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

В 1984 году Международной Организацией по Стандартизации (International Standard Organization, ISO) была разработана **модель взаимодействия открытых систем** (Open Systems Interconnection, OSI). Модель представляет собой международный стандарт для проектирования сетевых коммуникаций и предполагает уровневый подход к построению сетей. Каждый уровень модели обслуживает различные этапы процесса взаимодействия. Посредством деления на уровни сетевая модель OSI упрощает совместную работу оборудования и программного обеспечения. Модель OSI разделяет сетевые функции на семь уровней: прикладной, уровень представления, сеансовый, транспортный, сетевой, канальный и физический.

**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.

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

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

|  |  |  |
| --- | --- | --- |
| ISO/OSI | TCP/IP |  |
| Прикладной | Прикладной | На этом уровне находятся службы TCP/IP и прикладные системы пользователя. |
| Представительский |
| Сеансовый |
| Транспортный | Обеспечивает сквозную доставку данных произвольного размера по сети между прикладными процессами, запущенными на узлах сети (UDP – ориентирован на сообщения, TCP – на соединение). |
| Транспортный |
| Сетевой | Межсетевой | Осуществляет перенос между сетями различных типов адресной информации в унифицированной форме. Сборка и разборка пакетов при передаче их между сетями с различными максимальными значениями длины пакета (IP, ICMP, ARP). |
| Канальный |
| Уровень доступа к сети | Используются протоколы, обеспечивающие создание локальных сетей или соединений с глобальными сетями (Ethernet, PPP). |
| Физический |

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. Способы связи между клиентом и сервером могут быть различными и в общем случае зависят от интерфейсов, поддерживаемых операционной средой, в которой работает распределенное приложение. Клиент должен быть тоже распознан сервером, для того чтобы сервер, во-первых, мог его отличить от других клиентов, а во-вторых, чтобы смог обмениваться с клиентом данными. Если основная вычислительная нагрузка ложится на сервер, а клиент лишь обеспечивает интерфейсом пользователя с сервером, то такой клиент часто называют **тонким**.

По методу облуживания серверы подразделяются на **итеративные** и **параллельные** серверы. 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.

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

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

|  |  |  |
| --- | --- | --- |
| ISO/OSI | TCP/IP |  |
| Прикладной | Прикладной | На этом уровне находятся службы TCP/IP и прикладные системы пользователя. |
| Представительский |
| Сеансовый |
| Транспортный | Обеспечивает сквозную доставку данных произвольного размера по сети между прикладными процессами, запущенными на узлах сети (UDP – ориентирован на сообщения, TCP – на соединение). |
| Транспортный |
| Сетевой | Межсетевой | Осуществляет перенос между сетями различных типов адресной информации в унифицированной форме. Сборка и разборка пакетов при передаче их между сетями с различными максимальными значениями длины пакета (IP, ICMP, ARP). |
| Канальный |
| Уровень доступа к сети | Используются протоколы, обеспечивающие создание локальных сетей или соединений с глобальными сетями (Ethernet, PPP). |
| Физический |

**Публичный IP-адрес** – уникальный адрес.

**Частный IP-адрес** – адрес внутри локальной сети.

Публичные "белые" адреса используются в сети Интернет. **Публичным** IP-адресом называется IP-адрес, который используется для выхода в Интернет. Доступ к устройству с публичным IP-адресом можно получить из любой точки глобальной сети, т.к. публичные (глобальные) IP-адреса маршрутизируются в Интернете, в отличие от частных (серых) IP-адресов.

Однако, в связи с тем, что "белых" адресов существует ограниченное количество, в локальных сетях используются частные (серые) IP-адреса. Они не маршрутизируются в Интернете и на них нельзя отправить трафик из Интернета, только из конкретной частной локальной сети. Для домашних пользователей провайдер может предоставлять один или несколько публичных IP-адресов (как правило, это платная услуга). Маршрутизатор позволяет компьютерам локальной сети (с частными IP-адресами) использовать для выхода в Интернет публичный IP-адрес, установленный на внешнем интерфейсе (WAN) устройства. Именно этот внешний IP-адрес может быть использован для доступа из Интернета в локальную сеть.

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

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

Таблица 2.5.1

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

Хорошо известные номера портов присваиваются базовым системным службам (core services), имеющие системные привилегии. Зарегистрированные номера портов присваиваются промышленным приложениям и процессам. Распределение некоторых хорошо известных и зарегистрированных номеров портов приведено в таблице 2.5.2. 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)

22 TCP Secure Shell (SSH)

23 TCP Telnet

25 TCP Simple Mail Transfer Protocol (SMTP)

53 TCP Domain Name Server (DNS)

53 UDP Domain Name Server (DNS)

66 TCP Oracle SQL\*NET

67 UDP Dynamic Host Configuration Protocol (DHCP) Server

68 UDP Dynamic Host Configuration Protocol (DHCP) Client

80 TCP World Wide Web (WWW)

110 TCP Post Office Protocol, Version 3 (POP3)

111 TCP Remote Procedure Call (RPC)

111 UDP Remote Procedure Call (RPC)

143 TCP Internet Message Access Protocol (IMAP4)

1352 TCP Lotus Notes

1433 TCP Microsoft SQL Server

1522 TCP Oracle SQL Sever

Публичный IP-адрес - это 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.

# 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.** Термин Ethernet обычно связывают со стандартом, опубликованным в 1982 г. совместно корпорациями DEC, Intel и Xerox (DIX).

На сегодняшний день это наиболее распространенная технология локальных сетей. Ethernet применяет метод доступа CSMA/CD (множественный доступ с прослушиванием несущей частоты и обнаружением коллизий), использует 48-битную адресацию (стандарт IEE EUI-64) и обеспечивает передачу данных до 1 гигабита в секунду. Максимальная длина кадра, передаваемая в сети Ethernet составляет 1518 байт, при этом сами данные могут занимать от 46 до 1500 байт. Физически функции протокола Ethernet реализуются сетевой картой, которая может быть подключена к кабельной системе с фиксированным МАС-адресом производителя (в соответствии со стандартом ICANN).

**Протокол SLIP** (Serial Line IP). Аббревиатурой SLIP обозначают межсетевой протокол для последовательного канала. Раньше SLIP использовался для подключения домашних компьютеров к Internet через последовательный порт RS-232. Протокол использует простейшую инкапсуляцию кадра и 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. Ряд недостатков были исправлены в новой версии протокола именуемой CSLIP (Compressed SLIP).

**Протокол PPP** (Point-to-Point Protocol). PPP – универсальный протокол двухточечного соединения: поддерживается TCP/IP, NetBEUI, IPX/SPX, DECNet и многими стеками протоколов. Протокол может применяться для технологии ISDN (Integrated Services Digital Network) и SONET (Synchronous Optical Network). PPP поддерживает многоканальные реализации: можно сгруппировать несколько каналов с одинаковой пропускной способностью между отправителем и получателем. Кроме того, PPP обеспечивает циклический контроль для каждого кадра, динамическое определение адресов, управление каналом. 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.** В семействе протоколов TCP/IP протоколу IP отведена центральная роль. Его основной задачей является доставка дейтаграмм (так называется единица передачи данных в терминологии IP). При этом протокол по определению является **ненадежным** и **не поддерживающим соединения**.

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

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

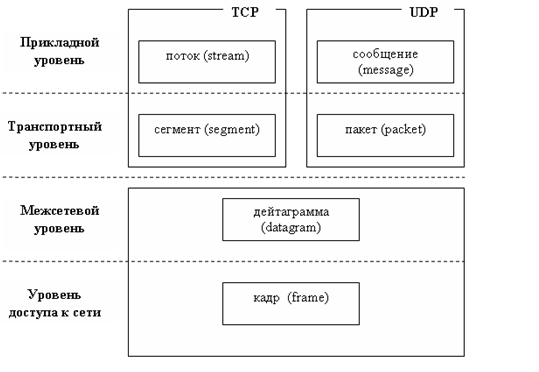
**Протокол IPv6.** Наиболее распространенной на настоящий момент версией протокола IP является IPv4 – именно об этой версии говорилось выше. Этот протокол оказался самым удачным сетевым протоколом из всех, когда-либо созданных. Поэтому IPv4 быстро превратился в стандарт. Можно сказать, что протокол IPv4 стал жертвой собственной популярности, т.к. предлагаемое полезное пространство адресов практически исчерпано (2^32). Как результат усилий, направленных на решение этой проблемы появился протокол IPv6, в котором попутно было реализовано много других новых возможностей. Главным отличительным признаком протокола IPv6 является *128-битный адрес*, позволяющий увеличить адресное пространство боле чем на 20 порядков. Основная концепция IPv6: каждый отдельный узел должен иметь собственный уникальный идентификатор интерфейса. Кроме того, протокол IPv6 требует соответствие идентификаторов интерфейсов формату IEEE EUI-64, позволяющему применять фиксированные (“зашитые” при изготовлении в специальную память сетевой платы) MAC-адреса сетевых плат. Например, 48-битный МАС-адрес платы Ethernet изначально предназначен для глобальной идентификации. Первые 24 бита этого адреса обозначают производителя платы (в соответствии с кодировкой ICANN) и индивидуальную партию изделия, а остальные 24 бита определяются производителем, с таким расчетом, чтобы каждый номер был уникален в пределах всей его продукции. Таким образом уникальный идентификатор интерфейса IPv6 на основе Ethernet cодержит в младших 64-х разрядах 128-битного адреса MAC-адрес платы Ethernet. Причем дополнение 48-бит адреса MAC-адреса до 64 бит осуществляется добавлением 16 бит (0xFFFF) между двумя его половина ми. На первоначальном этапе внедрения IPv6 предполагается совместное использование обеих версий IP-протокола. При этом предполагается использование, так называемых, IPv4-совместимых и IPv4-преобразованных адресов. Другой интересной особенностью IPv6 является возможность автоконфигурации. Автоконфигурация – это процесс, позволяющий хосту находить информацию для настройки собственных IP-параметров. В версии IPv6 основным средством, позволяющим выполнять подобную настройку является протокол DHCP. Пересмотр процесса автоконфигурации вызван сложность администрирования сетей с большим количеством хостов и в связи с необходимостью поддерживать мобильных (перемещающихся) пользователей. Большое внимание в новой версии протокола уделяется вопросам безопасности.

