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

**Шиман обращал внимание:** понятие распределенного приложения(в частности архитектура клиент\сервер);спецификации NDIS, POSIX и для чего нужны; ISO/OSI и TCP/IP уровни, для чего каждый предназначен, что на нём происходит и какие протоколы попадаются на уровне, где проходит граница программной и аппаратной, границы уровней.

**Протокол** - набор правил и соглашений, описывающих процедуры взаимодействия каждого уровня модели с соседними уровнями.

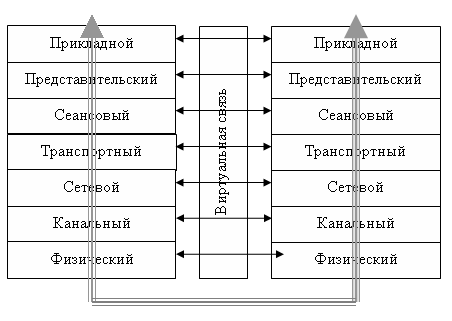
**Сетевой протокол** — набор правил, позволяющий осуществлять соединение и обмен данными между несколькими устройствами.

**Стандарт POXIS**(Portable Operating System Interface for Unix) - набор документов, описывающих интерфейсы между прикладной программой и ОС

Уровни модели OSI/ISO:

**Прикладной** – определение способа взаимодействия пользователей с системой (определение интерфейса). (HTTP, Telnet, FTP, SMTP)

По мере своего движения от отправителя к получателю на каждом уровне данные подвергаются необходимому преобразованию: 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.

Изображение выглядит как текст, диаграмма, снимок экрана, Параллельный

Контент, сгенерированный ИИ, может содержать ошибки.

На рис. 1.2.2 изображены два процесса (с именами C и S), функционирующие на разных компьютерах в среде соответствующих ОС. В составе ОС имеются службы (специальные программы) обеспечивающие поддержку протоколов канального, сетевого и транспортного уровней. Протоколы физического уровня, как правило, обеспечиваются сетевыми адаптерами. На рисунке граница ОС условно проходит по канальному уровню. Действительно, часто часть процедур канального уровня (обычно подуровня управления доступом к среде) обеспечивается аппаратно (сетевым адаптером), а другая часть процедур (обычно подуровня управления логическим каналом) реализована в виде драйвера, инсталлированного в состав ОС. Процессы, взаимодействуют со службами, обеспечивающими процедуры протоколов транспортного уровня с помощью набора специальных функций API (Application Program Interface), входящими в состав ОС. Функции сеансового, представительского и прикладного уровней обеспечивается самим распределенных приложением.

Для предоставления возможности разработчикам ПО использовать процедуры стека TCP/IP в состав ОС включаются специальные интерфейсы (***Application Programming Interface, API***), - набор специальных функций и технологических инструкций, обеспечивающих доступ к модулям протокола TCP/IP.

Наиболее распространенными API для обмена данными в сети, являются ***интерфейс сокетов*** и ***RPC*** ***(Remote Procedure Call)*** – вызов удаленных процедур.

**API сокетов** – название программного интерфейса, предназначенного для обмена данными между процессами, находящимися на одном или на разных объединенных сетью компьютерах. Описан в стандарте ***POSIX***.

**Представительский** – представление данных в едином формате и виде; процедуры этого уровня описывают способы кодирования, шифрования, сжатия данных.

**Сеансовый** – установка, поддержание и разрыв сеанса. Это точка взаимодействия программ и комп. сети (TCP, UPD)

**Транспортный** – подготовка и доставка пакетов данных между конечными точками без ошибок и в правильной последовательности, контроль передачи данных, разбиение данных на блоки, индексация блоков данных.

**Сетевой** – методы адресации и маршрутизации. Определяет единый метод адресации для всех компов в сети не зависимого от способа передачи данных. Результат – пакет, который обрабатывается транспортным уровнем. (IP, ICMP, ARP)

1. Указать где в OSI/ISO проходит ***граница между аппаратным и программным обеспечением***.

Канальный уровень

1. Определить понятие ***CSMA/CD***.

Множественный доступ с прослушиванием несущей и обнаружением коллизий

1. Как называется ***программное обеспечение реализующий подуровень LLC канального уровня***.

Драйвер

**Канальный** – 2 подуровня: управления доступом к среде передачи данных (определяет методы совместного использования сетевыми адаптерами среды передачи данных) и управления логическим каналом (канал между 2 сетевыми адаптерами, способы обнаружения и исправления ошибок передачи данных). Назначение: подготовить блок данных (кадр) для следующего сетевого уровня.

* Канальный уровень. Для обеспечения корректности передачи помещает в начало и конец каждого кадра специальную последовательность бит и высчитывает контрольную сумму. Определяет правила использования физического канала. При большом размере блока данных делит его на кадры меньшего размера. Функции этого уровня реализуются сетевыми адаптерами и их драйверами.

Делится на два подуровня: LLC (logical link control)-подуровень - логический контроль связи

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

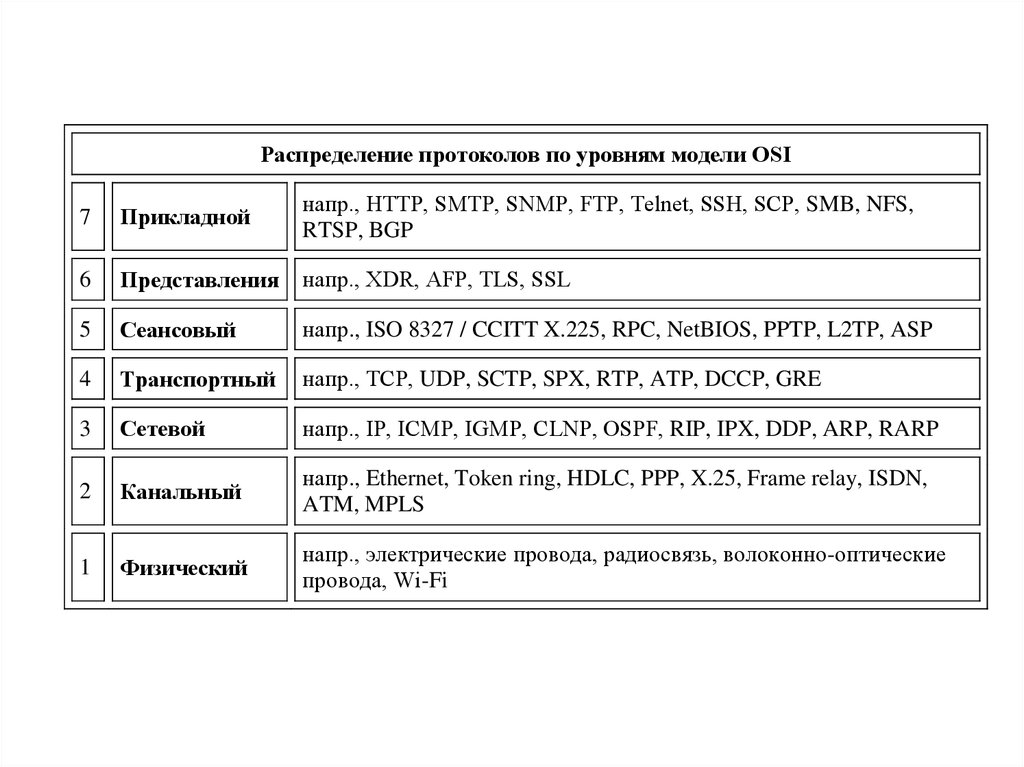
LLC находится выше MAC.

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 сможет расширяться, т.к. имеет ограниченные возможности адресации и не имеет функций маршрутизации.

**Физический** – определяет свойства среды передачи данных (коаксиальный кабель, витая пара) и способы ее соединения с сетевыми адаптерами. (Ethernet)

**Структура TCP/IP**

1. Уровень доступа к сети (Network Access Layer): - (определяет физ способы передачи данных по сети) для создания локальных сетей и соединение с глобальными сетями. Примеры: LAN, WAN, Ethernet, SLIP, PPP, Wi-Fi, Token Ring)
2. Межсетевой уровень (Internet Layer): маршрутизация пакетов между сетями. Примеры протоколов: IP, ICMP, ARP, RARP
3. Транспортный уровень (Transport Layer): передача данных по сети между процессами. Примеры протоколов: TCP, UDP.
4. Прикладной уровень (Application Layer): взаимодействие между приложениями и пользователями. Примеры протоколов: HTTP, FTP, SMTP, DNS, DHCP, POP3



**Архитектура клиент-сервер**

Если все процессы приложения можно условно разбить на две группы: клиент и сервер. Обмен данными осуществляется только между клиентом и сервером, инициатором обмена является клиент.

По методу обслуживания серверы подразделяются на итеративные (обслуживают запросы клиентов поочередно) и параллельные (обслуживают несколько клиентов одновременно)..

**Основные технологии создания распределенных программных систем:**

* Сетевые протоколы TCP/IP, HTTP, REST, SOAP, MQTT… используются для обмена данными между распределенными компонентами системы.
* RPC (Remote Procedure Call): Позволяет вызывать удаленные процедуры или функции на удаленных компьютерах через сеть. gRPC, XML-RPC, JSON-RPC.
* Межпроцессное взаимодействие (IPC): Позволяет взаимодействовать между процессами или потоками на разных узлах системы. Сокеты, разделяемая память, очередь сообщений и т.д.
* Сервисно-ориентированная архитектура (SOA): Подход к разработке, основанный на создании независимых сервисов, которые взаимодействуют через стандартизированные протоколы. 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 - стандартный интерфейс, определяющий способ взаимодействия между сетевыми адаптерами и драйверами сетевых адаптеров в ОC. Определяет API и протоколы, которые драйверы сетевых адаптеров должны реализовывать для обеспечения сетевой функциональности.

**NDIS (Network Driver Interface Specification) –** драйвер для сопряжения драйверов сетевых адаптеров с операционной системой.

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

**Сетевая служба** – программная реализация протоколов сетевого уровня TCP/IP. Примеры: DHCP, DNS, NBT, Telnet, FTP, WWW (протокол HTTP) и служба электронной почты (протоколы SMTP, POP3, IMAP4).

**DNS(Domain Name System)**: преобразует символические доменные имена в IP-адреса

**DHCP** (Dynamic Host Configuration Protocol): позволяет автоматически получать IP-адрес (маска и шлюз)

1. Организация, ведающая распределением ***IP-адресов***, поддержкой ***сетевых доменов Internet верхнего уровня***, ***регистрацией портов***.

**ICANN** (Internet Corporation for Assigned Names and Numbers).

1. Какая ***организация поддерживает сетевые*** протоколы Internet. Как называются ***документы, описывающие эти протоколы***.

