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

Модели взаимодействия описывает взаимодействия различных частей (процессов) распределенного приложения

Модель ISO/OSI – сетевая эталонная модель взаимодействия открытых систем. Модель OSI разделяет сетевые функции на семь уровней: прикладной, уровень представления, сеансовый, транспортный, сетевой, канальный и физический; описывает, как данные перемещаются по сетям и как происходит передача информации между приложениями на разных уровнях. Каждый уровень выполняет определенный набор задач, связанных с передачей данных между устройствами в сети.

**Физический уровень** определяет тип 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** (MAC) и **подуровень управления логическим каналом** (LLC). MAC определяет методы совместного использования сетевыми адаптерами среды передачи данных. LLC 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. Кадрирование пакетов – самое важное на этом уровне. Драйвер находится на канальном уровне на подуровне LLC.

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 уровня 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. Также, сетевой уровень проверяет, тому ли пришло сообщение.

**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. Также, этот уровень занимается индексацией пакетов и вычислением контрольной суммы.

**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.** Представляет данные в едином формате, а также кодирование, шифрование, сжатие.

**Прикладной уровень.** Основное назначения уровня: определить способы взаимодействия пользователей с системой (определить интерфейс).

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.

**TCP/IP** – см. вопрос ниже.

***Архитектура клиент–сервер*** (client-server architecture) – это концепция информационной сети, в которой основная часть ее ресурсов сосредоточена в серверах, обслуживающих своих клиентов

Распределенное приложение, имеющее **архитектуру клиент-сервер**, подразумевает наличие в своем составе два вида процессов: процессы-серверы и процессы-клиенты. Далее эти процессы будем назвать просто **серверами** и **клиентами**. Некоторые процессы распределенного приложения могут выступать клиентом для некоторых процессов-серверов и одновременно являться серверами других процессов-клиентов.

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. Клиент должен быть тоже распознан сервером, чтобы сервер, во-первых, мог его отличить от других клиентов, а во-вторых, чтобы смог обмениваться с клиентом данными.

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 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: например, TCP/IP, HTTP, REST, SOAP, MQTT и другие протоколы используются для обмена данными между распределенными компонентами системы.
* RPC (Remote Procedure Call): 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 к разработке, основанный на создании независимых сервисов, 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.

Спецификация NDIS определяет API и протоколы, которые драйверы сетевых адаптеров должны реализовывать для обеспечения сетевой функциональности.

NDIS предоставляет абстракцию для драйверов сетевых адаптеров, что позволяет разработчикам приложений взаимодействовать со сетевыми адаптерами независимо от их конкретной реализации.

NDIS — это специальный драйвер (ему соответствует файл ndis.sys), который содержит функции, используемые низкоуровневыми сетевыми драйверами. NDIS как бы обволакивает низкоуровневые сетевые драйверы и является посредником в их общении между собой и с железом. По сути NDIS можно считать третьим ядром Windows.

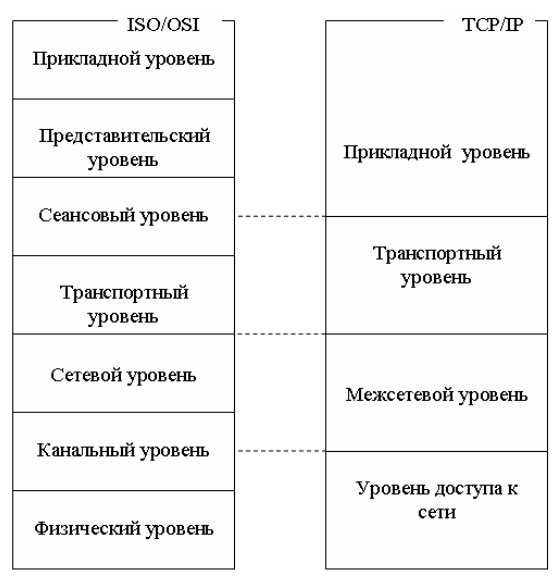
# Стек протоколов TCP/IP. Публичные и частные пространства адресов, типы портов. Параметры настройки TCP/IP.

***Стек протоколов TCP/IP*** — это набор стандартных протоколов, которые используются для передачи данных в сетях TCP/IP

Основой стека TCP/IPявляются 2 протокола:

* ***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.

Службы DNS и DHCP находятся на прикладном уровне, а протоколы соответствующих служб – на межсетевом.



1. Уровень доступа к сети (Network Access Layer) - обеспечивает физическое подключение к сети и передачу кадров данных. Сюда относятся протоколы, такие как Ethernet, PPP, SLIP.

2. Межсетевой уровень (Internet Layer) - отвечает за маршрутизацию пакетов между различными сетями. Соответствующие протоколы включают в себя IP (Internet Protocol), ICMP (Internet Control Message Protocol) и ARP (Address Resolution Protocol).

3. Транспортный уровень (Transport Layer) - обеспечивает более высокий уровень обработки данных, включая контроль ошибок и установление соединений. Здесь находятся протоколы TCP (Transmission Control Protocol) и UDP (User Datagram Protocol).

4. Прикладной уровень (Уровень приложений) (Application Layer) - обеспечивает интерфейс для конечных пользователей и приложений. Некоторые из наиболее широко используемых протоколов на этом уровне включают в себя HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol) и DNS (Domain Name System).

**Публичный IP-адрес** - это IP-адрес, который используется для связи с другими устройствами в Интернете.

**Частный IP-адрес** - это IP-адрес, который используется для связи с устройствами в локальной сети.

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

***Cетевой порт –*** id процесса, получающего или отправляющего данные с помощью транспортного уровня.

Номера портов делятся на три диапазона:

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

*Хорошо известные номера* портов присваиваются базовым системным службам, имеющие системные привилегии. *Зарегистрированные номера* портов присваиваются промышленным приложениям и процессам. *Динамические номера портов* (их часто называют ***эфемерными портами***) выделяются, как правило, прикладным процессам специализированной службой операционной системы. Некоторые системы TCP/IP применяют диапазон значений от 1024 до 5000 для назначения эфемерных номеров портов.

Таблица 2.5.2

Номер порта Протокол Описание

20 TCP File Transport Protocol (FTP)

21 TCP File Transport Protocol (FTP)

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

**Параметры настройки TCP/IP включают в себя:**

1. IP-адрес: может быть статическим, когда он устанавливается вручную, или динамическим, когда он назначается автоматически с помощью протокола DHCP.

2. Маска подсети: определяет область действия IP-адреса и разделяет адрес на сетевую часть и подсетевую часть. Маска подсети используется для определения, какие устройства находятся в данной сети и какие — в других.

3. Шлюз по умолчанию: определяет IP-адрес маршрутизатора, который обеспечивает связь между устройством и сетями, находящимися за маршрутизатором.

4. DNS-серверы: определяют серверы доменных имён, которые используются для преобразования имен хостов в IP-адреса и обратно.

5. WINS-серверы: определяют серверы, которые используются для решения NetBIOS имен.

# 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.***

* Наиболее распространенная технология *локальных сетей.*
* Применяет метод доступа CSMA/CD (множественный доступ с прослушиванием несущей частоты и обнаружением коллизий) для контроля доступа к среде передачи
* Использует 48-битную адресацию
* Обеспечивает передачу данных до 1 гигабита в секунду.
* Обеспечивает передачу данных до 1500 байт.
* Физически функции протокола Ethernet реализуются сетевой картой

***Протокол SLIP*** (Serial Line IP). Аббревиатурой SLIP обозначают межсетевой протокол для последовательного канала.

* Раньше использовался для подключения компьютеров к Internet *через последовательный порт*
* Протокол использует простейшую инкапсуляцию кадра
* имеет ряд недостатков:
  + хост с одной стороны должен знать IP-адрес другого, т.к. SLIP не дает возможности сообщить свой IP-адрес;
  + если линия задействована SLIP, то она не может быть использована никаким другим протоколом;
  + SLIP не добавляет контрольной информации к пакету передаваемой информации – весь контроль возложен на протоколы более высокого уровня.
* Используется редко

Ряд недостатков были исправлены в новой версии протокола именуемой CSLIP (Compressed SLIP).

***Протокол PPP*** *(Point-to-Point Protocol).*

* PPP – универсальный протокол для соединения между двумя узлами в сети: поддерживается TCP/IP, NetBEUI, IPX/SPX, DECNet и многими стеками протоколов.
* Протокол может применяться для технологии ISDN (используется для передачи голоса, видео и данных *по сети телефонной связи*) и SONET (высокоскоростная технология передачи данных *по оптическим каналам*).
* PPP поддерживает многоканальные реализации: можно сгруппировать несколько каналов с одинаковой пропускной способностью между отправителем и получателем.
* PPP обеспечивает циклический контроль для каждого кадра, динамическое определение адресов, управление каналом.
* Протокол для передачи пакетов IP через последовательный порт, модем или Интернет.

В настоящее время это наиболее широко используемый протокол для последовательного канала, обеспечивающий соединение компьютера с сетью Internet и практически вытеснил протокол SLIP.

**Протоколы межсетевого уровня**

***Протокол IPv4.***

* Обеспечивает адресацию и маршрутизацию пакетов данных (Основная задача – доставка дейтаграмм
* Наиболее распространенная версия протокола IP
* Использует 32-битные IP-адреса
* **ненадежный** и **не поддерживающий соединения**.

***Протокол IPv6.***

* Использует 128-битные адреса IP
* Решает проблему нехватки адресов IPv4
* Повышает эффективность маршрутизации
* Обеспечивает безопасность соединения.
* Основная концепция IPv6: каждый отдельный узел должен иметь собственный уникальный идентификатор интерфейса.
* Имеет возможность автоконфигурации. Автоконфигурация – это процесс, позволяющий хосту находить информацию для настройки собственных IP-параметров.

***Протокол ICMP.***

* Обеспечивает функции диагностики и контроля сети.
* Протокол контроля сообщений в Internet
* С помощью протокола ICMP осуществляется деятельность утилит достижимости (ping, traceroute);
* регулирует частоту отправки IP-дейтаграмм,
* доставляет хостам, маршрутизаторам и шлюзам служебную информацию
* осуществляет поиск и переадресацию маршрутизаторов;
* оптимизируют маршруты;
* диагностируют ошибки
* оповещают узлы IP-сети.

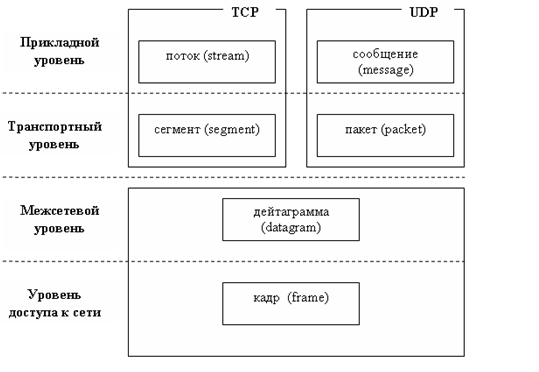
**Протокол ARP.** IP-адреса могут восприниматься только на сетевом уровне и вышестоящих уровнях

* Определяет физический адрес (МАС-адреса) устройства по его IP-адресу динамически (без вмешательства администратора, пользователя, прикладной программы)
* Использует кеш адресов, чтобы уменьшить количество запросов на связь между устройствами в сети.
* Время существования записи в arp-таблице составляет обычно 10-20 минут с момента ее создания и может быть изменено с помощью параметров реестра.
* Используется для проверки существования в сети дублированного IP-адреса и разрешения запроса о собственном MAC-адресе хоста во время начальной загрузки.

***Протокол RARP.*** *(Reverse ARP)*

* Применятся для получения IP-адреса по MAC-адресу.
* В настоящее время протокол заменен на протокол Прикладного уровня DHCP, предлагающий более гибкий метод присвоение адресов.

**Протоколы транспортного уровня**

****

***Протокол TCP.***

* **Надежный.** Проверяет целостность (наличие повреждений, потерь, дублирования) по контрольной сумме и порядок прихода данных. Все октеты в потоке данных пронумерованы в возрастающем порядке. Заголовок каждого сегмента содержит число октетов и порядковый номер первого октета данных в данном сегменте. Также подтверждает прием данных отправляющей стороне, если подтверждение не приходит – предпринимается повторная попытка переслать данные.
* **с установлением соединения**

Протокол TCP обеспечивает одновременно нескольких соединений. Поэтому говорят о **разделении каналов**. Каждый процесс Прикладного уровня идентифицируется номером порта. Заголовок TCP-сегмента содержит номера портов отправителя и получателя.

***Протокол UDP.***

* **Без установления соединения**. каждый пакет данных (или сообщение) может быть передан независимо друг от друга и без установки соединения. UDP не имеет методов установления, управления и завершения соединения между отправителем и получателем данных;
* **Ненадёжный**. Не гарантирует доставку пакетов данных в правильном порядке и может потерять данные. Это возлагается на программу пользователя.
* отсутствие буферизации (временное хранение нескольких кадров в памяти). UDP оперирует только одним пакетом, и вся работа по буферизации ложится на процесс Прикладного уровня;
* UDP не содержит средств, позволяющих разбивать сообщение на несколько пакетов (фрагментировать) – вся эта работа возложена на процесс Прикладного уровня.

***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. Пример: TCP

**Ненадежный** протокол, напротив, не гарантирует доставку данных. Он может потерять, дублировать или изменить порядок пакетов данных без уведомления об этом. Пример: 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 является TCP.

**Протоколы без установки соединения**, наоборот, не требуют предварительной установки логического канала связи. Они передают данные независимо друг от друга. Примером является UDP

**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

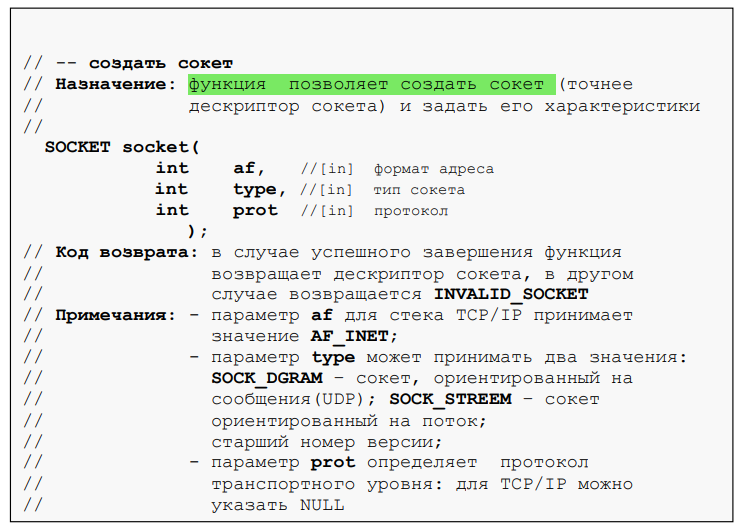
# Понятие сокета. Основные параметры сокета.

