированием внимания именно на этой области. Идею его необходимости поддержали крупные специалисты по распределённым вычислениям, но данный протокол до сих пор находится на стадии разработки.

**Всё программное обеспечение для работы с протоколом HTTP разделяется на три больших категории:**

\* **Серверы** как основные поставщики услуг хранения и обработки информации (обработка запросов).

\* **Клиенты** — конечные потребители услуг сервера (отправка запроса).

\* **Прокси** для выполнения транспортных служб.

**Структура протокола**

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

1. **Стартовая строка** (англ. Starting line) — определяет тип сообщения;

2. **Заголовки** (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;

3. **Тело сообщения** (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Заголовки и тело сообщения могут отсутствовать, **но стартовая строка является обязательным** элементом, так как указывает на тип запроса/ответа

**Стартовые строки** различаются для запроса и ответа. **Строка запроса** выглядит так:

**GET URI** — для версии протокола 0.9.

**Метод URI HTTP/Версия** — для остальных версий.

Здесь:

\* **Метод (англ. Method)** — название запроса, одно слово заглавными буквами. В версии HTTP 0.9 использовался только метод GET, список запросов для версии 1.1 представлен ниже.

\* **URI** определяет путь к запрашиваемому документу.

\* **Версия (англ. Version)** — пара разделённых точкой арабских цифр. Например: 1.0.

Для запроса страницы, клиент должен передать строку:

**GET /net/index.html HTTP/1.0**

Стартовая строка ответа сервера имеет следующий формат:

**HTTP/Версия КодСостояния Пояснение**

Здесь:

\* **Версия** — пара разделённых точкой арабских цифр как в запросе.

\* **КодСостояния (англ. Status Code)** — три арабские цифры. По коду статуса определяется дальнейшее содержимое сообщения и поведение клиента.

\* **Пояснение (англ. Reason Phrase)** — текстовое короткое пояснение к коду ответа для пользователя. Никак не влияет на сообщение и является необязательным.

Например, на предыдущий наш запрос клиентом страницы сервер ответил строкой:

**HTTP/1.0 200 OK**

**Методы**

Метод HTTP (англ. HTTP Method) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. чувствительно к регистру.

Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented). Если серверу метод известен, но он не применим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed).

**Типы методов:**

**OPTIONS**: Метод OPTIONS используется для запроса информации о возможностях и настройках сервера или о доступных методах запросов для конкретного ресурса. Он позволяет клиенту определить, какие методы запросов поддерживаются сервером и какие дополнительные операции можно выполнять над ресурсом. Сервер должен вернуть ответ со списком поддерживаемых методов и другой соответствующей информацией.

**GET**: Метод GET используется для получения ресурса с сервера. Клиент отправляет GET-запрос на указанный URL-адрес, и сервер возвращает содержимое запрошенного ресурса. GET-запросы не должны изменять состояние сервера и могут быть кэшированы клиентом для повторного использования. Примером GET-запроса является получение веб-страницы в браузере.

**HEAD**: Метод HEAD аналогичен методу GET, но сервер возвращает только заголовки ответа без фактического содержимого ресурса. HEAD-запросы полезны, когда клиенту требуется только получить информацию о заголовках, например, для проверки доступности или проверки обновлений ресурса.

**POST**: Метод POST используется для отправки данных на сервер для создания нового ресурса или выполнения какой-либо операции на сервере. Клиент отправляет POST-запрос с данными в теле запроса, и сервер обрабатывает эти данные соответствующим образом. POST-запросы могут изменять состояние сервера или создавать новые ресурсы. Примером POST-запроса является отправка данных формы на сервер.

**PUT**: Метод PUT используется для загрузки или замены ресурса на сервере. Клиент отправляет PUT-запрос с данными ресурса в теле запроса на указанный URL-адрес, и сервер сохраняет или заменяет ресурс по указанному адресу. PUT-запросы полезны, когда клиент хочет обновить существующий ресурс или создать новый ресурс с определенным URL-адресом.

**Код состояния** является частью первой строки ответа сервера. Он представляет собой целое число из трех арабских цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа

**Классы кодов состояния** (status codes) в протоколе HTTP используются для передачи информации о результате выполнения запроса клиента на сервер. Код состояния состоит из трех цифр и разделен на пять классов, каждый из которых имеет свое значение и семантику: **Информационные** (Informational) (100-199): Коды состояния в этом классе предназначены для передачи информационных сообщений о статусе запроса. Они указывают, что запрос принят сервером и находится в процессе обработки. Примером кода состояния из этого класса является 100 (Continue), который сообщает клиенту, что сервер принял часть запроса и ожидает получения остальных данных.

**Успешные** (Successful) (200-299): Коды состояния в этом классе указывают на успешное выполнение запроса. Они сообщают клиенту, что запрос был обработан успешно и результаты находятся в ответе сервера. Наиболее известным кодом состояния из этого класса является 200 (OK), который указывает на успешное выполнение запроса и возвращает запрошенные данные. **Перенаправления** (Redirection) (300-399): Коды состояния в этом классе указывают на необходимость выполнения дополнительных действий для завершения запроса. Они сообщают клиенту о необходимости перенаправления на другой ресурс или выполнения других действий. Примером кода состояния из этого класса является 302 (Found), который указывает на временное перенаправление на другой ресурс.

**Ошибки** **клиента** (Client Errors) (400-499): Коды состояния в этом классе указывают на ошибки, связанные с запросом, отправленным клиентом. Они указывают на неправильный или некорректный запрос со стороны клиента. Примером кода состояния из этого класса является 404 (Not Found), который указывает на то, что запрашиваемый ресурс не найден на сервере.

**Ошибки сервера (**Server Errors) (500-599): Коды состояния в этом классе указывают на ошибки, связанные с обработкой запроса на сервере. Они указывают на проблемы или сбои в работе сервера. Примером кода состояния из этого класса является 500 (Internal Server Error), который указывает на внутреннюю ошибку сервера.

Классы кодов состояния HTTP предоставляют информацию о результате выполнения запроса и позволяют клиенту и серверу

**Заголовки:**

**Заголовки HTTP** (англ. HTTP Headers) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений ARPA (см. RFC 822). Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

**Примеры заголовков:**

Server: Apache/2.2.11 (Win32) PHP/5.3.0

Last-Modified: Sat, 16 Jan 2017 21:16:42 GMT

Content-Type: text/plain; charset=windows-1251

Content-Language: ru

В примере выше каждая строка представляет собой один заголовок. При этом то, что находится до первого двоеточия, называется **именем** (англ. name), а что после неё — **значением** (англ. value).

Группы заголовков: . **General Headers** (русск. Основные заголовки) — должны включаться в любое сообщение клиента и сервера.

2. **Request Headers** (русск. Заголовки запроса) — используются только в запросах клиента.

3. **Response Headers** (русск. Заголовки ответа) — только для ответов от сервера.

4. **Entity Headers** (русск. Заголовки сущности) — сопровождают каждую сущность сообщения.

Именно в таком порядке рекомендуется посылать заголовки получателю.

**Докачка** (Resume Downloading): Докачка, также известная как продолжение загрузки или возобновление загрузки, является процессом, при котором клиентское приложение или браузер возобновляет загрузку файла с сервера с того места, где она была прервана или приостановлена в предыдущий раз. Когда клиент и сервер поддерживают докачку, клиент может отправить запрос с заголовком Range (Диапазон), указывающим на диапазон байтов файла, который требуется загрузить. Сервер может ответить частичным содержимым файла, начиная с указанного диапазона байтов. Клиент затем объединяет полученные части с ранее загруженными данными, чтобы получить полный файл.

**Фрагментарное скачивание** (Chunked Downloading): Фрагментарное скачивание, или загрузка порциями, это процесс, при котором файл разбивается на небольшие фрагменты (части), которые загружаются постепенно по мере доступности. Вместо загрузки всего файла целиком, клиент скачивает фрагменты по мере их готовности на сервере. Это позволяет начать воспроизведение или использование файла быстрее, чем при полной загрузке. Фрагментарное скачивание часто применяется в потоковой передаче мультимедийных контентов, таких как видео и аудио. Например, при просмотре видео веб-страница может загружать фрагменты видеофайла по мере проигрывания, чтобы обеспечить непрерывное воспроизведение и уменьшить задержку перед началом просмотра.

**Особенности протокола**

HTTP же устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP было разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включённые в неё объекты (картинки, каскадные стили и т. п.), а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (неанонимного) доступа в HTTP используются cookies; причём такой способ авторизации позволяет сохранить сессию даже после перезагрузки клиента и сервера.

При доступе к данным по FTP или по файловым протоколам тип файла (точнее, тип содержащихся в нём данных) определяется по расширению имени файла, что не всегда удобно. HTTP перед тем, как передать сами данные, передаёт в заголовке строчку «Content-Type: тип/подтип», позволяющую клиенту однозначно определить, каким образом обрабатывать присланные данные. Это особенно важно при работе с CGI-скриптами, когда расширение имени файла указывает не на тип присылаемых клиенту данных, а на необходимость запуска данного файла на сервере и отправки клиенту результатов работы программы, записанной в этом файле (при этом один и тот же файл в зависимости от аргументов запроса и своих собственных соображений может порождать ответы разных типов — в простейшем случае картинки в разных форматах).

Кроме того, HTTP позволяет клиенту прислать на сервер параметры, которые будут переданы запускаемому CGI-скрипту. Для этого же в HTML были введены формы.

# Служба RPC.

**Вызов удаленных процедур (RPC)**

**Концепция удаленного вызова процедур**

Идея вызова удаленных процедур (Remote Procedure Call - RPC) состоит в расширении хорошо известного и понятного механизма передачи управления и данных внутри программы, выполняющейся на одной машине, на передачу управления и данных через сеть. Средства удаленного вызова процедур предназначены для облегчения организации распределенных вычислений. Наибольшая эффективность использования RPC достигается в тех приложениях, в которых существует интерактивная связь между удаленными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются RPC-ориентированными.

Характерными чертами вызова локальных процедур являются:

* Асимметричность, то есть одна из взаимодействующих сторон является инициатором;
* Синхронность, то есть выполнение вызывающей процедуры при останавливается с момента выдачи запроса и возобновляется только после возврата из вызываемой процедуры.

Реализация удаленных вызовов существенно сложнее реализации вызовов локальных процедур. Начнем с того, что поскольку вызывающая и вызываемая процедуры выполняются на разных машинах, то они имеют разные адресные пространства, и это создает проблемы при передаче параметров и результатов, особенно если машины не идентичны. Так как RPC не может рассчитывать на разделяемую память, то это означает, что параметры RPC не должны содержать указателей на ячейки нестековой памяти и что значения параметров должны копироваться с одного компьютера на другой. Следующим отличием RPC от локального вызова является то, что он обязательно использует нижележащую систему связи, однако это не должно быть явно видно ни в определении процедур, ни в самих процедурах. Удаленность вносит дополнительные проблемы. Выполнение вызывающей программы и вызываемой локальной процедуры в одной машине реализуется в рамках единого процесса. Но в реализации RPC участвуют как минимум два процесса - по одному в каждой машине. В случае, если один из них аварийно завершится, могут возникнуть следующие ситуации: при аварии вызывающей процедуры удаленно вызванные процедуры станут "осиротевшими", а при аварийном завершении удаленных процедур станут "обездоленными родителями" вызывающие процедуры, которые будут безрезультатно ожидать ответа от удаленных процедур.

Кроме того, существует ряд проблем, связанных с неоднородностью языков программирования и операционных сред: структуры данных и структуры вызова процедур, поддерживаемые в каком-либо одном языке программирования, не поддерживаются точно так же во всех других языках.

Эти и некоторые другие проблемы решает широко распространенная технология RPC, лежащая в основе многих распределенных операционных систем.

**Базовые операции RPC**

Чтобы понять работу RPC, рассмотрим вначале выполнение вызова локальной процедуры в обычной машине, работающей автономно. Пусть это, например, будет системный вызов

count=read (fd,buf,nbytes);

где fd - целое число, buf - массив символов, nbytes - целое число.

Чтобы осуществить вызов, вызывающая процедура заталкивает параметры в стек в обратном порядке (рисунок 3.1). После того, как вызов read выполнен, он помещает возвращаемое значение в регистр, перемещает адрес возврата и возвращает управление вызывающей процедуре, которая выбирает параметры из стека, возвращая его в исходное состояние.

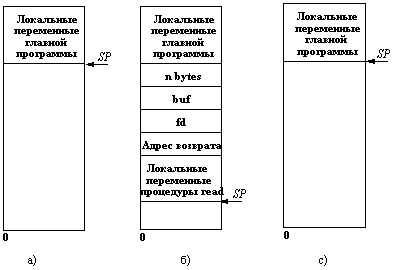


Рис. 3.1. а) Стек до выполнения вызова read;   
б) Стек во время выполнения процедуры;   
в) Стек после возврата в вызывающую программу

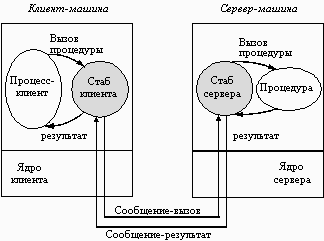
Идея, положенная в основу RPC, состоит в том, чтобы сделать вызов удаленной процедуры выглядящим по возможности также, как и вызов локальной процедуры. Другими словами - сделать RPC прозрачным: вызывающей процедуре не требуется знать, что вызываемая процедура находится на другой машине, и наоборот.