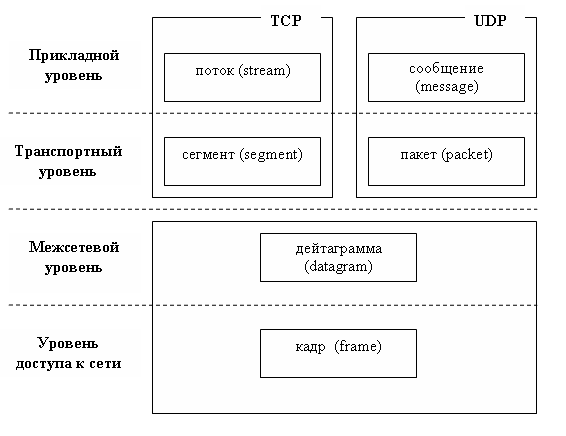
IETF (Internet Engineering Task Force). RFC (Request for Comments).

Хост - все устройства, имеющие IP-адрес

Адрес хоста – ip. Имя хоста – DNS имя.

1. Поясните понятие ***маска подсети***.

**Маска подсети -**  32-битовая маска для определения по IP-адресу адреса подсети и адреса узла (хоста, компьютера, устройства) этой подсети.



**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.

**Протоколы уровня доступа к сети (канальный):**

1) LAN (Local-Area Network) – протокол для создания локальных сетей;

2) WAN (Wide-Area Network) – протокол для соединения с глобальными сетями;

3) Ethernet – протокол для передачи данных по локальной сети. Применяет метод доступа CSMA/CD, использует 48-битную адресацию и обеспечивает передачу данных до 1 ГБ/с. Максимальная длина кадра составляет 1518 байт, при этом сами данные могут занимать от 46 до 1500 байт;

4) SLIP (Serial Line IP) – протокол для последовательного канала. Протокол использует простейшую инкапсуляцию кадра. Недостатки: 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.

5) PPP (Point-to-Point Protocol) – протокол двухточечного соединения (Двухточечные соединения обычно используются для соединения двух систем в глобальной сети). 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. Это наиболее широко используемый протокол для последовательного канала, практически вытеснил SLIP.

Одной из основных характеристик протоколов канального уровня является ***максимальная единица передачи данных*** ***MTU*** (Maximum Transmission Unit), которая определяет максимальную длину в байтах данных передаваемых в одном кадре. От значения MTU зависит скорость передачи по каналу. Если IP-модулю требуется отправить данные (дейтаграмму) имеющие дину большую MTU, то он производит фрагментацию разбивая дейтаграмму на части имеющие длину меньшую, чем MTU.

|  |  |
| --- | --- |
| **Сеть** | **MTU (байты )** |
| FDDI | 4464 |
| Ethernet | 1500 |
| IEEE802.3/802.2 | 1492 |
| X.25 | 576 |
| SLIP, PPP (с минимальной задержкой) | 256 |

**Протоколы межсетевого уровня:**

1. IP – доставка дейтаграмм. Не надежный и не поддерживающий соединение;

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

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

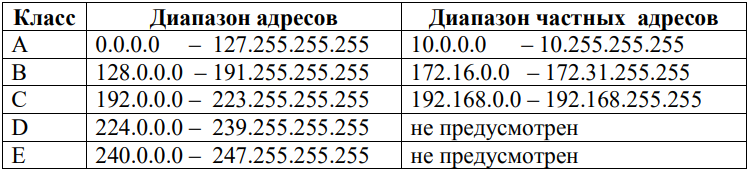
Если протоколы Уровня доступа к сети при передаче данных используют MAC-адреса, то на Межсетевом уровне применяется IP-адресация. Главной особенностью IP-адреса является его независимость от физической устройства, подключенного к сети.

IP-адрес – последовательность из 32 бит, разделенное на группы по 8 бит, называемых ***октетами***. 2 части: *номера подсети* (IDподсети) и *номера узла* (ID хоста) в этой подсети. При передаче пакета из одной подсети в другую используется ID подсети. Когда пакет попал в подсеть назначения, ID хоста указывает на конкретный узел в рамках этой подсети. Существует 2^32 уникальных ip адресов

**Тип: динамическая и статическая.**

**Бесклассовая** - вычисление адреса сети выполняется с помощью операции конъюнкции между IP-адресом и маской подсети.

**Классовая** длина маски строго фиксирована.



Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

1. Поясните понятие ***интерфейс внутренней петли***.

интерфейс, который позволяет клиенту и серверу, находящимся на одном хосте, обмениваться данными посредством протокола TCP/IP, не покидающими пределы этого хоста. 127.0.0.1

В качестве адреса хоста нулевая последовательность битов (этот адрес считался адресом самой сети) и последовательность битов, состоящую из одних единиц (этот адрес использовался для широковещательных сообщений в сети).

**Составная IP-сеть** - объединение нескольких IP-сетей с помощью специальных устройств (***шлюзы)***. Обычно шлюз - компьютер, на котором установлены несколько интерфейсов IP и ПО, реализующее протоколы Межсетевого уровня. Для обмена данными с хостом, который находится в другой сети, используется ***таблица маршрутов***. Таблица маршрутов имеется на каждом ***узле*** сети (хост или шлюз) и содержит информацию об адресах сетей, адресах шлюзов и т.п. Процесс определения адреса следующего узла в пути следования дейтаграммы и пересылка ее по этому адресу называется ***маршрутизацией***.

2) ICMP (Internet Control Message Protocol) – транспортировка информации о сетевой деятельности и маршрутизации. IPCM сообщения - отформатированные IP-дейтаграммы, которым соответствуют определенные типы (15 типов) и коды сообщений. С помощью этого протокола 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;

3) ARP (Address Resolution Protocol) – динамическая проекция IP-адресов в соответствующие MAC-адреса аппаратных средств. Каждый хост кэширует специальную ARP-таблицу. Время существования записи в таблице обычно составляет 10-20 минут с момента ее создания и может быть изменено с помощью параметров реестра;

4) RARP (Reverse ARP) – получение IP-адреса по MAC-адресу;

5) IPv6 (длина 128 бит) – новая версия протокола IPv4 призванная решить недостатки IPv4.

1. Основные ***отличия между IPv4 и IPv6***.

IPv4 более распространенный, но пространство адресов практически исчерпано, решение проблемы – IPv6. В IPv6 длина адреса расширена до 128 бит (против 32 в IPv4), что позволяет обеспечить больше уровней иерархии адресации, увеличить число адресуемых узлов, упростить автоконфигурацию(процесс, позволяющий хосту находить информацию для настройки собственных IP-параметров(DHCP)). В ipv6 реализована повышенная система безопасности и не использует классовую адресацию.

**Основная концепция IPv6**: каждый отдельный узел должен иметь собственный уникальный идентификатор интерфейса. Например, 48-битный МАС-адрес платы Ethernet изначально предназначен для глобальной идентификации. Первые 24 бита этого адреса обозначают производителя платы и индивидуальную партию изделия, а остальные 24 бита (уникальный идентификатор интерфейса) определяются производителем, с таким расчетом, чтобы каждый номер был уникален в пределах всей его продукции.

МАС-адрес (Media Access Control address) – это уникальный идентификатор, присваиваемый хосту.

MAC-адрес состоит из 48 бит, представленных в шестнадцатиричном формате.

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

1 бит – групповой 1, индивидуальный 0

1 бит обычно в 0. Установка бита U/L в 1 означает, что адрес задан не производителем сетевого адаптера, а организацией.

22 бита – уникальный идентификатор (код производителя)

24 бита – уникальный адрес (выбираются изготовителем для каждого устройства)

**Протоколы транспортного уровня:**

1) TCP;

2) UDP (User Datagram Protocol) – протокол передачи дейтаграмм пользователя.

**Сетевой порт** - идентификационный номер процесса, получающего или отправляющего данные с помощью транспортного уровня. (TCP/UDP)

Адресат в сети TCP/IP полностью определяется: IP-адресом, номером порта и типом протокола транспортного уровня (UDP или TCP). Основным отличием протоколов UDP и TCP является, то, что UDP – протокол без установления соединения (ориентированным на сообщения), а TCP – протокол на основе соединения (ориентированный на поток). Движение информации с верхних уровней на нижние сопровождается ***инкапсуляцией данных*** (упаковкой по принципу матрешки), а движение в обратном направлении – распаковкой.

**Протоколы прикладного уровня:**

1) HTTP (HyperText Transfer Protocol) – протокол для передачи произвольных данных. Использует порт 80 по умолчанию.

2) FTP (File Transfer Protocol) – протокол передачи файлов по сети;

3) SMTP (Simple Mail Transport Protocol) – протокол для электронной почты.

|  |  |
| --- | --- |
| Хорошо известные номера портов (системные службы) | 0 – 1023 |
| Зарегистрированные номера портов (промышленные приложения и процессы) | 1024 – 49151 |
| Динамические номера портов (процессам спец.службой ос) | 49152– 65535 |

DNS -53(TCP и UDP), Oracle SQL – 66(TCP), DHCP – 67-68(UDP), MSSQL – 1433(TCP), Oracle - 1522(TCP)

1. Поясните понятия ***публичный IP-адрес*** и ***частный IP-адрес***.

**Публичный -** присваивается провайдером при подключении к интернету(он уникален)

**Частный -** присваивается администратором локальной сети компьютерам, объединенным в одну сеть

Альтернативой классов модели является **бесклассовая междоменная маршрутизация – CIDR (Classes Inter-Domain Routing).** CIDR позволяет произвольным образом назначить границу адреса сети (биты установлены) и адреса хоста (биты сброшены) IP-адреса с помощью маски подсети. Вычисление адреса сети происходит с помощью конъюнкции между IP-адресом и маской подсети.

**Параметры настройки TCP/IP:**

1) Использовать фиксированный IP-адрес, при этом задать маску подсети и адрес основного шлюза или применить протокол DHCP;

2) Использовать фиксированный адрес DNS-сервера или получить автоматически.

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

**Шиман обращал внимание:** Главным параметром у протоколов на физическом уровне является МДУ?(максимальная длина кадров).Все параметры протоколов знать не обязательно, но только самого распространенного **протокола Ethernet 1500 байт** вот эту информацию в голове нужно держать, соответственно знать про IPv4,IPv6 структура пакета **структуры в зависимости от реализации могут быть разные но в классике жанра 64 ip адрес и 64 мак адрес ну и мак адрес 48 поэтому 16 получается зарезервировано на будущее.** ICMP нужно знать для того, чтобы знать о загруженности сети, ARP/RARP - сопоставление символьного имени и IP адреса, на основании этих двух работают DNS и службы DNS, TCP\UDP - сказать 6 основных свойств разницы их, но их больше, и потом если их принимать как постулаты закона, на все каверзные вопросы преподавателя можно отвечать, т.к на основании закона так-то так-то работает так-то. Вспомним что такое сокеты, как создаются сокеты, какие параметры нужно прописать, чтобы эти сокеты были ориентированы на сообщения или поток.