Сокет – совокупность IP-адреса и номера порта; используется для идентификации прикладного процесса в сети.

API сокетов – это название программного интерфейса для обмена данными между процессами, находящимися на одном или на разных объединенных сетью компьютерах.

Следует помнить, что одни и те же номера портов могут быть использованы как для протокола UDP, так и для протокола TCP.

В операционной системе Windows интерфейс сокетов имеет название **Windows Sockets API**. API сокетов включает в себя функции создания сокета (имеется в виду объект ОС, описывающий соединение), установки параметров сокета (сетевой адрес, номер порта и т.д.), функции создания канала и обмена данными между сокетами. Кроме того, есть набор функций, позволяющий управлять передачей данных, синхронизировать процессы передачи и приема данных, обрабатывать ошибки и т.п.



|  |  |
| --- | --- |
| **Наименование функции** | **Назначение** |
| **SOKET accept (**  **Socket s, сокет**  **Sockaddr\_in\* указатель на Sockaddr\_in**  **Int\* la указатель на длину Sockaddr\_in);** | функция используется для создания канала на стороне сервера и создает сокет для обмена данными по этому каналу |
| **int bind( Socket s сокет**  **sockaddr\_in\* указатель на сокадр**  **int la длина сокадр в байтах)** | Связать сокет с параметрами (структура сокадр) |
| **SOCKET closesocket(socket s, дескриптор сокета)** | Закрыть существующий сокет |
| **int connect(**  **socket s сокет**  **sockaddr\_in\* указатель на сокадр**  **int la длина сокадр** | Установить соединение с сокетом |
| **gethostbyaddr(**  **char\* addr адрес в сетевом форм**  **int la длина адреса**  **int ta тип адреса (AF\_INET))** | Получить имя хоста по его адресу |
| **gethostbyname(**  **char\* name символическое имя хоста)** | Получить адрес хоста по его имени |
| **gethostname(**  **char\* name буфер имени**  **int ln длина имени)** | Получить имя хоста |
| **getsockopt** | Получить текущие опции сокета |
| **inet\_addr(char\* addr адрес в строке)** | Преобразовать символьное представление IPv4-адреса в формат TCP/IP |
| **inet\_ntoa** | Преобразовать сетевое представление IPv4-адреса в символьный формат |
| **ioctlsocket** | Установить режим ввода-вывода сокета |
| **int listen (**  **socket s, сокет**  **int mcq максимальная длина очереди)** | Переключить сокет в режим прослушивания |
| **int recv(**  **socket сокет**  **char\* buf буфер для данных**  **int lbuf, длина буфера**  **int flags флаги (индикатор))** | Принять данные по установленному каналу |
| **int recvfrom(**  **socket s сокет**  **char\* buf буфер**  **int len длина буфера**  **int flag индикатор**  **sockaddr\* to указатель на сокадр**  **int tolen длина сокадра)** | Принять сообщение |
| **int send(**  **socket s сокет**  **char\* buf буфер данных**  **int lbuf длина буфера**  **int flags индикатор особого режима)** | Отправить данные по установленному каналу |
| **int sendto(**  **socket s сокет**  **char\* buf буфер**  **int len размер буфера**  **int flags индикатор**  **sockaddr\* to указатель на сокадр**  **int tolen длина сокадра)** | Отправить сообщение |
| **setsockopt(**  **socket s сокет**  **int level уровень действия режима**  **int optnamr режим сокета для установки**  **char\* optval значение режима сокета**  **int fromlen длина буфера optval)** | Установить опции сокета |
| **socket (  int af формат адреса**  **int type тип, int prot протокол)** | Создать сокет |
| **WSACleanup(void)** | Завершить использование библиотеки WS2\_32.DLL |
| **WSAGetLastError(void)** | Получить диагностирующий код ошибки |
| **WSAStartup(**  **WORD ver, версия Windows socket**  **lpWSAData wsd указатель на WSADATA)** | Инициализировать библиотеку WS2\_32.DLL |

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.

**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

**};**

# Интерфейс Nimed Pipe.

**Именованный канал –** это объект ядра операционной системы, который обеспечивает обмен данными между процессами, выполняющимися на компьютерах в одной локальной сети.

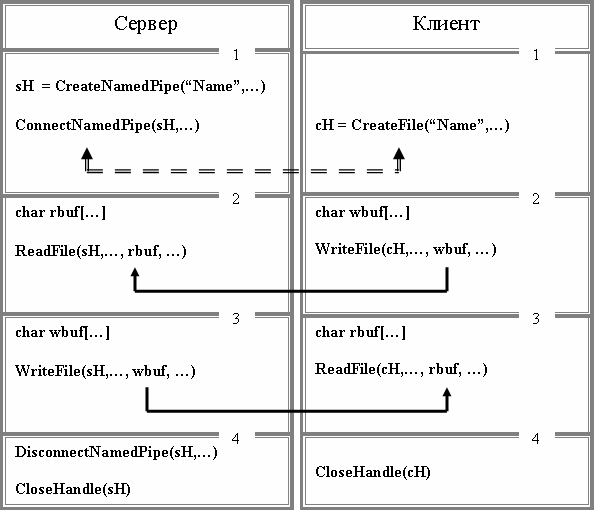
Процесс, создающий именованный канал, называется **сервером именованного канала**. Процессы, которые связываются с именованным каналом, называются **клиентами именованного канала**. Любой именованный канал идентифицируется своим именем, которое задается при создании канала.

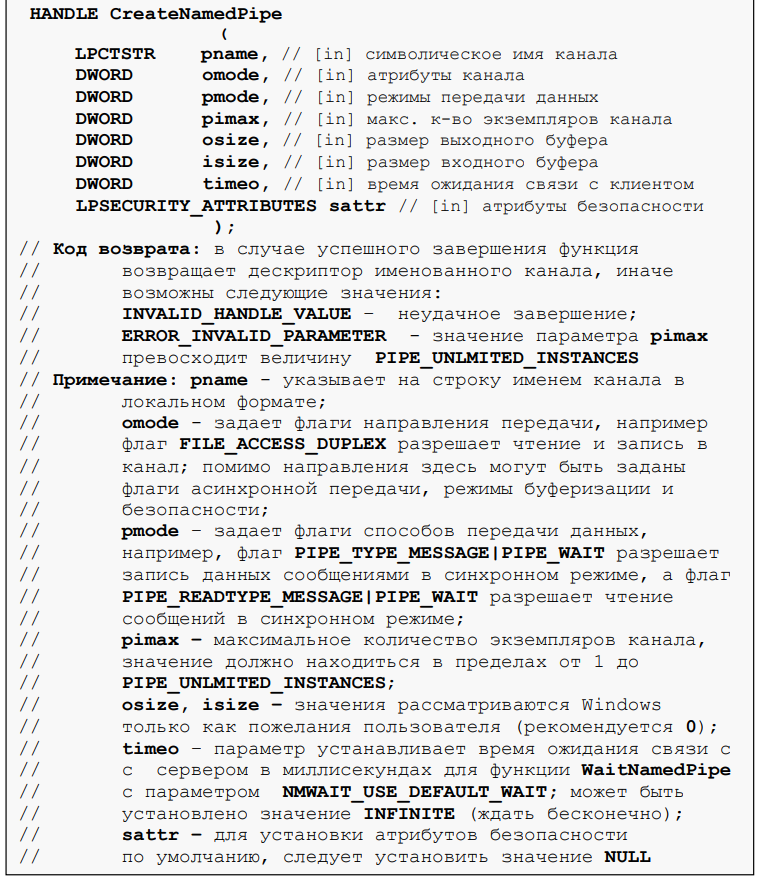
Именованные каналы бывают: ***дуплексные*** (позволяющие передавать данные в обе стороны) и ***полудуплексные***(позволяющие передавать данные только в одну сторону). Передача данных в именованном канале может осуществляться как потоком, так и сообщениями. Обмен данными в канале может быть синхронным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

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**.

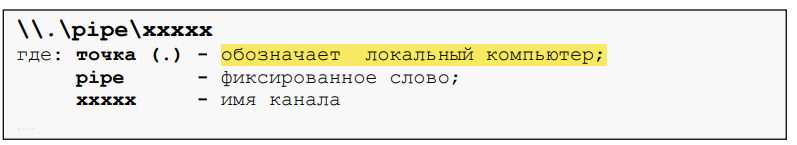
Для использования функций интерфейса Named Pipe в программе на языке C++ необходимо включить в ее текст заголовочный файл Windows.h. Сами функции интерфейса располагаются в библиотеке KERNEL32.DLL ядра операционной системы.

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **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);** | Выполнить одну транзакцию |
| **ConnectNamedPipe(**  **HANDLE hp дескриптор канала**  **LPOVERLAPPED ol исп для ассинхр связи** | Соединить сервер с каналом  для ожидания клиентов |
| **HANDLE CreateFile(**  **LPCTSR pname символическое имя канала**  **DWORD accss чтение или запись в канал**  **DWORD share режимы совместного использования**  **LPSECURITY\_ATTR атрибуты безопасности**  **DWORD oflag флаг открытия канала**  **DWORD aflag флаги и атрибуты**  **HANDLE exten дополнительные атрибуты** | Открыть канал |
| **HANDLE CreateNamedPipe(**  **LPCTSR pname символическое имя канала**  **DWORD omode атрибуты канала**  **DWORD pmode режимы передачи канала**  **DWORD pimax макс к-во экземпляров канала**  **DWORD osize размер выходного буфера**  **DWORD isize размер входного буфера**  **DWORD timeo время ожидания связи с клиентом**  **LPSECURITY\_ATTRIBUTES sattr атрибуты безопастности)** | Создать именованный канал |
| **DisconnectNamedPipe** | Закончить обмен данными |
| **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);** | Получить состояние канала |
| **GetNamedPipeInfo(**  **HANDLE hp дескриптор канала**  **LPDWORD pfg указатель на флаг тип канала**  **LPDWORD psw указатель на размер выходного буфера**  **LPDWORD psr указатель на размер входного буфера**  **LPDWORD pmi указатель на макс к-во экземпляров канала)** | Получить информацию об именованном канале |
| **PeekNamedPipe(**  **HANDLE hp дескриптор канала**  **LPVOID pb указатель на буфер**  **DWORD sb размер буфера**  **LPDWORD pi кол-во прочитанных байт**  **LPDWORD pa кол-во доступных байт**  **LPDWORD pr кол-во непрочитанных байт)** | Копировать данные канала |
| **BOOL ReadFile(**  **HANDLE hp дескриптор канала**  **LPVOID pb указатель на буфер ввода**  **DWORD sb кол-во читаемых байт**  **LPDWORD ps кол-во прочитанных байт**  **LPPOVERLAPPED ol для ассинхрон обработки)** | Читать данные из канала |
| **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);** | Изменить характеристики канала |
| **TrasactNamedPipe (**  **HANDLE hp дескриптор канала**  **LPVOID pw указатель на буфер записи**  **DWORD sw размер буфера записи**  **LPDVOID pr указатель на буфер чтения**  **DWORD sr размер буфера чтения**  **LPDWORD pi кол-во прочитанных байт**  **LPPOVERLAPPED ol для ассинхрон обработки)** | Писать и читать данные канала |
| **WaitNamedPipe(**  **LPCTSTR pn симвл имя канала**  **DWORD to интервал ожидания)** | Определить доступность канала |
| **BOOL WriteFile(**  **HANDLE hp дескриптор канала**  **LPVOID pb указатель на буфер ввода**  **DWORD sb кол-во записываемых байт**  **LPDWORD ps кол-во записанных байт**  **LPPOVERLAPPED ol для ассинхрон обработки)** | Писать данные в канал |





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



Функция ConnectNamedPipe приостанавливает выполнение программы клиента до момента, пока программа клиента не выполнит функцию CreateFile(Открыть канал).

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.

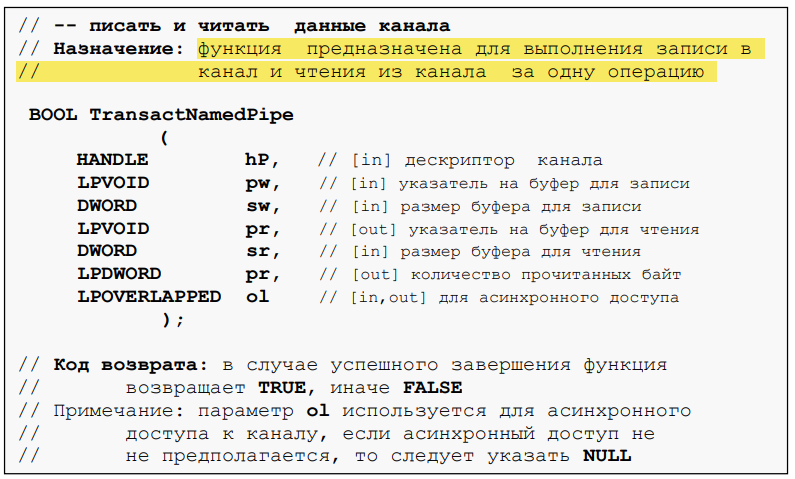
В последнем четвертом блоке программы сервер разрывает соединение с помощью функции DisconnectNamedPipe и закрывает дескриптор именованного канала.

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. Назначение всех функций будет пояснено ниже.

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.