RPC достигает прозрачности следующим путем. Когда вызываемая процедура действительно является удаленной, в библиотеку помещается вместо локальной процедуры другая версия процедуры, называемая клиентским стабом (stub - заглушка). Подобно оригинальной процедуре, стаб вызывается с использованием вызывающей последовательности (как на рисунке 3.1), так же происходит прерывание при обращении к ядру. Только в отличие от оригинальной процедуры он не помещает параметры в регистры и не запрашивает у ядра данные, вместо этого он формирует сообщение для отправки ядру удаленной машины.

**Этапы выполнения RPC**

Взаимодействие программных компонентов при выполнении удаленного вызова процедуры иллюстрируется рисунком 3.2. После того, как клиентский стаб был вызван программой-клиентом, его первой задачей является заполнение буфера отправляемым сообщением. В некоторых системах клиентский стаб имеет единственный буфер фиксированной длины, заполняемый каждый раз с самого начала при поступлении каждого нового запроса. В других системах буфер сообщения представляет собой пул буферов для отдельных полей сообщения, причем некоторые из этих буферов уже заполнены. Этот метод особенно подходит для тех случаев, когда пакет имеет формат, состоящий из большого числа полей, но значения многих из этих полей не меняются от вызова к вызову.

Затем параметры должны быть преобразованы в соответствующий формат и вставлены в буфер сообщения. К этому моменту сообщение готово к передаче, поэтому выполняется прерывание по вызову ядра.



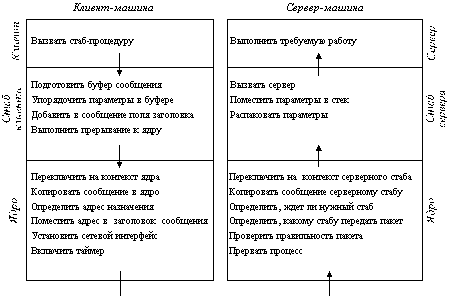
*Рис. 3.2. Remote Procedure Call*

Когда ядро получает управление, оно переключает контексты, сохраняет регистры процессора и карту памяти (дескрипторы страниц), устанавливает новую карту памяти, которая будет использоваться для работы в режиме ядра. Поскольку контексты ядра и пользователя различаются, ядро должно точно скопировать сообщение в свое собственное адресное пространство, так, чтобы иметь к нему доступ, запомнить адрес назначения (а, возможно, и другие поля заголовка), а также оно должно передать его сетевому интерфейсу. На этом завершается работа на клиентской стороне. Включается таймер передачи, и ядро может либо выполнять циклический опрос наличия ответа, либо передать управление планировщику, который выберет какой-либо другой процесс на выполнение. В первом случае ускоряется выполнение запроса, но отсутствует мультипрограммирование.

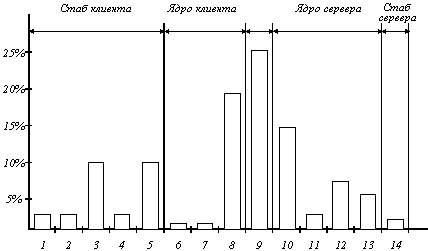
На стороне сервера поступающие биты помещаются принимающей аппаратурой либо во встроенный буфер, либо в оперативную память. Когда вся информация будет получена, генерируется прерывание. Обработчик прерывания проверяет правильность данных пакета и определяет, какому стабу следует их передать. Если ни один из стабов не ожидает этот пакет, обработчик должен либо поместить его в буфер, либо вообще отказаться от него. Если имеется ожидающий стаб, то сообщение копируется ему. Наконец, выполняется переключение контекстов, в результате чего восстанавливаются регистры и карта памяти, принимая те значения, которые они имели в момент, когда стаб сделал вызов receive.

Теперь начинает работу серверный стаб. Он распаковывает параметры и помещает их соответствующим образом в стек. Когда все готово, выполняется вызов сервера. После выполнения процедуры сервер передает результаты клиенту. Для этого выполняются все описанные выше этапы, только в обратном порядке.

Рисунок 3.3 показывает последовательность команд, которую необходимо выполнить для каждого RPC-вызова, а рисунок 3.4 - какая доля общего времени выполнения RPC тратится на выполнение каждого их описанных 14 этапов



*Рис. 3.3. Этапы выполнения процедуры RPC*



*Рис. 3.4. Распределение времени между 14 этапами выполнения RPC*

Последовательность команд, которую необходимо выполнить для каждого RPC-вызова:

***1. Вызов стаба***

***2. Подготовить буфер***

***3. Упаковать параметры***

***4. Заполнить поле заголовка***

***5. Вычислить контрольную сумму в сообщении***

***6. Прерывание к ядру***

***7. Очередь пакета на выполнение***

***8. Передача сообщения контроллеру по шине QBUS***

***9. Время передачи по сети Ethernet***

***10. Получить пакет от контроллера***

***11. Процедура обработки прерывания***

***12. Вычисление контрольной суммы***

***13. Переключение контекста в пространство пользователя***

***14. Выполнение серверного стаба***

**Динамическое связывание**

Рассмотрим вопрос о том, как клиент задает месторасположение сервера. Одним из методов решения этой проблемы является непосредственное использование сетевого адреса сервера в клиентской программе. Недостаток такого подхода - его чрезвычайная негибкость: при перемещении сервера, или при увеличении числа серверов, или при изменении интерфейса во всех этих и многих других случаях необходимо перекомпилировать все программы, которые использовали жесткое задание адреса сервера. Для того, чтобы избежать всех этих проблем, в некоторых распределенных системах используется так называемое динамическое связывание.

Начальным моментом для динамического связывания является формальное определение (спецификация) сервера. Спецификация содержит имя файл-сервера, номер версии и список процедур-услуг, предоставляемых данным сервером для клиентов. Для каждой процедуры дается описание ее параметров с указанием того, является ли данный параметр входным или выходным относительно сервера. Некоторые параметры могут быть одновременно входными и выходными - например, некоторый массив, который посылается клиентом на сервер, модифицируется там, а затем возвращается обратно клиенту (операция copy/ restore).

Формальная спецификация сервера используется в качестве исходных данных для программы-генератора стабов, которая создает как клиентские, так и серверные стабы. Затем они помещаются в соответствующие библиотеки. Когда пользовательская (клиентская) программа вызывает любую процедуру, определенную в спецификации сервера, соответствующая стаб-процедура связывается с двоичным кодом программы. Аналогично, когда компилируется сервер, с ним связываются серверные стабы.

При запуске сервера самым первым его действием является передача своего серверного интерфейса специальной программе, называемой binder'ом. Этот процесс, известный как процесс регистрации сервера, включает передачу сервером своего имени, номера версии, уникального идентификатора и описателя местонахождения сервера. Описатель системно независим и может представлять собой IP, Ethernet, X.500 или еще какой-либо адрес. Кроме того, он может содержать и другую информацию, например, относящуюся к аутентификации.

Когда клиент вызывает одну из удаленных процедур первый раз, например, read, клиентский стаб видит, что он еще не подсоединен к серверу, и посылает сообщение binder-программе с просьбой об импорте интерфейса нужной версии нужного сервера. Если такой сервер существует, то binder передает описатель и уникальный идентификатор клиентскому стабу.

Клиентский стаб при посылке сообщения с запросом использует в качестве адреса описатель. В сообщении содержатся параметры и уникальный идентификатор, который ядро сервера использует для того, чтобы направить поступившее сообщение в нужный сервер в случае, если их несколько на этой машине.

Этот метод, заключающийся в импорте/экспорте интерфейсов, обладает высокой гибкостью.

Однако у динамического связывания имеются недостатки, например, дополнительные накладные расходы (временные затраты) на экспорт и импорт интерфейсов. Кроме того, в больших распределенных системах может стать узким местом программа binder, а создание нескольких программ аналогичного назначения также увеличивает накладные расходы на создание и синхронизацию процессов.

**Семантика RPC в случае отказов**

В идеале RPC должен функционировать правильно и в случае отказов. Рассмотрим следующие классы отказов:

1. Клиент не может определить местонахождения сервера, например, в случае отказа нужного сервера, или из-за того, что программа клиента была скомпилирована давно и использовала старую версию интерфейса сервера. В этом случае в ответ на запрос клиента поступает сообщение, содержащее код ошибки.
2. Потерян запрос от клиента к серверу. Самое простое решение - через определенное время повторить запрос.
3. Потеряно ответное сообщение от сервера клиенту. Этот вариант сложнее предыдущего, так как некоторые процедуры не являются идемпотентными. Идемпотентной называется процедура, запрос на выполнение которой можно повторить несколько раз, и результат при этом не изменится. Примером такой процедуры может служить чтение файла. Но вот процедура снятия некоторой суммы с банковского счета не является идемпотентной, и в случае потери ответа повторный запрос может существенно изменить состояние счета клиента. Одним из возможных решений является приведение всех процедур к идемпотентному виду. Однако на практике это не всегда удается, поэтому может быть использован другой метод - последовательная нумерация всех запросов клиентским ядром. Ядро сервера запоминает номер самого последнего запроса от каждого из клиентов, и при получении каждого запроса выполняет анализ - является ли этот запрос первичным или повторным.
4. Сервер потерпел аварию после получения запроса. Здесь также важно свойство идемпотентности, но к сожалению не может быть применен подход с нумерацией запросов. В данном случае имеет значение, когда произошел отказ - до или после выполнения операции. Но клиентское ядро не может распознать эти ситуации, для него известно только то, что время ответа истекло. Существует три подхода к этой проблеме:

* Ждать до тех пор, пока сервер не перезагрузится и пытаться выполнить операцию снова. Этот подход гарантирует, что RPC был выполнен до конца по крайней мере один раз, а возможно и более.
* Сразу сообщить приложению об ошибке. Этот подход гарантирует, что RPC был выполнен не более одного раза.
* Третий подход не гарантирует ничего. Когда сервер отказывает, клиенту не оказывается никакой поддержки. RPC может быть или не выполнен вообще, или выполнен много раз. Во всяком случае этот способ очень легко реализовать.

Ни один из этих подходов не является очень привлекательным. А идеальный вариант, который бы гарантировал ровно одно выполнение RPC, в общем случае не может быть реализован по принципиальным соображениям. Пусть, например, удаленной операцией является печать некоторого текста, которая включает загрузку буфера принтера и установку одного бита в некотором управляющем регистре принтера, в результате которой принтер стартует. Авария сервера может произойти как за микросекунду до, так и за микросекунду после установки управляющего бита. Момент сбоя целиком определяет процедуру восстановления, но клиент о моменте сбоя узнать не может. Короче говоря, возможность аварии сервера радикально меняет природу RPC и ясно отражает разницу между централизованной и распределенной системой. В первом случае крах сервера ведет к краху клиента, и восстановление невозможно. Во втором случае действия по восстановлению системы выполнить и возможно, и необходимо.

1. Клиент потерпел аварию после отсылки запроса. В этом случае выполняются вычисления результатов, которых никто не ожидает. Такие вычисления называют "сиротами". Наличие сирот может вызвать различные проблемы: непроизводительные затраты процессорного времени, блокирование ресурсов, подмена ответа на текущий запрос ответом на запрос, который был выдан клиентской машиной еще до перезапуска системы.

Как поступать с сиротами? Рассмотрим 4 возможных решения.

* *Уничтожение.* До того, как клиентский стаб посылает RPC-сообщение, он делает отметку в журнале, оповещая о том, что он будет сейчас делать. Журнал хранится на диске или в другой памяти, устойчивой к сбоям. После аварии система перезагружается, журнал анализируется и сироты ликвидируются. К недостаткам такого подхода относятся, во-первых, повышенные затраты, связанные с записью о каждом RPC на диск, а, во-вторых, возможная неэффективность из-за появления сирот второго поколения, порожденных RPC-вызовами, выданными сиротами первого поколения.
* *Перевоплощение*. В этом случае все проблемы решаются без использования записи на диск. Метод состоит в делении времени на последовательно пронумерованные периоды. Когда клиент перезагружается, он передает широковещательное сообщение всем машинам о начале нового периода. После приема этого сообщения все удаленные вычисления ликвидируются. Конечно, если сеть сегментированная, то некоторые сироты могут и уцелеть.
* *Мягкое перевоплощение* аналогично предыдущему случаю, за исключением того, что отыскиваются и уничтожаются не все удаленные вычисления, а только вычисления перезагружающегося клиента.
* *Истечение срока*. Каждому запросу отводится стандартный отрезок времени Т, в течение которого он должен быть выполнен. Если запрос не выполняется за отведенное время, то выделяется дополнительный квант. Хотя это и требует дополнительной работы, но если после аварии клиента сервер ждет в течение интервала Т до перезагрузки клиента, то все сироты обязательно уничтожаются.