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

1. отсутствие механизмов обеспечения надежности (пакеты не упорядочиваются, их прием не подтверждается);
2. отсутствие обработки соединений (нет методов установления, управления и завершения соединения между отправителем и получателем);

Все следующие свойства должен обеспечивать процесс прикладного уровня:

3) отсутствие гарантий доставки;

4) отсутствие буферизации (оперирует только одним пакетом);

5) отсутствие фрагментации (не разбивает сообщение на несколько пакетов);

6) отсутствие проверки соответствия контрольной суммы.

Примеры ненадежных протоколов: UDP(игры радио аудио видео), ICMP.

Примеры надежных протоколов: TCP, SMTP.

**Протокол TCP**. Протокол TCP является ***надежным*** байт-ориентированным протоколом с ***установлением соединения***. При получении дейтаграммы IP-протокол извлекает из дейтаграммы данные. На транспортном 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 модуль TCP переправляет данные процессу получателя.

Протокол TCP рассматривает данные отправителя как непрерывный не интерпретируемый поток октетов. При этом TCP при отправке разделяет (если это необходимо) этот поток на TCP-сегменты и объединяет полученные от протокола IP-дейтаграммы при приеме данных.

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

**Надежный и ненадежный протоколы**:

**Надежный** обеспечивает доставку данных без потерь, дублирования или изменения порядка передачи. Он гарантирует, что данные будут успешно доставлены получателю и подтверждает получение каждого пакета данных.

**Ненадежный** протокол не гарантирует доставку данных.

**Протоколы с установкой соединени**я требуют предварительного установления логического канала связи между отправителем и получателем перед передачей данных.

**Протоколы без установки соединения** передают данные независимо друг от друга.

**Ориентированные на поток** гарантируют, что данные будут доставлены в том же порядке, в котором были отправлены, без разделения на отдельные сообщения.

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

# 4 Понятие сокета. Основные параметры сокета.

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

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

### Handle от сокета

**Handle (дескриптор сокета)** — это уникальный идентификатор, который используется для доступа к сокету в операционной системе. Он представляет собой целое число, которое позволяет системе различать сокеты, не сохраняя сам объект сокета.

* **Сокет:** Это объект, который содержит все параметры и состояние соединения, включая настройки протокола, адреса, буферы и т.д.
* **Дескриптор сокета (Handle):** Это абстракция, предоставляемая операционной системой, для взаимодействия с конкретным сокетом. Он не содержит информации о самом сокете, а лишь указывает на его место в памяти.

Для создания сокета используется функция socket.

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

// дескриптор сокета) и задать его характеристики

//

**SOCKET socket(**

**int af,** //[in] формат адреса

**int type,** //[in] тип сокета

**int prot** //[in] протокол

**);**

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

// возвращает дескриптор сокета, в другом

// случае возвращается **INVALID\_SOCKET**

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

// значение **AF\_INET**;

// - 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

После завершения работы с сокетом, обычно, его закрывают (освобождают ресурс).

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

// освобождает все ресурсы связанные с ним

//

**SOCKET closesocket(**

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

**);**

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

// возвращает нуль, в другом случае

// возвращается **SOCKET\_ERROR**

Для установки параметров существующего сокета используется функция bind.

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

// с параметрами, находящимися в структуре

// SOCKADDR\_IN

//

**int bind(**

**SOCKET s,** //[in] сокет

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

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

**)**

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

// возвращает нуль, в случае ошибки

// возвращается **SOCKET\_ERROR**

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

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

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

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

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

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

**struct 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

**};**

Рисунок 3.8.2. Структура SOCKADDR\_IN

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

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

// -- **преобразовать u\_short в формат TCP/IP**

// **Назначение**: функция преобразовывает два байта данных

// формата u\_short (unsigned short) в два

// два байта, сетевого формата

//

**u\_short htons (**

**u\_short hp** //[in] 16 битов данных

**);**

//

// **Код возврата:** 16 битов в формате TCP/IP

//

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.

// -- **преобразовать символьное представление IPv4-адреса в формат TCP/IP**

// **Назначение**: функция преобразует общепринятое символьное

// представление IPv4-адреса (n.n.n.n) в

// четырехбайтовый IP-адрес в формате TCP/IP

//

**unsigned long** **inet\_addr(**

**const char\* stra** //[in] строка символов, закачивающаяся 0x00

**);**

//

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

// IP-адрес в формате TCP/IP, иначе

// возвращается **INADDR\_NONE**

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

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

**SOCKET sS;** // серверный сокет

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

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

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

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

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

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

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

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

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

### 1. **socket**

* **Описание**: Создает новый сокет.
* **Аргументы**:
  + **af**: Семейство адресов (например, AF\_INET для IPv4).
  + **type**: Тип сокета (SOCK\_STREAM для TCP, SOCK\_DGRAM для UDP).
  + **protocol**: Протокол, обычно передается 0 для использования стандартного.

### 2. **bind**

* **Описание**: Привязывает сокет к указанному адресу и порту.
* **Аргументы**:
  + **s**: Дескриптор сокета.
  + **name**: Указатель на структуру sockaddr с адресом и портом.
  + **namelen**: Размер структуры sockaddr.

### 3. **listen**

* **Описание**: Устанавливает сокет в режим прослушивания входящих соединений.
* **Аргументы**:
  + **s**: Дескриптор сокета, который будет прослушивать.
  + **backlog**: Максимальное количество ожидающих соединений в очереди.

### 4. **accept**

* **Описание**: Принимает входящее соединение на сокете и создает новый сокет для общения с клиентом.
* **Аргументы**:
  + **s**: Дескриптор слушающего сокета.
  + **addr**: Указатель на структуру sockaddr, в которую будет записан адрес клиента.
  + **addrlen**: Указатель на переменную, где будет храниться длина структуры sockaddr.

### 5. **connect**

* **Описание**: Устанавливает соединение с удаленным сокетом.
* **Аргументы**:
  + **s**: Дескриптор сокета, который нужно подключить.
  + **name**: Указатель на структуру sockaddr с адресом сервера.
  + **namelen**: Размер структуры sockaddr.

### 6. **send**

* **Описание**: Отправляет данные через сокет.
* **Аргументы**:
  + **s**: Дескриптор сокета для отправки данных.
  + **buf**: Указатель на буфер с данными.
  + **len**: Количество байт для отправки.
  + **flags**: Флаги, определяющие поведение функции.

### 7. **recv**

* **Описание**: Принимает данные из сокета.
* **Аргументы**:
  + **s**: Дескриптор сокета для получения данных.
  + **buf**: Указатель на буфер для записи полученных данных.
  + **len**: Максимальное количество байт для чтения.
  + **flags**: Флаги, определяющие поведение функции.

### 8. **close**

* **Описание**: Закрывает сокет и освобождает ресурсы.
* **Аргументы**:
  + **s**: Дескриптор сокета, который нужно закрыть.

### 9. **getsockname**

* **Описание**: Получает информацию о привязанном адресе сокета.
* **Аргументы**:
  + **s**: Дескриптор сокета.
  + **name**: Указатель на структуру sockaddr для записи адреса.
  + **namelen**: Указатель на переменную для хранения длины структуры sockaddr.

### 10. **getpeername**

* **Описание**: Получает адрес удаленного узла, к которому подключен сокет.
* **Аргументы**:
  + **s**: Дескриптор сокета.
  + **name**: Указатель на структуру sockaddr для записи адреса.
  + **namelen**: Указатель на переменную для хранения длины структуры sockaddr.

# Интерфейс Nimed Pipe.

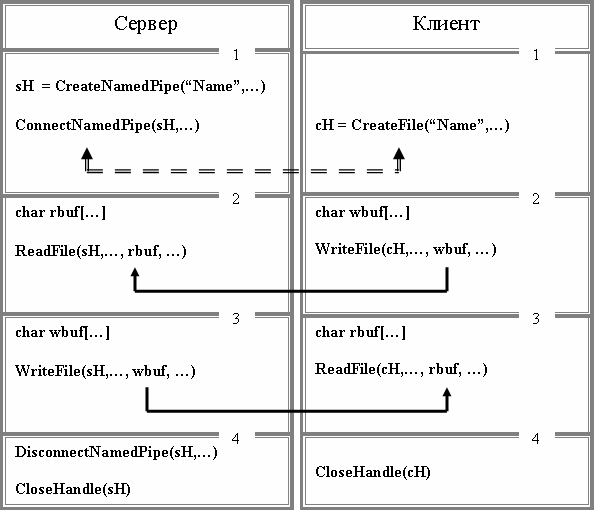
IPC (Inter-Process Communication) — это набор методов, позволяющий различным процессам взаимодействовать друг с другом и обмениваться данными.

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

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

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

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



**\\.\pipe\xxxxx**

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

**pipe -** фиксированное слово;

**xxxxx -** имя канала

**\\servname\pipe\xxxxx**

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

**pipe -** фиксированное слово;

// **-- создать именованный канал**

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

**HANDLE CreateNamedPipe**   **(**

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

**DWORD omode,** // [in] атрибуты канала

**DWORD pmode,** // [in] режимы передачи данных

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

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

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

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

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

**);**

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

// возвращает дескриптор именованного канала, иначе

// возможны следующие значения:

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

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

// превосходит величину **PIPE\_UNLMITED\_INSTANCES**

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

// локальном формате;

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

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

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

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

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

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

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

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

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

// подсоединения к экземпляру именованного канала

// клиента

**BOOL ConnectNamedPipe**

**(**

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

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

**);**

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

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

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

// если используется асинхронная связь, в случае

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

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

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

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

// **-- определить доступность канала**

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

// доступного именованного канала

**BOOL WаitNamedPipe**

**(**

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

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

**);**

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

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

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

// канала задается в локальном формате, если же канал

// создан на другом компьютере, то имя канала следует

// задавать в сетевом формате;

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

// ожидания (в миллисекундах) освобождения экземпляра

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

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

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

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

// ожидания бесконечно

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

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

При использовании форматов имени канала, необходимо помнить, что:

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

// **-- открыть канал**

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

// к именованному каналу

**HANDLE CreateFile**

**(**

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

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

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

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

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

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

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

**);**

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

// возвращает дескриптор именованного канала, иначе

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

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

// сетевом формате: в зависимости от способа применения

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

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

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

// - параметр **share** можетпринимать значения

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

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

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

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

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

// - значение параметра **oflag** всегдаустанавливаетсяв

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

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

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

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

**Обмен данными по именованному каналу**

Для обмена данными по именованному каналу используются три функции: ReadFile, WriteFile и PeekNamedPipe.

// **-- читать данные из канала**

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

// именованного канала

**BOOL ReadFile**

**(**

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

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

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

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

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

**);**

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

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

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

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

// **-- писать данные в канал**

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

// именованный канал

**BOOL WriteFile**

**(**

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

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

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

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

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

**);**

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

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

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

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

Функция PeekNamedPipe копирует данные из канала в буфер. При этом данные не извлекаются и их еще можно считать (извлечь) с помощью функции ReadFile.