**Протокол ICMP.** Спецификация протокола ICMP (Протокол контроля сообщений в Internet) изложена в документе RFC 792. ICMP является неотъемлемой частью TCP/IP и предназначен для транспортировки информации о сетевой деятельности и маршрутизации. IPCM сообщения представляют собой специально отформатированные IP-дейтаграммы, которым соответствуют определенные типы (15 типов) и коды сообщений. Описание типов и кодов IPCM-сообщений содержится в [5]. С помощью протокола ICMP осуществляется деятельность утилит достижимости (ping, traceroute); регулируется частота отправки IP-дейтаграмм, оптимизируется MTU для маршрута передачи IP-дейтаграмм; доставляется хостам, маршрутизаторам и шлюзам всевозможная служебная информация; осуществляется поиск и переадресация маршрутизаторов; оптимизируются маршруты; диагностируются ошибки и оповещаются узлы IP-сети.

**Протокол ARP.** IP-адреса могут восприниматься только на сетевом уровне и вышестоящих уровнях TCP/IP. На канальном уровне всегда действует другая схема адресации, которая зависит от используемого протокола. Например, в сетях Ethernet используются 48-битные адреса. Для установки соответствия между 32-разрядными IP-адресами и теми или иными MAC-адресами, действующими на канальном уровне, применяется механизм привязки адресов по протоколу ARP, спецификация которого приведена в документе RFC 826. Основной задачей ARP является динамическая (без вмешательства администратора, пользователя, прикладной программы) проекция IP-адресов в соответствующие МАС-адреса аппаратных средств. Эффективность работы ARP обеспечивается тем, что каждый хост кэширует специальную ARP-таблицу. Время существования записи в этой таблице составляет обычно 10-20 минут с момента ее создания и может быть изменено с помощью параметров реестра. Просмотреть текущее состояние ARP-таблицы можно с помощью команда arp. Кроме того, протокол ARP используется для проверки существования в сети дублированного IP-адреса и разрешения запроса о собственном MAC-адресе хоста во время начальной загрузки.

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

**Протоколы транспортного уровня**

****

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

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

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

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.** Протокол UDP является протоколом **без установления соединения**. Спецификация протокола описывается в документе RFC 768. Основными свойствами протокола являются:

1) отсутствие механизмов обеспечения надежности: пакеты не упорядочиваются, и их прием не подтверждается;

2) отсутствие гарантий доставки: пакеты оправляются без гарантии доставки, поэтому процесс Прикладного уровня (программа пользователя) должен сам отслеживать и обеспечивать (если это необходимо повторную передачу);

3) отсутствие обработки соединений: каждый оправляемый или получаемый пакет является независимой единицей работы; UDP не имеет методов установления, управления и завершения соединения между отправителем и получателем данных;

4) UDP может по требованию вычислять контрольную сумму для пакета данных, но проверка соответствия контрольной суммы ложится на процесс Прикладного уровня;

5) отсутствие буферизации: UDP оперирует только одним пакетом, и вся работа по буферизации ложится на процесс Прикладного уровня;

6) UDP не содержит средств, позволяющих разбивать сообщение на несколько пакетов (фрагментировать) – вся эта работа возложена на процесс Прикладного уровня.

Следует обратить внимания, что протокол UDP характеризуется тем, что он не обеспечивает. Все перечисленные отсутствующие характеристики присутствуют в протоколе TCP. Фактически UDP – это тонкая прослойка интерфейса, обеспечивающая доступ процессов Прикладного уровня непосредственно к протоколу IP.

**Надежный и ненадежный протоколы**:

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

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

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

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

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

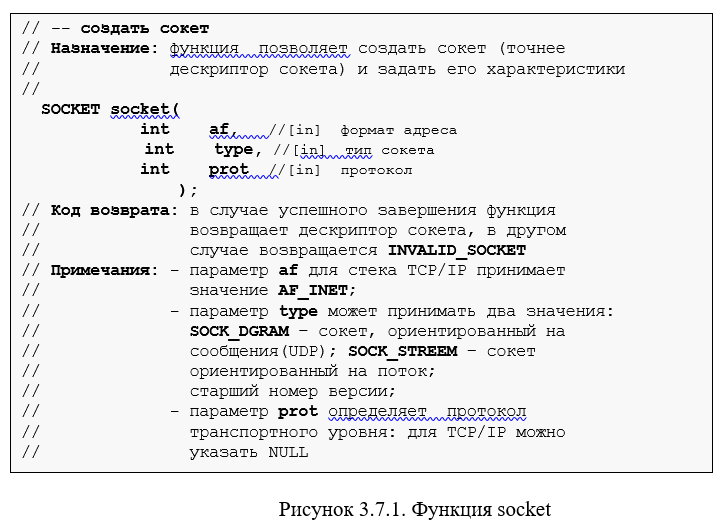
**Протоколы, ориентированные на сообщения**, передают данные в виде отдельных сообщений или пакетов данных. Каждое сообщение может быть обработано независимо от других сообщений. UDP является

# Понятие сокета. Основные параметры сокета.

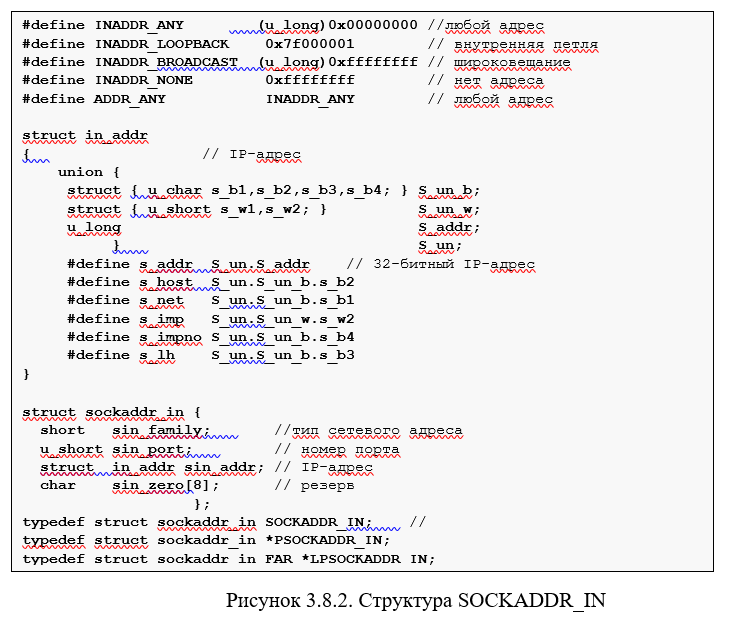
Интерфейс, который представляет совокупность IP-адреса и номера порта называется **сокетом**. Сокет однозначно идентифицирует прикладной процесс в сети TCP/IP. Следует помнить, что одни и те же номера портов могут быть использованы как для протокола UDP, так и для протокола TCP.

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

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** |
| **accept(**  **Socket s, сокет**  **Sockaddr\_in\* указатель на сокадр**  **Int\* la указатель на длину сокадр);** | Разрешить подключение к сокету |
| **bind( Socket s сокет**  **sockaddr\_in\* указатель на сокадр**  **int la длина сокадр в байтах)** | Связать сокет с параметрами (структура сокадр) |
| **closesocket(socket s, дескриптор сокета)** | Закрыть существующий сокет |
| **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 |
| **listen(**  **socket s, сокет**  **int mcq максимальная длина очереди)** | Переключить сокет в режим прослушивания |
| **recv(**  **socket сокет**  **char\* buf буфер для данных**  **int lbuf, длина буфера**  **int flags флаги (индикатор))** | Принять данные по установленному каналу |
| **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 |
| **send(**  **socket s сокет**  **char\* buf буфер данных**  **int lbuf длина буфера**  **int flags индикатор особого режима)** | Отправить данные по установленному каналу |
| **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.

Структура **SOCKADDR\_IN** содержит три значения (параметры сокета):

• тип используемого адреса (константа AF\_INET используется для обозначения семейства IP-адресов);

• номер порта (устанавливается значение 2000 с помощью функции htons)

• адрес интерфейса.

Последний параметр определяет собственный IP-адрес сервера. При этом предполагается, что хост, в общем случае, может иметь несколько IP-интерфейсов. Если требуется использовать определенный IP-интерфейс хоста, то необходимо его здесь указать. Если выбор IP-адреса не является важным или IP-интерфейс один на хосте, то следует указать значение INADDR\_ANY (как это сделано в примере).