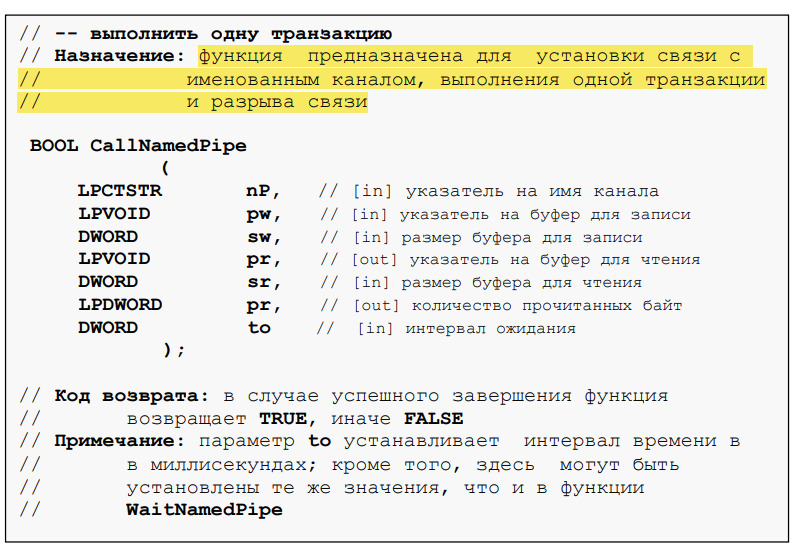
**Передача транзакций по именованному каналу**

Для обмена сообщениями по сети может использоваться функция **TansactNamedPipe**, 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 оправить ответ на пришедшее сообщение.

Если требуется передать только одну транзакцию, то используют функцию **CallNamedPipe**, 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.



# Интерфейс MailSlot.

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

**Почтовый ящик** (Mailslot) – это объект ядра операционной системы, который обеспечивает передачу данных от процессов-клиентов к процессам-серверам, выполняющимся на компьютерах в одной локальной сети.

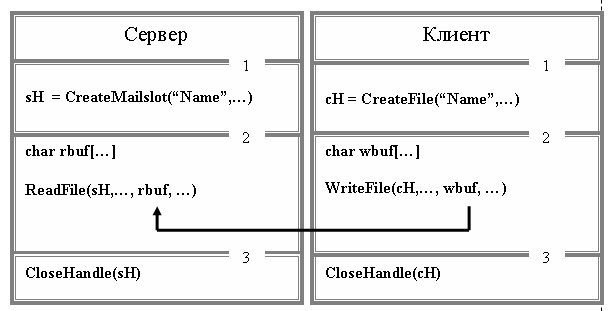
Процесс, создающий почтовый ящик называется **сервером** почтового ящика. Процессы, которые связываются с почтовым ящиком, называются **клиентами** почтового ящика.

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 оправляемых 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 ниже. Функции CreateFile, ReadFile, WriteFile являются универсальными и используются также для работы с именованными каналами, файловой системой, сокетами и т.д.

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **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);** | Открыть почтовый ящик |
| **CreateMailslot(**  **LPCTSTR pname, [in] символическое имя ящика**  **DWORD maxms, [in] максимальная длина сообщения**  **DWORD timeo, [in] интервал ожидания**  **LPSECURITY\_ATTRIBUTES sattr [in] атрибуты безопасности);** | Создать почтовый ящик |
| **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);** | Получить информацию о почтовом ящике (только на стороне сервера) |
| **ReadFile** | Читать данные из почтового ящика |
| **SetMailslotInfo(**  **HANDLE hM, [in] дескриптор почтового ящика**  **PDWORD to [in] новое значение интервала );** | Изменить время ожидания сообщения |
| **WriteFile** | Писать данные в почтовый ящик |



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.

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 \*.

# 7, 8. Структура программы TCP-сервера. Структура программы TCP-клиента.

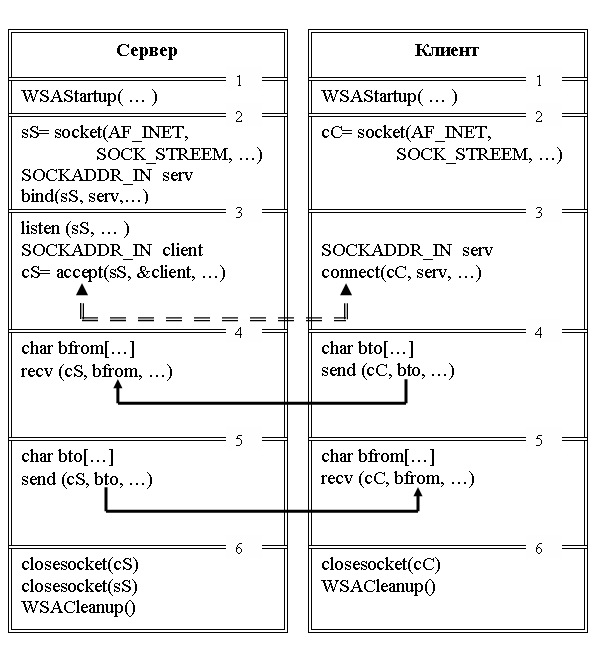
Существование двух различных протоколов на транспортном уровне TCP/IP, определяет две схемы взаимодействия процессов распределенного приложения: схема, ориентированная на сообщения, и схема, ориентированная на поток.

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 инициатором обмена является клиент, а сервер ждет запросы клиента и обслуживает их.

****

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

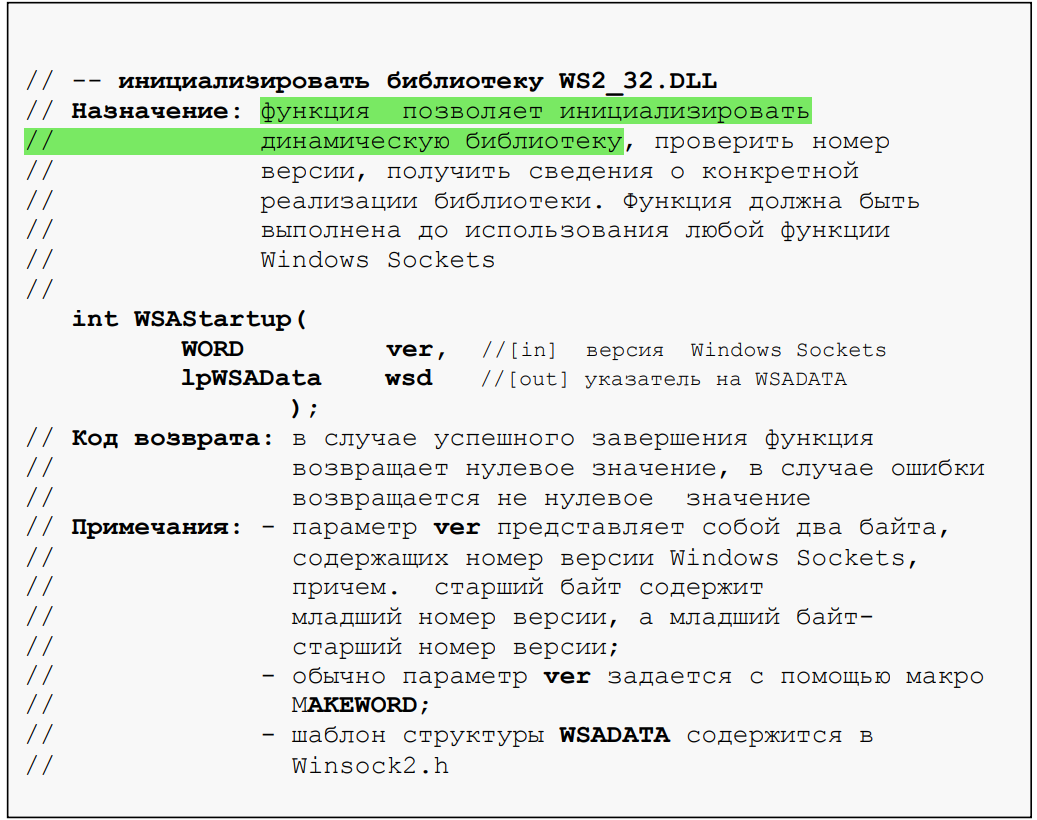
Второй блок программы сервера создает сокет (функция soсket) и устанавливает параметры этого сокета. Единственным отличием является значение SOCK\_STREEM параметра функции socket, указывающий, что сокет будет использоваться для соединения (сокет, ориентированный на поток). Для установки параметров сокета, используется функция bind. При этом говорят, что сокет связывают с параметрами. Для хранения параметров сокета в Winsock2 предусмотрена специальная структура SOCKADDR\_IN

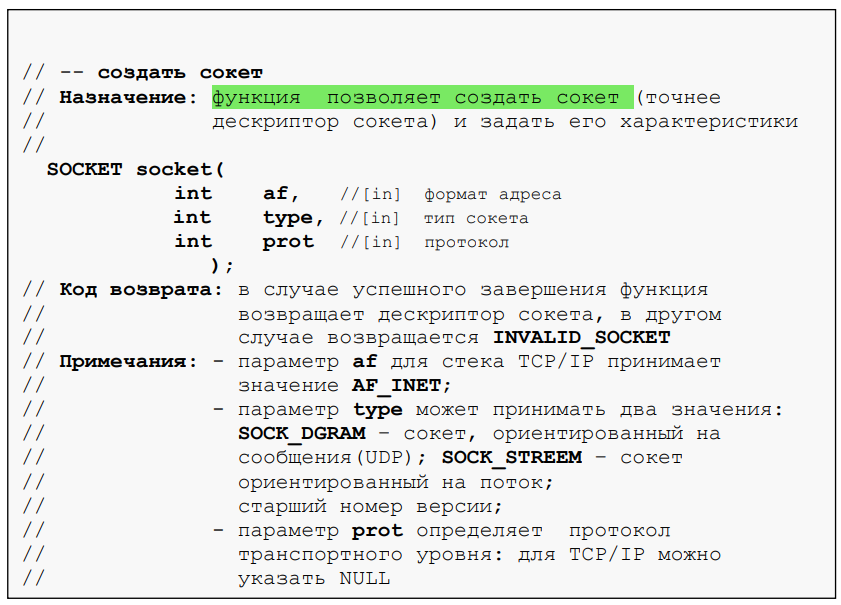
В третьем блоке программы сервера выполняются две функции Winsock2: listen и accept. Функция listen переводит сокет, ориентированный на поток, в состояния прослушивания (открывает доступ к сокету) и задает некоторые параметры очереди соединений. Функция accept переводит процесс сервера в состояние ожидания, 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.

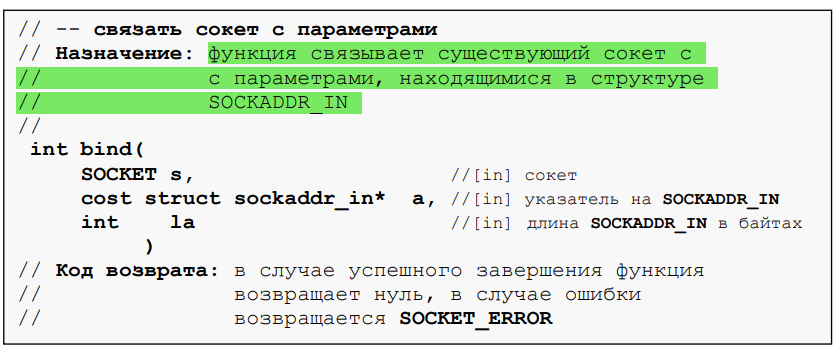
Четвертый и пятый блоки программы сервера предназначены для обмена данными по созданному соединению. Следует обратить внимание, что, во-первых, используются функции send и recv, а во-вторых, в качестве параметра эти функции используют сокет, созданный командой accept.

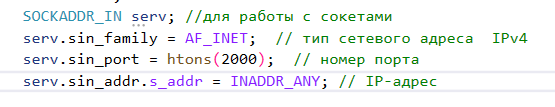
В шестом блоке для закрытия сокета применяется функция closesocket. Для завершения работы с библиотекой WS2\_32.DLL используется функция WSACleanup

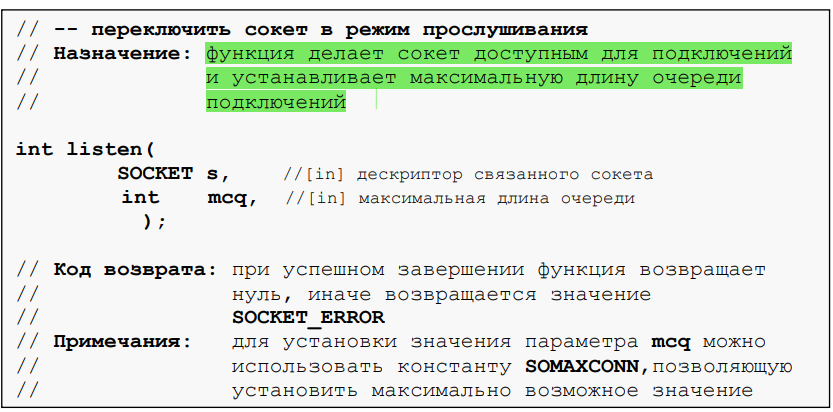
В программе клиента осталось пояснить, только работу третьего блока. В этом блоке выполняется функция connect, предназначенная для установки соединения с сокетом сервера. Функция в качестве параметров имеет, созданный в предыдущем блоке, дескриптор сокета (ориентированного на поток) и структуру SOCKADDR\_IN с параметрами сокета сервера.