// **-- копировать данные канала**

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

// из канала без извлечения

**BOOL PeekNamedPipe**

**(**

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

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

**DWORD sb,** //[in] размер буфера

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

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

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

**);**

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

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

**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.

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

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

// канал и чтения из канала за одну операцию

**BOOL TransactNamedPipe**

**(**

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

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

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

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

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

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

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

**);**

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

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

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

// доступа к каналу, если асинхронный доступ не

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

Если требуется передать только одну транзакцию, то используют функцию CallNamedPipe: сначала осуществляется установка связи с именованным каналом, имя которого указывается в параметрах функции. При этом именованный канал должен быть открыт в режиме данных сообщениями. После установки связи функция пересылает в канал единственное сообщение и получает одно сообщение в ответ. После обмена данными осуществляется разрыв связи с именованным каналом.

// **-- выполнить одну транзакцию**

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

// именованным каналом, выполнения одной транзакции

// и разрыва связи

**BOOL CallNamedPipe**

**(**

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

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

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

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

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

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

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

**);**

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

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

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

// в миллисекундах; кроме того, здесь могут быть

// установлены те же значения, что и в функции

// **WaitNamedPipe**

**Определение состояния и изменение характеристик именованного канала**

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

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

// статических характеристик именованного канала

**BOOL GetNamedPipeInfo**

**(**

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

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

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

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

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

**);**

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

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

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

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

// атрибуты которого запрашиваются; для установки

// этой переменной должны использоваться константы:

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

// типа используемого в функции дескриптора;

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

// типа передачи (поток и сообщения)

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

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

// динамических характеристик именованного канала

**BOOL GetNamedPipeHandleState**

**(**

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

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

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

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

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

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

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

**);**

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

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

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

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

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

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

// передачи сообщениями;

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

// количество байтов, которые клиент именного канала

// должен записать в канал перед передачей серверу;

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

// которые должно пройти прежде, чем данные будут

// переданы

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

// **-- изменить характеристики канала**

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

// динамических характеристик именованного канала

**BOOL SetNamedPipeHandleState**

**(**

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

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

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

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

**);**

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

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

# Интерфейс MailSlot.

IPC (Inter-Process Communication) — это набор методов, позволяющий различным процессам взаимодействовать друг с другом и обмениваться данными.

В этой главе рассматривается еще один IPC – механизм, поддерживаемый операционной системой Windows и имеющий название **Mailslots** (почтовый ящик).

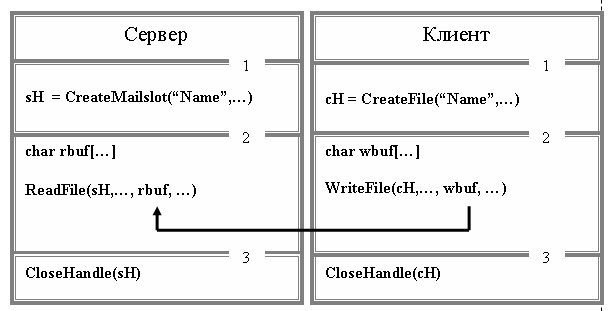
**Назначение и состав интерфейса Mailslot**

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

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

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

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



**\\.\mailslot\xxxxx**

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

**mailslot -** фиксированное слово;

**xxxxx -** имя почтового ящика

**\\servname\mailslot\xxxxx**

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

**mailslot -** фиксированное слово;

**\\domain\mailslot\xxxxx**

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

**mailslot -** фиксированное слово;

// **-- создать почтовый ящик**

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

// ящика

**HANDLE CreateMailslot**

**(**

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

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

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

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

**);**

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

// возвращает дескриптор почтового ящика, иначе

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

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

// **локальном формате**;

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

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

// ожидания, следует установить значение

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

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

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

// возвращает дескриптор почтового ящика, иначе

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

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

// **локальном формате**;

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

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

// ожидания, следует установить значение

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

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

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

// **-- открыть почтовый ящик**

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

// к почтовому ящику

**HANDLE CreateFile**

**(**

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

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

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

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

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

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

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

**);**

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

// возвращает дескриптор именованного канала, иначе

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

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

// сетевом или доменном формате: в зависимости от

// способа применения;

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

// **GENERIC\_WRITE**

// - параметр **share** можетпринимать значения

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

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

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

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

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

// - значение параметра **oflag** всегдаустанавливаетсяв

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

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

// что определяет значения флагов и атрибутов по

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

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

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

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

**char rbuf[100];** // буфер ввода

**try**

**{**

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

**NULL,**

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

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

**throw "CreateMailslotError";**

**if(!ReadFile(hM,**

**rbuf,** // буфер

**sizeof(rbuf),** // размер буфера

**&rb,** // прочитано

**NULL))**

**throw "ReadFileError";**

**}**

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

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

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

**try**

**{**

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

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

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

**NULL,**

**OPEN\_EXISTING,** // ящик уже есть

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

**throw "CreateFileError";**

**if(!WriteFile(hM,**

**wbuf,** // буфер

**sizeof(wbuf),** // размер буфера

**&wb,** // записано

**NULL))**

**throw "ReadFileError";**

**}**

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

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

// характеристик созданного почтового ящика

**BOOL GetMailslotInfo**

**(**

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

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

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

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

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

**);**

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

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

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

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

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

// одной характеристики почтового ящика –

// интервала времени ожидания

**BOOL SetMailslotInfo**

**(**

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

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

**);**

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

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

# Структура программы TCP-сервера.

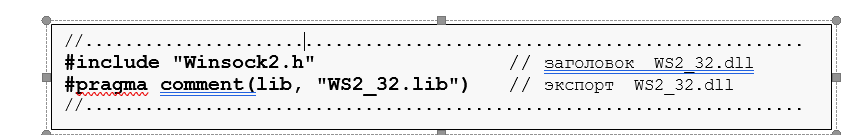
### TCP-сервер

**Определение**:  
TCP (Transmission Control Protocol) — это протокол, обеспечивающий надежную, ориентированную на соединение передачу данных.

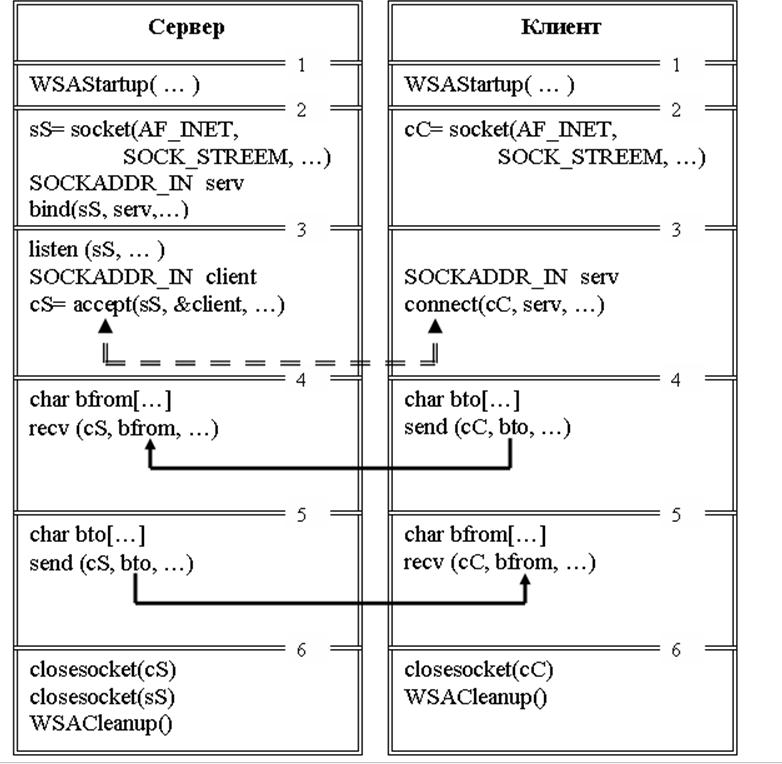
**Особенности**:

* **С соединением**: Требует установления соединения между клиентом и сервером перед передачей данных (тройное рукопожатие).
* **Подтвержденная доставка**: Гарантирует доставку данных, использует механизмы для подтверждения получения и повторной передачи потерянных пакетов.
* **Более медленный**: Из-за дополнительных задержек на установление соединения и подтверждение доставки.
* **Контроль потока**: Обеспечивает управление потоком и перегрузкой, что снижает риск потери данных.
* **Использование**: Подходит для приложений, где важна надежность, например, веб-сайты, электронная почта и передача файлов.

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



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



### 1. **WSAStartup**

* **Описание**: Инициализирует библиотеку Windows Sockets (Winsock).
* **Аргументы**:
  + **wVersionRequested**: Указывает версию Winsock, которую вы хотите использовать. Обычно передается MAKEWORD(2, 2) для версии 2.2.
  + **lpWSAData**: Указатель на структуру WSADATA, в которую будет записана информация о версии и других характеристиках Winsock (
* для хранения информации о Winsock
  + ). Обычно передается NULL, если не нужно получать эту информацию.

### 2. **s = socket(AF\_INET, SOCK\_STREAM, 0)**

* **Описание**: Создает новый сокет.
* **Аргументы**:
  + **af**: Указывает семейство адресов. Например, AF\_INET для IPv4.
  + **type**: Тип сокета. Используйте SOCK\_STREAM для потокового соединения (TCP).
  + **protocol**: Обычно передается NULL для использования стандартного протокола (TCP).

### 3. bind(sS, (SOCKADDR\_IN)&serv, sizeof(serv))\*

* **Описание**: Привязывает сокет к указанному адресу и порту.
* **Аргументы**:
  + **s**: Дескриптор сокета.
  + **name**: Указатель на структуру SOCKADDR\_IN, которая содержит информацию о IP-адресе и порту.
  + **namelen**: Размер структуры SOCKADDR\_IN (обычно используется sizeof(serv)).

### 4. **listen(sS, SOMAXCONN)**

* **Описание**: Устанавливает сокет в режим прослушивания входящих соединений.
* **Аргументы**:
  + **s**: Дескриптор сокета, который будет прослушивать входящие соединения.
  + **backlog**: Максимальное количество ожидающих соединений в очереди. Используйте SOMAXCONN для определения максимального значения.

### 5. cS = accept(sS, (SOCKADDR\_IN)&client, &clientSize)\*

**Описание**: 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.

* **Аргументы**:
  + **s**: Дескриптор слушающего сокета.
  + **addr**: Указатель на структуру SOCKADDR\_IN, в которую будет записан адрес клиента, подключающегося к серверу.
  + **addrlen**: Указатель на переменную, где будет храниться длина структуры SOCKADDR\_IN.

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

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

// канала с определенным сокетом сервера

**int connect (**

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

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

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

**);**

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

// нуль, иначе возвращается значение

// **SOCKET\_ERROR**

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

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

// инициализирована параметрами серверного

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

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

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

### 6. **char bfrom[...]**

* **Описание**: Объявление буфера для получения данных от клиента.