**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

**};**

**SOCKET socket (**

**int af,** //[in] формат адреса

**int type,** //[in] тип сокета

**int prot** //[in] протокол

**);**

- параметр **af** для стека TCP/IP принимает значение AF\_INET;

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

• **SOCK\_DGRAM** – сокет, ориентированный на сообщения(UDP);

• **SOCK\_STREEM** – сокет, ориентированный на поток;

• старший номер версии;

- параметр **prot** определяет протокол транспортного уровня:

• для TCP/IP можно указать NULL

Все функции интерфейса Winsock2 могут завершаться успешно или с ошибкой. При описании каждой функции будет указано, каким образом можно проверить успешность ее завершения. В том случае, если функция завершает свою работу с ошибкой, формируется дополнительный диагностирующий код, позволяющий уточнить причину ошибки.

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

# Интерфейс Nimed Pipe.

Современные операционные системы имеют встроенные механизмы межпроцессного взаимодействия - IPC (InterProcess Communication), предназначенные для обмена данными между процессами и для синхронизации процессов.

Разработчики программного обеспечения могут использовать IPC с помощью предоставляемых операционными системами программных интерфейсов (API).

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

Именованные каналы бывают: **дуплексные** (позволяющие передавать данные в обе стороны) и **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.

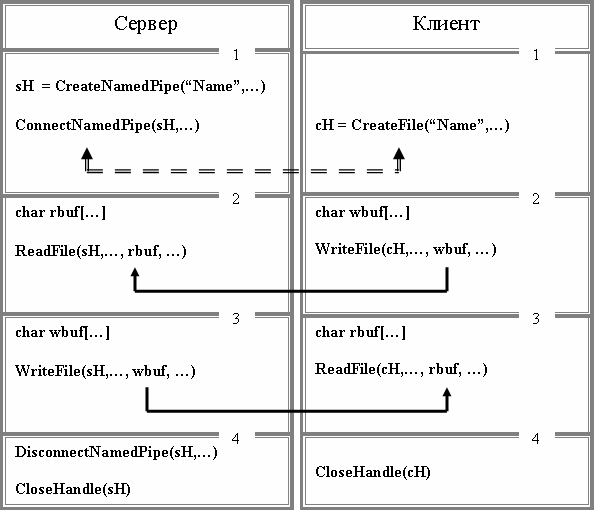
Все функции Named Pipe API можно разбить на три группы: **функции управления каналом** (создать канал, соединить сервер с каналом, открыть канал, получить информацию об именованном канале, получить состояние канала, изменить характеристики канала); **функции обмена данными** (писать в канал, читать из 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 |
| **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 |
| **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 |
| **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 |
| **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 таблице 4.2.1 перечислены основные функции интерфейса Named Pipe. Следует отметить, что функции CreateFile, ReadFile, WriteFile, которые тоже перечислены в таблице 4.2.1, применяются не только для работы с именованными каналами, но и для работы с файловой системой, с сокетами и т.д. Поэтому эти универсальные функции часто не указывают в составе Named Pipe API.

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



На рисунке 4.2.1 изображены две программы, реализующие два процесса распределенного приложения. Каждая из программ разбита на четыре блока.

Сплошными направленными линиями обозначается движение данных от одного процесса к другому. Прерывистой линией обозначается синхронизация процессов.

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

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

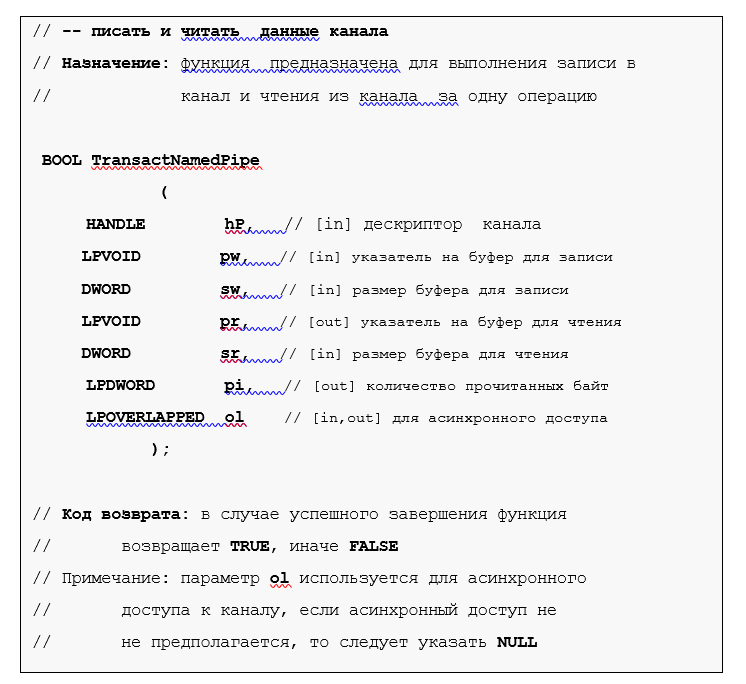
В последнем четвертом блоке программы сервер разрывает соединение с помощью функции DisconnectNamedPipe и закрывает дескриптор именованного канала.

Для программы клиента остается пояснить только первый блок, т.к. всем остальным блокам, есть аналогичные в программе сервера. В первом блоке программы клиента выполняется функция CreateFile, одним из параметров которой является строка с именем канала. Если к моменту выполнения функции канал уже создан и сервер подсоединился к каналу, то функция CreateFile возвращает дескриптор именованного канала, который потом используется в других функциях программы клиента. Иногда перед выполнением функции CreateFile, выполняют функцию WaitNamedPipe, позволяющую определить доступность экземпляра канала. 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. Схема изображенная на рисунке 4.2.1 является общей для этих двух случаев.

**Передача транзакций по именованному каналу**

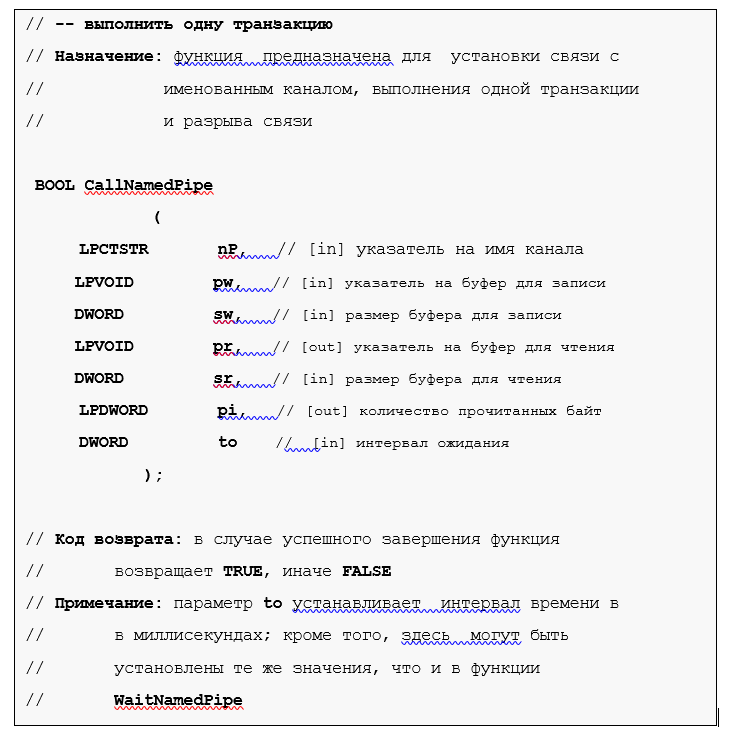
Для обмена сообщениями по сети может использоваться функция **TansactNamedPipe**, которая объединяет операции чтения и записи в одну операцию. Такую объединенную операцию называют **транзакцией именованного канала**. Функция TansactNamedPipe может быть использована только в том случае, если сервер именованного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



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

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

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

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

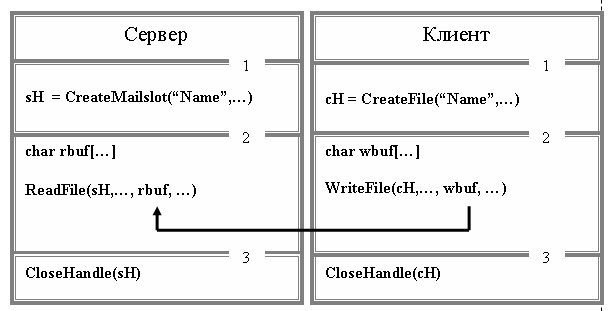
**Почтовым ящиком** (Mailslot) называется объект ядра операционной системы, который обеспечивает передачу данных от процессов-клиентов к процессам-серверам, выполняющимся на компьютерах в одной локальной сети. Процесс, создающий почтовый ящик называется **сервером** почтового ящика. Процессы, которые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

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



# 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.

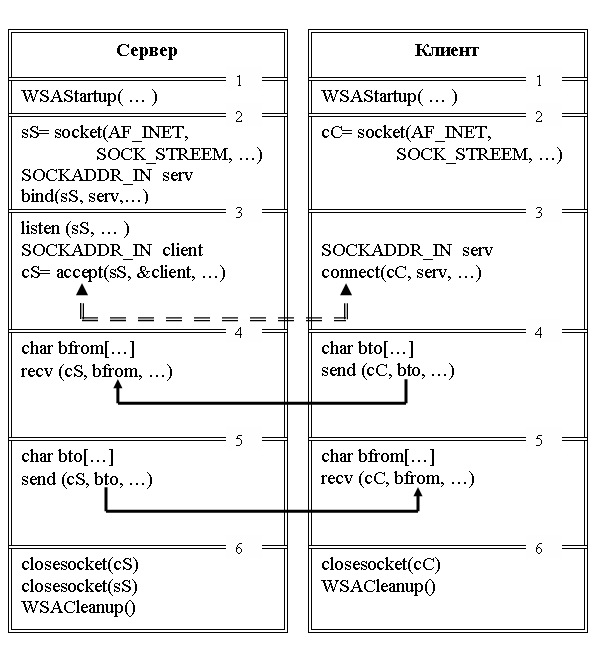
Принципиальное различие этих схем, заключается в следующем.