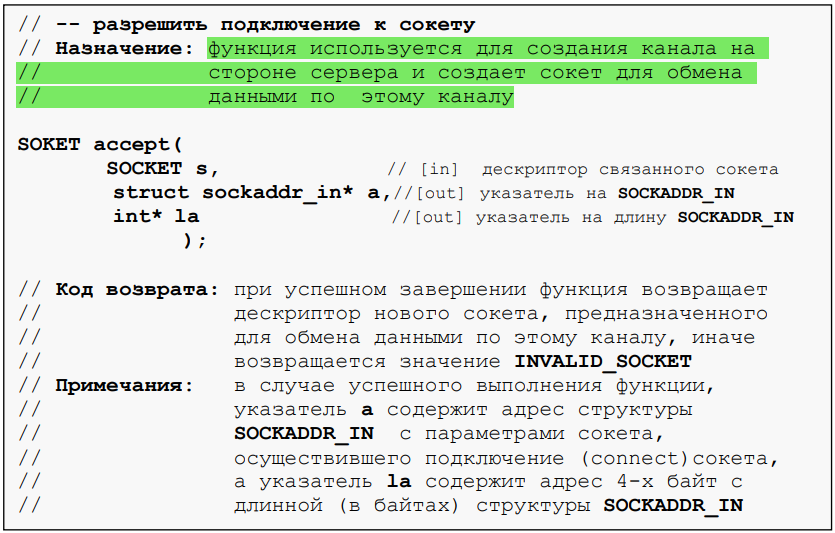


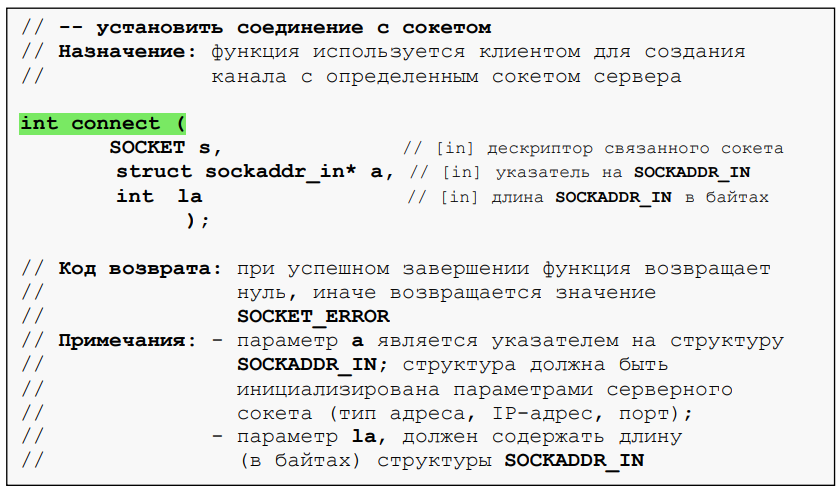


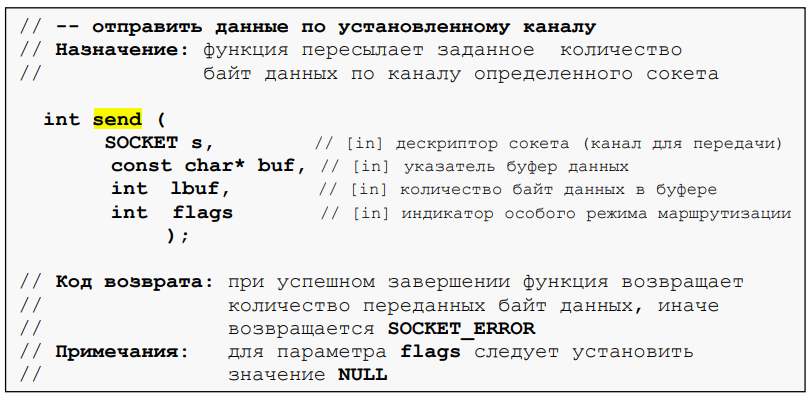




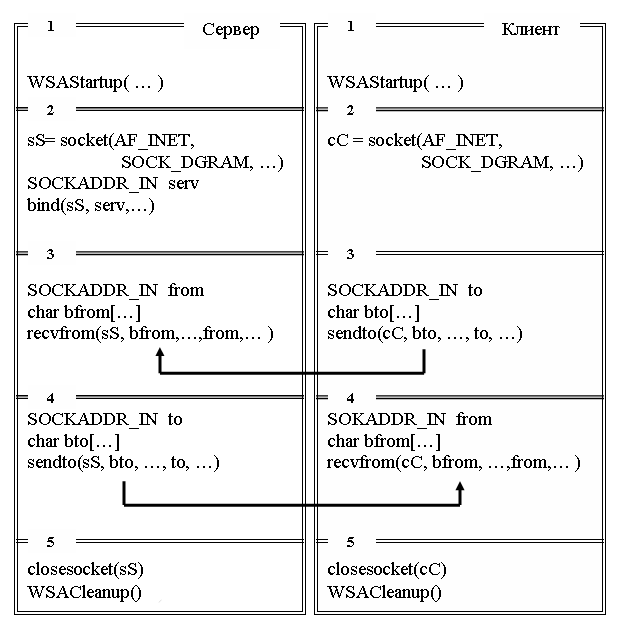








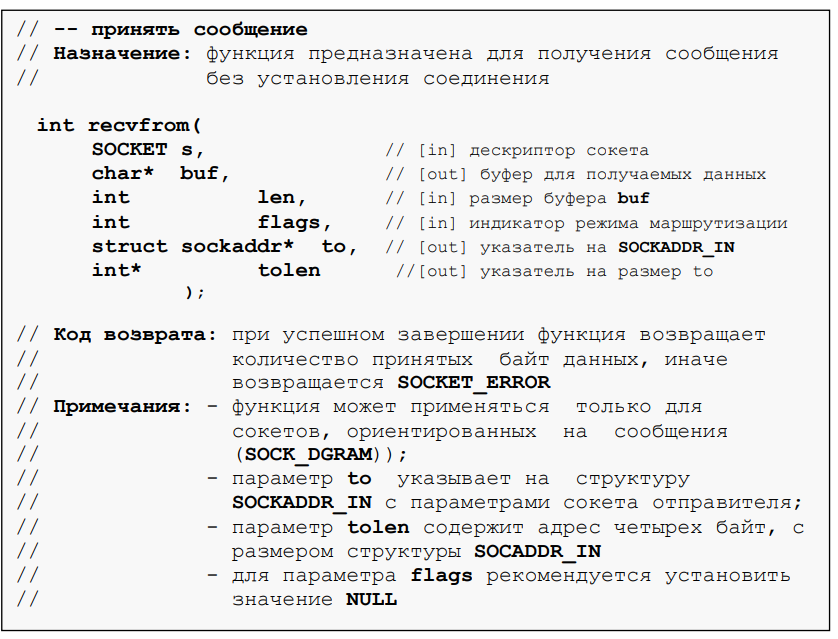
# 9, 10. Структура программы UDP-сервера. Структура программы UPD-клиента.



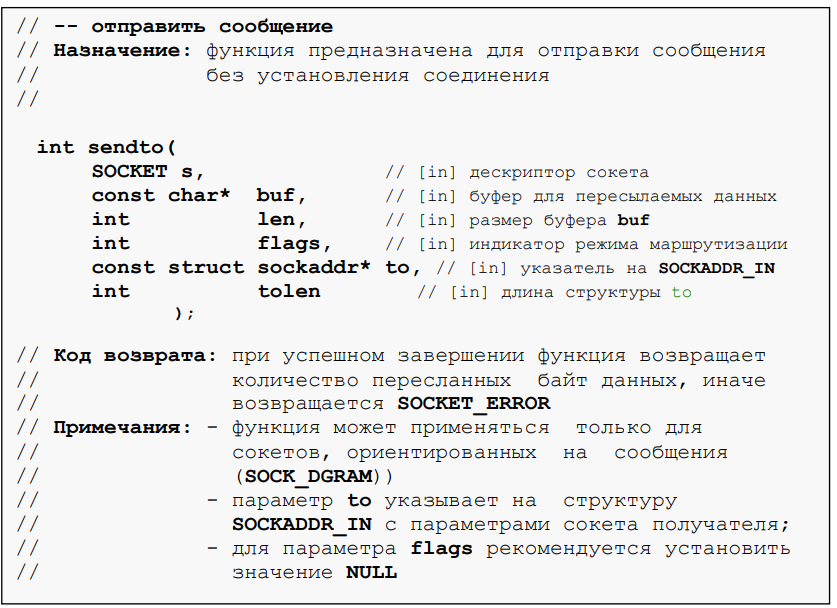
Первые блоки обеих программ одинаковые и предназначены для инициализации библиотеки WS2\_32.DLL.

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. 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.

# 11, 12. Структура параллельного сервера. Структура параллельного сервера. Accept Server. GarbageCleaner.

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.

Если предполагаются короткие запросы от клиентов, не требующие больших затрат на обработку и длинных ответов сервера. Как правило, подобные серверы работают над UDP.

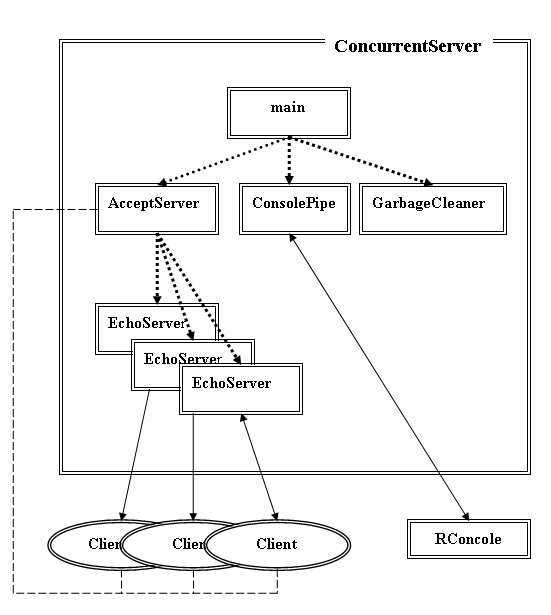
Параллельные серверы имеют другой цикл работы:

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. Параллельные серверы целесообразно использовать, если предполагается наличие относительно длительного сеанса связи между клиентом и сервером. Работают над TCP.



На рисунке изображена 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. Сплошными двунаправленными линиями обозначается перемещение данных.

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. Данные перемещаются между EchoServer-процессами сервера и клиентскими программами (с именем Client), 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.

**Процесс main.** Основным назначением процесса main, является 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 GarbageCleaner.

**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 использования. 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 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.

11. Обычно сервер должен распознавать некоторое количество команд (запросов), которые клиент может направить в его адрес.

Основным отличием новой структуры, является промежуточный поток DispatchServer между AcceptServer и обслуживающими потоками ServiceServer (раньше это был единственный поток называемый EchoServer). *Теперь предполагается, что сначала программа клиента осуществляет процедуру подключения (для этого используется поток AcceptServer), потом поток DispatchServer принимает от клиента запрос (команду) на обслуживание и после этого уже запускается соответствующий поток ServiceServer,* который исполняет команду и в случае необходимости обменивается данными с клиентом.

Введение промежуточного звена, обуславливается тем, что после этапа подключения, клиент (в общем случае) может достаточно долго не запрашивать у сервера услугу (не выполнять функцию send, пересылающую команду сервера). Ожидать поступление команды в потоке AcceptServer не целесообразно, т.к. его основное назначение – подключение клиентов.

Поток DispatchServer получает информацию (сокет и его параметры) о новом подключении через список ListContact (элементы списка создаются и заполняются в потоке AcceptServer).

поток DispatchServer должен отследить интервал времени (с помощью ожидающего таймера) от момента подключения до получения первой команды, проанализировать команду на правильность, а также запустить поток соответствующий запрошенному сервису (и) или оправить диагностирующее сообщение.

**Использование динамически подключаемых библиотек**

При разработке параллельного сервера часто бывает полезным разделить процедуры управления сервером и обслуживания клиентов. Одним из способов такого разделения является размещения функций обслуживающих потоков в динамически подключаемой библиотеке (dll-библиотеке)

Рассмотренный выше обслуживающий разнотипные запросы сервер использовал таблицу TableService для того, чтобы сопоставить запросу клиента функцию обслуживающего потока. Удобно поместить эту таблицу вместе с функциями обслуживающих потоков в dll-библиотеке. В этом случае сервер может динамически загружать dll-библиотеку.

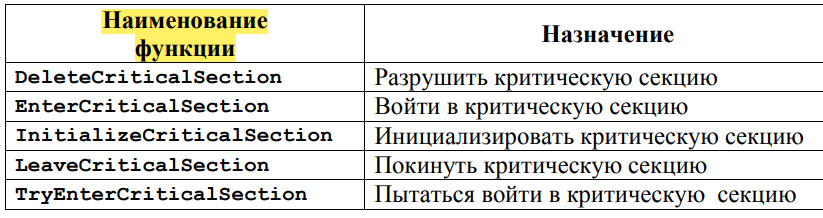
Таким образом, теперь параллельный сервер состоит из двух частей: управляющий сервер и dll-библиотека функций потоков обслуживания. При такой структуре, набор сервисных услуг предоставляемых сервером зависит от версии динамической библиотеки.

Библиотека загружается в память при инициализации сервера

12.

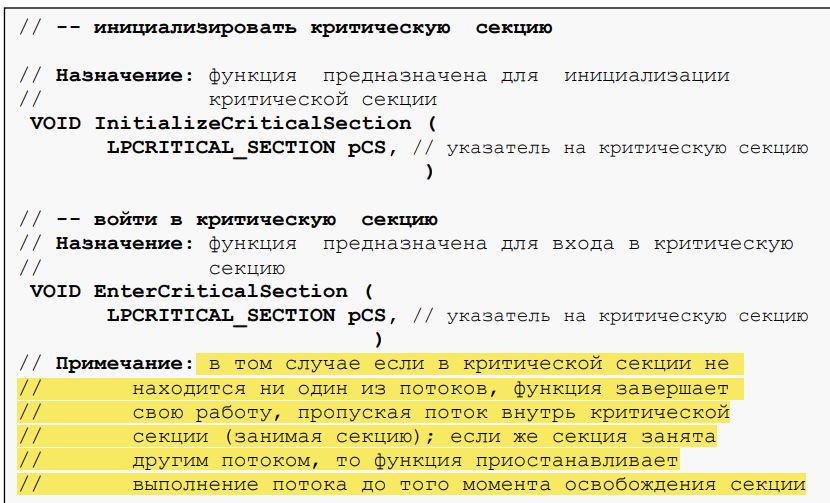
Синхронизация будет осуществляться между двумя потоками: AcceptServer и GarbageCleaner. Как уже описывалось выше, синхронизация необходима в связи с тем, что одновременное добавление элемента в список ListContact, выполняемое потоком AcceptServer, и удаление элемента списка, выполняемое потоком GarbageCleaner может привести к непредсказуемым последствиям.

Операционная система Windows обладает широким спектром механизмов синхронизации потоков и процессов: критические секции, мьютексы, события, семафоры, ожидающий таймер



Критическая секция является объектом операционной системы типа CRITICAL\_SECTION

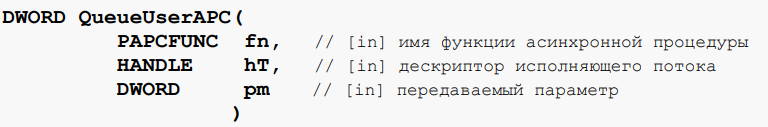
Для того, чтобы добавление и удаление элементов списка не осуществлялось одновременно, эти операции помещают внутри критической секции соответствующего потока. Каждая критическая секция начинается функцией EnterCriticalSection, а заканчивается функцией LeaveCriticalSection.



Асинхронной процедурой называется функция, которая выполняется асинхронно в контексте какого-нибудь потока.