### 7. **char bto[...]**

* **Описание**: Объявление буфера для отправки данных клиенту.

### 8. **recv(cS, bfrom, sizeof(bfrom), 0)**

* **Описание**: Принимает данные из сокета, подключенного к клиенту.
* **Аргументы**:
  + **s**: Дескриптор сокета для получения данных (в данном случае cS).
  + **buf**: Указатель на буфер (bfrom), в который будут записаны полученные данные.
  + **len**: Максимальное количество байт для чтения из сокета (обычно используется sizeof(bfrom)).
  + **flags**: Флаги, определяющие поведение функции (обычно передается 0). вместо 0 можно передать, например, MSG\_WAITALL, который указывает, что функция должна ждать получения всех запрашиваемых байт.

### 9. **send(cS, bto, sizeof(bto), 0)**

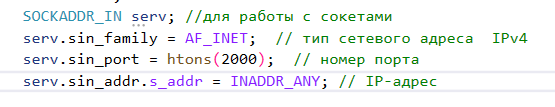
* **Описание**: Отправляет данные через сокет, подключенный к клиенту.
* **Аргументы**:
  + **s**: Дескриптор сокета для отправки данных (cS).
  + **buf**: Указатель на буфер (bto), содержащий данные для отправки.
  + **len**: Количество байт для отправки (обычно используется sizeof(bto)).
  + **flags**: Флаги, определяющие поведение функции (обычно передается 0).

### 10. **closesocket(cS)**

* **Описание**: Закрывает сокет, связанный с клиентом, и освобождает ресурсы.
* **Аргументы**:
  + **s**: Дескриптор сокета, который нужно закрыть (cS).

### 11. **WSACleanup()**

* **Описание**: Завершает работу библиотеки Winsock и освобождает все ресурсы.



# Структура программы TCP-клиента.

# Структура программы UDP-сервера.

# Структура программы UDP-клиента.

### UDP-сервер

**Определение**:  
UDP (User Datagram Protocol) — это протокол передачи данных, который работает на основе датаграмм и не требует установления соединения перед передачей данных.

**Особенности**:

* **Без соединения**: Не требует предварительного установления соединения между клиентом и сервером.
* **Неподтвержденная доставка**: Нет механизма для подтверждения доставки данных, что может привести к потере пакетов.
* **Быстрота**: Более быстрый, так как не требует дополнительной задержки на установление соединения и подтверждение получения.
* **Простота**: Проще в реализации, подходит для приложений, где важна скорость, например, потоковое видео, онлайн-игры и VoIP.
* **Отсутствие контроля потока**: Нет управления потоком, что может привести к перегрузке сети.



Второй блок, в сравнении с сервером, не использует команду bind. 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.

### WSAStartup

* **Сервер и Клиент**:
  + **Аргументы**:
    - MAKEWORD(2, 2): Указывает требуемую версию Winsock (здесь версия 2.2).
    - &wsaData: Указатель на структуру WSADATA, которая будет заполнена информацией о версии Winsock и других параметрах.
  + **Назначение**: Инициализация библиотеки Winsock перед использованием сокетов.

### 2. socket

* **Сервер (sS) и Клиент (cC)**:
  + **Аргументы**:
    - AF\_INET: Указывает, что используется IPv4.
    - SOCK\_DGRAM: Указывает, что используется UDP (для сервера) или другой тип сокета.
    - IPPROTO\_UDP: Указывает конкретный протокол, если он используется (для UDP это может быть 0).
  + **Назначение**: Создание сокета для сетевого взаимодействия.

### 3. bind

* **Сервер**:
  + **Аргументы**:
    - sS: Дескриптор сокета, созданного ранее.
    - &serv: Указатель на структуру sockaddr\_in, содержащую информацию о порте и IP-адресе, к которому будет привязан сокет.
    - sizeof(serv): Размер структуры sockaddr\_in.
  + **Назначение**: Привязка сокета к определенному адресу и порту, чтобы сервер мог принимать входящие сообщения.

### 4. recvfrom

* **Сервер**:
  + **Аргументы**:
    - sS: Дескриптор сокета, через который сервер будет принимать данные.
    - bfrom: Указатель на буфер, в который будут записаны принятые данные.
    - sizeof(bfrom): Максимальное количество байт для чтения из сокета.
    - 0: Флаги, определяющие поведение функции (обычно передается 0).
    - &from: Указатель на структуру sockaddr\_in, в которую будет записан адрес клиента, от которого пришли данные.
    - &fromSize: Указатель на переменную, содержащую размер структуры адреса клиента.
  + **Назначение**: Получение данных от клиента через сокет.

### 5. sendto

* **Сервер и Клиент**:
  + **Аргументы**:
    - sS или cC: Дескриптор сокета, через который отправляются данные (сервер или клиент).
    - bto: Указатель на буфер, содержащий данные, которые необходимо отправить.
    - strlen(bto): Количество байт для отправки.
    - 0: Флаги, определяющие поведение функции (обычно передается 0).
    - &to: Указатель на структуру sockaddr\_in, содержащую адрес получателя (клиента или сервера).
    - toSize: Размер структуры адреса получателя.
  + **Назначение**: Отправка данных на указанный адрес.

### 6. closesocket

* **Сервер и Клиент**:
  + **Аргументы**:
    - sS или cC: Дескриптор сокета, который необходимо закрыть.
  + **Назначение**: Закрытие сокета и освобождение ресурсов.

### 7. WSACleanup

* **Сервер и Клиент**:
  + **Аргументы**: Нет.
  + **Назначение**: Освобождение ресурсов, выделенных для библиотеки Winsock, после завершения работы с сокетами.

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

**try**

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

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

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

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

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

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

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

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

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

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

**char ibuf[50];** //буфер ввода

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

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

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

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

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

**}**

**catch (string errorMsgText)**

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

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

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

**try**

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

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

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

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

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

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

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

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

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

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

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

**}**

**catch (string errorMsgText)**

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

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

# Структура параллельного сервера.

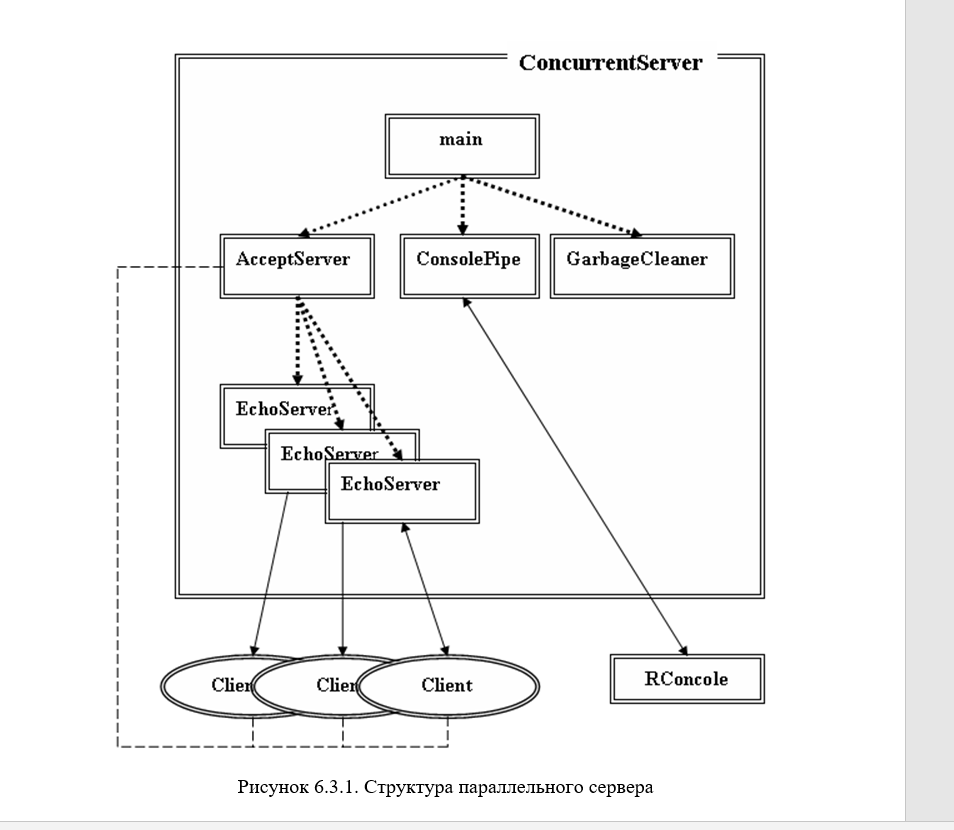
12 Структура параллельного сервера. Accept Server. GarbageCleaner

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

**Работа итеративного сервера** описывается циклом из четырех шагов: 1) ожидание запроса от клиента; 2) обработка запроса; 3) отправка результата запроса; 4) возврат в ждущее состояние 1. 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. 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 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



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.

Все процессы, работающие в рамках сервера, изображены на рисунке прямоугольниками. Пунктирными направленными линиями обозначается создание (запуск) одного процесса другим. Сплошными двунаправленными линиями обозначается перемещение данных. Данные перемещаются между EchoServer-процессами сервера и клиентскими программами, а также между программой с именем RConsole, реализующей клиентскую часть консоли управления, и процессом ConsolePipe, который реализует серверную часть консоли управления. Штриховой линией, соединяющей изображение клиентских программ и изображение процесса AcceptServer, обозначается процедура создания соединения между клиентом и сервером.

**Процесс main –** основное назначение является запуск, инициализация и завершение работы сервера. Именно этот процесс первым получает управление от операционной системы. Процесс main запускает основные процессы: AcceptServer, ConsolePipe и GarbageCleaner.

**Процесс 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 называется ListContact. При подключении очередного клиента, процесс AcceptServer добавляет в ListContact элемент, предназначенный для хранения информации о состоянии данного подключения.

**Процесс ConsolePipe –** создается процессом main и является сервером именованного канала, по которому осуществляется связь между программной RConsole и параллельным сервером.