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.

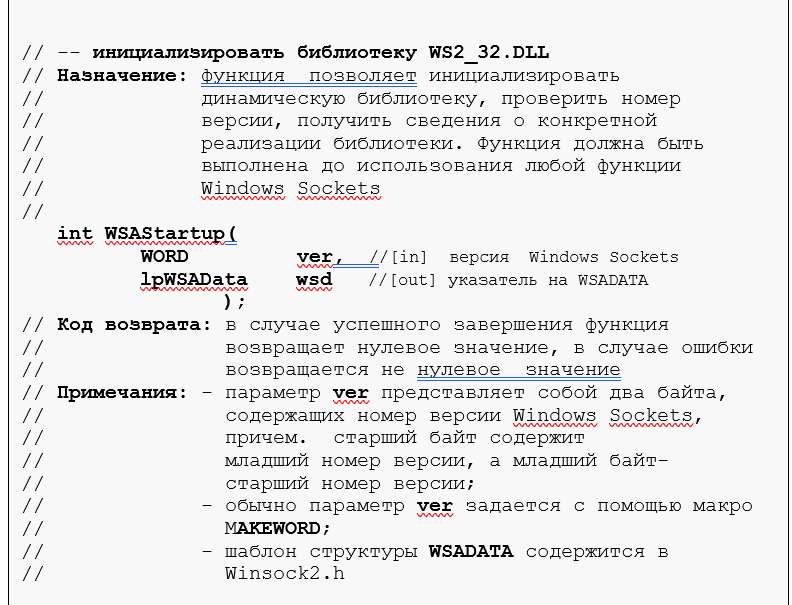
Во втором случае между сокетами устанавливается TCP-соединение и весь обмен данных осуществляется в рамках этого соединения. Передача по каналу является надежной и данные поступают в порядке их отправления.

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

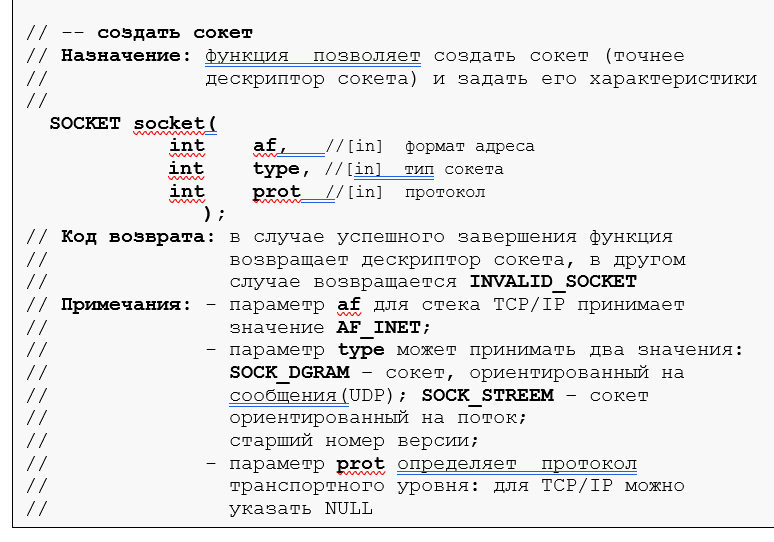
Взаимодействие программ клиента и сервера в случае установки соединения схематично изображено на рисунке. Как и в предыдущем случае обе программы разбиты на блоки. Сплошными направленными линиями обозначается движение данных по сети TCP/IP, прерывистой – синхронизация (ожидание) процессов.

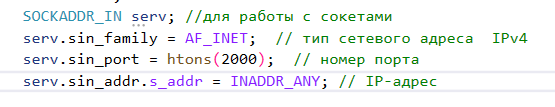
****

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

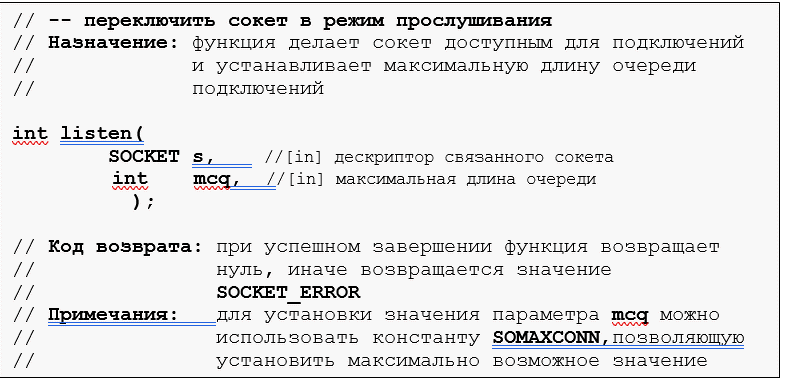


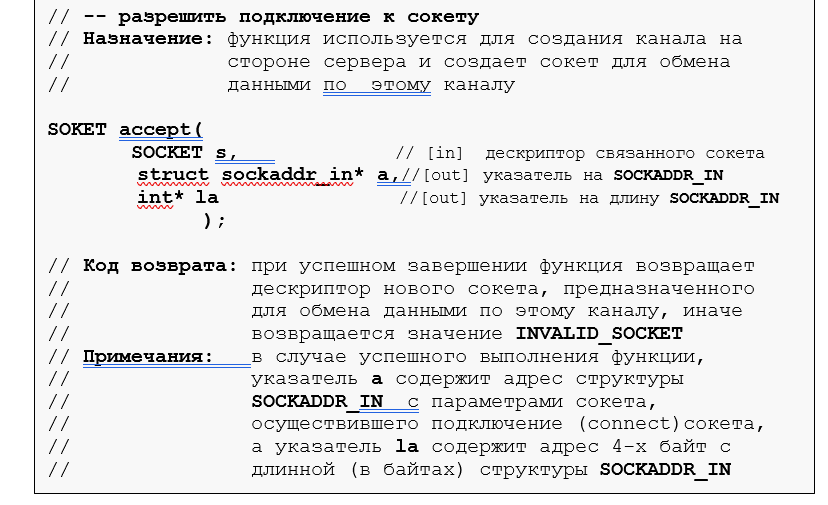
Второй блок сервера имеет то же предназначение, что и в предыдущем случае. Единственным отличием является значение SOCK\_STREEM параметра функции socket, указывающий, что сокет будет использоваться для соединения (сокет, ориентированный на поток).



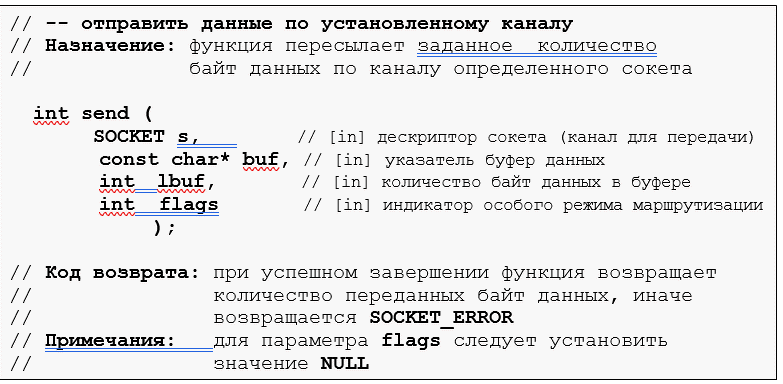
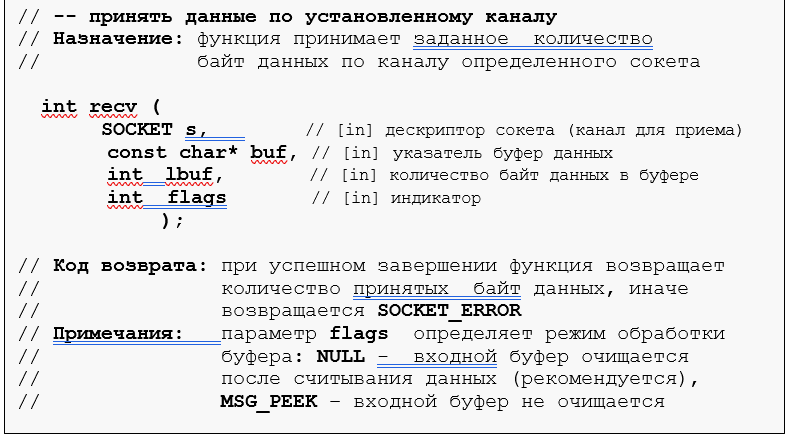


В третьем блоке программы сервера выполняются две функции Winsock2: listen и accept. Функция listen переводит сокет, ориентированный на поток, в состояния прослушивания (открывает доступ к сокету) и задает некоторые параметры очереди соединений. Функция accept переводит процесс сервера в состояние ожидания,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

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

Принципиальное различие этих схем, заключается в следующем.