При подключении очередного клиента, функция AcceptServer запускает обслуживающий поток EchoServer. С этого момента AcceptServer теряет связь с потоком EchoServer и, в принципе, даже не знает о моменте окончания его работы. В этом случае поток EchoServer может поставить функцию (асинхронную процедуру) в специальную очередь к потоку AcceptServer. (в начале и при окончании работы выведется уведомление)

В начале своей работы функция EchoServer выполняет функцию QueueUserAPC, которая помещает асинхронную процедуру ASStartMessage в очередь к потоку AcceptServer. Эта очередь обеспечивается операционной системой Windows и работает по алгоритму FIFO. После отключения программы клиента, перед самым завершением своей работы, функция EchoServer вновь исполняет функцию QueueUserAPC, но уже для постановки в очередь асинхронной процедуры ASFinishMessage. Выполнение всех асинхронных процедур, находящихся к этому моменту в очереди потока, осуществляется последовательно в потоке AcceptServer, после того, как в рамках этого потока будет выполнена функция SleepEx



Асинхронные процедуры не могут возвращать никакого значения и принимают только один параметр.

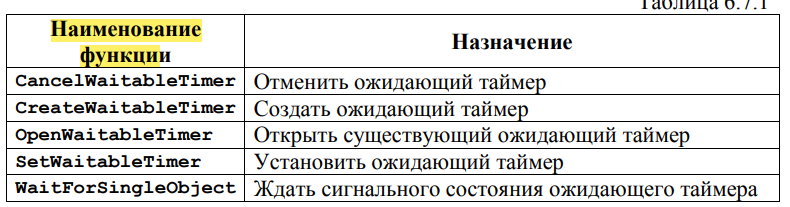
**Использование ожидающего таймера**

Производительность параллельного сервера в значительной степени зависит от количества одновременно подключившихся клиентов: с ростом подключившихся клиентов, производительность убывает. Обслуживание каждого клиента связано с выделением ему определенных ресурсов: процессорного времени, оперативной памяти, сетевого трафика и т.п. В связи с ограниченностью ресурсов возникает необходимость управлять процессом обслуживания. Одной из задач управления сервером является выявление слишком продолжительных подключений. Для реализации такого решения может быть использован механизм ожидающего таймера.

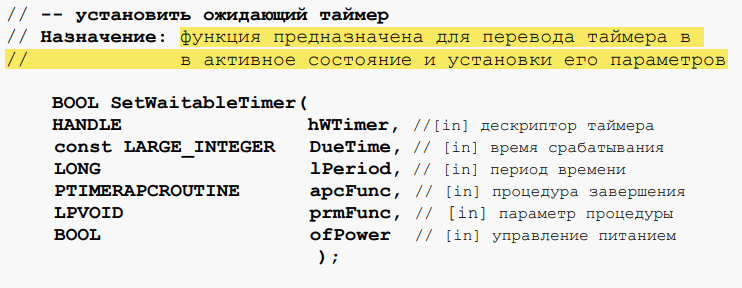
**Ожидающим таймером** в Windows, называется объект синхронизации, который переходит в *сигнальное состояние* при наступлении заданного момента времени. Если ожидающий таймер ждет момента перехода в сигнальное состояние, то говорят, что он находится в *активном состоянии*. Другое состояние ожидающего таймера *пассивное* – из этого состояния он не может перейти в сигнальное состояние.

По способу перехода **из сигнального стояния в несигнальное,** ожидающие таймеры разделяются на *таймеры с ручным сбросом* и *таймеры с автоматическим сбросом*, иначе называемые таймерами синхронизации. По способу перехода **из несигнального состояния в сигнальное**, ожидающие таймеры бывают *периодические* и *непериодические*. Периодические таймеры работают по циклу: активное состояние – сигнальное состояние – активное состояние. Непериодические таймеры могут только один раз перейти из активного состояния в сигнальное.

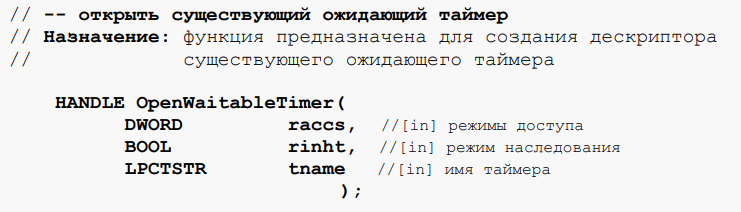
Ожидающий таймер будет создаваться потоком AcceptServer для каждого запущенного потока EchoSever. Сразу же после перехода одного из ожидающих таймеров, в очередь асинхронных процедур потока AcceptServer будет поставлена соответствующая процедура завершения.



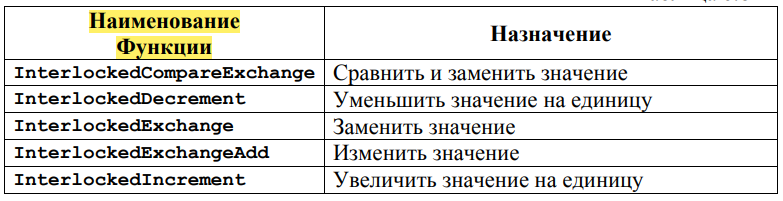




Если создан поименованный таймер, то он может быть использован в контексте другого процесса с помощью функции OpenWaitableTimer (рисунок 6.7.3). Для перевода таймера в неактивное состояние применяется функция CancelWaitableTimer



Блокирующие функции выполняют несколько элементарных операций, которые объединяются в одну неделимую операцию, называемую атомарной операцией.



В модели параллельного сервера ConcurrentServer атомарные операции могут быть использованы, например, для управления количеством одновременно обслуживаемых клиентов.

Предположим, что число одновременно обслуживаемых клиентов хранится в переменной с именем ClientServiceNumber, а максимальное количество одновременно работающих клиентов в глобальной переменной MaxClientServiceNumber. Тогда удобно в асинхронных процедурах ASStartMessage и ASFinishMessage выполнять увеличение и уменьшение значения переменной ClientServiceNumber.

**Не блокирующий режим работы сокета**

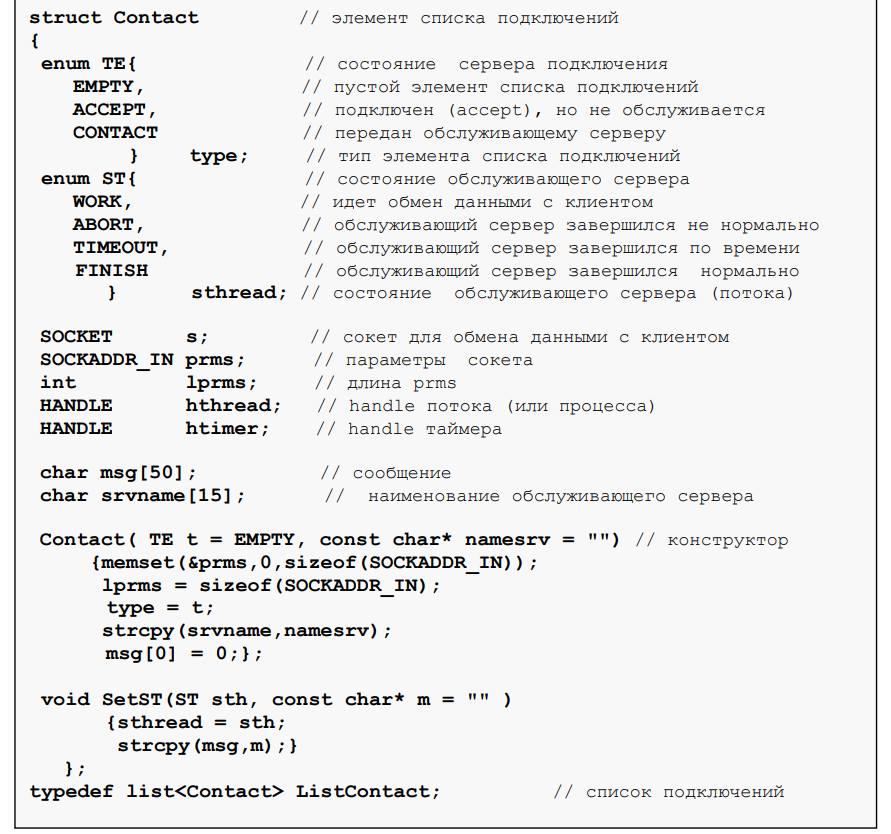
Для подключения клиентских программ AcceptServer использует функцию Winsock2 accept, которая приостанавливает работу потока AcceptServer до момента подключения клиента. После того, как клиентская программа выполнит функцию connect, функция accept сервера сформирует новый сокет для обмена данными между сервером и клиентом, а также возобновит исполнение потока AcceptServer.

Если в момент ввода команды stop, поток AcceptServer был приостановлен функцией accept, то действие этой команды наступит только после подключения следующего клиента, т.к. нет возможности проверить наличие, введенной и переданной функцией ConsolePipe команды. Введенная следующая команда может “затереть” предыдущую и команда stop не выполнена вообще.

Переключение сокета **в режим без блокировки** (nonblocking mode), позволяет избежать приостановки программы. В режиме без блокировки выполнение accept, не приостанавливает выполнение потока, как это было прежде, а возвращает значение нового сокета, если обнаружен запрос на создание канала (функция connect, выполненная клиентом).

Для переключения режимов сокета применяется функция ioctlsocket





12

# Широковещание. Обнаружение сервера с помощью широковещания.

Распределенное приложение не должно быть привязано к конкретным параметрам сокетов, т.к. это делает ограниченным его применение.

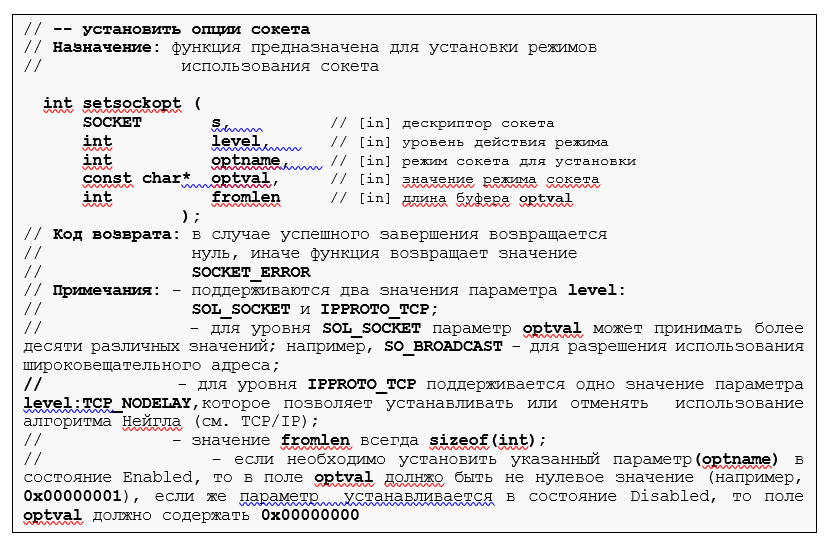
Для обеспечения независимости приложения от параметров сокета сервера (сетевой адрес и номера порта), как правило, номер порта делают одним из параметров инициализации сервера и хранят в специальных конфигурационных файлах, которые считывается сервером при загрузке

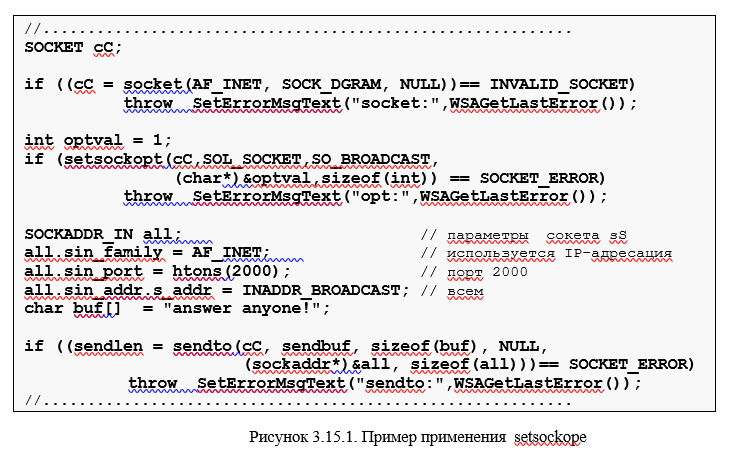
В некоторых случаях удобно возложить поиск сетевого адреса сервера на само клиентское приложение (при условии, что номер порта сервера известен). В этих случаях используются широковещательные сетевые адреса, позволяющие адресовать сообщение о поиске сервера всем компьютерам сети.

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. Сразу следует оговориться, что реально данный метод можно применять только внутри сегмент локальной сети, т.к. широковещательные пакеты, как правило, не пропускаются маршрутизаторами и шлюзами.

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

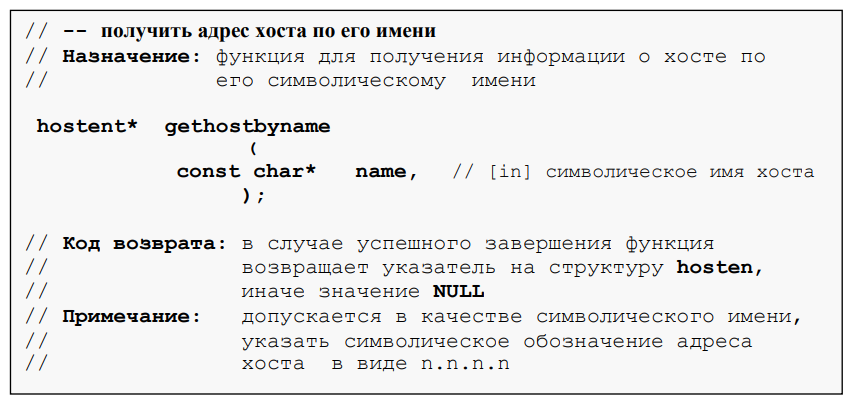
Стандартный широковещательный адрес в формате TCP/IP задается с помощью константы INADDR\_BROADCAST, которая определена в Winsock2.h. По умолчанию использование стандартного широковещательного адреса не допускается и для его применения необходимо установить специальный режим использования сокета SO\_BROADCAST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





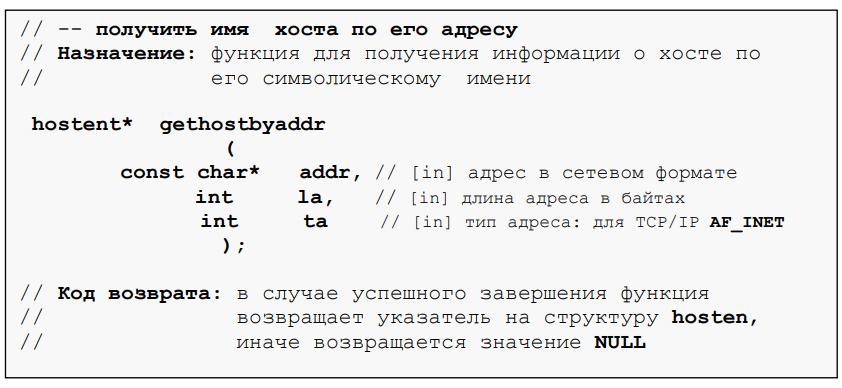
# Применение символического адреса хоста.