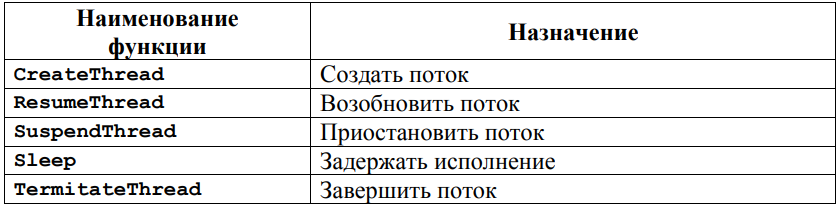
**Процесс GarbageCleaner –** удаление элемента списка подключений ListContact, после отключения программы клиента. Следует отметить, что ListContact является ресурсом, требующим последовательного использования. Одновременная запись и (или) удаление элементов списка может привести к разрушению списка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

**Процесс EchoServer –** создается 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).

**Процесс 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.

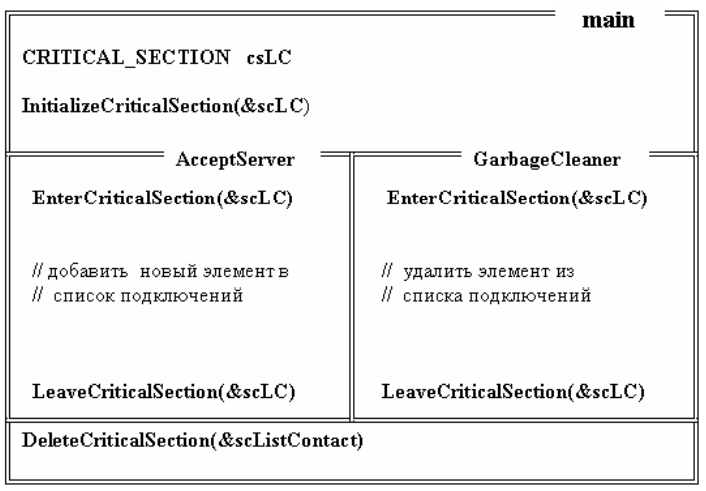
**Процесс RConsole –** предназначена для ввода команд управления сервером и для вывода диагностических сообщений, полученных от сервера. RConsole является клиентом именованного канала.

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

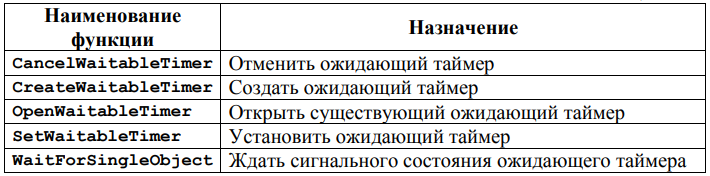


Синхронизация в параллельном сервере будет осуществляться между AcceptServer, который добавляет клиента в ListContact и GarbageCleaner, который удаляет клиента из ListContact.

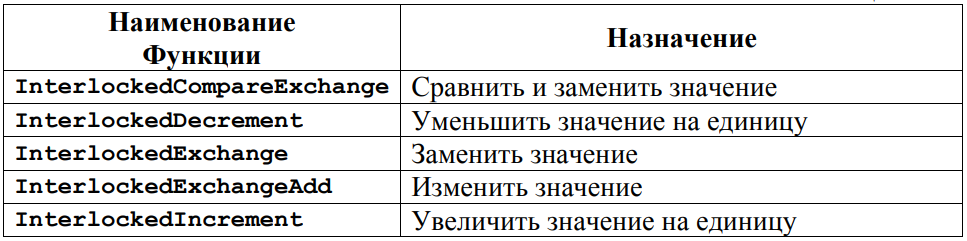
**Критические секции:**

****

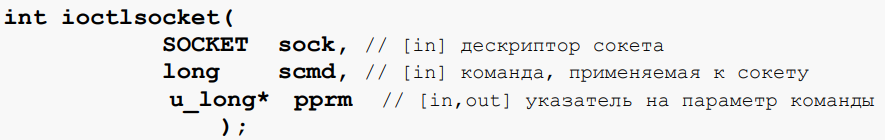
**Ожидающим таймер** в Windows называется объект синхронизации, который переходит в сигнальное состояние при наступлении заданного момента времени.



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

****

**Переключение сокета в режим без блокировки**. В этом режиме выполнение функции accept(), не приостанавливает выполнение потока, как это было прежде, а возвращает значение нового сокета, если обнаружен запрос на создание канала (функция connect(), выполненная клиентом), или значение INVLID\_SOCKET, если запроса на создание канала нет в очереди запросов или возникла ошибка. Для переключение сокета применяется функция ioctlsocket().



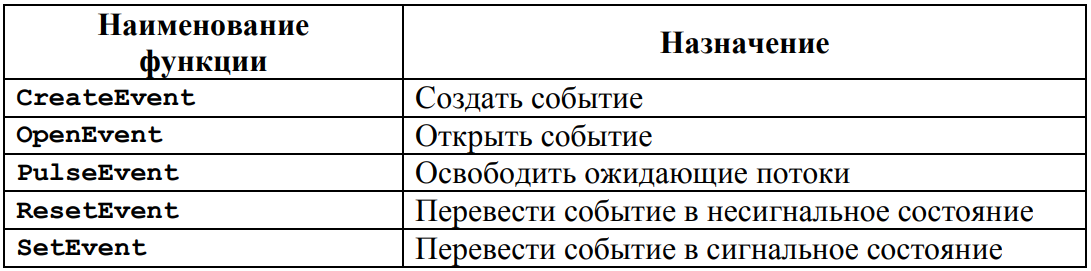


**Обработка запросов клиента.** Основным отличием новой структуры, является промежуточный поток DispatchServer между AcceptServer и обслуживающими потоками ServiceServer (раньше это был единственный поток EchoServer). Теперь предполагается, что сначала программа клиента осуществляет процедуру подключения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

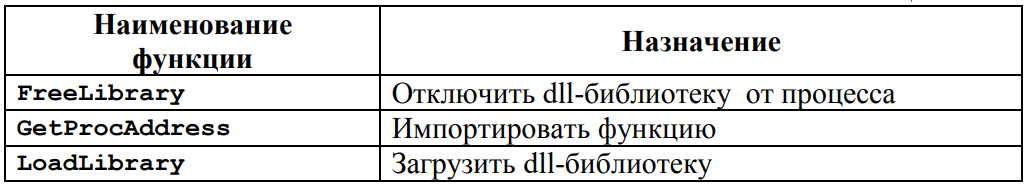
Введение промежуточного звена, обуславливается тем, что после этапа подключения, клиент (в общем случае) может достаточно долго не запрашивать у сервера услугу (не выполнять функцию send()), пересылающую команду сервера). Ожидать поступление команды в потоке AcceptServer не целесообразно, т.к. его основное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

На поток DispatchServer возлагается прием первой команды от клиента после подключения.

**Механизм события,** позволяет оповестить поток о некотором выполненном действии, произошедшем за пределами потока.



**Динамически подключаемые библиотеки** позволяют менять поддерживаемый сервером сервис при неизменной логике управления сервером и обслуживания клиентов.



# 13 Широковещание. Обнаружение сервера с помощью широковещания.

Широковещание — это метод передачи данных в компьютерных сетях, при котором сообщение отправляется всем узлам сети одновременно. Это позволяет эффективно распространять информацию, не зная адресов конкретных получателей.

### Принципы работы:

1. **Широковещательные адреса**:
   * Используются для отправки сообщений о поиске сервера.
   * Предполагается, что сервер находится в состоянии ожидания и откликнется на запрос клиента.
2. **Алгоритм клиента**:
   * При получении откликов от нескольких серверов клиент должен обработать их и выбрать подходящий.
3. **Ограничения**:
   * Широковещательные пакеты работают только внутри локальной сети, так как маршрутизаторы обычно их не пропускают.

### Протокол и сокеты:

* **Протокол UDP**: Для широковещательной связи используется только UDP.
* При создании сокетов используется:
  + type = SOCK\_DGRAM
  + Для обмена данными используются функции sendto и recvfrom.

### Примеры функций:

// **-- установить опции сокета**

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

// использования сокета

**int setsockopt (**

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

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

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

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

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

**);**

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

// нуль, иначе функция возвращает значение

// **SOCKET\_ERROR**

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

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

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

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

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

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

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

**SOCKET cC;**

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

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

**int optval = 1;**

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

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

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

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

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

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

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

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

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

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

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

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

1. **sendto**
   * **Описание:** Отправляет сообщение по сокету.
   * **Аргументы:**
     + SOCKET s: Дескриптор сокета.
     + const char \*buf: Указатель на буфер с данными.
     + int len: Длина данных.
     + unsigned long flags: Флаги (обычно 0).
     + const struct sockaddr \*to: Указатель на адрес получателя.
     + int tolen: Длина адреса получателя.
   * **Возвращаемое значение:** Количество отправленных байт или -1 в случае ошибки.
2. **recvfrom**
   * **Описание:** Принимает сообщение из сокета.
   * **Аргументы:**
     + SOCKET s: Дескриптор сокета.
     + char \*buf: Указатель на буфер для получаемых данных.
     + int len: Максимальная длина данных.
     + unsigned long flags: Флаги (обычно 0).
     + struct sockaddr \*from: Указатель на адрес отправителя.
     + int \*fromlen: Указатель на длину адреса отправителя.
   * **Возвращаемое значение:** Количество полученных байт или -1 в случае ошибки.

### Настройка сокетов:

* **Стандартный широковещательный адрес** задается с помощью INADDR\_BROADCAST.
* Для его использования необходимо установить опцию сокета SO\_BROADCAST через функцию setsockopt.

# Применение символического адреса хоста.

**Символические имена компьютеров** позволяют идентифицировать узлы в сети в удобной для чтения форме, вместо использования числовых IP-адресов. Это делает обращения к узлам более понятными и запоминаемыми.

**Основные функции:**

// -- получить адрес хоста по его имени

// Назначение: функция для получения информации о хосте по

// его символическому имени

hostent\* gethostbyname

(

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

);

// Код возврата: в случае успешного завершения функция

// возвращает указатель на структуру hosten,

// иначе значение NULL

// Примечание: допускается в качестве символического имени,

// указать символическое обозначение адреса

// хоста в виде n.n.n.n

// -- получить имя хоста по его адресу

// Назначение: функция для получения информации о хосте по

// его символическому имени

hostent\* gethostbyaddr

(

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

int la, // [in] длина адреса в байтах

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

);

// Код возврата: в случае успешного завершения функция

// возвращает указатель на структуру hosten,

// иначе возвращается значение NULL

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

**char FAR\* h\_name;** // имя хоста

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

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

**short h\_length;** // длина адреса

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

**} hostent;**

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

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

// -- получить имя хоста

// Назначение: функция для получения собственного имени хоста

int gethostname

(

char\* name , // [out] имя хоста

int ln // [in] длина буфера name

);

// Код возврата: в случае успешного завершения функция

// возвращает нуль, иначе возвращается значение

// SOCKET\_ERROR

**Примеры символических адресов:**

* **Доменные имена:** Например, "google.com" или "facebook.com".
* **Имена в локальной сети:** Например, "laptop" или "printer".

**Процесс разрешения имен:**

* **DNS (Domain Name System):** Служба, которая преобразует символические адреса в IP-адреса, позволяя устанавливать сетевые соединения.

# Основные сетевые утилиты и их назначение.

1. Как ***посмотреть MAC-адрес сетевой карты*** на компьютере?

ipconfig /all и getmac

1. Как посмотреть ***IP-адрес компьютера***.

Ipconfig /all

1. Как протестировать ***IP-соединение в локальной сети*** ?

ping

1. Как получить ***перечень сетевых узлов*** между двумя хостами?