На практике ни один из этих подходов не желателен, более того, уничтожение сирот может усугубить ситуацию. Например, пусть сирота заблокировал один или более файлов базы данных. Если сирота будет вдруг уничтожен, то эти блокировки останутся, кроме того уничтоженные сироты могут остаться стоять в различных системных очередях, в будущем они могут вызвать выполнение новых процессов и т.п.

# NAT, proxy-серверы, межсетевые экраны, ремайлеры.

**NAT (от англ. Network Address Translation — «преобразование сетевых адресов»)** — это механизм в сетях TCP/IP, позволяющий преобразовывать IP-адреса транзитных пакетов.

Механизм NAT определён в RFC 1631, RFC 3022.

Преобразование адресов методом NAT может производиться почти любым маршрутизирующим устройством — маршрутизатором, сервером доступа, межсетевым экраном. Наиболее популярным является SNAT, суть механизма которого состоит в замене адреса источника (англ. source) при прохождении пакета в одну сторону и обратной замене адреса назначения (англ. destination) в ответном пакете. Наряду с адресами источник/назначение могут также заменяться номера портов источника и назначения.

Ваш компьютер может быть подключен к интернету напрямую. Тогда говорят, что у него внешний IP адрес. Обычно это значит, что компьютер подключен сразу к модему (DSL, кабельному или обычному аналоговому).

За NAT означает, что ваш компьютер подключен не к интернету, а к локальной сети. Тогда у него внутренний IP адрес, из интернета сам по себе недоступный.

Доступ к интернету ваш компьютер получает через NAT - процесс трансляции внутренних адресов во внешние и обратно. NAT-устройство обычно называют роутером.

NAT позволяет хостам из intranet прозрачным образом обращаться к хостам в общедоступном пространстве, при этом для внутренних хостов не требуется наличия зарегистрированных (и наиболее дефицитных) сетевых Internet-адресов.

Internet Engineering Task Force вот уже почти десять лет предупреждает о грядущем дефиците адресов в существующем сейчас адресном пространстве, формируемом IPv4. И хотя, по-видимому, новая версия протокола IPv6 будет способна достаточно долгое время удовлетворять требованиям постоянно растущего Internet, предложенные за последние несколько лет меры уже получили законный статус.

Одно из подобных предложений описывает опубликованный в 1994 году стандарт RFC 1631, "Транслятор сетевых IP-адресов" (The IP Network Address Translator). В период становления Internet потребители были вынуждены использовать уникальные в глобальном масштабе сетевые адреса вне зависимости от того, планировали они подключаться к глобальному пространству Internet или нет. Идея состояла в том, чтобы избежать потенциальных конфликтов в том случае, если бывшая ранее частной сеть в конце концов присоединялась к общедоступной Internet.

Простейшее устройство NAT имеет два сетевых соединения: одно с Internet и одно с частной сетью. Хосты в частной сети, использующие частные IP-адреса (иногда также называемые адресами Network 10, которые начинаются с 10.0.0.0 в диапазоне, выделенном для частного использования), связываются с Internet, передавая пакеты непосредственно на устройство NAT. В отличие от обычных маршрутизаторов, которые просто считывают адреса отправителя и получателя каждого пакета, прежде чем передать их по назначению, устройства NAT модифицируют заголовки пакетов, заменяя адрес отправителя в частной сети на свой собственный Internet-адрес.

Теперь рассмотрим процесс установления сессии при использовании NAT-маршрутизатора на границе внутренней сети и сети Интернет.

Когда клиент внутренней сети устанавливает связь с сервером внешней сети, то, как и в случае установки соединения между двумя ПК, открывается сокет, определяемый IP-адресом источника, портом источника, IP-адресом назначения, портом назначения и сетевым протоколом. Когда приложение передает данные через этот сокет, IP-адрес источника и порт источника вставляются в пакет в поля параметров источника. Поля параметров пункта назначения будут содержать IP-адрес сервера и порт сервера. К примеру, компьютер внутренней сети с IP-адресом 192.168.0.1 может обратиться к Web-серверу Глобальной сети с IP-адресом 64.233.188.104. В этом случае операционная система клиента может назначить установленной сессии порт 1251 (порт источника), а порт назначения — это порт Web-сервиса, то есть 80.

IP-адрес источника: 192.168.0.1;

порт источника: 1251;

IP-адрес получателя: 64.233.183.104;

порт получателя: 80;

протокол: TCP.

Устройство NAT (маршрутизатор) перехватывает исходящий из внутренней сети пакет и заносит в свою внутреннюю таблицу сопоставление портов источника и получателя пакета, используя IP-адрес назначения, порт назначения, внешний IP-адрес устройства NAT, внешний порт, сетевой протокол, а также внутренние IP-адрес и порт клиента.

Благодаря NAT-маршрутизатору любой ПК внутренней сети получает возможность передавать данные в Глобальную сеть с использованием внешнего IP-адреса и порта маршрутизатора. При этом IP-адреса внутренней сети, как присвоенные сессиям порты, остаются невидимыми со стороны внешней сети.

Помимо source NAT (предоставления пользователям локальной сети с внутренними адресами доступа к сети Интернет) часто применяется также destination NAT, когда обращения извне транслируются межсетевым экраном на сервер в локальной сети, имеющий внутренний адрес и потому недоступный извне сети непосредственно (без NAT).

Существует 3 базовых концепции трансляции адресов: статическая (Static Network Address Translation), динамическая (Dynamic Address Translation ), маскарадная (NAPT, NAT Overload, PAT).

Статический NAT — Отображение незарегистрированного IP адреса на зарегистрированный IP адрес на основании один к одному. Особенно полезно, когда устройство должно быть доступным снаружи сети.

Динамический NAT — Отображает незарегистрированный IP адрес на зарегистрированный адрес от группы зарегистрированных IP адресов. Динамический NAT также устанавливает непосредственное отображение между незарегистрированным и зарегистрированным адресом, но отображение может меняться в зависимости от зарегистрированного адреса, доступного в пуле адресов, во время коммуникации.

Перегруженный NAT (NAPT, NAT Overload, PAT, маскарадинг) — форма динамического NAT, который отображает несколько незарегистрированных адресов в единственный зарегистрированный IP адрес, используя различные порты. Известен также как PAT (Port Address Translation)

При перегрузке, каждый компьютер в частной сети транслируется в тот же самый адрес, но с различным номером порта.

Маскарадинг (англ. Masquerading) — тип трансляции сетевого адреса, при которой адрес отправителя подставляется динамически, в зависимости от назначенного интерфейсу адреса. Разделение SNAT (source NAT) и маскарадинга характерно для iptables, в маршрутизаторах cisco операция трансляции (ip nat source) умеет работать как с заданными адресами, так и с интерфейсами (используя автоматически назначенный интерфейсу адрес).

Ещё одной особенностью маскарадинга (в iptables) является «забывание» про установленные трансляции при остановке (down) интерфейса. Это связано с тем, что после поднятия интерфейса его адрес, вероятнее всего (в случае DHCP/Dialup) будет другим, и записи о ранее выполненных трансляциях не будут иметь смысла.

**NAT выполняет три важных функции.**

Позволяет сэкономить IP-адреса (только в случае использования NAT в режиме PAT), транслируя несколько внутренних IP-адресов в один внешний публичный IP-адрес (или в несколько, но меньшим количеством, чем внутренних). По такому принципу построено большинство сетей в мире: на небольшой район домашней сети местного провайдера или на офис выделяется 1 «белый» (то есть внешний) IP-адрес, за которым работают и получают доступ вовне все «серые» (то есть внутренние) IP-адреса.

Позволяет предотвратить или ограничить обращение снаружи ко внутренним хостам, оставляя возможность обращения изнутри наружу. При инициации соединения изнутри сети создаётся трансляция. Ответные пакеты, поступающие снаружи, соответствуют созданной трансляции и поэтому пропускаются. Если для пакетов, поступающих снаружи, соответствующей трансляции не существует (а она может быть созданной при инициации соединения или статической), они не пропускаются.

Позволяет скрыть определённые внутренние сервисы внутренних хостов/серверов. По сути, выполняется та же указанная выше трансляция на определённый порт, но возможно подменить внутренний порт официально зарегистрированной службы (например, 80-й порт TCP (HTTP-сервер) на внешний 54055-й). Тем самым, снаружи, на внешнем IP-адресе после трансляции адресов на сайт (или форум) для осведомлённых посетителей можно будет попасть по адресу http://example.org:54055, но на внутреннем сервере, находящимся за NAT, он будет работать на обычном 80-м порту. Повышение безопасности и скрытие «непубличных» ресурсов.

**Недостатки**

При использовании NAT хосты Internet взаимодействуют напрямую с NAT-устройством, а не с реальным хостом в частной сети. Входящие пакеты передаются по IP-адресу устройства NAT, а устройство меняет адрес назначения в заголовке пакета со своего собственного Internet-адреса на адрес частной сети реального хоста назначения.

В результате - теоретически - единый уникальный в глобальном масштабе IP-адрес может скрывать за собой сотни, тысячи или даже миллионы хостов, чьи адреса определены в частной сети. На практике, однако, такой подход имеет ряд недостатков. Например, многие Internet-протоколы и приложения существенно опираются на тот факт, что сеть в действительности поддерживает соединение из конца в конец, когда пакеты передаются без каких-либо изменений от отправителя получателю. Архитектура IP-защиты, к примеру, не может поддерживать работу через устройство NAT, поскольку оригинальные заголовки, в которых указаны исходные IP-адреса отправителей, защищены с помощью цифровой подписи. Измените исходный адрес - и цифровая подпись потеряет свою целостность.

Использование NAT к тому же усложняет работу администраторов. Конечно, NAT - превосходное решение для организации, филиала или даже подразделения, которые не могут получить достаточное количество уникальных Internet-адресов. Однако при реорганизации, объединении или покупке другой компанией, когда требуется объединение двух или нескольких частных сетей, возникают большие сложности. Даже если структура организации остается достаточно стабильной, системы NAT могут случайно оказаться вложенными структурами, превращая маршрутизацию в настоящий кошмар.

Не все протоколы могут «преодолеть» NAT. Некоторые не в состоянии работать, если на пути между взаимодействующими хостами есть трансляция адресов. Некоторые межсетевые экраны, осуществляющие трансляцию IP-адресов, могут исправить этот недостаток, соответствующим образом заменяя IP-адреса не только в заголовках IP, но и на более высоких уровнях (например, в командах протокола FTP)..

Из-за трансляции адресов «много в один» появляются дополнительные сложности с идентификацией пользователей и необходимость хранить полные логи трансляций.

DoS со стороны узла, осуществляющего NAT — если NAT используется для подключения многих пользователей к одному и тому же сервису, это может вызвать иллюзию DoS-атаки на сервис (множество успешных и неуспешных попыток). Например, избыточное количество пользователей мессенджера за NAT приводит к проблеме с подключением к серверу некоторых пользователей из-за превышения допустимой скорости подключений. Частичным решением проблемы является использование пула адресов (группы адресов), для которых осуществляется трансляция.

В некоторых случаях, необходимость в дополнительной настройке при работе с пиринговыми сетями и некоторыми другими программами (Battle.net-игры), в которых необходимо не только инициировать исходящие соединения, но также принимать входящие. Однако, если NAT устройство и ПО требующее дополнительной настройки поддерживает технологию Universal Plug and Play, то в этом случае настройка произойдет полностью автоматически и прозрачно для пользователя.

**Межсетевой экран или сетевой экран** — комплекс аппаратных или программных средств, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами.

Также сетевые экраны часто называют фильтрами, так как их основная задача — не пропускать (фильтровать) пакеты, не подходящие под критерии, определённые в конфигурации.

Некоторые сетевые экраны также позволяют осуществлять трансляцию адресов — динамическую замену внутрисетевых (серых) адресов или портов на внешние, используемые за пределами ЛВС.

Другие названия

Брандма́уэр (нем. Brandmauer) — заимствованный из немецкого языка термин, являющийся аналогом английского firewall в его оригинальном значении (стена, которая разделяет смежные здания, предохраняя от распространения пожара). Интересно, что в области компьютерных технологий в немецком языке употребляется слово «firewall».

Файрво́лл, файрво́л, файерво́л, фаерво́л — образовано транслитерацией английского термина firewall, эквивалентного термину межсетевой экран.

Основная задача любого брандмауэра в конечном счете сводится к обеспечению безопасности внутренней сети. Для решения этой задачи брандмауэры должны уметь маскировать защищаемую сеть, блокировать все известные типы хакерских атак, блокировать утечку информации из внутренней сети, контролировать приложения, получающие доступ во внешнюю сеть.

Для того чтобы реализовать указанные функции, брандмауэры анализируют весь трафик между внешней и внутренней сетями на предмет его соответствия тем или иным установленным критериям или правилам, определяющим условия прохождения трафика из одной сети в другую. Если трафик отвечает заданным критериям, то брандмауэр пропускает его через себя. В противном случае, то есть если установленные критерии не соблюдены, трафик блокируется брандмауэром. Брандмауэры фильтруют как входящий, так и исходящий трафики, а также позволяют управлять доступом к определенным сетевым ресурсам или приложениям. Они могут фиксировать все попытки несанкционированного доступа к ресурсам локальной сети и выдавать предупреждения о попытках проникновения.

