Содержание

[1. Алгоритм: базовые конструкции, способы задания, оценка временной сложности на примере алгоритмов поиска](#h.w1g6utvik2fc)

[2. Тип данных. Структуры данных. Классификация структур данных](#h.315hmzjxgn9x)

[3. Базовые и улучшенные сортировки на основе выбора, включения и обмена. Сравнительный анализ алгоритмов сортировки](#h.ladilxi2x1n3)

[4. Структуры данных типа стек. Реализация стека как отображения на массив и односвязный список. Примеры применения](#h.w78q7dgjcbeq)

[5. Структуры данных типа очередь. Реализация очереди как отображение на массив и односвязный список. Примеры применения.](#h.15y4gwh6nupw)

[6. Структуры данных типа таблица. Прямого доступа, хеш-таблица. Разрешение коллизий с помощью цепочек и открытой адресации и анализ их алгоритмов](#h.4vj7oo81u386)

[7. Структуры данных бинарное дерево. Операции включения, исключения. Алгоритмы поиска и прохождения](#h.g1yr5lr3bnvj)

[8. Программные продукты как сложные системы. Признаки сложных систем](#h.5one8lxve30g)

[Декомпозиция. Преимущества и недостатки объектно- ориентирован- ного подхода в программировании](#h.opxhu5m8vbhy)

[9. Основные понятия объектно-ориентированного программирования: объект, класс, виды отношений между классами](#h.bd4co06wmsuj)

[10. Основные принципы объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм](#h.hffu154yrxs4)

[11. Общая характеристика объектов в объектно-ориентированном программировании. Объектная декомпозиция. Виды отношений между объектами](#h.kt39txk0c7wp)

[13. Понятие «базы данных». Основные компоненты базы данных](#h.ipki02lfg55r)

[14. Системы баз данных. Архитектура систем баз данных. Современные СУБД.](#h.h2m7z36cm66j)

[22. Процессы и потоки в многозадачной операционной системе. Алгоритмы планирования и управления.](#h.hh436tojwty3)

[23. Управление памятью в операционной системе. Физическая память. Виртуальная память. Адресное пространство. Алгоритмы замещения страниц памяти.](#h.8j88xobk5bi9)

[24. Межпроцессное взаимодействие. Механизмы взаимодействия процессов.](#h.9qwv529vp9z1)

[25. Файловые системы. Основные особенности и физическая организация на примере одной из современных файловых систем.](#h.m4r4sy1rjulz)

[26. Каналы передачи данных. Физический канал. Логический канал. Понятие блока данных. Пример формата блока данных любого протокола](#h.lykol7tf66jq)

[27. Структуризация сетей. Понятие и характеристики основных сетевых топологий. Структурообразующие аппаратные средства и программное обеспечение](#h.ei1lkifdctu0)

[28. Понятие адресации в сетях. Типы адресов. Адресация в IP-сетях](#h.6et20adnhjjr)

[29. Характеристика протоколов IP, TCP, ARP, ICMP, POP3, SMTP](#h.qxvpplqhddmm)

[30. Эталонная модель взаимодействия открытых систем. Понятие уровня и уровневых сетевых сервисов. Характеристика уровней](#h.qjfit9spujzr)

[31. Этапы сбора и анализа требований к программной системе, требований заказчика и детальных требований](#h.jybajph9kid)

[32.Требования к программному обеспечению. Типы требований. Уровни описания требований](#h.fo5r2sxm3acc)

# **1.** **Алгоритм: базовые конструкции, способы задания, оценка временной сложности на примере алгоритмов поиска**

Базовые конструкции: последовательное выполнение, цикл, условие. Цикл с постусловием, предусловием.

Способы задания: словесный — произвольное изложение на естественном языке. Графический - последовательность связанных между собой функциональных блоков,

каждый из которых соответствует выполнению одного или нескольких действий. Псевдокод занимает промежуточное положение между естественным языком и формальным. С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи. В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.