Tracert, pathping

1. Как посмотреть какими программами заняты сетевые порты на компьютере?

netstat

Утилиты - внешние команды ОС для диагностики сети. При этом утилиты, наименования которых выделено жирным шрифтом считаются стандартными для протокола TCP/IP и присутствуют в большинстве ОС.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Утилита ping.** Использует протокол ICMP 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). ping

**ping** hostname

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

**Утилит 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. tracert

**tracert** hostname

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

**Утилита route**. Позволяет манипулировать таблицей сетевых маршрутов, которая имеется на каждом компьютере с TCP/IP. Утилита обеспечивает выполнение 4 команд: print (распечатка таблицы сетевых маршрутов), add (добавить маршрут в таблицу), change (изменение существующего маршрута), delete (удаление маршрута). route

**route** print

**Утилита netstat.** Отражает состояние текущих TCP/IP-соединений хоста, а также статистику работы протоколов. Можно распечатать номера ожидающих портов всех соединений TCP/IP, имена исполняемых файлов, участвующих в подключениях, идентификаторы соответствующих Windows-процессов…

**netstat** -?

Активные соединения TCP/IP на компьютере:

**netstat** -a

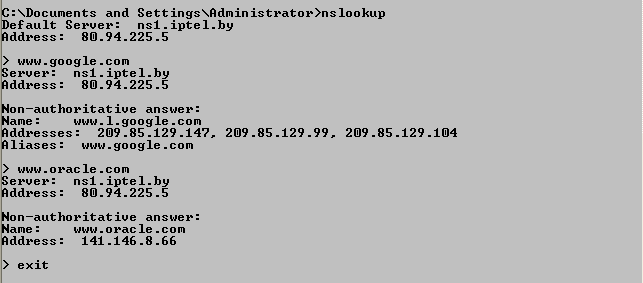
**Утилита arp**. Просмотра и модификации ARP-таблицы, используемой для трансляции IP-адресов в MAC-адреса. Можно распечатывать таблицу, удалять и добавлять данные ARP-таблицы. Корректировку ARP-таблицы может осуществлять только пользователь справами администратора. arp

**arp** -a

Текущее состояние ARP-таблицы:

**Утилита nslookup.** Проверка правильности работы DNS-серверов. Пользователь может выполнять запросы к DNS-серверам на получение адреса хоста по его DNS-имени. 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.

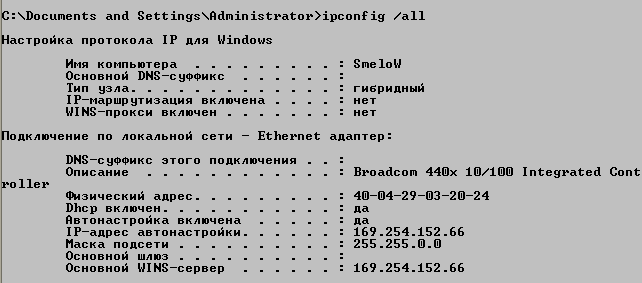


**Утилита hostname**. Вывода на консоль имени хоста. hostname не имеет никаких параметров.

**Утилита ipconfig.** Определить конфигурацию IP-интерфейса и значения всех сетевых параметров.

**ipconfig**  /?

Короткий отчет о конфигурации TCP/IP можно получить выдав команду ipconfig без параметров. Для получения полного отчета /all.



**Утилита nbtstat.** Просматривать статистику текущих соединений, использующих протокол NBT (NetBIOS over TCP/IP). Утилита в чем-то подобна утилите netstat, но применительно к протоколу NBT. nbtstat

**Утилита net**. Основное средство управления сетью для сетевого клиента Windows. 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:

**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

**16) Служба DNS.**

**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:**

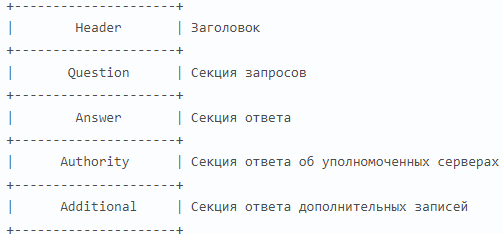
1) Стандартная основная (Standard primary) – главная копия стандартной зоны. Только в данном экземпляре зоны допускается производить какие-либо изменения, которые затем реплицируются на серверы, хранящие дополнительные зоны;

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;

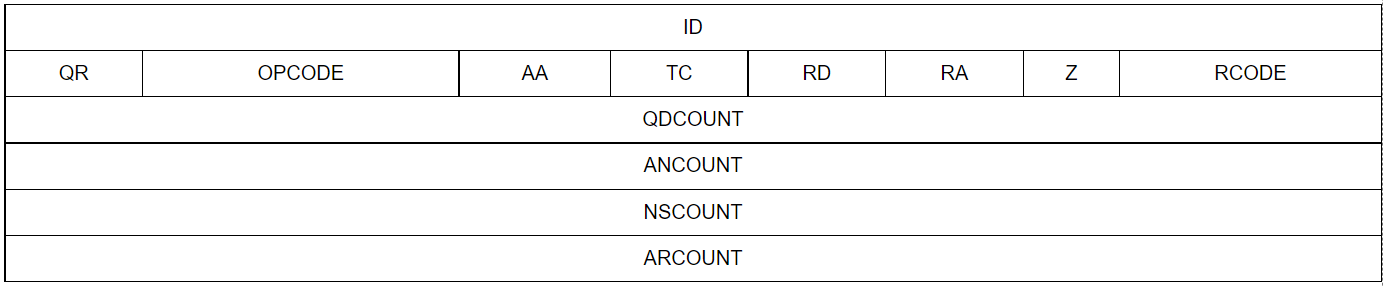
3) NS – указывает на серверы DNS, ответственные за конкретный домен и его поддомены;

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.

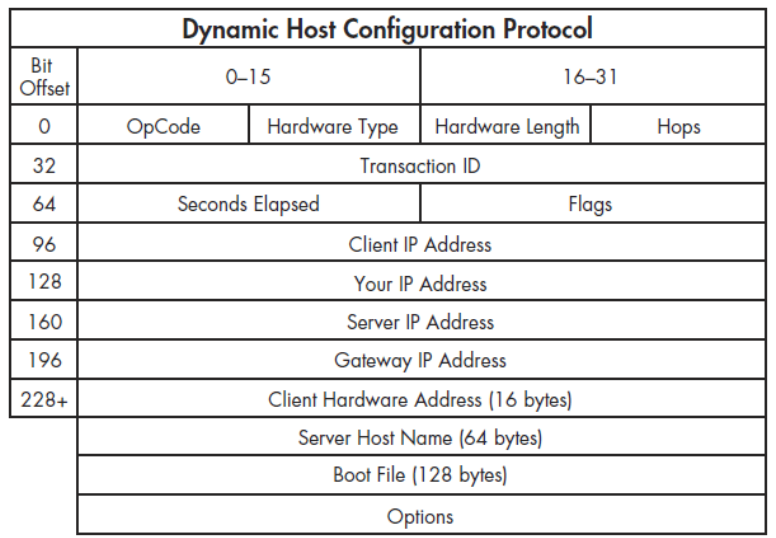
**ARCOUNT** – количество записей в Additional Record Section.

**17) Служба DHCP.**

**DHCP (Dynamic Host Configuration Protocol) –** это сетевая служба прикладного уровня TCP/IP, обеспечивающая выделение и доставку IP-адресов и сопутствующей конфигурационной информации (маска подсети, адрес локального шлюза, адреса серверов DNS и т.п.) хостам. Позволяет отказаться от фиксированных IP-адресов в зоне действия сервера DHCP.

**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.

**18) Стандарты сообщений Internet.**

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

Данная спецификация вносит 2 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) и горизонтальные табуляции.

**Фальцовка –** разбиение строки на несколько строк.

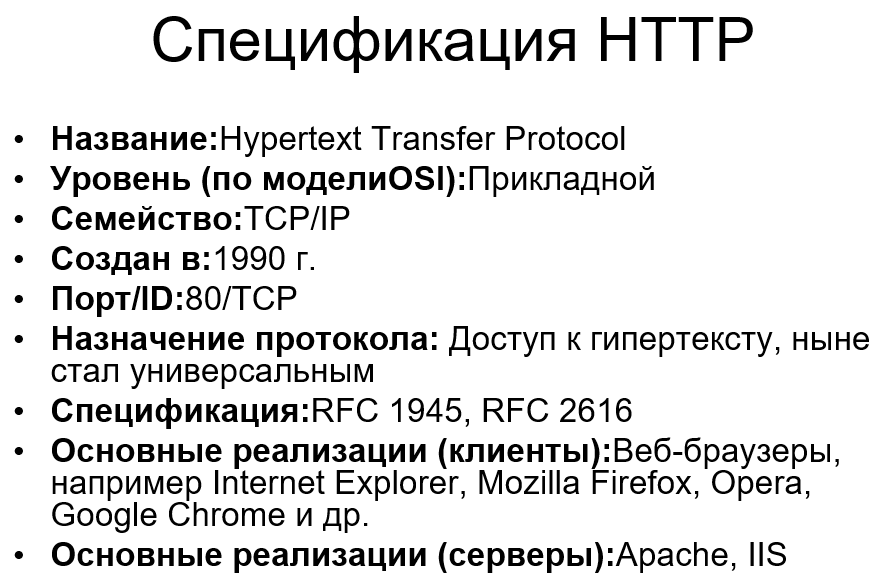
**Расфальцовка –** процесс преобразования фальцованного многострочного представления поля в обычное однострочное.

Некоторые символы имеют специальное значение (например, используются в качестве границ лексем). Для использования таких символов в общепринятом смысле служит механизм квотирования (добавления кавычек).

Поле addr-spec представляет собой специфический для Internet идентификатор, содержащий локально интерпретируемую строку, за которой следует символ “@” и доменное имя Internet. Это локально интерпретируемая строка представляет собой строку в кавычках или атом с точкой.

**19) Протокол HTTP.**

**HTTP (HyperText Transfer Protocol) –** протокол передачи данных.



**Каждое HTTP-сообщение состоит из 3 частей:**

1) Стартовая строка – определяет тип сообщения;

2) Заголовки – характеризуют тело сообщения, параметры передачи и прочие сведения;

3) Тело сообщения – непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строки.

Заголовки и тело могут отсутствовать, но стартовая строка является обязательной.

**Методы HTTP:**

1) GET – запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные;

2) HEAD – запрашивает ресурс так же, как и метод GET, но без тела ответа;

3) POST – используется для отправки сущностей к определенному ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере;

4) PUT – заменяет все текущие представления ресурса данными запроса;

5) DELETE – удаляет указанный ресурс;

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

7) OPTIONS – используется для описания параметров соединения с ресурсом;

8) TRACE – выполняет вызов возвращаемого тестового сообщения с ресурса;

9) PATCH – используется для частичного изменения ресурса.