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

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

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

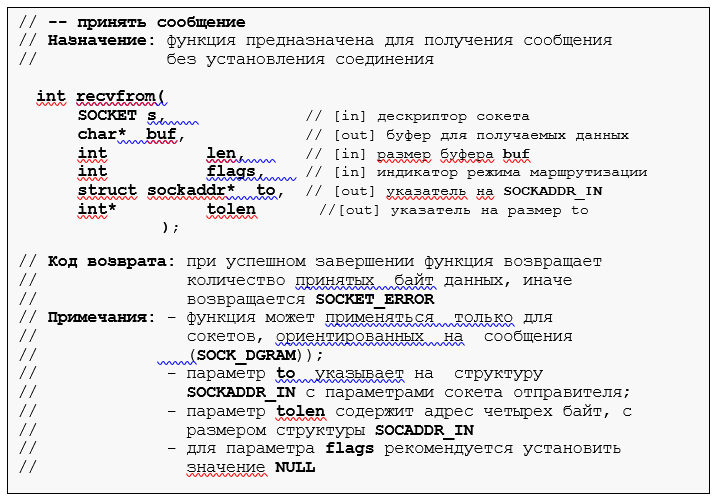


На рисунке схематично изображены две программы, реализующие два процесса распределенного приложения. Рассматриваемое приложение имеет архитектуру клиент-сервер (на рисунке сделаны соответствующие обозначения). Обе программы разбиты на пять блоков, а стрелками обозначается движение информации по сети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

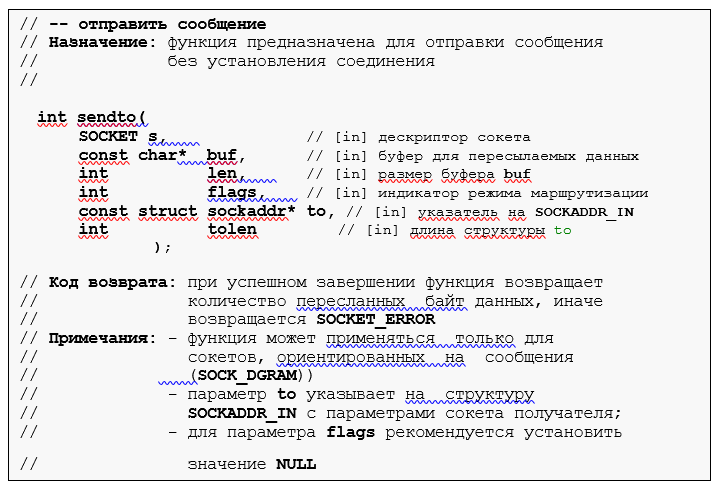
Первые блоки обеих программ одинаковые и предназначены для инициализации библиотеки WS2\_32.DLL.

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

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



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



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

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

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

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

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

Работа итеративного сервера описывается циклом из четырех шагов:

1) ожидание запроса от клиента;

2) обработка запроса;

3) отправка результата запроса;

4) возврат в ждущее состояние 1.

Очевидно, что сервер этого класса может применяться в том случае, если предполагаются короткие запросы от клиентов, не требующие больших затрат на обработку и длинных ответов сервера. Как правило, подобные серверы работают над UDP, когда нет необходимости создавать отдельный канал связи для каждого клиента. Консоль управления в этом случае может быть выполнена в виде специального клиента, запросы которого и есть команды управления сервером.

Параллельные серверы имеют другой цикл работы:

1) ожидание запроса от клиента;

2) запуск нового сервера для обработки текущего запроса;

3) возврат в ждущее состояние 1.

Преимущество параллельных серверов заключается в том, что он лишь порождает новые серверы, которые и занимаются обработкой запросов клиентов. Очевидно, что для создания параллельных серверов необходимым условием является мультизадачность операционной среды сервера. По всей видимости, параллельные серверы целесообразно использовать, если предполагается наличие относительно длительного сеанса связи между клиентом и сервером. Как правило, параллельные серверы работают над TCP. Консоль управления может быть создана, как отдельный процесс или поток (в зависимости от возможностей операционной системы) в рамках сервера или так же, как предлагалось для итеративного сервера, выполнить в виде специализированного клиента.

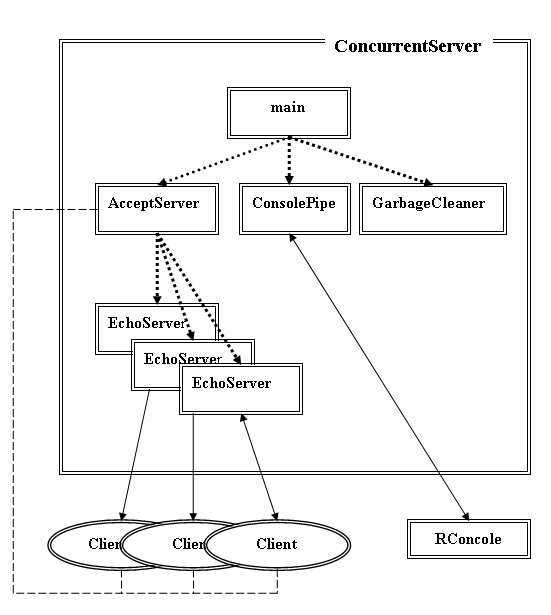
Сложности в разработке: Критическим по времени для параллельного сервера является момент подключения клиента. Сервер не должен тратить много времени на подключение клиента

С другой стороны, может потребоваться управлять процессом подключения.

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

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

Структура параллельного сервера, зависит, в конечном счете, от характера решаемой сервером задачи, но все же существуют общие структурные свойства сервера, на которых следует остановиться. Для того, чтобы упростить изложение будем рассматривать далее конкретную реализацию (модель) параллельного сервера с именем **ConcurrentServer** структура которого, по мнению автора, отражает все требующие внимания моменты.



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.

Процесс c именем main, является главным процессом сервера, который получает управление от операционной системы. Этот процесс создает и запускает новые процессы AcceptServer, ConsolePipe и GarbageCleaner. Процесс AcceptServer, в свою очередь создает несколько процессов с именем EchoServer. Сплошными двунаправленными линиями обозначается перемещение данных. Данные перемещаются между EchoServer-процессами сервера и клиентскими программами (с именем Client), обозначенными на рисунке овалами, а также между программой с именем RConsole, реализующей клиентскую часть консоли управления, и процессом ConsolePipe, который реализует серверную часть консоли управления. Штриховой линией, соединяющей изображение клиентских программ и изображение процесса AcceptServer обозначается процедура создания соединения между клиентом и сервером.

Опишем назначение компонентов изображенного на рисунке распределенного приложения.

**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 RConsole.

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

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

**Процесс GarbageCleaner.** Основным назначением процесса GarbageCleaner является удаление элемента списка подключений ListContact, после отключения программы клиента. Следует сразу отметить, что ListContact является ресурсом, требующим последовательного 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.

# Широковещание. Обнаружение сервера с помощью широковещания.

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

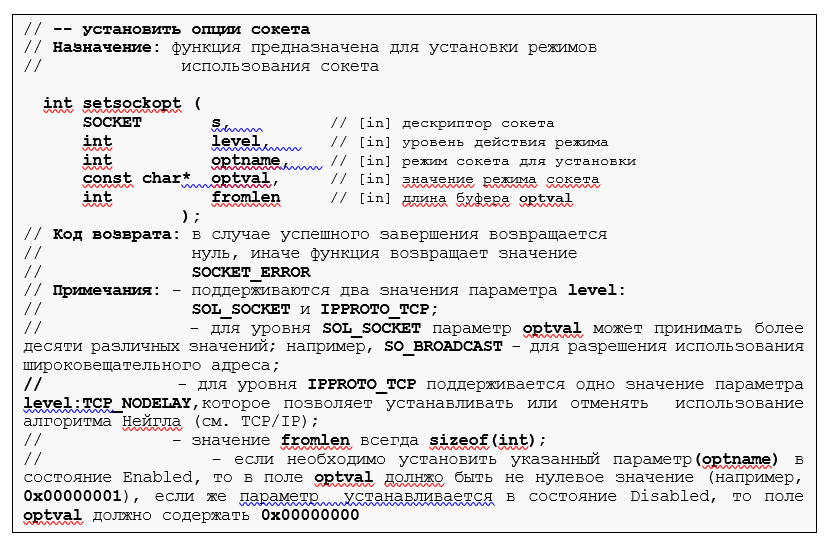
Для обеспечения независимости приложения от параметров сокета сервера (сетевой адрес и номера порта), как правило, номер порта делают одним из параметров инициализации сервера и хранят в специальных конфигурационных файлах, которые считывается сервером при загрузке (реже номер порта передается в виде параметра в командной строке). Так, например, большинство серверов баз данных в качестве одного из параметров инициализации используют номер порта, а при конфигурации (или инсталляции) клиентских приложений указывается сетевой адрес и порт сервера.