Оценка временной сложности — функция времени выполнения алгоритма от размера входных данных. Это функция размера входных и выходных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера. Сложность бинарного поиска: Θ( log( n ) ). Поиск продолжаться, пока не достигнем единицы. Если мы захотим узнать, на какой итерации это произошло, нам нужно будет просто решить следующее уравнение:

1 = n / 2^i

2^i = n

i = log( n )

# **2.** **Тип данных. Структуры данных. Классификация структур данных**

Тип данных - множество значений и операций на этих значениях.

Структуры данных - программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике.

Классификация структур данных

Простые — состоят из простых типов. Составные — состоят из других структур.

Физическая структура данных - это способ физического представления данных в памяти компьютера.

Логическая или абстрактная структура - это рассмотрение структуры данных без учета его представления в машинной памяти.

Несвязные структуры характеризуются отсутствием связей между элементами структуры.

Связные структуры характеризуются наличием связи. Примерами несвязных структур есть векторы, массивы, строки, стеки, очереди; примеры связных структур - связные списки.

Статические - к этой группе относят массивы, множества, записи, таблицы.

Полустатические - это стеки, очереди, деки, дерева.

Динамические - линейные и разветвленные связные списки, графы, дерева.

Структуры данных для оперативной памяти - это данные, размещенные в статической и динамической памяти компьютера. Все вышеприведенные структуры данных - это структуры для оперативной памяти.

Структуры данных для внешней памяти называют файловыми структурами или файлами. Примерами файловых структур есть последовательные файлы, файлы, организованные разделами, В- деревья.

Линейные структуры с **последовательным** распределением элементов в памяти (векторы, строки, массивы, стеки, очереди) и структуры с **произвольным** связным распределением элементов в памяти (односвязные и двусвязные линейные списки).

Нелинейные структуры - многосвязные списки, деревья, графы.

# **3.** **Базовые и улучшенные сортировки на основе выбора, включения и обмена. Сравнительный анализ алгоритмов сортировки**

**Сортировка пузырьком**. Идея метода: шаг сортировки состоит в проходе снизу вверх по массиву. По пути просматриваются пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами. После нулевого прохода по массиву "вверху" оказывается самый "легкий" элемент - отсюда аналогия с пузырьком. Следующий проход делается до второго сверху элемента, таким образом второй по величине элемент поднимается на правильную позицию...

Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

Среднее число сравнений и обменов имеют квадратичный порядок роста: Theta(n^2), отсюда можно заключить, что алгоритм пузырька очень медленен и малоэффективен.

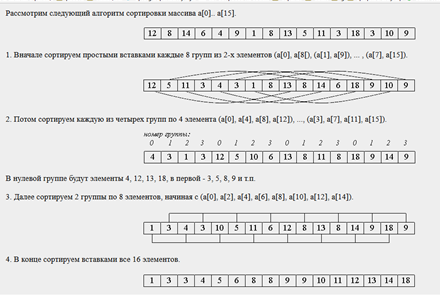
**Сортировка выбором**. Идея метода состоит в том, чтобы создавать отсортированную последовательность путем присоединения к ней одного элемента за другим в правильном порядке. На i-м шаге выбираем наименьший из элементов a[i] ... a[n] и меняем его местами с a[i]. С учетом того, что количество рассматриваемых на очередном шаге элементов уменьшается на единицу, общее количество операций: Theta(n^2).

**Сортировка простыми вставками**. На следующем, (i+1)-м каждом шаге алгоритма берем a[i+1] и вставляем на нужное место в готовую часть массива.

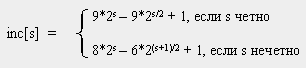
Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним. В зависимости от результата сравнения элемент либо остается на текущем месте(вставка завершена), либо они меняются местами и процесс повторяется. Аналогично сортировке выбором, среднее, а также худшее число сравнений и пересылок оцениваются как Theta(n^2).