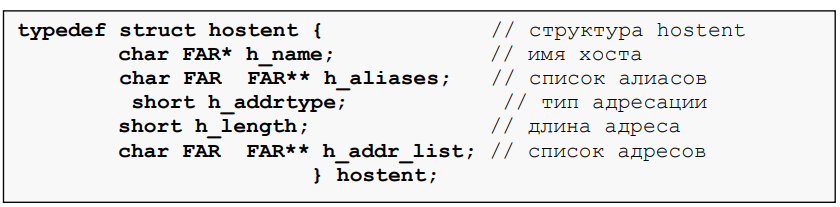
При наличии специальной службы в сети способной разрешить адрес компьютера по его символическому имени (например, DNS или некоторые протоколы, работающие поверх TCP/IP) поиск серверного компьютера можно осуществить с помощью функции gethostbyname. При этом предполагается, что известно символическое имя компьютера, на котором находится программа сервера.



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.

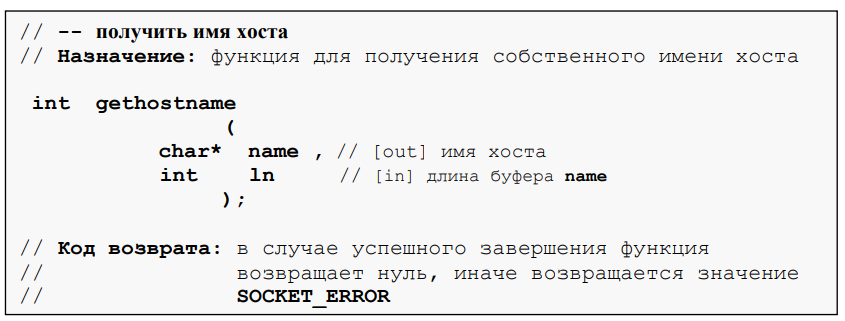
Помимо функции gethostbyname в составе Winsock2 имеется функция gethostbyaddr, назначение которой противоположно: получение символического имени компьютера по сетевому адресу. Обе функции используют структуру hosten, содержащуюся в Winsock2.h.





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 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.



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.

Сетевые утилиты – это внешние команды операционной системы, предназначенные для диагностики сети.

|  |  |
| --- | --- |
| **Наименование** | **Назначение** |
| **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.** 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 ping следует выполнить команду *ping без параметров*. В простейшем случае команда может быть применена с одним параметром:

**ping** hostname

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

ping -l 1000 hostname // размер буфера отправки 1000 байт

ping -n 17 hostname // количество отправляемых запросов равно 17

**Утилита 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*. В простейшем случае команда может быть применена с одним параметром:

**tracert** hostname

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

-h число узлов

**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).

Для получения справки о параметрах утилиты route следует выполнить команду *route без параметров.* В простейшем случае команда может быть использована для распечатки таблицы сетевых маршрутов:

**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

**netstat -b** для отображения исполняемых файлов участвующих в создании подключений

-p протокол Отображение подключений для протокола, заданного соответствующим

**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 справами администратора.

Для получения справки о параметрах утилиты arp, следует выполнить команду *arp без параметров*. 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.

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

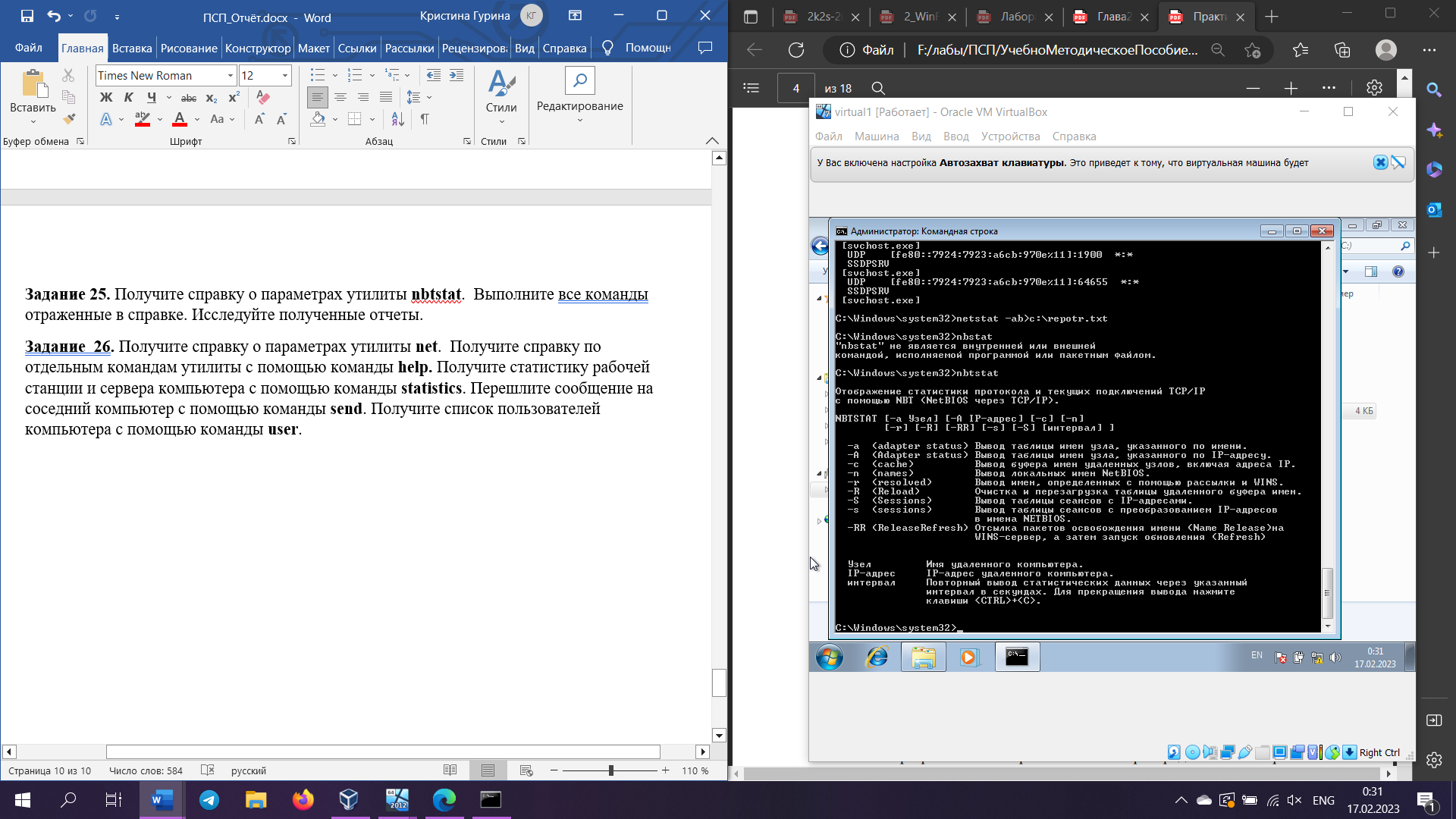
**Утилита ipconfig.** Утилита ipconfig является наиболее востребованной сетевой утилитой. С ее помощью можно определить конфигурацию IP-интерфейса и значения всех сетевых параметров. Особенно эта утилита полезна на компьютерах, работающих с протоколом DHCP: команда позволяет проверить параметры IP-интерфейсов,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

**Утилита nbtstat.** Утилита nbtstat позволяет просматривать статистику текущих соединений, использующих протокол NBT (NetBIOS over TCP/IP). Утилита в чем-то подобна утилите netstat, но применительно к протоколу NBT. Для получения справки о параметрах команды, необходимо ее выполнить без указания параметров.



**Утилита net.** Утилита net является основным средством управления сетью для сетевого клиента Windows. Команду net часто включают в скрипты регистрации и командные файлы. С помощью этой команды можно зарегистрировать пользователя в рабочей группе Windows, можно осуществить выход из сети, запустить или остановить сетевой сервис, управлять списком имен, пересылать сообщения в сети, синхронизировать время и т.д.

Для вывода списка параметров (команд) утилиты net следует выполнить следующую команду.

**net** help

Справка может быть уточнена для каждого отдельного параметра команды. Например, для того, чтобы получить справку для параметра send (пересылка сообщений в сети) следует добавить соответствующий параметр.

**net** help send

# 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).

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

Есть два 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**

Раньше, когда сети содержали небольшое количество узлов, достаточно было на каждом компьютере хранить и поддерживать актуальное состояние простого текстового файла, в котором содержался список сетевых узлов данной сети. В каждой строке текстового файла содержится пара «IP-адрес — имя сетевого узла».

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. Также эта 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, которая связывает имена сетевых узлов с их IP-адресами. 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.

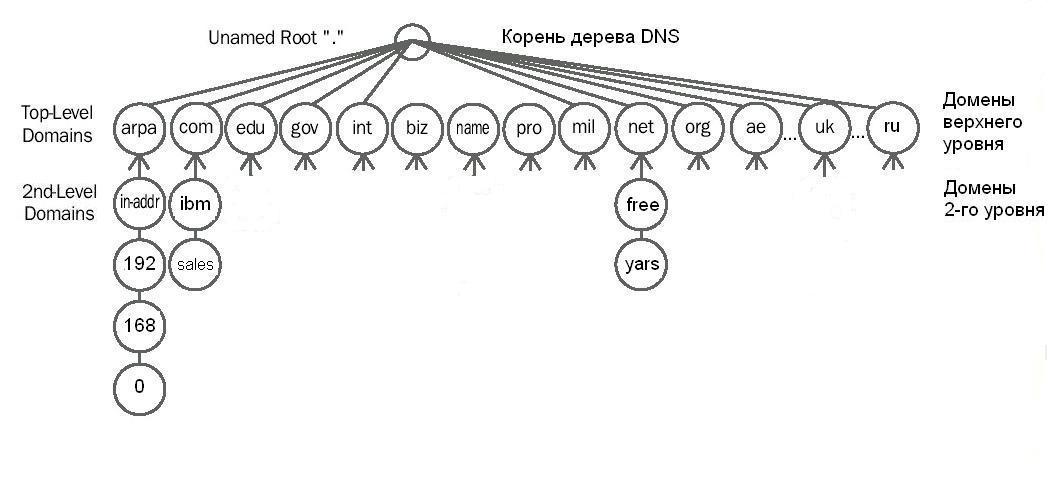
Процесс поиска в БД службы DNS имени некого сетевого узла и сопоставления этому имени IP-адреса называется «*разрешением имени узла в пространстве имен DNS»*.

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** 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.

**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*)
* **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.

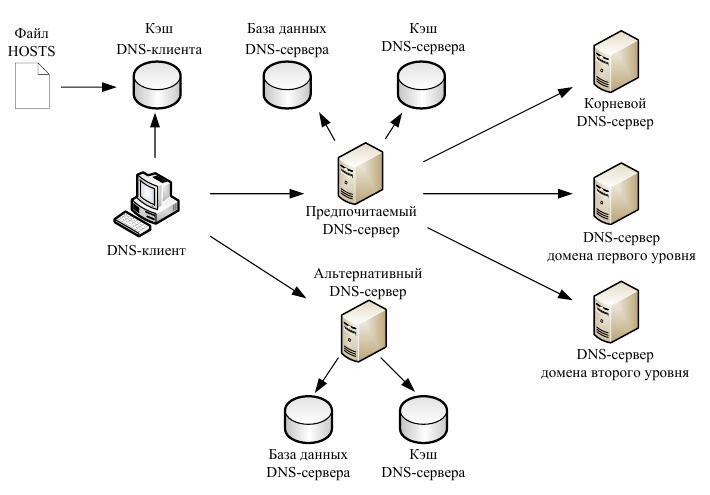


Рис. 5.3. Процесс обработки рекурсивного DNS-запроса

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 выдают адреса, контролируют их использование и сообщают клиентам требуемые параметры конфигурации. Сетевой администратор может задать диапазон адресов, распределяемых сервером среди компьютеров. Сервер ведет собственную регистрационную базу данных.

**Устройство протокола**

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 байт |
| 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 |

**Распределение IP-адресов**

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

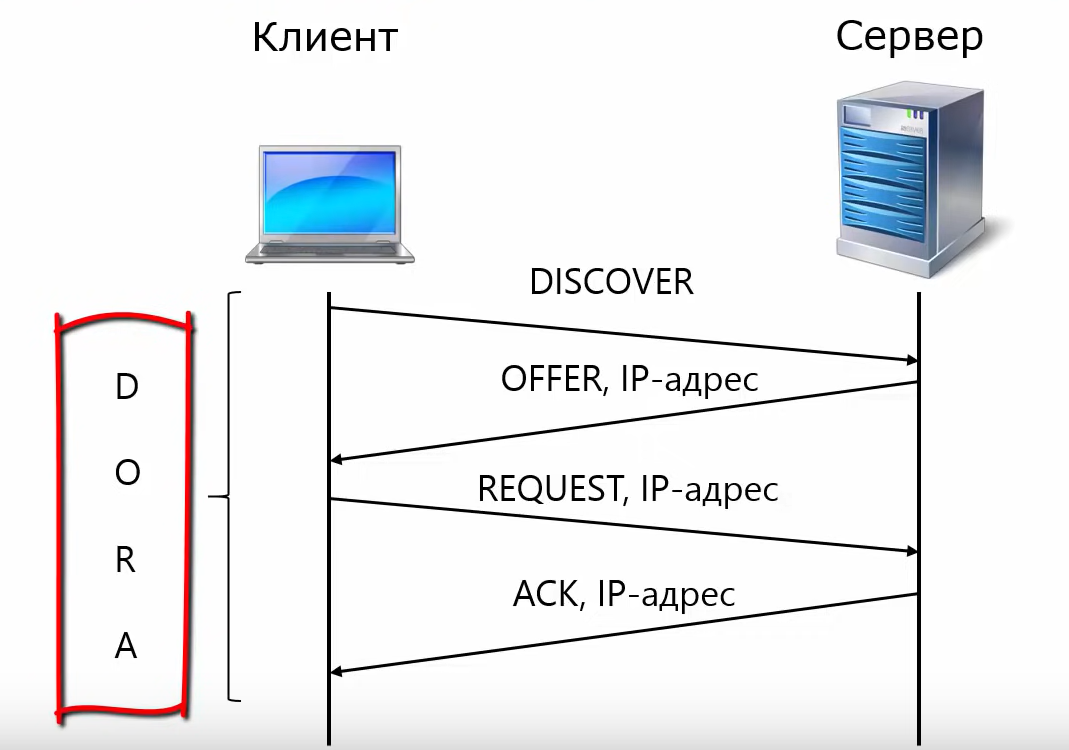
**Ручное распределение.** При этом способе сетевой администратор сопоставляет аппаратному адресу (обычно MAC-адресу) каждого клиентского компьютера определённый IP-адрес. Фактически, данный способ распределения адресов отличается от ручной настройки каждого компьютера лишь тем, что сведения об адресах 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).