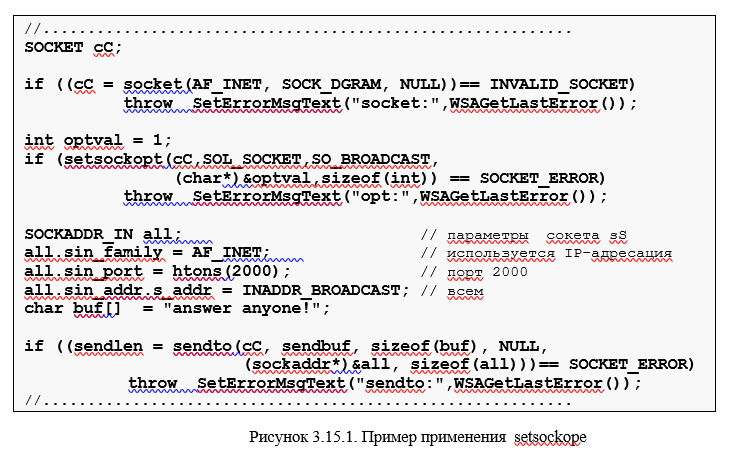
В некоторых случаях удобно возложить поиск сетевого адреса сервера на само клиентское приложение (при условии, что номер порта сервера известен). В этих случаях используются широковещательные сетевые адреса, позволяющие адресовать сообщение о поиске сервера всем компьютерам сети.

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.

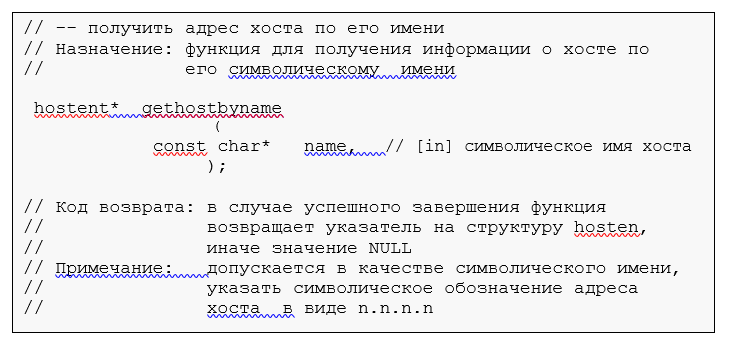
Стандартный широковещательный адрес в формате TCP/IP задается с помощью константы INADDR\_BROADCAST, которая определена в Winsock2.h. По умолчанию использование стандартного широковещательного адреса не допускается и для его применения необходимо установить специальный режим использования сокета SO\_BROADCAST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





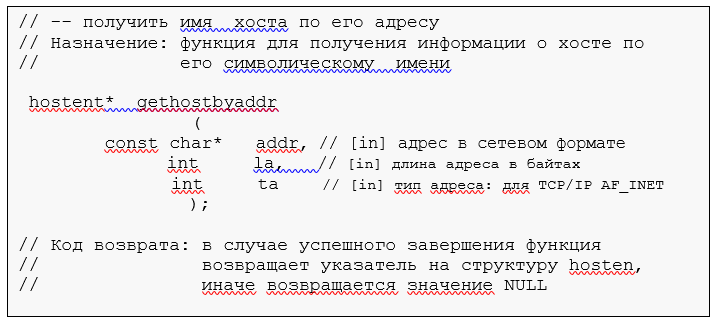
# Применение символического адреса хоста.

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.



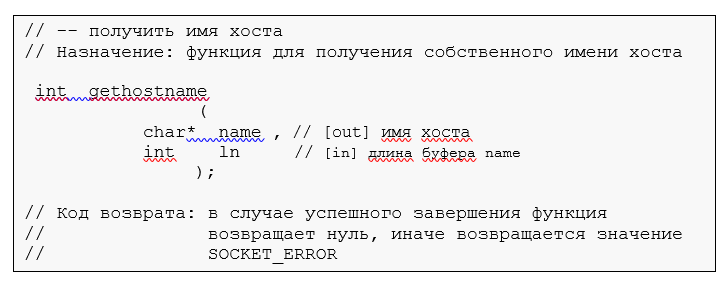
Такое решение достаточно часто применяется разработчиками распределенных систем. Связав набор программ-серверов с определенными стандартными именами компьютеров, распределенное приложение становится не зависимым от адресации в сети. Естественно при этом необходимо позаботиться, чтобы существовала служба,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

Помимо функции gethostbyname в составе 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.

Утилиты представляют собой внешние команды операционной системы и предназначаются для диагностики сети. Сетевые утилиты, поддерживаемые операционной системой Windows, перечислены в таблице ниже. При этом утилиты, наименования которых выделено жирным шрифтом считаются стандартными для протокола TCP/P и присутствуют в большинстве операционных систем.

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

Для получения справки о параметрах утилиты 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.

**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.

**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

**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).

**DNS работает на прикладному уровне модели ISO/OSI.**

**Службу DNS (Domain Name System**) можно рассматривать, как распределенную иерархическую базу данных, основное назначение которой отвечать на два вида запросов: выдавать IP-адрес по символическому имени хоста и наоборот. База данных имеет древовидную структуру, в корне которой ничего нет, а сразу под корнем находятся первичные сегменты (домены): .com, .edu, .gov, …, .ru, .by, … Наименование этих первичных доменов отражает деление базы данных DNS по отраслевому и национальному признакам. Домен в терминологии DNS называется любое поддерево дерева базы данных DNS.

**Служба DNS состоит из 3 основных компонент:**

1) Пространство имен DNS и соответствующие ресурсные записи (RR, resource record) – это сама распределенная база данных DNS;

2) Серверы имен DNS – компьютеры, хранящие базу данных DNS и отвечающие на запросы DNS-клиентов;

3) DNS-клиенты – компьютеры, посылающие запросы серверам DNS для получения ресурсных записей.

Информация о доменах, хранящаяся в базе данных сервера DNS, организуется в особые единицы, называемые **зонами.**

**Типы зон:**

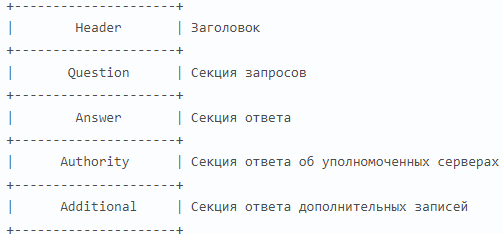
1) Стандартная основная (Standard primary) – главная копия стандартной зоны. Только в данном экземпляре зоны допускается производить какие-либо изменения, которые затем реплицируются на серверы, хранящие дополнительные зоны;

2) 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;

3) Интегрированная в Active Directory (Active Directory-integrated) – вся информация о зоне хранится в виде одной записи в базе данных Active Directory;

4) Зона-заглушка (stub, только в Windows 2003) – особый тип зоны, которая для данной части пространства имен DNS содержит самый минимальный набор ресурсных записей.

**Структура DNS:**

****

**Типы DNS-записей (RR):**

1) A – связывает доменное имя и IP-адрес;

2) MX – управляет маршрутами почтовых сообщений для протокола SMTP;

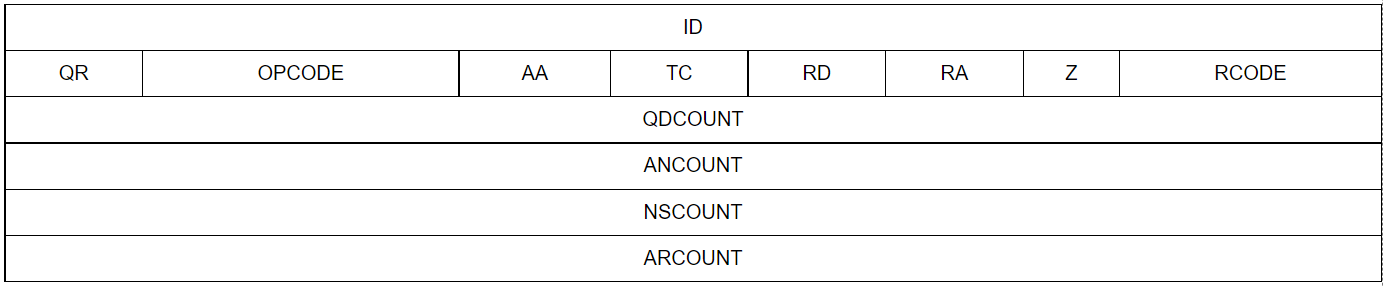
3) NS – указывает на серверы DNS, ответственные за конкретный домен и его поддомены;

4) SOA – используется для указания основного сервера для данной зоны и описания свойств зоны;

5) PTR – используется для обратного разрешения IP-адресов в имена узлов в домене in-addr.arpa;

6) SRV – используется для поиска серверов, на которых функционируют определенные службы;

7) CNAME – отображает одно имя на другое.

****

**ID –** идентификатор. 2 байта.

**QR** – идентификатор, обозначает, данное сообщение запрос от клиента или ответ от сервера.

**OPCODE** – клиент указывает тип запроса к серверу DNS.

**AA** – имеет смысл только в пакетах ответа от сервера клиенту. Если установлен данный флаг, значит сервер сгенерировал ответ.

**TC** – данный флаг устанавливается в пакете ответа в том случае, если сервер не смог поместить всю необходимую информацию в пакет из-за существующих ограничений.

**RD** – заголовок DNS. Клиент просит сервер не сообщать ему промежуточный ответ, а сообщить клиенту только конечный результат.

**RA** – данный флаг устанавливается в ответах. Служит для ответа клиенту поддерживает ли сервер рекурсивное обслуживание.

**RCODE** – поле кода ответа, либо с ошибкой, либо успешно. 0 – без ошибок, 1 – ошибка разрешения имен, и эта ошибка связана с DNS сервером который не смог понять формат запроса клиента, 2 – означает что сервер сигнализирует о неисправностях сервера, 3 – имени не существует в данном домене, 4 – сервер не может выполнить запрос данного типа.