По своему назначению брандмауэры больше всего напоминают контрольно-пропускной пункт (КПП) охраняемого объекта, где производится проверка документов всех входящих на территорию объекта и всех покидающих ее. Если пропуск в порядке — доступ на территорию разрешен. Совершенно аналогично действуют и брандмауэры, только в роли людей, проходящих через КПП, выступают сетевые пакеты, а пропуском является соответствие заголовков этих пакетов предопределенному набору правил.

Можно ли утверждать, что брандмауэр обеспечивает 100-процентную безопасность пользовательской сети или персонального ПК? Безусловно, нет. Хотя бы потому, что вообще ни одна система не дает 100-процентной гарантии безопасности. К брандмауэру стоит относиться как к средству, которое, при правильной его настройке, способно в значительной мере осложнить задачу злоумышленника по проникновению в персональный компьютер пользователя. Подчеркнем: лишь осложнить, но вовсе не гарантировать абсолютной безопасности. Кстати, если речь заходит не о защите локальной сети, а о защите отдельного ПК, имеющего доступ в Интернет, то с обеспечением его персональной безопасности успешно справляется и брандмауэр ICF (Internet Connection Firewall), встроенный в операционную систему начиная с Windows XP. Поэтому в дальнейшем мы будем говорить лишь о корпоративных аппаратных брандмауэрах, ориентированных на защиту небольших сетей.

Возможности брандмауэров и степень их интеллектуальности зависят от того, на каком уровне эталонной модели OSI они функционируют. Чем выше уровень OSI, на основе которого построен брандмауэр, тем выше обеспечиваемый им уровень защиты.

Разновидности сетевых экранов

Сетевые экраны подразделяются на различные типы в зависимости от следующих характеристик:

обеспечивает ли экран соединение между одним узлом и сетью или между двумя или более различными сетями;

происходит ли контроль потока данных на сетевом уровне или более высоких уровнях модели OSI;

отслеживаются ли состояния активных соединений или нет.

В зависимости от охвата контролируемых потоков данных сетевые экраны делятся на:

традиционный сетевой (или межсетевой) экран — программа (или неотъемлемая часть операционной системы) на шлюзе (сервере передающем трафик между сетями) или аппаратное решение, контролирующие входящие и исходящие потоки данных между подключенными сетями.

персональный сетевой экран — программа, установленная на пользовательском компьютере и предназначенная для защиты от несанкционированного доступа только этого компьютера.

Вырожденный случай — использование традиционного сетевого экрана сервером, для ограничения доступа к собственным ресурсам.

В зависимости от уровня, на котором происходит контроль доступа, существует разделение на сетевые экраны, работающие на:

сетевом уровне, когда фильтрация происходит на основе адресов отправителя и получателя пакетов, номеров портов транспортного уровня модели OSI и статических правил, заданных администратором;

сеансовом уровне (также известные как stateful) — отслеживающие сеансы между приложениями, не пропускающие пакеты нарушающих спецификации TCP/IP, часто используемых в злонамеренных операциях — сканировании ресурсов, взломах через неправильные реализации TCP/IP, обрыв/замедление соединений, инъекция данных.

уровне приложений, фильтрация на основании анализа данных приложения, передаваемых внутри пакета. Такие типы экранов позволяют блокировать передачу нежелательной и потенциально опасной информации, на основании политик и настроек.

Некоторые решения, относимые к сетевым экранам уровня приложения, представляют собой прокси-серверы с некоторыми возможностями сетевого экрана, реализуя прозрачные прокси-серверы, со специализацией по протоколам. Возможности прокси-сервера и многопротокольная специализация делают фильтрацию значительно более гибкой, чем на классических сетевых экранах, но такие приложения имеют все недостатки прокси-серверов (например, анонимизация трафика).

**Прокси-сервер (от англ. proxy — «представитель, уполномоченный»)** — служба в компьютерных сетях, позволяющая клиентам выполнять косвенные запросы к другим сетевым службам. Сначала клиент подключается к прокси-серверу и запрашивает какой-либо ресурс (например, e-mail), расположенный на другом сервере. Затем прокси-сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного кэша (в случаях, если прокси имеет свой кэш). В некоторых случаях запрос клиента или ответ сервера может быть изменён прокси-сервером в определённых целях. Также прокси-сервер позволяет защищать клиентский компьютер от некоторых сетевых атак и помогает сохранять анонимность клиента.

Использование

Чаще всего прокси-серверы применяются для следующих целей:

Обеспечение доступа с компьютеров локальной сети в Интернет.

Кэширование данных: если часто происходят обращения к одним и тем же внешним ресурсам, то можно держать их копию на прокси-сервере и выдавать по запросу, снижая тем самым нагрузку на канал во внешнюю сеть и ускоряя получение клиентом запрошенной информации.

Сжатие данных: прокси-сервер загружает информацию из Интернета и передаёт информацию конечному пользователю в сжатом виде. Такие прокси-серверы используются в основном с целью экономии внешнего трафика.

Защита локальной сети от внешнего доступа: например, можно настроить прокси-сервер так, что локальные компьютеры будут обращаться к внешним ресурсам только через него, а внешние компьютеры не смогут обращаться к локальным вообще (они «видят» только прокси-сервер).

Ограничение доступа из локальной сети к внешней: например, можно запретить доступ к определённым веб-сайтам, ограничить использование интернета каким-то локальным пользователям, устанавливать квоты на трафик или полосу пропускания, фильтровать рекламу и вирусы.

Анонимизация доступа к различным ресурсам. Прокси-сервер может скрывать сведения об источнике запроса или пользователе. В таком случае целевой сервер видит лишь информацию о прокси-сервере, например, IP-адрес, но не имеет возможности определить истинный источник запроса. Существуют также искажающие прокси-серверы, которые передают целевому серверу ложную информацию об истинном пользователе.

Прокси-сервер, к которому может получить доступ любой пользователь сети интернет, называется открытым.

Виды прокси-серверов

Прозрачный прокси — схема связи, при которой трафик, или его часть, перенаправляется на прокси-сервер неявно (средствами маршрутизатора). При этом клиент может использовать все преимущества прокси-сервера без дополнительных настроек, но с другой стороны, не имеет выбора.

Обратный прокси — прокси-сервер, который в отличие от прямого, ретранслирует запросы клиентов из внешней сети на один или несколько серверов, логически расположенных во внутренней сети. Часто используется для балансировки сетевой нагрузки между несколькими веб-серверами и повышения их безопасности, играя при этом роль межсетевого экрана на прикладном уровне.

Веб-прокси (англ. «web-based proxy») — это прокси-сервер и анонимайзер особого вида, представляющий собой веб-приложение (чаще всего PHP или Perl скрипт) установленное на веб-сервере, выступающее в роли посредника для загрузки контента различных веб-сайтов.

Веб-прокси могут быть использованы для:

ускорения загрузки веб-сайтов;

тестирования онлайн сервисов;

обхода ограничений Администратора локальной сети на доступ к определенным адресам веб-сайтов;

сокрытия реального IP-адреса и анонимного доступа к веб-сайтам;

получения доступа к веб-сайтам закрытым для просмотра пользователей определенных стран;

и многих других целей.

**Ремейлер (англ. remailer)** — это сервер, получающий сообщение электронной почты и переправляющий его по адресу, указанному отправителем. В процессе переадресации вся информация об отправителе уничтожается, поэтому конечный получатель лишён возможности выяснить, кто является автором сообщения.[1] Некоторые из ремейлеров позволяют также шифровать письма и указывать фиктивный адрес отправителя, но большинство из них прямо указывают в заголовке, что электронное сообщение анонимно. В качестве ремейлеров могут выступать специализированные веб-сайты, открытые SMTP-серверы и анонимные сети вроде Mixminion.

Виды ремейлеров

Ремейлеры делятся на анонимные и псевдо-анонимные.

При использовании псевдо-анонимного ремейлера, его оператор знает адрес электронной почты, который необходим для получения ответа на письмо. Тайна связи полностью зависит от оператора, который может стать жертвой угроз, шантажа или социальной инженерии. Преимуществом псевдо-анонимных ремейлеров является их юзабилити, за которое пользователь расплачивается меньшей защищённостью.

Анонимные ремейлеры обеспечивают гораздо более высокую секретность, но при этом они и сложнее в использовании. Их операторы не могут знать, какие данные пересылаются через них, а поэтому нет гарантии своевременной доставки сообщения, которое может и вовсе затеряться.[2] В обмен на высокое время ожидания анонимные ремейлеры достаточно надёжно скрывают от посторонних глаз реальный адрес и содержимое сообщения.

Mixminion — стандарт реализации третьего типа протокола анонимной пересылки электронной почты. Mixminion может отсылать и принимать анонимные сообщения электронной почты.

# Web-сервисы: SOAP, XML, WSDL, UDDI.

**Веб-сервисы**

Веб-сервисы меняют все! Точно так же, как железнодорожные перевозки влияют на экономику страны, так и веб-сервисы коренным образом изменяют правила электронной коммерции. Передавая большой объем данных более эффективно и дешевле, чем ранее, они обеспечивают связь между приложениями, выполняемыми в любых удаленных точках земного шара. В результате между предпринимателями и потребителями устанавливается быстрая, удобная и продуктивная связь.

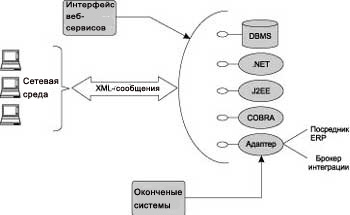
Сеть в ее современном состоянии плохо поддерживает программно-ориентированное взаимодействие. Частным лицам и компаниям, занимающимся коммерцией с использованием сети Интернет, необходим способ публикации ссылок на их приложения и данные, практически такой же, как при публикации ссылок на их веб-страницы. Интернет-приложения должны иметь возможность находить другие интернет-приложения, обращаться к ним и автоматически взаимодействовать с ними. Обеспечивая "общение" между программами, веб-сервисы расширяют возможности Интернета. За счет поддержки веб-сервисов размещенные в различных узлах Интернета приложения могут взаимодействовать непосредственно, как будто они являются частью одной крупной информационной системы.

**Основы веб-сервисов**

Веб-сервисы преобразуют XML-документы (Extensible Markup Language, XML) в ИТ-системах. Веб-сервисы - это XML-приложения, осуществляющие связывание данных с программами, объектами, базами данных либо с деловыми операциями целиком. Между веб-сервисом и программой осуществляется обмен XML-документами, оформленными в виде сообщений. Стандарты веб-сервисов определяют формат таких сообщений, интерфейс, которому передается сообщение, правила привязки содержания сообщения к реализующему сервис приложению и обратно, а также механизмы публикации и поиска интерфейсов.

Веб-сервисы могут использоваться во многих приложениях. Независимо от того, откуда запускаются веб-сервисы, с настольных компьютеров клиентов или с переносных, они могут использоваться для обращения к таким интернет-приложениям, как система предварительных заказов или контроля выполнения заказов. Веб-сервисы пригодны для В2В-интеграции (business-to-business), замыкая приложения, выполняемые различными организациями, в один производственный процесс. Веб-сервисы также могут решать более широкую проблему интеграции приложений предприятия (Enterprise Application Integration, EAI), осуществляя связь нескольких приложений одного предприятия с несколькими другими приложениями, размещенными как "до", так и "после" брандмауэра. Во всех перечисленных случаях технологии веб-сервисов являются "связующим звеном", объединяющим различные части программного обеспечения.

Как видно из рис. 1.1, веб-сервисы представляют собой оболочку, обеспечивающую стандартный способ взаимодействия с прикладными программными средами, такими как системы управления базами данных (СУБД), .NET, J2EE (Java2 Platform, Enterprise Edition), CORBA (Common Object Request Broker Architecture), посредники пакетов планирования ресурсов предприятия (Enterprise Resource Planning, ERP), брокеров интеграции и пр.



*Рис. 1.1. Веб-сервисы взаимодействуют с прикладными системами*

Интерфейсы веб-сервисов получают из сетевой среды стандартные XML-сообщения, преобразуют XML-данные в формат, "понимаемый" конкретной прикладной программной системой, и отправляют ответное сообщение (последнее - не обязательно). Программная реализация веб-сервисов (базовое программное обеспечение, нижний уровень) может быть создана на любом языке программирования с использованием любой операционной системы и любого связующего программного обеспечения (middleware).

Веб-сервисы объединяют программирование и концепции Сети. Веб-сервисы сочетают параметры программных приложений и абстрактные характеристики Сети. Современные интернет-технологии частично достигают своих целей, поскольку они определены на очень высоком отвлеченном уровне, что обеспечивает совместимость с любой операционной системой, любым программным и аппаратным обеспечением. Инфраструктура, основанная на применении веб-сервисов, пользуется этим уровнем абстракции и включает в себя связанную с данными семантическую информацию, то есть веб-сервисы определяют не только данные, но и порядок обработки и преобразования этих данных в базовые программные приложения и обратно.