**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).

**20) Служба RPC.**

**RPC (Remote Procedure Call) –** удаленный вызов процедур. Позволяет вызывать функции и процедуру в другом адресном пространстве.

**Порядок действий:**

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.

**Маршалинг –** процесс укомплектования параметров в сообщение.

**Самый длительный процесс – процесс передачи.**

**Дополнительно**

Интерфейс RPC определяет программный механизм, который первоначально был разработан в компании Sun Microsystems и предназначался для того, чтобы упростить разработку распределенных приложений. Спецификация RPC компании Sun Microsystems содержится в документах RFC 1059, 1057, 1257.

RPC Sun Microsystems реализована в двух модификациях: одна выполнена на основе API сокетов для работы над TCP и UDP, другая, названная ***TI-RPC*** ***(Transport Independent RPC)***, использует API TLI (Transport Layer Interface, компании AT&T) и способна работать с любым транспортным протоколом, поддерживаемый ядром операционной системы.

Идея, положенная в основу RPC, заключается в разработке специального API, позволяющего осуществлять вызов ***удаленной процедуры*** (процедуры, которая находится и исполняется на другом хосте) способом, по возможности, ничем не отличающимся от вызова локальной процедуры из динамической библиотеки. Реализация этой идеи осложняется необходимостью учитывать возможность различия операционных сред, в которых работают вызывающая и вызываемая процедуры (отсюда, различные типы данных, невозможность обрабатывать адресные указатели и т.п.). Кроме того, следует предусмотреть обработку внепланового завершения процедуры на одной из сторон распределенного приложения. Все эти проблемы сделали интерфейс RPC достаточно сложным. Прозрачность механизма вызова достигается созданием вместо вызываемой и вызывающей процедур специальных программных заглушек, называемых ***клиентским*** и ***серверным*** ***стабами.***

Клиентским стабом называется тот стаб, который находится на хосте с вызывающей процедурой. Его основной задачей является преобразовать передаваемые параметры в формат стандарта ***XDR (External Data Representation)*** и скрыть (подменив вызываемую удаленную процедуру локальным вызовом стаба) от пользователя механизм RPC.

Серверный стаб находится на том же хосте, что и вызываемая процедура и предназначен для преобразования полученных параметров из формата XDR в формат, воспринимаемый вызываемой процедуры, а также для сокрытия (серверный стаб подменяет вызывающую процедуру на стороне сервера) RPC-механизма от вызываемой процедуры.

Стандарт XDR предназначен кодирования полей в запросах и ответах интерфейса RPC. Стандарт регламентирует все типы данных и уточняет способ их передачи в RPC-сообщениях. Спецификация стандарта XDR приведена в RFC 1014.

Число вызываемых удаленных процедур не регламентируется спецификацией RPC. Поэтому за ними не закрепляются конкретные TCP-порты. Порты получают сами удаленные процедуры динамическим образом (эфемерные порты). Учет соответствия портов вызываемым процедурам осуществляет специальная программа ***PortMapper*** (регистратор портов). Сам PortMapper доступен по 111 порту и тоже является удаленной процедурой. Процедура PortMapper – это связующее звено между различными компонентами системы. Всякая вызываемая процедура, должна быть зарегистрирована в базе данных PortMapper с помощью специальных служебных функций. Вызывающая сторона (клиентский стаб) с помощью все тех же служебных функций может получить спецификацию вызываемой процедуры.

Развитием технологии RPC для объектно-ориентированного программирования в операционной системе Windows являются технологии ***COM*** и ***DCOM***, которые позволяют создавать удаленные объекты.

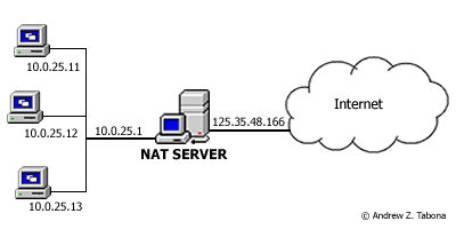
Аналогом RPC в Java-технологиях является механизм ***RMI (Remote Method Invocation)***, позволяющий работать с удаленными Java-объектами. Информацию об объекте и его методах вызывающая сторона может получить, обратившись к реестру RMI (аналогу PortMapper).

Сетевая файловая система ***NFS (Network File System)***, повсеместно используемая в качестве службы прозрачного удаленного доступа к файлам основана на механизме RPC Sun Microsystems.

Следует отметить, что кроме Sun-реализации интерфейса RPC, широко применяется и конкурирующий программный продукт, разработанный объединением OSF (Open Software Foundation).

**21) NAT, proxy-серверы, межсетевые экраны, ремайлеры.**

**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.

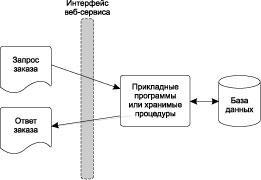
Стандарт MIXMINIOM анонимной перессылки почты, по этому стандарту можно отправлять электронную почту.

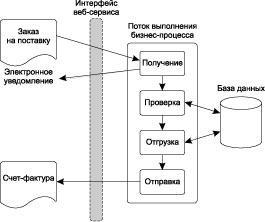
**22) Web-сервисы: SOAP, XML, WSDL, UDDI.**

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 withAttachments ("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.

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.

SOAP — это стандарт для отсылки и получения сообщений по Internet. Изначально этот протокол был предложен фирмой Microsoft в качестве средства для удаленного вызова процедур (RPC, Remote Procedure Call) по протоколу HTTP, а спецификация SOAP 1.0 (Userland, Microsoft, Developmentor) была тесно связана с Component Object Model. Фирма IBM и ряд других компаний, в том числе Lotus, внесли определенный вклад в развитие этого протокола, и его спецификация была направлена на рассмотрение комитетом W3C.  
  
Спецификация SOAP определяет XML-«конверт» для передачи сообщений, метод для кодирования программных структур данных в формате XML, а также средства связи по протоколу HTTP.  
  
SOAP-сообщения бывают двух типов: запрос (Request) и ответ (Response). Запрос вызывает метод удаленного объекта, ответ возвращает результат выполнения данного метода.

**23) Национальная инфраструктура информационной безопасности.**

информационная безопасность - состояние защищенности сбалансированных интересов личности, общества и государства от внешних и внутренних угроз в информационной сфере;

**Статья 349. Несанкционированный доступ к компьютерной информации**

1. Несанкционированный доступ к информации, хранящейся в компьютерной системе, сети или на машинных носителях, сопровождающийся нарушением системы защиты (несанкционированный доступ к компьютерной информации), повлекший по неосторожности изменение, уничтожение, блокирование информации или вывод из строя компьютерного оборудования либо причинение иного существенного вреда, –

наказывается штрафом или арестом на срок до шести месяцев.

1. Несанкционированный доступ к компьютерной информации, совершенный из корыстной или иной личной заинтересованности, либо группой лиц по предварительному сговору, либо лицом, имеющим доступ к компьютерной системе или сети, –

наказывается штрафом, или лишением права занимать определенные должности или заниматься определенной деятельностью, или арестом на срок от трех до шести месяцев, или ограничением свободы на срок до двух лет, или лишением свободы на тот же срок.

1. Несанкционированный доступ к компьютерной информации либо самовольное пользование электронной вычислительной техникой, средствами связи компьютеризованной системы, компьютерной сети, повлекшие по неосторожности крушение, аварию, катастрофу, несчастные случаи с людьми, отрицательные изменения в окружающей среде или иные тяжкие последствия, –

наказываются ограничением свободы на срок до пяти лет или лишением свободы на срок до семи лет.

**Статья 350. Модификация компьютерной информации**

1. Изменение информации, хранящейся в компьютерной системе, сети или на машинных носителях, либо внесение заведомо ложной информации, причинившие существенный вред, при отсутствии признаков преступления против собственности (модификация компьютерной информации) –

наказываются штрафом, или лишением права занимать определенные должности или заниматься определенной деятельностью, или арестом на срок от трех до шести месяцев, или ограничением свободы на срок до трех лет, или лишением свободы на тот же срок.

1. Модификация компьютерной информации, сопряженная с несанкционированным доступом к компьютерной системе или сети либо повлекшая по неосторожности последствия, указанные в части третьей статьи 349 настоящего Кодекса, –

наказывается ограничением свободы на срок до пяти лет или лишением свободы на срок до семи лет с лишением права занимать определенные должности или заниматься определенной деятельностью или без лишения.

**Статья 351. Компьютерный саботаж**

1. Умышленные уничтожение, блокирование, приведение в непригодное состояние компьютерной информации или программы, либо вывод из строя компьютерного оборудования, либо разрушение компьютерной системы, сети или машинного носителя (компьютерный саботаж) –

наказываются штрафом, или лишением права занимать определенные должности или заниматься определенной деятельностью, или арестом на срок от трех до шести месяцев, или ограничением свободы на срок до пяти лет, или лишением свободы на срок от одного года до пяти лет.

1. Компьютерный саботаж, сопряженный с несанкционированным доступом к компьютерной системе или сети либо повлекший тяжкие последствия, –

наказывается лишением свободы на срок от трех до десяти лет.

**Статья 352. Неправомерное завладение компьютерной информацией**

Несанкционированное копирование либо иное неправомерное завладение информацией, хранящейся в компьютерной системе, сети или на машинных носителях, либо перехват информации, передаваемой с использованием средств компьютерной связи, повлекшие причинение существенного вреда, –

наказываются общественными работами, или штрафом, или арестом на срок до шести месяцев, или ограничением свободы на срок до двух лет, или лишением свободы на тот же срок.

**Статья 353. Изготовление либо сбыт специальных средств для получения неправомерного доступа к компьютерной системе или сети**

Изготовление с целью сбыта либо сбыт специальных программных или аппаратных средств для получения неправомерного доступа к защищенной компьютерной системе или сети –

наказываются штрафом, или арестом на срок от трех до шести месяцев, или ограничением свободы на срок до двух лет.

**Статья 354. Разработка, использование либо распространение вредоносных программ**

1. Разработка компьютерных программ или внесение изменений в существующие программы с целью несанкционированного уничтожения, блокирования, модификации или копирования информации, хранящейся в компьютерной системе, сети или на машинных носителях, либо разработка специальных вирусных программ, либо заведомое их использование, либо распространение носителей с такими программами –

наказываются штрафом, или арестом на срок от трех до шести месяцев, или ограничением свободы на срок до двух лет, или лишением свободы на тот же срок.

1. Те же действия, повлекшие тяжкие последствия, –

наказываются лишением свободы на срок от трех до десяти лет.

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

• Секретность

• Аутентификация

• Обеспечение строго выполнения обязательств

• Обеспечение целостности

**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 и аутентичности.

• Криптосистема

• Криптоанализ

• Криптология

• Криптографическая стойкость

**Криптогра́фия**— 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.

**Криптоло́гия**  — 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.