Выдача адреса в аренду производится по запросу клиента. DHCP-сервер (или группа серверов) гарантирует, что выделенный адрес до истечения срока его аренды не будет выдан другому клиенту; при повторных обращениях сервер старается предложить клиенту адрес, которым тот пользовался ранее. Со своей стороны, клиент может запросить пролонгацию срока аренды адреса либо, наоборот, досрочно отказаться от него. При острой нехватке адресов сервер может сократить срок аренды адреса по сравнению с запрошенным.

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.

Выбор адреса DHCP-сервером. Если на момент получения запроса DHCPDISCOVER сервер не располагает свободными IP-адресами, он может направить уведомление о возникшей проблеме администратору. В противном случае при выборе адреса обычно применяется следующий алгоритм. Клиенту выделяется адрес, записанный за ним в данный момент. Если это невозможно, сервер предложит адрес, которым пользовался клиент до окончания срока последней аренды (при условии, что данный адрес свободен), либо адрес, запрошенный самим клиентом при помощи соответствующей опции (опять же, если адрес не занят). Наконец, в том случае, когда все предыдущие варианты не проходят, новый адрес выбирается из пула доступных адресов с учетом подсети, из которой поступил клиентский запрос.

Заметим, что исходя из определенной сетевым администратором политики сервер может выдать клиенту адрес, отличающийся от запрошенного (даже при доступности последнего), вообще отказать в предоставлении адреса или предложить адрес, относящийся к другой подсети. Более того, DHCP-сервер вообще не обязан реагировать на каждый поступивший запрос DHCPDISCOVER. Это предоставляет администратору возможность контролировать доступ к сети, например, разрешив серверу отвечать только тем клиентам, которые предварительно зарегистрировались с помощью специальной процедуры.

**Недостатки DHCP**

* низкий уровень информационной безопасности, что обусловлено непосредственным использованием протоколов UDP и IP.
* Нет никакой защиты от появления в сети несанкционированных DHCP-серверов

**Опции DHCP**

Помимо IP-адреса, DHCP также может сообщать клиенту дополнительные параметры, необходимые для нормальной работы в сети. Эти параметры называются опциями DHCP.

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.

# Стандарты сообщений Internet.

Сообщения, соответствующие спецификации RFC-5322, включают символы с десятичными кодами от 1 до 127, интерпретируемые в соответствии с кодировкой US-ASCII.

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.

Для удобства и с учетом ограничения размеров строки (998/78 символов), значение поля может быть разбито на несколько строк; это называется «**фальцовкой**» (folding). FWS (пробельные символы для фальцовки) означает указание места, где возможно выполнение фальцовки.

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).

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. Тело сообщения представляет собой последовательность символов, которая следует после раздела заголовков и отделена от него пустой строкой (строкой, содержащей только CRLF).

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.

Обязательными полями заголовка являются только поле даты и поле адреса отправителя сообщения. 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.

Поле from содержит имя «From» (*автора (авторов) сообщения*) и разделенный запятыми список из одного или нескольких имен почтовых ящиков. Если это поле содержит более одного адреса почтового ящика в списке, в сообщение должно включаться поле «Sender» *(реальный отправитель*) с указанием одного почтового ящика. Дополнительно может также включаться поле «Reply-To*», (адрес по которому нужно направить ответ)* содержащее список разделенных запятыми почтовых ящиков (не менее одного).

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).

Поле «Message-ID:» указано, как необязательное, но его следует включать в каждое сообщение. *(содержит один уникальный идентификатор сообщения).* В ответные сообщения следует включать поля «In-Reply-To:» *(для идентификации сообщения, на которое отвечает данное сообщение)* и «References:». Каждое из полей «References:» и «In-Reply-To:» содержит один или множество уникальных идентификаторов сообщений, опционально разделенных символами CFWS.

Поле «Subject:» описывает тему сообщения.

При использовании полей пересылки должны передаваться поля «Resent-From:» и «Resent-Date:». Следует передавать также поле «Resent-Message-ID:».

Некоторые символы имеют специальное значение (например, используются в качестве границ лексем). Для использования таких символов в общепринятом смысле служит механизм квотирования (добавления «кавычек»). Некоторые конструкции в теле структурированных полей заголовков представляют собой просто строки некоторых базовых символов. Такие конструкции называют атомами.

Задание адреса

Адреса появляются в нескольких полях заголовка для индикации отправителей и получателей сообщения. Адрес может задавать персональный почтовый ящик или группу почтовых ящиков.

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.

**Сообщение от одного лица другому с простой адресацией**

----

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.

**Пересылка сообщений**

----

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".

----

# 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

По умолчанию HTTP использует порт 80.

Запросы клиентов содержат URI (Uniform Resource Identifier) - универсальный идентификатор ресурса, позволяющий определить у сервера затребованный ресурс. **URI** представляет собой сочетание URL (Uniform Resource Locator) и URN (Uniform Resource Name). **URL** – унифицированный адресатор ресурсов: предназначен для указания места нахождения ресурса в сети. **URN** – унифицированное имя ресурса: идентифицирует ресурс, по указанному месту его нахождения (подразумевается, что по данному адресу может быть представлено несколько различных ресурсов).

В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ».

**Достоинства HTTP:**

• Простота

• Расширяемость

• Распространённость

**Недостатки HTTP:**

• Большой размер сообщений

• Отсутствие «навигации»

• Нет поддержки распределённости

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. Исключением является версия 0.9 протокола, у которой сообщение запроса содержит только стартовую строку, а сообщения ответа только тело сообщения.

***Стартовая строка***

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/1.1 200 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. В ответ серверу следует включить заголовок Allow со списком поддерживаемых методов. Также в заголовки ответа может включаться информация о поддерживаемых расширениях.

Для того чтобы узнать возможности всего сервера, клиент должен указать в URI звёздочку — «\*». Запросы «OPTIONS \* HTTP/1.1» могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1.

**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. Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»

**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. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.

**POST**: Применяется для передачи пользовательских данных заданному ресурсу. 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. Многократное повторение одних и тех же запросов POST может возвращать разные результаты (не кушируется). При результатах выполнения **200 (Ok)** и **204 (No Content)** в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ **201 (Created)** с указанием URI нового ресурса в заголовке Location

**PUT**: Применяется для загрузки содержимого запроса на указанный в запросе URI. 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. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус **201 (Created).** Если же был изменён ресурс, то сервер возвращает **200 (Ok)** или **204 (No Content).** Сообщения ответов сервера на метод PUT не кэшируются. различие методов **POST** и **PUT**

**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:**

1хх Информационные

2хх Успешные

3хх Перенаправление

4хх Ошибки клиента

5хх Ошибки сервера

**Заголовки HTTP** (англ. HTTP Headers) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение

1. **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.

Отделяются пустой строкой от тела сообщения.

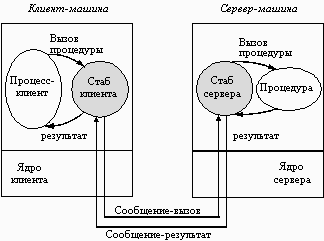
# Служба RPC.

**RPC** – удаленный вывод процедур.

**Remote Procedure Call или «удаленный вызов процедур»** представляет собой технологию межпроцессного взаимодействия [IPC](https://learn.microsoft.com/en-us/windows/win32/ipc/interprocess-communications#using-rpc-for-ipc). Она позволяет программам вызывать функции и процедуры удаленно таким образом

***Процесс передачи параметров при вызове удаленной процедуры называется маршалингом параметров (parameter marshaling)***

Главная особенность – вычислительная машина делает вычисления, клиенты посылают данные серверу на вычисления. Работа бывает синхронная и асинхронная. Синхронная клиент ждет ответ от сервера, асинхронная клиент отправляет данные на вычисление и продолжат свою работу. Процесс укомплектования параметров в сообщения называется – **маршалинг**. На стороне клиента имеется *СТАБ клиента – это заглушка которая осуществляет маршалинг.* Ядро клиента находится в операционной системе клиента. Процесс клиента передает СТАБУ клиента функцию, СТАБ переводит все в сообщение и передает ядру. Ядро клиента передает ядру сервера, ядро сервера передает СТАБУ сервера. СТАБ распаковывает сообщение вызывает функцию и передает параметры. Процедура отправляет СТАБУ и т.д. Клиентский СТАБ распаковывает данные и передает их процессу, который отправлял сообщение.



1. Процедура клиента вызывает клиентскую заглушку.

2. Клиентская заглушка создает сообщение и вызывает функцию RPC локальной ОС.

3. Служба RPC пересылает сообщение серверу

4. Служба RPC вызывает серверную заглушку и передает ей сообщение.

5. Серверная заглушка извлекает из сообщения параметры и вызывает удаленную процедуру.

6. Удаленная процедура выполняет код и возвращает параметры и значения серверной заглушке.

7. Серверная заглушка формирует сообщение и вызывает службу RPC своей локальной ОС.

8. Служба RPC сервера пересылает сообщение RPC ОС клиента.

9. RPC клиента возвращает сообщение заглушке.

10. Заглушка извлекает данные и передает их процессу.

Проблемы:

* Клиент и сервер могут применять разные кодировки.
* Клиент и сервере могут иметь разные представления данных.
* Передача параметров по ссылке. Клиент и сервер имеют разные адресные пространства. Один из способов решения пересылка всего параметра (копирование и восстановлении). Передача будет эффективнее если параметры будут помечены признаками **in**, **out**, **inout**.
* В каждой из частей необходимо договориться о внутреннем представлении данных.

Существует специальный язык **IDL** (Interface Definition Language) – кроссплатформенный язык определения интерфейсов. В шаблоне IDL необходимо определить название процедуры и ее параметры. Кроме этого, чтобы унифицировать вызов необходимо сделать уникальный идентификатор. Он создается в студии. Он состоит из букв, цифр и знаков, размером 128 бит. MIDL – при помощи этого компилятора на выходе 3 фала: Библиотека \*.h, клиентский СТАП и серверный СТАП с расширением \*.cpp. После компиляции на сервере получаем исполняемые файлы сервера и клиента.

IDL обычно описывает интерфейс между клиентом и сервером, который может быть реализован на разных языках программирования. Это позволяет клиентам и серверам, написанным на разных языках программирования, взаимодействовать друг с другом.

Могут случаться ситуации типа потери отправленных данных. Потери ответных сообщений от сервера к клиенту. Невозможность клиента определить местоположение сервера (сервер сменил имя). Сервер может претерпеть некую аварию и необходимо ждать пока сервер не перезагрузится и т.д и т.п.

XML-RPC (сокр. от англ. Extensible Markup Language Remote Procedure Call — XML-вызов удалённых процедур) — основанный на XML стандарт (протокол) вызова удалённых процедур, является прародителем SOAP, отличается исключительной простотой применения. XML-RPC, как и любой другой интерфейс RPC, определяет набор стандартных типов данных и команд, которые программист может использовать для доступа к функциональности другой программы, находящейся на другом компьютере в сети.

XML-RPC (XML Remote Procedure Call) – это простой протокол удаленного вызова процедур, который использует XML для передачи данных между клиентскими и серверными приложениями через Интернет.

Корпорация Майкрософт вскоре сочла этот протокол слишком упрощённым, и начала расширять его функциональность. После нескольких циклов по расширению функциональности, появилась система, ныне известная как SOAP. Позднее Майкрософт начала широко рекламировать и внедрять SOAP, а изначальный XML-RPC был отвергнут. Но, несмотря на отвержение Майкрософт, стандарт XML-RPC очаровал многих программистов своей необычайной простотой и, за счёт этого, существует по сей день и даже постепенно набирает популярность.

**Динамическое связывание**

1. Процедура клиента вызывает клиентскую заглушку.

2. Клиентская заглушка создает сообщение и вызывает функцию RPC локальной ОС.

3. RPC локальной ОС связывается с биндером, котлрый находит удаленный сервер по идентификатору и возвращает соединение

4. по этому соединению RPC локальной ОС RPC пересылает сообщение серверу

4. Служба RPC вызывает серверную заглушку и передает ей сообщение.

5. Серверная заглушка извлекает из сообщения параметры и вызывает удаленную процедуру.

6. Удаленная процедура выполняет код и возвращает параметры и значения серверной заглушке.

7. Серверная заглушка формирует сообщение и вызывает службу RPC своей локальной ОС.

8. Служба RPC сервера пересылает сообщение RPC ОС клиента.

9. RPC клиента возвращает сообщение заглушке.

10. Заглушка извлекает данные и передает их процессу.

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.

# NAT, proxy-серверы, межсетевые экраны, ремейлеры.

**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.

**Функционирование**

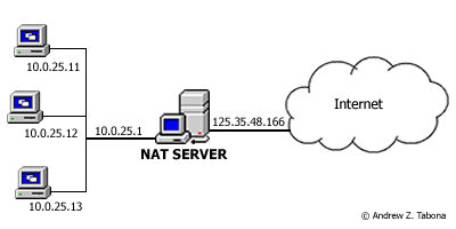
Преобразование адресов методом NAT может производиться почти любым маршрутизирующим устройством — маршрутизатором, сервером доступа, межсетевым экраном. Наиболее популярным является SNAT, суть механизма которого состоит в замене адреса источника (англ. source) при прохождении пакета в одну сторону и 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 раутером.

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.