**Простой пример: поиск информации**

В настоящее время большинство сервисов вызываются по Сети посредством ввода данных в HTML-формы и отправки этих данных сервису путем добавления их в строку унифицированного указателя информационного ресурса (Uniform Resource Locator, URL):

<http://www.google.com/search?q=Skate+boots&btnG=Google+Search>

Этот пример иллюстрирует простоту веб-взаимодействия (например, поиска, покупки акций или запроса маршрута движения), где параметры и ключевые слова внедряются непосредственно в URL. В данном случае представлен простой запрос поиска skate boots (ботинки с коньками) в строке обращения к поисковой машине Google. Ключевое слово search (искать) представляет сервис, к которому будет осуществлено обращение, а параметр Skate+boots является строкой поиска, которая была введена в HTML-форме на странице веб-сайта Google. Сервис поиска Google передаст этот запрос к различным поисковым машинам, которые вернут список URL для страниц, на которых имеется соответствие параметру поиска Skate+boots. Данный малоэффективный способ поиска в Сети полностью основан на установлении соответствия указанной текстовой строки и индексированных HTML-страниц.

XML - лучший способ отправки данных. XML предоставляет значительные преимущества при передаче данных через Интернет. Теперь предыдущий запрос можно представить в виде XML-документа:

<SOAP-ENV:Body>

<s:SearchRequest

xmlns:s="www.xmlbus.com/SearchService">

<pl>Skate</pl>

<p2>boots</p2>

<p3>size 7.5</p3>

</s:SearchRequest>

</SOAP-ENV:Body>

Отправка запроса в виде XML-документа имеет следующие преимущества: возможность определения типов данных и структур, большую гибкость и расширяемость. XML может представлять структурированные данные или данные определенного типа (например, допустимо указывать значение поля size (размер) как в виде строки цифр, так и в форме числа с плавающей точкой) и содержать больший объем информации, чем это допускает URL.

Данный пример представлен в форме SOAP-сообщения (Simple Object Access Protocol) - стандартной формы обмена XML-сообщениями - одной из технологий, лежащих в основе веб-сервисов. В SOAP-сообщении имя запрашиваемого сервиса и входные параметры представлены в виде отдельных XML-элементов. Рассматриваемый пример также иллюстрирует использование пространства имен XML (xmlns:), еще одного важного элемента веб-сервисов. Благодаря тому, что XML-документы поддерживают разные типы данных, сложные структуры и объединение схем, современные технологии веб-сервисов обеспечивают значительное преимущество над существующими возможностями обращения к программным приложениям посредством HTML и URL.

**Следующее поколение Сети**

Следующее поколение Сети будет основано на программно-ориентированных взаимодействиях. Веб-сервисы предполагают использовать созданные для взаимодействия людей глобальные сети совершенно в иных целях. Программно-ориентированные взаимодействия будут автоматически выполнять операции, которые ранее обязательно требовали "ручного" вмешательства:

· поиск и покупка товаров и услуг по самой выгодной цене;

· согласование заказов авиабилетов и мест в ресторане на определенную дату (планирование путешествий);

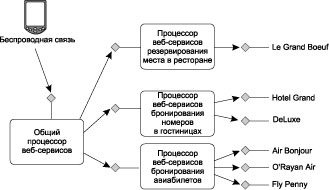
· оптимизация коммерческих операций закупки товаров, выписки счетов и доставки.

Следующее поколение Сети будет использовать программно-ориентированные сервисы для непосредственного взаимодействия с приложениями, построенными на основе любой комбинации объектов, программ и баз данных.

Веб-сервисы являются не только интерфейсом объектов, программ, связующего программного обеспечения и баз данных для доступа по Сети. Объединение ряда веб-сервисов позволяет осуществлять новые типы взаимодействия.

Веб-сервисы предоставляют новые варианты взаимодействия. Представим, например, что вы живете в Сан-Франциско и желаете зарезервировать столик в любимом парижском ресторане, а затем составить план путешествия, позволяющий оказаться там в необходимое время. В настоящее время для осуществления заказа необходимо непосредственно позвонить в ресторан, учитывая 9-часовую разницу во времени и различия в языке, а затем связаться с агентом бюро путешествий и подобрать подходящий рейс и гостиницу. Однако, используя веб-сервисы, можно спланировать обед с помощью календаря карманного компьютера (Personal Digital Assistant, PDA), а для автоматического резервирования столика на определенное время просто щелкнуть на кнопке. После выполнения резервирования веб-сервис запустит другие сервисы, которые забронируют наиболее дешевые авиабилеты и закажут номер в самом ближайшем четырехзвездочном отеле.

На рис. 1.2 представлен порядок осуществления взаимодействия веб-сервисов с карманным компьютером, подключенным по беспроводному каналу к процессору веб-сервисов, который для заказа столика в любимом ресторане использует веб-сервис этого ресторана. Процессор веб-сервисов получает запросы от функции "календарь" карманного компьютера и находит веб-сервисы, связанные с расширенной функцией календаря, такой как заказ столика. Для завершения планирования путешествия после успешного резервирования места в ресторане процессор веб-сервисов контактирует с веб-сервисами бронирования авиабилетов и номеров в гостиницах.

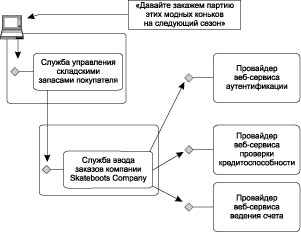


*Рис. 1.2. Приложения могут использовать веб-сервисы для резервирования места в ресторане   
и бронирования авиабилетов и номеров в гостиницах*

Веб-сервисы способны находить другие сервисы и взаимодействовать с ними. Веб-сервисы также полезны при поиске и осуществлении взаимодействия с сайтами, являющимися интерактивными системами заказов, например с системой продаж модных коньковых ботинок с выдвигающимися лезвиями.

Розничные продавцы спортивных товаров, заинтересованные в создании запасов таких ботинок, могут использовать веб-сервисы для размещения предварительных заказов, для проверки состояния выполнения заказа или размещения заказов сезонного возобновления запасов. Они также получат немедленный ответ при отсутствии коньков на складе. "Строительные" блоки веб-сервиса предоставляют стандартные компоненты конкретного приложения для компании Skateboots Company, которая не настолько велика, чтобы содержать свою собственную инфраструктуру приложений. Компания-владелец веб-сервиса также обеспечивает безопасность, гарантируя получение заказов компанией Skateboots Company только от уполномоченных оптовых покупателей, и предлагает сервисы проверки полномочий для крупных предварительных заказов. Другие компании помогут Skateboots Company создать электронный каталог и выполнить расчеты.

Целиком реализованная как веб-сервис система ввода заказов компании Skateboots зависит от Интернета, но необходимая функциональность обеспечивается за счет совместной работы ряда других веб-сервисов. На рис. 1.3 продемонстрировано, как веб-сервисы могут изменить способ построения и использования коммерческого приложения. Заинтересованный в закупке коньков оптовый покупатель вводит запрос с помощью собственной локальной системы управления складскими запасами, доступной компьютерам магазинов как веб-сервис. Локальная система управления складскими запасами контактирует через Интернет с веб-сервисом производителя и посылает ему заказ на определенное число коньков, основываясь на объеме имеющихся складских помещений и учитывая наиболее ходовые размеры ботинок.



*Рис. 1.3. Служба ввода заказов Skateboots Company содержит   
несколько других веб-сервисов*

Система ввода заказов компании Skateboots Company состоит из нескольких сервисов, включая специальную часть, которая отвечает уникальным аспектам ее продукции, и несколько торговых компонентов, выполняющих стандартные функции, например проверку полномочий пользователя, проверку кредитоспособности и отслеживание счета. Все эти сервисы предоставляются другими компаниями, специализирующимися на оказании таких услуг через Интернет.

Создание использующих веб-сервисы коммерческих приложений влечет за собой необходимость установления между этими сервисами соответствующих взаимоотношений, что реализуется с помощью любой комбинации языков программирования, операционных систем, пакетов программ, размещенных "до" или "после" брандмауэра. (Это также является способом решения сложных EAI-проблем.) Установление необходимых отношений (или потока управления) связанных веб-сервисов также приводит к автоматизации соответствующих бизнес-процессов и процедур.

Использование веб-сервисов очень выгодно с коммерческой точки зрения. За счет повсеместного распространения веб-сервисов Интернет становится более эффективным, особенно при осуществлении коммерческих сделок. Сочетая прямой доступ к программным приложениям и коммерческим документам, веб-сервисы следующего поколения Сети обеспечат полностью автоматическое взаимодействие, что позволит обращаться непосредственно к данным программ, игнорируя знакомые веб-страницы. Более того, основные компоненты веб-сервисов, скорее всего, будут предоставляться и публиковаться множеством различных компаний, специализирующихся на отдельных функциональных элементах (проверка полномочий, координация сделок, ведение счетов). Это обеспечит непосредственное взаимодействие "приложение-приложение" - принцип, лежащий в основе веб-сервисов и определяющий их суть и реализацию.

**ОБЩЕЕ ВЗАИМОПОНИМАНИЕ**

Технология веб-сервисов существует на очень высоком уровне абстракции, позволяя поддерживать множество одновременных определений, которые иногда бывают противоречивы. На простейшем уровне веб-сервисы могут восприниматься как интернет-ориентированные текстовые брокеры интеграции. Любые данные могут преобразовываться в ASCII-текст и обратно, и этот подход в течение долгого времени был общим знаменателем для систем графического вывода и систем управления базами данных. Как говорится, "если ничего не получается - преобразуй данные в текстовую форму". Ориентированные на использование текста системы также лежат в основе успешного развития Интернета, на котором базируется дополнительная абстракция веб-сервисов. Любой компьютер или операционная система может поддерживать HTML, браузеры и веб-сервисы; и при получении по сети файлов им совершенно безразлично и даже неизвестно, с каким типом прикладной системы они взаимодействуют.

То же самое можно сказать и о веб-сервисах, которые зачастую вызывают замешательство, когда разработчики традиционного, устоявшегося компьютерного окружения пытаются воспринимать веб-сервисы как отдельный тип распределенной программной среды, такой как CORBA, J2EE или .NET. Поскольку веб-сервисы гораздо более абстрактны - как посредники (преобразователи), а не как интерфейсы - так и будет продолжаться, пока они не будут приведены в соответствие с общими описаниями и соглашениями.

**Взаимодействие с веб-сервисами**

Веб-сервисы поддерживают несколько парадигм обмена сообщениями. Уровень абстракции, на котором оперируют веб-сервисы, подразумевает такие стили взаимодействия, как эмуляцию удаленного вызова процедуры (Remote Procedure Call, RPC), асинхронный обмен сообщениями, однонаправленную передачу сообщений, широковещание и публикацию/подписку. Основные СУБД, такие как Oracle, SQL Server и DB2, поддерживают анализ XML и службы преобразования, обеспечивая непосредственное взаимодействие между веб-сервисами и СУБД. Производители связующего программного обеспечения обычно также предоставляют возможность привязки веб-сервисов к своим программным системам (серверам приложений и брокерам интеграции). Следовательно, для пользователя взаимодействие с веб-сервисами может проявляться в интерактивной или пакетной форме, поддерживающей синхронную и асинхронную модели связи; а также как пользовательский интерфейс, написанный с использованием Java, VB (Visual Basic), офисных приложений, браузеров или "толстых" клиентов СУБД. Такое взаимодействие может привязываться к любому типу базовой (более низкого уровня) программной системы.

Веб-сервисы выполняют RPC- и документно-ориентированное взаимодействия. Стандарты и технологии веб-сервисов обычно подразумевают два основных типа моделей взаимодействия приложений:

· удаленный вызов процедуры (онлайновая);

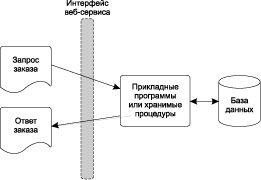
· документно-ориентированный (пакетная).

Эти два типа взаимодействия мы и рассмотрим в последующих разделах.

**RPC-ориентированные взаимодействия**

RPC-ориентированные взаимодействия удобны для краткого обмена данными. В RPC-ориентированном взаимодействии запросы веб-сервисов приобретают форму вызова метода или процедуры с соответствующими входными или выходными параметрами. В отличие от документно-ориентированного взаимодействия, RPC-ориентированное взаимодействие производит отправку документа, специально отформатированного для передачи в отдельную логическую программу или базу данных (рис. 1.4). Поскольку, например, заказ коньков в режиме реального времени зависит от их наличия на складе, программа обращается к базе данных с проверкой наличия заказываемого товара. Если получено подтверждение, то программа вернет заказчику XML-документ в формате "запрос/ответ", сообщающий о принятии заказа с его последующим исполнением.

Если поставка невозможна, будет получено сообщение об отсрочке выполнения заказа или о полном отказе от его выполнения. В отличие от документно-ориентированного стиля взаимодействия, запрос и ответ моделируются как синхронные сообщения, то есть приложение, посылающее сообщение, ждет реакции на него.