**QDCOUNT** – количество запросов.

**ANCOUNT** – количество ответов.

**NSCOUNT** – количество записей в Authority Section.

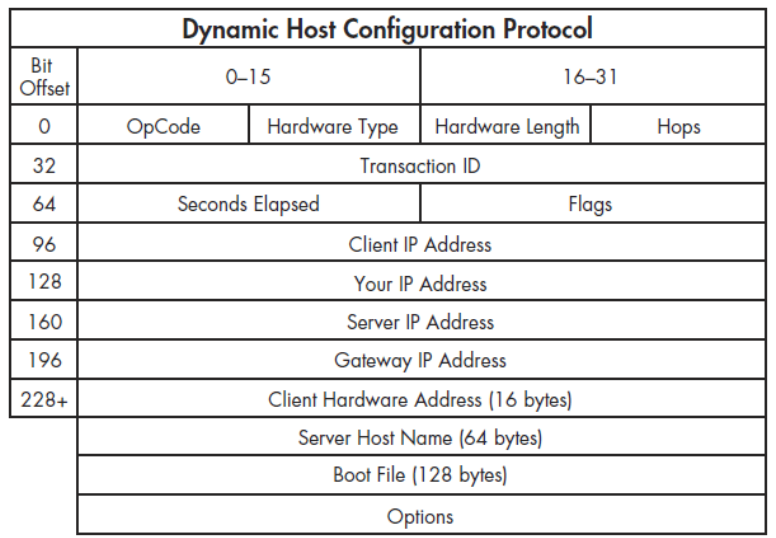
**ARCOUNT** – количество записей в Additional Record Section.

# 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.

**Служба DHCP состоит из 3 модулей:** сервера DHCP, клиента DHCP и ретранслятора DHCP (используется в том случае, если на первоначальном этапе подключения к сети широковещательные запросы DHCP-клиента не могут быть доставлены (по разным причинам) DHCP-серверу. Играет роль посредника между ними).

Для обнаружения DHCP-сервера DHCP-клиент выдает в сеть широковещательный запрос. Если в этом домене есть DHCP-сервер, то он окликается, посылая клиенту специальное сообщение, содержащее IP-адрес DHCP-сервера. Если доступны несколько DHCP-серверов, то, как правило, выбирается первый ответивший. Получив адрес сервера, клиент формирует запрос на выделение IP-адреса из пула адресов DHCP-сервера. В ответ на запрос, DHCP-сервер выделяет адрес клиенту на определенный период времени. После получения IP-адреса TCP/IP-стек клиента начинает его использовать. Продолжительность аренды адреса устанавливается специально или по умолчанию (может колебаться от нескольких часов до нескольких недель). После истечения срока аренды DHCP-клиент пытается снова договорится с DHCP-сервером о продлении срока аренды или о выделении нового IP-адреса.

****

**OpCode –** тип сообщения.

**Hops –** количество промежуточных маршрутизаторов.

**Second Elapsed –** время в секундах с момента начала процесса получения адреса.

**CSMA/CD (Carrier Sense Multiple Access with Collision Detection) –** множественный доступ с прослушиванием несущей и обнаружением коллизий.

Если во время передачи кадра рабочая станция обнаруживает другой сигнал, занимающий передающую среду, то она останавливает передачу, посылает сигнал преднамеренной помехи и ждет в течении случайного промежутка времени, перед тем как снова отправить кадр.

# Стандарты сообщений Internet.

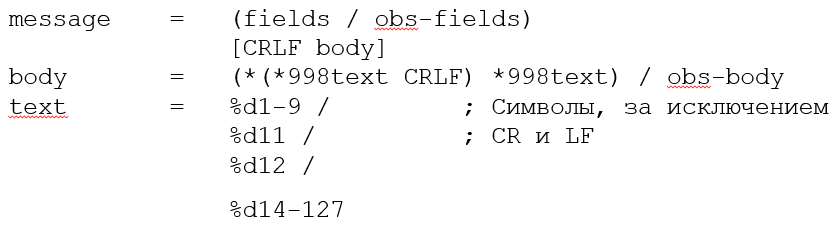
Сообщения, соответствующие спецификации RFC-5322, включают символы с десятичными кодами от 1 до 127, интерпретируемые в соответствии с кодировкой US-ASCII. Данная спецификация вносит два ограничения на число символов в строке. Строка должна содержать не более 998 символов; следует использовать строки размером не более 78, без учета CRLF. 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. Для удобства и с учетом ограничения размеров строки (998/78 символов), значение поля может быть разбито на несколько строк; это называется «**фальцовкой**» (folding).

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).

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.

Поле addr-spec представляет собой специфический для Internet идентификатор, содержащий локально интерпретируемую строку, за которой следует символ @ (код ASCII - 64) и доменное имя Internet. Эта локально интерпретируемая строка представляет собой строку в кавычках или атом с точкой.

Общий синтаксис сообщений:



# Протокол HTTP.

**Протокол HTTP** (Hypertext Transfer Protocol) – это протокол Прикладного уровня, доставляющий информацию между различными гипермедийными системами. Под понятием гипермедийной системы понимается компьютерное представление системы данных, элементы которой представляются в различных форматах (гипертекст, графические изображения, видеоизображения, звук и т.д.) и обеспечивается автоматическая поддержка смысловых связей между представлениями элементов.

По умолчанию HTTP использует порт 80 и предназначен для построения систем архитектуры клиент-сервер. 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).

**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.

**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.

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.

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**

**Метод HTTP** (англ. HTTP Method) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. чувствительно к регистру.

Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented). Если серверу метод известен, но он не применим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed).

**Типы методов:**

**OPTIONS**: Метод OPTIONS используется для запроса информации о возможностях и настройках сервера или о доступных методах запросов для конкретного ресурса. Он позволяет клиенту определить, какие методы запросов поддерживаются сервером и какие дополнительные операции можно выполнять над ресурсом. Сервер должен вернуть ответ со списком поддерживаемых методов и другой соответствующей информацией.

**GET**: Метод GET используется для получения ресурса с сервера. Клиент отправляет GET-запрос на указанный URL-адрес, и сервер возвращает содержимое запрошенного ресурса. GET-запросы не должны изменять состояние сервера и могут быть кэшированы клиентом для повторного использования. Примером GET-запроса является получение веб-страницы в браузере.

**HEAD**: Метод HEAD аналогичен методу GET, но сервер возвращает только заголовки ответа без фактического содержимого ресурса. HEAD-запросы полезны, когда клиенту требуется только получить информацию о заголовках, например, для проверки доступности или проверки обновлений ресурса.

**POST**: Метод POST используется для отправки данных на сервер для создания нового ресурса или выполнения какой-либо операции на сервере. Клиент отправляет POST-запрос с данными в теле запроса, и сервер обрабатывает эти данные соответствующим образом. POST-запросы могут изменять состояние сервера или создавать новые ресурсы. Примером POST-запроса является отправка данных формы на сервер.

**PUT**: Метод PUT используется для загрузки или замены ресурса на сервере. Клиент отправляет PUT-запрос с данными ресурса в теле запроса на указанный URL-адрес, и сервер сохраняет или заменяет ресурс по указанному адресу. PUT-запросы полезны, когда клиент хочет обновить существующий ресурс или создать новый ресурс с определенным URL-адресом.

DELETE – удаляет указанный ресурс;

CONNECT – устанавливает “туннель” к серверу, определенному по ресурсу;

TRACE – выполняет вызов возвращаемого тестового сообщения с ресурса;

PATCH – используется для частичного изменения ресурса.

**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

**Коды состояния:**

**1)** 1xx – коды, информирующие о процессе передачи;

**2)** 2xx – сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента (200 – успешно);

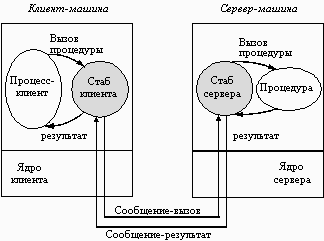
**3)** 3xx – сообщают клиенту, что для успешного выполнения операции необходимо сделать другой запрос;

**4)** 4xx – ошибки со стороны клиента (400 – плохой запрос, 401 – не авторизирован, 404 – не найден);

**5)** 5xx – ошибки со стороны сервера, должен включать в тело сообщение, которое клиент отобразит пользователю (500 – внутренняя ошибка сервера, 501 – не выполнено, 502 – плохой шлюз, 503 – недоступен, 505 – HTTP-версия не поддерживается).

# Служба RPC.

**RPC** – удаленный вывод процедур. 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.

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.

Проблемы:

• Клиент и сервер могут применять разные кодировки.

• Клиент и сервере могут иметь разные представления данных.

• В каждой из частей необходимо договориться о внутреннем представлении данных.

• 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.

Протокол XML-RPC был изначально разработан Дэйвом Винером из компании «UserLand Software» в сотрудничестве с Майкрософт в 1995 году. Однако 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**

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.

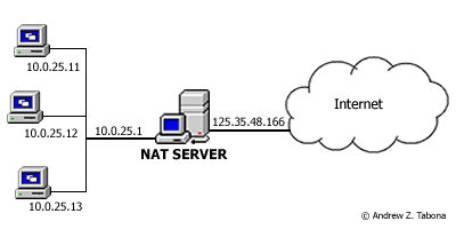
Клиентский стаб при посылке сообщения с запросом использует в качестве адреса описатель. В сообщении содержатся параметры и уникальный идентификатор, который ядро сервера использует для того, чтобы направить поступившее сообщение в нужный сервер в случае, если их несколько на этой машине.