Существует 3 вида NAT устройств:

• **Статический** – отображает не зарегистрированный на зарегистрированный на основании 1 к 1. Полезен, когда необходимо скрыть доступ к интернету

• **Динамический** – отображает незарегистрированный адрес на зарегистрированный адрес из группы зарегистрированных IP адресов.

• **Маскарадный** (Перегруженный) – форма динамического НАТ которая отображает несколько не зарегистрированных адресов в единый зарегистрированный IP адрес (т.е. использует порты).

**Преимущества**:

• Позволяет предотвратить и ограничить обращение снаружи ко внутренним хостам, оставляя возможность обращения изнутри наружу.

Позволяет сэкономить IP-адреса

• Позволяет скрыть определенные внутренние сервисы определенных внутренних хостов и серверов.

**Недостатки**:

• При использовании NAT хосты интернет взаимодействуют напрямую с NAT устройствами и не взаимодействуют напрямую с реальными хостами

• Использование NAT усложняет работу администратора

• Не все протоколы могут преодолевать NAT устройства

**Межсетевой экран (сетевой экран)** – это комплекс аппаратных или программных средств осуществляющий контроль и фильтрацию проходящих через него пакетов в соответствии с заданными правилами. (Фаервол и Брандмауэр). При передаче любых пакетов через фаерволы этот пакет проходит несколько стадий проверки. Если он проходит все стадии, то пакет передается, если не проходит хотя бы одну, то пакет удаляется. В ISO / OSI находятся на сетевом, сеансовом, и прикладном уровнях.

пакетных фильтров (сетевой уровень)

Шлюзы сеансового уровня

Шлюзы прикладного уровня

SPI брандмауэры объединяют в своей работе все 3 уровня с их фаерволами.

На сеансовом уровне блокируются TCP соединения. На прикладном уровне брандмауэры отвечают за доступ приложения в сеть, обмен почтовыми сообщениями, и определяют содержимое пакетов.

**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

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.

**Ремейлеры** – это сервер, получающий сообщение электронной почты и перенаправляющий его по адресу, указанному отправителем. В процессе переадресации вся информация об отправителе уничтожается, поэтому получатель лишен возможности узнать кто отправил сообщение.

Делятся на анонимные и псевдоанонимные:

1.Псевдоанонимные – сервер знает адрес электронной почты, который необходим для получения ответа на письмо.

2.Анонимные ремэйлеры полностью уничтожают адрес отправителя. При этом обеспечивается очень высокая безопасность, и нет гарантии своевременной доставки.

Стандарт MIXMINIOM анонимной перессылки почты. Mixminion может отсылать и принимать анонимные сообщения электронной почты.

# Web-сервисы: SOAP, XML, WSDL, UDDI.

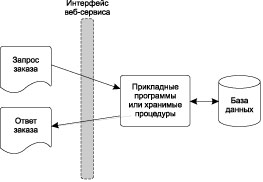
**Веб-сервисы** - это XML-приложения, могут взаимодействовать друг с другом и со сторонними приложениями посредством XML-сообщений, основанных на определённых протоколах ([SOAP](https://ru.wikipedia.org/wiki/SOAP), [XML-RPC](https://ru.wikipedia.org/wiki/XML-RPC) и т. д.)

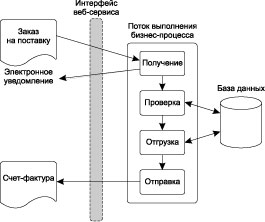
Стандарты веб-сервисов определяют формат таких сообщений, интерфейс, которому передается сообщение, правила привязки содержания сообщения к реализующему сервис приложению и обратно, а также механизмы публикации и поиска интерфейсов.

Стандарты и технологии веб-сервисов обычно подразумевают два основных типа моделей взаимодействия приложений:

• удаленный вызов процедуры (онлайновая);

• документно-ориентированный (пакетная).





**SOAP** (Simple Object Access Protocol) – это протокол обмена сообщениями в формате XML. SOAP определяет структуру сообщений, а также правила их передачи (т. е. какая информация должна пересылаться и как.)

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.

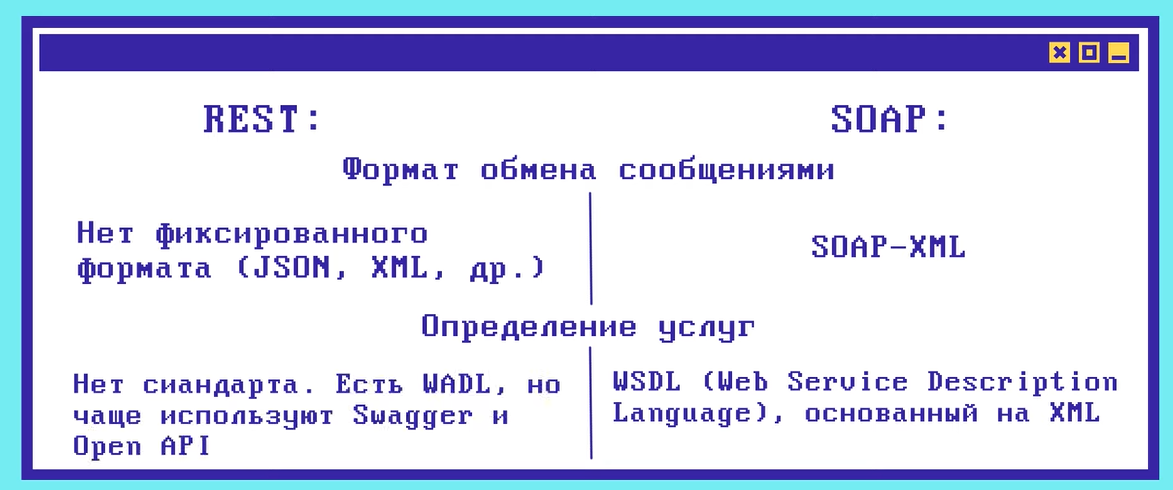
- **Body (тело сообщения)** - содержит XML-данные, передаваемые данным сообщением.

- 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.



**WSDL** – язык описания [веб-сервисов](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) и доступа к ним, основанный на языке [XML](https://ru.wikipedia.org/wiki/XML). WSDL описывают, как получить доступ к веб-службе и какие операции она будет выполнять.

Клиентская программа, подключающаяся к веб-службе, может прочитать WSDL, чтобы определить, какие функции доступны на сервере. Затем клиент может использовать SOAP для фактического вызова одной из функций, перечисленных в WSDL.

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 состоит из трех элементов:

• определение типов данных;

• операции *(список доступных операций веб-сервиса)*

• связывание сервисов. *(описывает, какой протокол и метод передачи сообщений следует использовать (например, HTTP или SOAP), а также какой конкретный порт должен использоваться для доступа к сервису)*

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** – это реестр веб-сервисов, который используется для поиска и доступа к удаленным сервисам. UDDI предоставляет каталог сервисов, а также связанных с ними описаний и классификаций. UDDI использует язык WSDL для описания веб-служб.





# Национальная инфраструктура информационной безопасности.

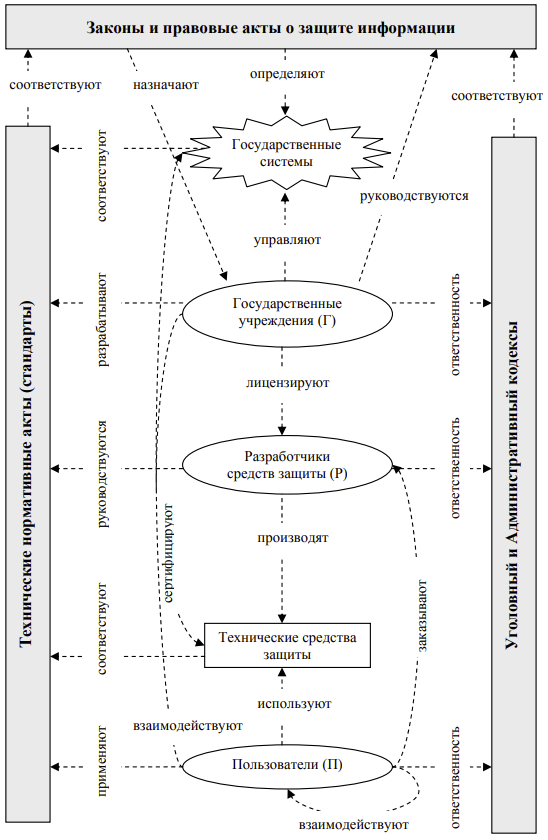
Инфраструктура национальной информационной безопасности – это комплекс организаций, средств и мероприятий, направленных на обеспечение защиты целостности, конфиденциальности и доступности информации. ИНИБ является составной частью общей системы национальной безопасности страны.

Государственные учреждения в сфере защиты информации разрабатывают Технические нормативные акты (стандарты), которым должно соответствовать ПО. Также они выдают лицензию разработчикам и осуществляет экспертизу (сертификацию) технических средств информационной безопасности.

Разработчики разрабатывают средства 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).

Статьи

1. Несанкционированный доступ к компьютерной информации
2. Модификация компьютерной информации
3. Компьютерный саботаж (Умышленные уничтожение, блокирование, приведение в непригодное состояние компьютерной информации или программы)
4. Неправомерное завладение компьютерной информацией
5. Изготовление либо сбыт специальных средств для получения неправомерного доступа к компьютерной системе или сети
6. Разработка, использование либо распространение вредоносных программ
7. Нарушение правил эксплуатации компьютерной системы или сети



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

**Конфиденциальность** – предотвращение попадание информации неавторизованным пользователям.

**Аутентификация** – проверка принадлежности субъекту предъявленного им идентификатора, подтверждающего личность. Процесс аутентификации может осуществляться:

• логин и пароль

• электронный сертификат

• смарт-карт

• идентификация личности по биометрическим данным

**Идентификация** – процесс присвоения субъектам идентификатора и сравнение идентификатора с перечнем идентификаторов.

**Авторизация** – процесс проверки прав субъекта на выполнение некоторых действий. Иногдапроцесс предоставление прав доступа (говорят: администратор системы авторизует пользователя – наделяет его перечнем прав и полномочий).

**Обеспечение целостности** – это защита данных от изменения и модификации.

Безопасность охватывает все уровни протоколов:

1. На физическом уровне можно поместить сетевой кабель в специальные герметические трубы, наполненные специальным газом (если просверлить, то утечка газа вызывает сигнал тревоги)

2. На канальном – аппаратное сжатие, шифрование, перемешивание

3. На сетевом – фаервол и брандмауэр

4. На транспортном – можно поддерживать зашифрованное соединение между процессами

5. На сеансовом – продолжительность действия ключей для шифров

6. На представительском – методы шифрования

7. На прикладном - процессы аутентификации

**Криптография**— наука о методах обеспечения конфиденциальности (невозможности прочтения информации посторонним) и аутентичности (целостности и подлинности авторства, а также невозможности отказа от авторства) информации.

Изначально криптография изучала методы шифрования информации — обратимого преобразования открытого (исходного) текста на основе секретного алгоритма и/или ключа в шифрованный текст (шифротекст) (Самый эффективный способ борьбы со сниффингом пакетов не предотвращает перехвата и не распознает работу снифферов).

**Криптоанализ** — наука о методах получения исходного значения зашифрованной информации, не имея доступа к секретной информации (ключу), необходимой для этого. В большинстве случаев под этим подразумевается нахождение ключа.

**криптология** = криптография + криптоанализ;

**Криптология** — наука, занимающаяся методами шифрования и дешифрования. Криптология состоит из двух частей — криптографии и криптоанализа. А также разработкой методов, позволяющих взламывать криптосистемы.

**Шифр**  — система обратимых преобразований, зависящая от некоторого секретного параметра ([ключа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F))) и предназначенная для обеспечения секретности передаваемой информации.

**Код** — алгоритм, который изменяет оригинальный текст, заменяя его символы на другие символы или цифры, что затрудняет понимание содержания сообщения без знания соответствующего ключа декодирования.

**Ключ шифра** — это параметр, который используется при шифровании данных и их расшифровке.

параметр криптографического алгоритма, обеспечивающий выбор одного преобразования из совокупности преобразований, возможных для этого алгоритма. В современной криптографии предполагается, что вся секретность криптографического алгоритма сосредоточена в ключе, но не деталях самого алгоритма.

**IPSec** – это комплекс протоколов касающихся вопросов шифрования, аутентификации и обеспечения защиты при транспортировке IP пакетов. Он включает около 20ти предложений по стандартам и 18ти RFC.

Основными функциями IPSec являются:

• Обеспечение конфиденциальности - отправитель должен иметь возможность шифровать пакеты до их отправки.

• Обеспечение целостности

• Обеспечение защиты от воспроизведения пакетов

Протоколы IPSec:

• IKE – обеспечение аутентификации сторон

• AH – обеспечивает аутентификацию пакетов и выявление их воспроизведение

• ESP – обеспечивает конфиденциальность, аутентификацию источника и целостность данных, а также сервис защиты от воспроизведения пакетов

• HMAC – механизм аутентификации сообщений с использованием хэш функций

• DES – стандарты шифрования данных

Существует 2 режима работы:

**Транспортный** – шифруется только информативная часть IP пакета.

**Туннельный** – IP пакет шифруется целиком. IP пакет вкладывается в другой IP пакет.

**SSL/TSL** TLS (что есть Transport Layer Security), он же ранее известный как SSL (Secure Sockets Layer), на данный момент является стандартом де-факто для защиты протоколов транспортного уровня от различных методов вмешательства извне. Много кто его использует.

протоколы, которые обеспечивают защищенное соединение между клиентом и сервером в интернете.

**HTTPS** (Hypertext Transfer Protocol Secure) — расширение протокола HTTP, поддерживающее шифрование. Данные, передаваемые по протоколу HTTPS, «упаковываются» в криптографический протокол SSL или TLS, тем самым обеспечивается защита этих данных. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443.

это защищенная версия протокола HTTP, который использует SSL/TLS для обеспечения безопасности передачи данных

**DNSSEC** (англ. Domain Name System Security Extensions) — это расширение протокола DNS для обеспечения безопасности. DNSsec используется для защиты пользовательских данных от перехвата или подмены.

набор спецификаций IETF, обеспечивающих безопасность информации, предоставляемой средствами DNS в IP-сетях. Он обеспечивает DNS-клиентам аутентификацию данных DNS либо аутентификацию информации о факте отсутствия данных и их целостность. Не обеспечивается доступность данных и конфиденциальность запросов.