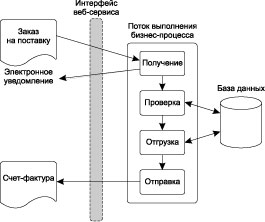


*Рис. 1.4. Веб-сервисы поддерживают интерактивный   
заказ в форме запроса/ответа*

**Документно-ориентированные взаимодействия**

Документно-ориентированные взаимодействия удобны для обмена большими объемами данных. При документно-ориентированном взаимодействии запросы веб-сервиса имеют форму завершенного XML-документа, предназначенного для обработки целиком. Например, веб-сервис, который представляет заказ на поставку (оптовый предварительный заказ на поставку коньков), должен сразу предъявлять производителю полную форму заказа (рис. 1.5). Это напоминает помещение сообщения в очередь для асинхронной обработки. Производитель обычно по электронной почте либо в какой-либо иной форме высылает заказчику подтверждение, свидетельствующее о том, что заказ принят и будет выполнен в соответствии с предопределенной последовательностью выполнения бизнес-процесса. Последовательность выполнения может содержать такие этапы, как проверка в базе данных предыдущих заказов данного оптового покупателя на предмет исчерпания кредитного лимита, согласованных объемов или графика поставок по данному заказу. В реальном потоке выполнения заказа и отправки счета, конечно же, существуют и другие этапы, но в данном примере рассматривается только заключительная стадия: отправка XML-счета для оплаты после доставки и получения заказа.

Необходимые взаимодействия определяются в договоре торговых партнеров. Документно-ориентированные взаимодействия зачастую предполагают, что использующие веб-сервисы стороны заранее согласовали порядок оформления общих документов, таких как заказ на приобретение, счет за доставку или общий счет. Эти стороны обычно идентифицируются как "торговые партнеры" или "сотрудничающие партнеры". Торговые партнеры также обыкновенно согласовывают общий поток выполнения процесса или модель взаимодействия при обмене документами, например, оговаривают необходимость подтверждения квитанции заказа на приобретение, передачу специальной информации о состоянии в ответ на запрос заказа или отправку сигнала оповещения по электронной почте после отгрузки заказа. В ходе реализации бизнес-процесса необходим обмен полными документами. Если до этого документ содержал общую, фрагментированную информацию, то теперь требуется согласованное заполнение специальных разделов, таких как цена покупки или обязательная дата доставки.



*Рис. 1.5. Веб-сервис обрабатывает полный заказ на поставку*

Для синхронных/асинхронных парадигм обмена сообщений удобны два стиля привязки. В нашем примере предварительный оптовый заказ выполняется в компании Skateboots Company с использованием заказов на покупку, представляемых в виде пакетов в соответствии с предварительными договоренностями, что помогает производителю спланировать объемы. В период активного спроса заказы на немедленное пополнение обрабатываются более интерактивными сервисами, которые зависят от заказов на пополнение с имеющихся складов и позволяют тотчас идентифицировать отложенные заказы. Таким образом, сайт Skateboots.com предоставляет веб-сервисы, поддерживающие оба основных типа взаимодействия.

**Технология веб-сервисов**

Порядок описания, поиска и взаимодействия веб-сервисов друг с другом определяют стандарты. Взаимодействующие через Интернет программы должны уметь обнаруживать друг друга, находить информацию, позволяющую им осуществить связь, понимать, какая модель контактирования должна быть применена (простая, типа "запрос/ответ", или более сложная последовательность), и договариваться об использовании таких услуг, как защита информации, подтверждение передачи сообщений и составление сделок. Некоторые из этих сервисов реализуются существующими технологиями и предлагаемыми стандартами, а другие - нет. Использующее веб-сервисы сообщество стремится удовлетворить все эти требования, но это - эволюционный процесс, как и сам Интернет. С самого начала инфраструктура и стандарты веб-сервисов подразумевали возможность расширения (так же как до них XML и HTML), что позволяет использовать их сразу же после появления новых стандартов и технологий.

Веб-сервисы требуют использования нескольких смежных XML-технологий. Для транспортировки и преобразования данных в программы и обратно веб-сервисы требуют использования нескольких смежных XML-технологий.

· Язык XML (Extensible Markup Language) - фундамент, на котором строятся веб-сервисы. Он предоставляет язык определения данных и порядок их обработки. XML представляет семейство связанных спецификаций, публикуемых и поддерживаемых интернет-консорциумом (World Wide Web Consortium, W3C) и другими организациями.

· WSDL (Web Services Description Language) - технология, основанная на XML, определяющая интерфейсы веб-сервисов, типы данных и сообщений, а также модели взаимодействия и протоколы связывания.

**НОВАЯ "СЕРЕБРЯНАЯ ПУЛЯ"?**

С точки зрения современных проблем вычислительной техники веб-сервисы иногда представляются как решение типа "серебряная пуля", исполняя роль, которая ранее отводилась Сети, реляционным базам данных, языкам программирования 4-го поколения и искусственному интеллекту. К сожалению, сами по себе веб-сервисы не в состоянии решить многие проблемы. Они представляют новый уровень (или другой способ выполнения), а не фундаментальное изменение, которое приходит на смену существующей компьютерной инфраструктуре. Этот новый технологический уровень выполняет новую функцию, но, что более важно, обеспечивает механизм интеграции, определенный на более высоком уровне абстракции.

Важность веб-сервисов заключается в том, что они могут служить "связующим мостом", а не заменой существующих технологий. Можно сказать, что новые языки программирования, такие как Visual Basic, C#, C/C++ и Java, пришли на смену старым языкам, таким как COBOL и FORTRAN, хотя множество программ на этих языках все еще используются, как и привязки веб-сервисов для них. В качестве веб-серверов веб-сервисы являются дополнением и не конфликтуют с существующими приложениями, программами и базами данных. Разработка приложений по-прежнему требует знаний Java, VB и C#. Все, что является новым, - это способ преобразования данных для передачи в приложение и обратно на основе стандартных форматов XML-данных и протоколов, что позволяет обеспечить новый уровень интеграции и взаимодействия.

При проектировании и разработке новых программ и баз данных разработчики могут учитывать веб-сервисы, но эти программы и базы данных все равно потребуют использования упаковщиков веб-сервисов. Веб-сервисы сами по себе не являются исполняемыми; они полагаются на исполняемые программы, написанные с помощью языков программирования или сценариев. Веб-сервисы определяют мощный уровень абстракции, который может применяться для межпрограммного взаимодействия на основе существующей сетевой инфраструктуры, но без этой инфраструктуры они ничего не значат.

· Протокол SOAP (Simple Object Access Protocol) - совокупность XML-технологий, определяющих "конверт" для связи веб-сервисов (привязанный к HTTP и другим транспортам), - специализированный формат для передачи XML-документов по Сети, а также соглашения RPC-взаимодействий.

· Технология UDDI (Universal Description, Discovery and Integration) - реестр веб-сервисов и механизм поиска. Он используется для хранения и упорядочения деловой информации, а также для нахождения указателей на интерфейсы веб-сервисов.

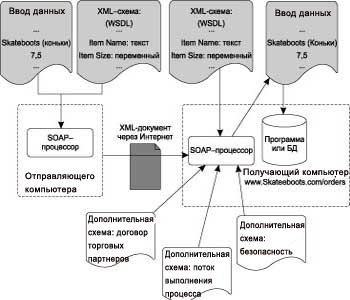
**Пример использования**

Стандарты веб-сервисов обычно используются совместно. Основные стандарты веб-сервисов используются согласованно. После обнаружения WSDL в UDDI или другом месте генерируется SOAP-сообщение для отправки на удаленный сайт.

Как видно из рис. 1.6, при предоставлении документа по адресу веб-сервиса программа использует XML-схему определенного типа (такую как WSDL), позволяющую преобразовать данные из ее входного источника (в этом примере структурированный файл) и на основе того же WSDL-файла создать экземпляр XML-документа в формате, согласованном с целевым веб-сервисом. WSDL-файл используется для определения как входного, так и выходного преобразования данных.

SOAP-процессор отправляющего компьютера преобразует данные из собственного ("родного") формата в тип данных, предопределенный в соответствии с содержащейся в WSDL-файле XML-схемой на основе таблиц преобразования для текстов, значений с плавающей точкой и других данных. Таблицы преобразования "связывают" собственные типы данных с соответствующими конкретной XML-схеме. (Стандартное преобразование типов широко используется в Java, Visual Basic, CORBA и других известных системах. Многие средства XML позволяют настраивать специальные преобразования типов.) SOAP-процессор получающего компьютера выполняет обратное преобразование данных из типов XML-схемы в собственные типы данных.

Файлы описаний веб-сервисов обычно регистрируются с помощью URL. URL, повсеместно используемый в Сети, указывает на IP-адрес, соответствующий веб-ресурсу. Схемы веб-сервисов являются одной из форм веб-ресурса, они содержатся в доступных через Интернет файлах и к ним применим тот же механизм, что используется при загрузке HTML-файлов. Главное отличие между загрузкой HTML-файла и обращением к ресурсу веб-сервиса заключается в том, что веб-сервис оперирует XML-документами, а не HTML-документами и опирается на соответствующие технологии, такие как использование схем, преобразование, проверка подлинности, что и обеспечивает поддержку удаленного соединения приложений. Но способ, согласно которому схемы веб- сервисов публикуются и загружаются, одинаков: HTTP-операция по указанному URL.



*Рис. 1.6. Веб-сервисы используют XML-документы и   
осуществляют преобразование данных*

Для проверки достоверности сообщений веб-сервисы используют XML-схемы. После получения документа реализация веб-сервиса сначала должна проанализировать XML-сообщение и удостовериться в корректности данных, выполнить проверку качества услуг (Quality-of-Service), такую как проверку политики безопасности или соглашений торговых партнеров, а затем произвести последовательность связанных с данным документом коммерческих операций. Веб-сервис на вымышленном нами сайте skateboots.com размещен в папке skateboots.com/ order, на которую и указывает URL.

Веб-сервис, доступный по данному интернет-адресу, идентифицируется с помощью публичного WSDL-файла, который может быть загружен на отправляющий компьютер и использоваться при генерации сообщения. Компания Skateboots Company также осуществляет отправку в общедоступный каталог UDDI-листинга, позволяющего клиентам находить компанию с помощью технологии UDDI. В общем случае, любой, кто хочет взаимодействовать с веб-сервисом, размещающим или контролирующим по Сети заказы для Skateboots Company, для генерации сообщения должен найти способ получения и использования WSDL-файла.

Размещенные по адресу skateboots.com программы представляют собой прослушивающий HTTP-процесс, связанный с соответствующими веб-сервисами для распознавания XML-сообщений, определенных в данном формате. Эти программы включают в себя XML-анализаторы и преобразователи. Кроме того, они осуществляют конвертацию данных SOAP-сообщения в форматы, необходимые для системы ввода заказов компании Skateboots Company.

Технологии веб-сервисов сформировались из основных структур. Этих технологий достаточно для построения, развертывания и публикации базовых веб-сервисов. На самом деле, необходим лишь базовый протокол SOAP. С момента появления веб-сервисов к ним все время добавляются другие технологии. Хотя для деловой связи, а также для создания "моста" с несовместимыми технологиями вполне достаточно базовых принципов, данная форма веб-взаимодействия тем не менее была одобрена очень быстро.

**WSDL: описание веб-сервисов**

Язык описания веб-сервисов (Web Services Description Language, WSDL) - это формат XML-схем, определяющий расширенную структуру описания интерфейсов веб-сервисов. WSDL первоначально был разработан компаниями Microsoft и IBM. А затем его поддержали консорциум W3C и 25 компаний. WSDL - это сердце структуры веб-сервиса. Это общий способ представления передаваемых в сообщениях типов данных, указывающий действия, которые должны быть выполнены с данным сообщением и согласно которому сообщения привязываются к сетевым транспортам.

WSDL - это XML-формат, описывающий состав веб-сервиса. WSDL предназначен для использования как в процедурно-ориентированных, так и в документно-ориентированных приложениях. Так же как и другие XML-технологии, WSDL является расширяемым языком и имеет такое количество параметров, что обеспечение совместимости при организации взаимодействия между различными реализациями может вызвать сложности. Полное взаимопонимание возможно лишь в том случае, если отправитель и получатель сообщения могут совместно использовать и одинаково интерпретировать один и тот же WSDL-файл.

WSDL в соответствии с уровнем абстрагирования состоит из трех элементов. WSDL можно разделить на три основные составляющие:

· определение типов данных;

· абстрактные операции;

· связывание сервисов.

Каждая составляющая может быть указана в различных XML-документах и импортирована в виде различных комбинаций, что позволяет создавать окончательное описание веб-сервиса. Кроме того, все составляющие могут быть представлены и в одном XML-документе. "Определение типов данных" задает структуру и содержание сообщений. "Абстрактные операции" определяют операции, которые должны быть выполнены с содержанием сообщения, а "связывание сервисов" подразумевает сетевой транспорт, который доставит сообщение по месту назначения.