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 (Network Address Translation) –** это механизм в сетях TCP/IP, позволяющий преобразовывать IP-адреса транзитных пакетов.



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 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. 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.

**ProxyServer**

Это служба в компьютерных сетях, позволяющая клиентам выполнять косвенные запросы к другим сетевым служба. Могут использоваться для обеспечения доступа компьютерам локальной сети в интернет. Кэширование данных. Для сжатия данных, что обеспечивают быструю скорость передачи. Используется для защиты локальной сети от внешнего доступа. На Proxy можно наложить некоторые правила на доступ из локальной сети во внешнюю. Анонимизация доступа к программам и службам.

**РЕМЭЙЛЕРЫ**

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 псевдо анонимные. Псевдо анонимные – сервер знает адрес электронной почты, который необходим для получения ответа на письмо. Анонимные ремэйлеры полностью уничтожают адрес отправителя. При этом обеспечивается очень высокая безопасность, и нет гарантии своевременной доставки.

Стандарт MIXMINIOM анонимной перессылки почты, по этому стандарту можно отправлять электронную почту.

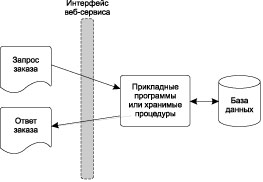
# Web-сервисы: SOAP, XML, WSDL, UDDI.

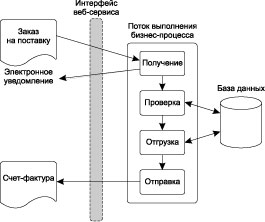
Веб-сервисы преобразуют XML-документы (Extensible Markup Language, XML) в ИТ-системах. Веб-сервисы - это XML-приложения, осуществляющие связывание данных с программами, объектами, базами данных либо с деловыми операциями целиком. Между веб-сервисом и программой осуществляется обмен XML-документами, оформленными в виде сообщений. Стандарты веб-сервисов определяют формат таких сообщений, интерфейс, которому передается сообщение, правила привязки содержания сообщения к реализующему сервис приложению и обратно, а также механизмы публикации и поиска интерфейсов.

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

• удаленный вызов процедуры (онлайновая);

• документно-ориентированный (пакетная).





**SOAP** - это XML-способ определения: 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** - это XML-формат, описывающий состав веб-сервиса. WSDL предназначен для использования как в процедурно-ориентированных, так и в документно-ориентированных приложениях. Так же как и другие XML-технологии, WSDL является расширяемым языком и имеет такое количество параметров, что обеспечение совместимости при организации взаимодействия между различными реализациями может вызвать сложности. Полное взаимопонимание возможно лишь в том случае, если отправитель и получатель сообщения могут совместно использовать и одинаково интерпретировать один и тот же WSDL-файл.

WSDL в соответствии с уровнем абстрагирования состоит из трех элементов. WSDL можно разделить на три основные составляющие:

• определение типов данных;

• абстрактные операции;

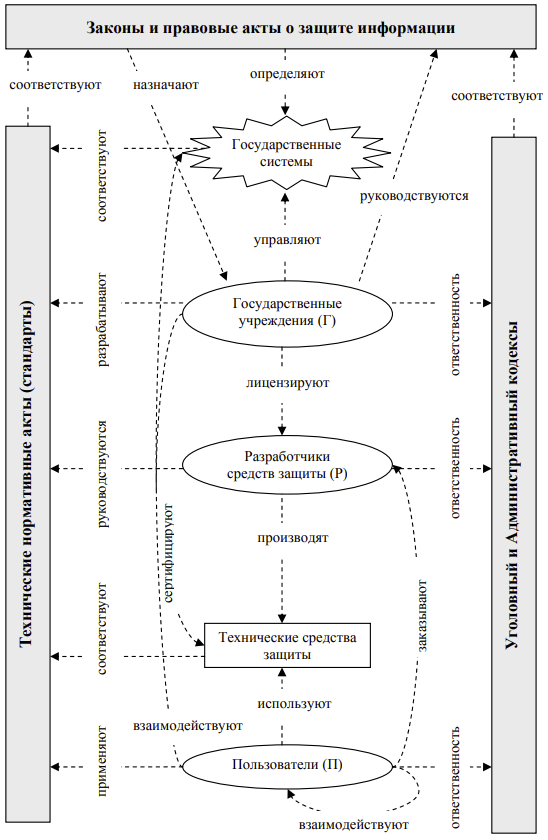
• связывание сервисов.

После того как вы определили данные в сообщении (XML), описали сервисы, которые будут принимать и обрабатывать сообщения (WSDL), и предъявили требования к средству их передачи и приема (SOAP), вам необходимы способы публикации предлагаемой вами службы, а также инструменты поиска нужных вам, но предоставляемых другими участниками услуг. Именно эти функции и выполняет реестр UDDI (universal distribution, discovery, and integration).

**UDDI** регистрирует и публикует определения веб-сервисов. Структура UDDI определяет модель данных в программных интерфейсах (API) XML и SOAP для регистрации и обнаружения коммерческой информации, включая веб-сервисы.

# Национальная инфраструктура информационной безопасности.

Инфраструктура национальной информационной безопасности (ИНИБ) представляет собой комплекс взаимосвязанных обслуживающих систем различной природы (организационных, правовых, информационных и т.п.) обеспечивающих основу для решения задачи информационной безопасности (ИБ) в масштабах страны. ИНИБ является составной частью общей системы национальной безопасности страны. На рис. 1 представлена обобщенная схема ИНИБ, исторически сложившаяся в большинстве развитых стран. На рисунке изображены основные элементы ИНИБ (замкнутые фигуры, различной геометрии) и подписанные стрелки, указывающие тип взаимоотношения между парами элементов.



Основой любой ИНИБ является законодательная и нормативно-правовая база обеспечения информационной безопасности (ЗБИБ), являющаяся выражением государственной политики в этой сфере. Государство исходит из того, что информационные ресурсы являются объектами собственности, участвующими в хозяйственном обороте. Законы и правовые акты ЗБИБ наделяют определенные органы государственной власти полномочиями осуществлять государственное регулирование, контроль (мониторинг) и управление в сфере информационной защиты, а также определяют взаимоотношение субъектов информационной деятельности. Технические нормативные акты представляют собой стандарты и технические регламенты, которые могут (или обязаны) использовать субъекты информационной деятельности. Ответственность субъектов информационной деятельности за нарушение законов в сфере информационной безопасности определяется Уголовным и Административным кодексами.

Условно субъекты в сфере информационной безопасности можно разбить на три группы: государственные учреждения, осуществляющие государственное регулирование в этой сфере (группа Г); разработчики технических средств защиты информации (группа Р) и пользователи (группа П). Очевидно, что все эти группы пересекаются.

Полномочия и деятельность субъектов группы Г, определена законом или специальными положениями. Группа Г координирует и лицензирует всю деятельность по технической защите информации, разрабатывает (или участвует в подготовке) нормативных правовых актов, осуществляет экспертизу (сертификацию) технических средств информационной безопасности. Кроме того, субъекты этой группы осуществляют управление государственными информационными системами, связанными с информационной безопасностью (например, удостоверяющие и регистрационные центры).

Основной деятельностью субъекты группы P является разработка технических средств защиты информации. Свою деятельность они осуществляют на основе лицензии, а все созданные ими средства защиты должны пройти сертификацию прежде, чем они будут применяться пользователями. Сертификация – это процедура подтверждения соответствия продукции определенным техническим нормативным документам (стандартам).

Пользователи информационных систем (группа П) представляют самую многочисленную группу субъектов в сфере информационной безопасности. Субъекты этой группы применяют средства технической защиты информации для защиты собственных информационных ресурсов и взаимодействуют с системами защиты других субъектов. В своей деятельности они могут использовать государственные автоматизированные системы, обеспечивающие сервисные услуги в области технической защиты информации.

Исторически первой ИНИБ сложилась в США. Первый закон о защите информации датируется 1906 годом, а на сегодняшний день ЗБИБ США насчитывает их более 500 [5, 6].

Появление глобальных информационных систем привело к необходимости выработки единого подхода к проблемам информационной безопасности на международном уровне.

Первый важный шаг, который сделан в этом направлении, является разработка международных стандартов по информационной безопасности.

По понятным причинам основой международных стандартов по информационной безопасности стали стандарты, разработанные стандартизирующими организациями ведущих индустриальных стран (прежде всего США и Великобритании). Нормативная документация национальной ЗБИБ, как правило, являются гармонизированными международными стандартами.

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

**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.

Безопасность охватывает все уровни протоколов:

1. На физическом уровне можно поместить сетевой кабель в специальные герметические трубы

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

6. На представительском – методы шифрования

7. На прикладном - процессы аутентификации

**Обеспечение целостности** (Целостность информации (также целостность данных) — термин в информатике и теории телекоммуникаций, который означает, что данные полны, условие того, что данные не были изменены при выполнении любой операции над ними, будь то передача, хранение или представление.) данных с помощью хэш-кодов они используются для цифровых подписей.

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

**Шифр** - какая-либо система преобразования текста (код) для обеспечения секретности передаваемой информации.

**Код** — совокупность алгоритмов криптографических преобразований (шифрования), отображающих множество возможных открытых данных на множество возможных зашифрованных данных, и обратных им преобразований.

**Ключ** — 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

• 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.

**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.

**DNSSEC** (англ. Domain Name System Security Extensions) — 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.