На рис. 1.7 представлены составляющие WSDL, упорядоченные в соответствии с их уровнем абстракции, который определяется независимо от транспорта, особенно когда для одного сервиса используется множество транспортов. Например, к одному и тому же сервису можно обращаться посредством "SOAP поверх HTTP" или "SOAP поверх JMS". Аналогично, определения типов данных размещены в разных разделах, что позволяет использовать их различными сервисами. Основные WSDL-составляющие разделены на компоненты.



*Рис. 1.7. WSDL состоит из трех основных составляющих  
и семи компонентов*

WSDL-интерфейсы подобны интерфейсам CORBA и DCOM. Части описания содержат описания типов данных, сообщения и абстрактные операции, которые схожи с интерфейсами CORBA и DCOM. Сообщения могут состоять из нескольких секций и могут быть предназначены для использования в случае процедурно-ориентированного стиля взаимодействия, документно-ориентированного стиля или при применении обоих стилей. В зависимости от уровня абстрагирования одни и те же сообщения могут определяться и использоваться для нескольких типов портов. Как и другие части WSDL, сообщения также поддерживают компоненты расширения, в частности, для включения в себя атрибутов других сообщений.

Типы данных веб-сервисов основаны на XML-схемах, но могут быть распространены на любые другие механизмы. WSDL-описания типов данных базируются на XML-схемах, но здесь годятся для использования и другие эквивалентные или аналогичные системы определения типов данных. Например, вместо типов данных XML-схемы может применяться язык описания интерфейса (Interface Definition Language, IDL) CORBA. (Если используется иная модель определения типов, ее должны "понимать" обе стороны.)

Абстрактные операции и сообщения отображаются на специальный транспорт. Связывание сервисов отображает абстрактные сообщения и операции на специальный транспорт, такой как SOAP. Для включения сведений, характерных для SOAP и других типов отображения, используется связывание расширяемых компонентов. Абстрактные определения могут привязываться к различным физическим транспортам. Спецификация WSDL v1.1 включает примеры однонаправленного SOAP-отображения для SMTP (Simple Mail Transfer Protocol), SOAP RPC-привязки к HTTP, SOAP-привязки к методам get и post протокола HTTP, а также пример отображения для MIME (Multipurpose Internet Mail Extensions) - многокомпонентного связывания SOAP.

Пространство имен гарантирует уникальность имен WSDL-элементов. Пространство имен XML предназначено для обеспечения уникальности имен, используемых в любой из трех основных WSDL-составляющих. Конечно же, при раздельной разработке WSDL-элементов и импортировании их в один файл описанные в отдельных файлах пространства имен не должны перекрываться. Связанные схемы используются для проверки достоверности как WSDL-файла и сообщений, так и для операций, определенных в WSDL-файле.

Можно смело сказать, что по мере развития веб-сервисов WSDL, по всей видимости, будет включать множество расширений, изменений и добавлений. Как и SOAP, WSDL разработан в виде расширяемой XML-структуры, которая может быть легко модифицирована для привязки множества типов данных, определений типов сообщений, операций и транспортов. Например, рабочие группы IETF (Internet Engineering Task Force) предлагают новый стандарт протоколов - Blocks Extensible Exchange Protocol (BEEP), - позволяющий определять транспорт с установлением соединений. (В отличие от HTTP, ориентированного на связь без установления соединения, что усложняет решение проблем качества обслуживания на транспортном уровне.) Компании, интересующиеся использованием веб-сервисов во внутренних приложениях или для осуществления интеграции, могут расширить WSDL, обеспечив его привязку к более традиционным протоколам, таким как DCOM или IIOP (Internet Inter-Orb Protocol).

**SOAP: доступ к веб-сервисам**

До сего момента вы определяли данные (XML) и абстрактно представили службу, необходимую для обеспечения соединения и обработки сообщения (WSDL). Теперь же необходимо уточнить, каким способом сообщение будет пересылаться с одного узла на другой, что позволит провести его обработку на конечном компьютере.

SOAP обеспечивает механизм подключения к веб-сервисам. Спецификация протокола SOAP определяет структуру сообщения, используемого для обмена форматированными XML-данными посредством сети Интернет. Структура сообщения достаточно проста, удобна при разработке и полностью нейтральна по отношению к операционной системе, языку программирования и компьютерной платформе. Протокол SOAP предназначен для обеспечения транспорта базового уровня, на основе которого могут возводиться более сложные протоколы и способы взаимодействия.

SOAP - это XML-способ определения: какая информация должна пересылаться и как. SOAP - это базовая однонаправленная модель соединения, обеспечивающая согласованную передачу сообщения от отправителя к получателю, потенциально допускающая наличие посредников, которые могут обрабатывать часть сообщения или добавлять к нему дополнительные элементы. Спецификация SOAP содержит соглашения по преобразованию однонаправленного обмена сообщениями в соответствии с принципом "запрос/ответ", популярным при организации соединения типа RPC, а также определяет, как осуществлять передачу всего XML-документа.

SOAP включает дополнительные правила кодирования для различных типов данных, но конечные узлы SOAP-соединения могут определить собственные правила посредством конфиденциального соглашения сторон. В ходе связи зачастую используется буквенное или встроенное в XML кодирование.

Как видно из рис. 1.8, SOAP предназначена для поддержания независимого абстрактного протокола связи, обеспечивающего коммуникацию двух и более предприятий или двух и более удаленных корпоративных сайтов. Связанные системы можно построить на основе любой комбинации аппаратного и программного обеспечения, поддерживающей доступ через Интернет к таким средам, как .NET и J2EE. Существующие системы обычно также представлены множественными инфраструктурами и пакетами программ. SOAP и остальные структуры XML предоставляют возможность двум и более предприятиям, торговым площадкам или партнерам выработать общий подход по вопросу предоставления услуг в Сети.

SOAP-сообщения содержат конверт, заголовок и тело сообщения. SOAP-сообщения состоят из нескольких основных частей.

Envelope (конверт) - определяет начало и конец сообщения.

Header (заголовок) - содержит любые дополнительные атрибуты сообщения, используемые в ходе обработки сообщения как посредником, так и конечным получателем.

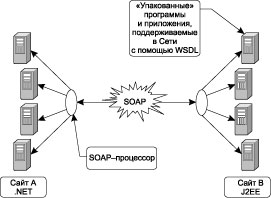
Body (тело сообщения) - содержит XML-данные, передаваемые данным сообщением.

Attachment (вложение) - состоит из одного и более документов, "прикрепленных" к основному сообщению. (Относится только к SOAP with Attachments ("SOAP с вложениями").)

RPC interaction (SOAP:RPC-взаимодействие) - определяет, как моделировать взаимодействия RPC-типа.

Encoding (кодировка) - определяет, как будут представлены простые и сложные данные, передаваемые в сообщении.

Обязательными являются только конверт и тело сообщения.



*Рис. 1.8. SOAP-сообщения осуществляют соединение удаленных узлов*

**UDDI: публикация и поиск веб-сервисов**

После того как вы определили данные в сообщении (XML), описали сервисы, которые будут принимать и обрабатывать сообщения (WSDL), и предъявили требования к средству их передачи и приема (SOAP), вам необходимы способы публикации предлагаемой вами службы, а также инструменты поиска нужных вам, но предоставляемых другими участниками услуг. Именно эти функции и выполняет реестр UDDI (universal distribution, discovery, and integration).

UDDI регистрирует и публикует определения веб-сервисов. Структура UDDI определяет модель данных в программных интерфейсах (API) XML и SOAP для регистрации и обнаружения коммерческой информации, включая веб-сервисы. Реестр UDDI был создан независимым консорциумом производителей (основанным компаниями Microsoft, IBM и Ariba), занимающимся разработкой стандарта для описания, регистрации и локализации веб-сервисов. Microsoft, IBM, Hewlett- Packard и SAP на начальном этапе предоставляют общедоступный UDDI-сервис, который концептуально копирует DNS (систему имен доменов Интернета, осуществляющую "перевод" имен Интернета в IP-адреса). На практике UDDI более похож на сервис реплицируемой базы данных, доступной через Интернет.

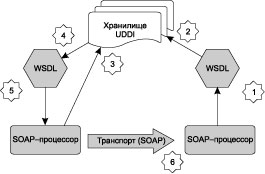
UDDI - это каталог веб-сервисов. UDDI идейно подобен каталогу "Желтые страницы". Предприниматели регистрируют свою контактную информацию, в том числе номер телефона, факса, почтовый адрес и адрес веб-сайта. Регистрация включает различные необходимые для поиска категории информации, такие как географическое положение, код отрасли промышленности, тип бизнеса и т. д. Другие предприниматели при поиске поставщиков комплектующих, служб доставки, а также аукционов и торгов могут найти нужные данные, сохраненные в UDDI. Можно также обнаружить информацию об определенном веб-сервисе обычным путем, пытаясь найти URL WSDL-файла, указывающего на веб-сервис поставщика.

Для регистрации и поиска информации UDDI использует SOAP. Для регистрации предпринимателей в UDDI служит SOAP; далее клиенты реестра для поиска информации необходимого им торгового партнера используют программный интерфейс SOAP-запроса. Первоначальный запрос может вернуть несколько соответствий, из которых может быть выбрано только одно. После выбора необходимого элемента для получения необходимой для осуществления сделки конкретной контактной информации производится заключительный вызов API.

На рис. 1.9 показано, как предприятие будет осуществлять регистрацию в UDDI сведений о веб-сервисе.

Сначала предприятие генерирует WSDL-файл, который описывает веб-сервисы, поддерживаемые SOAP-процессором (1), и с помощью API UDDI регистрирует информацию в хранилище (2). После отправки предприятием этих сведений (совместно с другой контактной информацией) элемент реестра будет содержать URL, указывающий на WSDL-файл сайта SOAP-сервера или на другой файл XML-схемы, описывающий данный веб-сервис.

После этого SOAP-процессор другого предприятия запрашивает реестр (3) для получения WSDL или другой схемы (4). Для отправки конкретной операции по определенному протоколу (6) клиент может сгенерировать соответствующее сообщение (5). Конечно же, клиент и сервер должны договориться о том, чтобы протокол был одинаковым (в данном примере "SOAP поверх HTTP"), и совместно использовать одинаковые понятия или семантические описания сервиса, который в данном примере представлен WSDL. При повсеместном распространении основных стандартов, скорее всего, общее понимание будет гарантироваться посредством WSDL.



*Рис. 1.9. Для обнаружения веб-сервиса может   
использоваться репозиторий UDDI*

# Национальная инфраструктура информационной безопасности.

Инфраструктура национальной информационной безопасности (ИНИБ)представляет собой комплекс взаимосвязанных обслуживающих систем различной природы (организационных, правовых, информационных и т.п.) обеспечивающих основу для решения задачи информационной безопасности (ИБ) в масштабах страны. ИНИБ является составной частью общей системы национальной безопасности страны. На рис. 1 представлена обобщенная схема ИНИБ, исторически сложившаяся в большинстве развитых стран. На рисунке изображены основные элементы ИНИБ (замкнутые фигуры, различной геометрии) и подписанные стрелки, указывающие тип взаимоотношения между парами элементов.

Основой любой ИНИБ является законодательная и нормативно-правовая база обеспечения информационной безопасности (ЗБИБ), являющаяся выражением государственной политики в этой сфере. Государство исходит из того, что информационные ресурсы являются объектами собственности, участвующими в хозяйственном обороте. Законы и правовые акты ЗБИБ наделяют определенные органы государственной власти полномочиями осуществлять государственное регулирование, контроль (мониторинг) и управление в сфере информационной защиты, а также определяют взаимоотношение субъектов информационной деятельности. Технические нормативные акты представляют собой стандарты и технические регламенты, которые могут (или обязаны) использовать субъекты информационной деятельности. Ответственность субъектов информационной деятельности за нарушение законов в сфере информационной безопасности определяется Уголовным и Административным кодексами.

Условно субъекты в сфере информационной безопасности можно разбить на три группы: государственные учреждения, осуществляющие государственное регулирование в этой сфере (группа Г); разработчики технических средств защиты информации (группа Р) и пользователи (группа П). Очевидно, что все эти группы пересекаются.

Полномочия и деятельность субъектов группы Г, определена законом или специальными положениями. Группа Г координирует и лицензирует всю деятельность по технической защите информации, разрабатывает (или участвует в подготовке) нормативных правовых актов, осуществляет экспертизу (сертификацию) технических средств информационной безопасности. Кроме того, субъекты этой группы осуществляют управление государственными информационными системами, связанными с информационной безопасностью (например, удостоверяющие и регистрационные центры).

Основной деятельностью субъекты группы P является разработка технических средств защиты информации. Свою деятельность они осуществляют на основе лицензии, а все созданные ими средства защиты должны пройти сертификацию, прежде, чем они будут применяться пользователями. Сертификация – это процедура подтверждения соответствия продукции определенным техническим нормативным документам (стандартам).

Пользователи информационных систем (группа П) представляют самую многочисленную группу субъектов в сфере информационной безопасности. Субъекты этой группы применяют средства технической защиты информации для защиты собственных информационных ресурсов и взаимодействуют с системами защиты других субъектов. В своей деятельности они могут использовать государственные автоматизированные системы, обеспечивающие сервисные услуги в области технической защиты информации.

Исторически первой ИНИБ сложилась в США. Первый закон о защите информации датируется 1906 годом, а на сегодняшний день ЗБИБ США насчитывает их более 500 [5, 6].

Появление глобальных информационных систем привело к необходимости выработки единого подхода к проблемам информационной безопасности на международном уровне.

Первый важный шаг, который сделан в этом направлении, является разработка международных стандартов по информационной безопасности.

По понятным причинам основой международных стандартов по информационной безопасности стали стандарты, разработанные стандартизирующими организациями ведущих индустриальных стран (прежде всего США и Великобритании). Нормативная документация национальной ЗБИБ, как правило, являются гармонизированными международными стандартами.

# Безопасность в сетях: конфиденциальность, аутентификация, обеспечение строгого выполнения обязательств, авторизация, обеспечение целостности, криптография, криптоанализ, криптология, шифр, код, ключ шифра, IPsec, SSL/TSL, HTTPS, DNSsec.

**Безопасность в сетях**

**Конфиденциальность** –предотвращение попадания информации неавторизованным пользователям.

**Аутентификация** - проверка принадлежности субъекту предъявленного им идентификатора (подтверждение личности). Процесс аутентификации может осуществляться различным способом:

* логин и пароль;
* электронный сертификат;
* старт-карта;
* идентификация личности по биометрическим данным.

**Обеспечение строгого выполнения обязательств** - электронная подпись документа, гарантирующая подлинность отправителя.

**Авторизация** -процесспроверки прав субъекта на выполнение некоторых действий (говорят: пользователь авторизовался в системе). Иногдапроцесс предоставление прав доступа (говорят: администратор системы авторизует пользователя – наделяет его перечнем прав и полномочий).

**Обеспечение целостности** – методы защиты электронных документов от модификации в процессе передаче по сети.

**Криптография** (греч. скрытое письмо) – наука о методах обеспечения конфиденциальности (секретности) и аутентичности (целостности и невозможность отказа от авторства).

**Криптоанализ** – наука, изучающая методы разрушения конфиденциальности и целостности;

**Криптология** = криптография + криптоанализ

**Шифр** – побитовое или посимвольное преобразование независимое от лингвистической особенности сообщения.

**Ключ** – параметр шифра, определяющий преобразование открытого текста.

**Код** – заменяет слово языка другим словом или символом (американцы использовали язык индейского племени навахо (использовали носителей языка) в ходе переговоров на войне с Японией). Электронная цифровая подпись – реквизит электронного документа, предназначенный для защиты документа от подделки и позволяющий установить подлинность отправителя.

**IPsec (RFC 2401-2412) –** набор протоколов длязащиты данных передаваемых с помощью IP.IPsec встроен в **IPv6** и является его неотъемлемой частью. **IPsec** состоит: 1)протоколы для защиты потока - **AH** (**Authentication Header,** гарантируется целостность, но данные не шифруются) и **ESP (Encapsulation Security Payload**, обеспечивает целостность и шифрование**);** 2) протокол обмена криптографическими ключами **IKE** **(Internet Key Exchange)**.

**IPSec** — это комплекс протоколов, касающихся вопросов шифрования, аутентификации и обеспечения защиты при транспортировке IP-пакетов; в его состав на данный момент входят около 20ти предложений по стандартам и 18ти RFC.

Большинство современных (**2007 год**) реализаций IPsec основано на **RFC 2401-2412**.

Вот наиболее использующие в настоящее время:

|  |  |  |
| --- | --- | --- |
| **RFC**  **Количество** | **Имя** | **Описание** |
| **2401** | Архитектура безопасности для Интернет-протокола | Основной документ, описывающий архитектуру IPSec. |
| **2402** | Аутентификация IP заголовка | Определяет AH протокол, используется для обеспечения целостности данных. |
| **2403** | Использование HMAC-MD5-96 в пределах ESP и AH | Описывает особенности алгоритма шифрования для использования AH и ESP называемой сообщения Сборник 5 (MD5). |
| **2404** | Использование HMAC-SHA-1-96 в пределах ESP и AH | Описывает особенности алгоритма шифрования для использования AH и ESP призвал Безопасный алгоритм хеширования 1 (SHA-1). |
| **2406** | IP Encapsulating безопасности полезной нагрузки (ESP) | Описывает IPSec Инкапсуляция безопасности полезной нагрузки (ESP) протокол, который обеспечивает шифрование данных для сохранения конфиденциальности. |
| **2408** | Интернет Ассоциация безопасности и управления ключами протокола (ISAKMP) | Определяет методы для обмена ключами и переговорных безопасности ассоциаций. |
| **2409** | IKE | Описывает IKE протокол используется для ведения переговоров безопасности ассоциациями и обмен ключами между устройствами для безопасной связи. Основываясь на ISAKMP и Oakley. |
| **2412** | Oakley ключевые определения протокола | Описывает общий протокол для обмена ключом. |

Средства безопасности для IP описываются семейством спецификаций IPsec. Следовательно, IPSec работает на третьем уровне модели OSI – сетевом, в результате чего защита передаваемых IP-пакетов становится прозрачной для сетевых приложений т.о. IPSec призван обеспечить низкоуровневую защиту.

**Основными функциями IPSec являются:**

* Обеспечение конфиденциальности.

Отправитель должен иметь возможность шифровать пакеты перед тем, как передавать их по сети.

* Обеспечение целостности.

Получатель должен иметь возможность аутентифицировать стороны, учавствующие в процессе обмена информацией, и пакеты IPSec, посылаемые этими сторонами, дабы быть уверенным в том, что передаваемые данные не были изменены в пути.

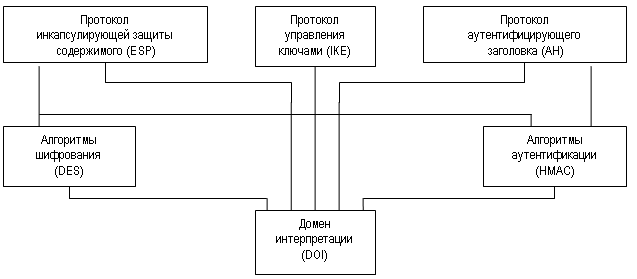
* Обеспечение защиты от воспроизведения пакетов.

Получатель должен иметь возможность обнаруживать и отбрасывать воспроизведенные пакеты, исключая таким образом проведение атак внедрения посредника.

**В комплекс спецификации IPSec входят следующие протоколы и стандарты:**

* **IKE**, обеспечивающий аутентификацию сторон, согласование параметов ассоциаций защиты (SA), а так же выбор ключей шифрования.
* **AH**, обеспечивающий аутентификацию пакетов и выявление их воспроизведения.
* **ESP** - обеспечивает конфиденциальность, аутентификацию источника и целостность данных, а также сервис защиты от воспроизведения пакетов.
* **HMAC** - механизм аутентификации сообщений с использованием хэш- функций.
* **DES**, 3DES – стандарты шифрования данных.

Взаимосвязи между протоколами представлены на рисунке 1:



**Рис 1**. Архитектура IPSec

**Режимы работы IPSec**

Существует два режима работы IPsec: **транспортный режим** и **туннельный режим**.

В транспортном режиме шифруется только информативная часть IP-пакета. Заголовок IP-пакета не изменяется (не шифруется). Транспортный режим как правило используется для установления соединения между хостами. Он может также использоваться между шлюзами, для защиты туннелей, организованных каким-нибудь другим способом (IP tunnel и др.).

В туннельном режиме IP-пакет шифруется целиком. Для того чтобы его можно было передать по сети, он помещается в другой IP-пакет. По существу, это защищённый IP-туннель. Туннельный режим может использоваться для подключения удалённых компьютеров к виртуальной частной сети или для организации безопасной передачи данных через открытые каналы связи (например, Интернет).

Режимы IPSec не являются взаимоисключающими. На одном и том же узле некоторые SA могут использовать транспортный режим, а другие — туннельный.

**Принципы работы**

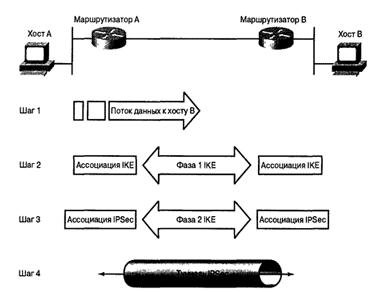
Целостность и конфиденциальность данных посредством IPSec обеспечивается за счет реализации механизмов аутентификации сторон и пакетов, а также шифрования, которые, в свою очередь, требуют предварительного согласования сторонами параметров информационного обмена – так называемого «контекста безопасности»- используемых криптографических алгоритмов, протоколов управления ключами и их параметров.

**Спецификация IPSec** является алгоритмонезависимой, то есть предусматривает возможность использования сторонами нескольких протоколов и параметров аутентификации и шифрования, различных схем распределения ключей. Результатом согласования контекста безопасности является индекс параметров безопасности (SPI), который представляет собой указатель на определенный элемент базы данных политики безопасности (SPD). На основании информации, содержащейся в SPD, для пакета данных может быть выбрано одно из трёх действий: отбросить пакет, обработать пакет без вмешательства IPSec или обработать пакет с помощью IPSec. В последнем случае в SPD также содержится указатель на SA, который необходимо использовать в процессе обмена данными.

В свою очередь SA – это согласованная политика или способ обработки данных, которыми обмениваются стороны. Два устройства с каждой стороны одной ассоциации защиты содержат данные о протоколах, алгоритмах и ключах, используемых в SA. Отдельно взятая ассоциация защиты используется для связи только в одном направлении, для двунаправленной связи их требуется две. Каждый SA реализует один режим и протокол, поэтому в случае, когда для анализа некоторого пакета требуется применить два протокола (например, AH и ESP, хотя этот случай и является очень редким), требуется создание двух различных SA.

**Процесс функционирования IPSec в общем случае можно разбить на следующие шаги:**

1. Инициация IPSec. Приложение, трафику которого требуется защита IPSec, начинает процесс обмена данными IКЕ-протокола.
2. Первая фаза IKE, в которой выполняется аутентификация сторон, и ведутся переговоры о параметрах ассоциаций защиты IKE, в результате чего создается защищенный канал для обмена параметрами защиты IPSec в ходе второй фазы IKE.
3. Вторая фаза IKE, по ходу которой ведутся переговоры о параметрах ассоциации защиты IPSec, создаются соответствующие SA для устройств обеих сторон.
4. Собственно передача данных, т.е. процесс обмена данными, основывающийся на параметрах IPSec и ключах, хранимых в SPD.
5. Завершение работы IPSec. SA IPSec завершают свою работу в результате либо их удаления, либо превышения лимита времени их существования.



**1.4. Оценка IPSec**

Спецификация IPSec получила неоднозначную оценку со стороны специалистов**.**

**С одной стороны**, отмечается, что протокол IPSec является лучшим среди всех других протоколов защиты передаваемых по сети данных, разработанных ранее (включая разработанный Microsoft PPTP).

**По мнению** **другой стороны**, присутствует чрезмерная сложность и избыточность протокола. Так, Niels Ferguson и Bruce Schneier в своей работе "A Cryptographic Evaluation of IPsec" отмечают, что они обнаружили серьезные проблемы безопасности практически во всех главных компонентах IPSec. Эти авторы также отмечают, что набор протоколов требует серьезной доработки для того, чтобы он обеспечивал хороший уровень безопасности.

**SSL (Secure Sockets Layer)** и **TLS (Transport Level Security)** — криптографические протоколы, обеспечивающие защищенную передачу данных в компьютерной сети. Они широко используются в веб-браузерах, а также при работе с электронной почтой, обмене мгновенными сообщениями и в IP-телефонии.

Соединение, защищенное протоколом TLS, обладает одним или несколькими следующими свойствами:

* *Безопасность*: симметричное шифрование защищает передаваемую информацию от прочтения посторонними лицами.
* *Аутентификация*: "личность" участника соединения можно проверить с помощью асимметричного шифрования.
* *Целостность*: каждое сообщение содержит код (**Message Authentication Code, MAC**), с помощью которого можно проверить, что данные не были изменены или потеряны в процессе передачи.

Так как большинство протоколов связи могут быть использованы как с TLS/SSL, так и без него, при установке соединения необходимо явно указать серверу, хочет ли клиент устанавливать TLS. Один способ добиться этого — использовать порт, по которому соединение всегда устанавливается с использованием TLS (например, 443 для HTTPS). Другой способ — использовать специальную команду серверу от клиента переключить соединение на TLS (например, STARTTLS для протоколов электронной почты).

**HTTPS –** расширение HTTP, поддерживающее шифрование; **HTTPS** – это HTTP над SSL/TSL, 443 порт

**DNSSEC** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Domain Name System Security Extensions) — набор расширений [IETF](https://ru.wikipedia.org/wiki/Internet_Engineering_Task_Force) протокола [DNS](https://ru.wikipedia.org/wiki/DNS), позволяющих минимизировать атаки, связанные с подменой IP-адреса при [разрешении доменных имён](https://ru.wikipedia.org/wiki/DNS). Он направлен на предоставление DNS-клиентам (англ. термин resolver) гарантии достоверности и целостности данных.