**Сортировка Шелла** является довольно интересной модификацией алгоритма сортировки простыми вставками.



Использованный в примере набор ..., 8, 4, 2, 1 - неплохой выбор, особенно, когда количество элементов - степень двойки. Однако гораздо лучший вариант предложил Р. Седжвик. Его последовательность имеет вид:



При использовании таких приращений среднее количество операций: O(n^(7/6)), в худшем случае - порядка O(n^(4/3)).

При использовании формулы Седжвика следует остановиться на значении inc[s-1], если 3\*inc[s] > size.

**Пирамидальная сортировка**. O(n log n). Строим из массива пирамиду.

·в a[0] хранится корень дерева

·левый и правый сыновья элемента a[i] хранятся, соответственнно, в a[2i+1] и a[2i+2]

Плюсы такого хранения пирамиды очевидны:

·никаких дополнительных переменных, нужно лишь понимать схему.

· узлы хранятся от вершины и далее вниз, уровень за уровнем.

· узлы одного уровня хранятся в массиве слева направо.

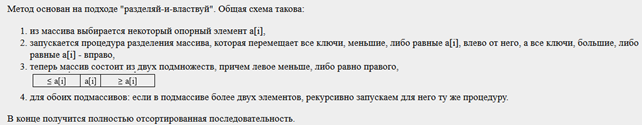
Итак, задача построения пирамиды из массива успешно решена. Как видно из свойств пирамиды, в корне всегда находится максимальный элемент. Отсюда вытекает алгоритм фазы 2:

Берем верхний элемент пирамиды a[0]...a[n] (первый в массиве) и меняем с последним местами. Теперь "забываем" об этом элементе и далее рассматриваем массив a[0]...a[n-1]. Для превращения его в пирамиду достаточно просеять лишь новый первый элемент.

Повторяем шаг 1, пока обрабатываемая часть массива не уменьшится до одного элемента.

Очевидно, в конец массива каждый раз попадает максимальный элемент из текущей пирамиды, поэтому в правой части постепенно возникает упорядоченная последовательность.

**Быстрая сортировка**



На входе массив a[0]...a[N] и опорный элемент p, по которому будет производиться разделение.

1. Введем два указателя: i и j. В начале алгоритма они указывают, соответственно, на левый и правый конец последовательности.

2. Будем двигать указатель i с шагом в 1 элемент по направлению к концу массива, пока не будет найден элемент a[i] >= p. Затем аналогичным образом начнем двигать указатель j от конца массива к началу, пока не будет найден a[j] <= p.

3. Далее, если i <= j, меняем a[i] и a[j] местами и продолжаем двигать i,j по тем же правилам...

4. Повторяем шаг 3, пока i <= j.

быстродействие: O(n log n), что и имеет место на практике.

# **4.** **Структуры данных типа стек. Реализация стека как отображения на массив и односвязный список. Примеры применения**

Стек - тип данных, представляющий собой список элементов, организованных по принципу последним пришёл — первым вышел. Программный вид стека используется для обхода структур данных, например, дерево или граф. При использовании рекурсивных функций также будет применяться стек, но аппаратный его вид. Кроме этих назначений, стек используется для организации стековой машины, реализующей вычисления в обратной инверсной записи.

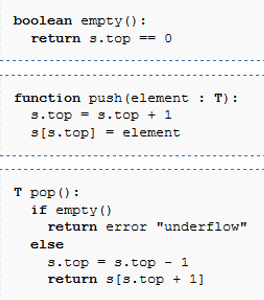
Перед реализацией стека выделим ключевые поля:

· {s[1] … n]} — массив, с помощью которого реализуется стек, способный вместить не более n элементов,

· s.top — индекс последнего помещенного в стек элемента.

Стек состоит из элементов {s[1] … s.top]}, где s[1] — элемент на дне

стека, [s.top] — элемент на его вершине. Если s.top = 0, то стек не содержит ни одного элемента и является пустым.

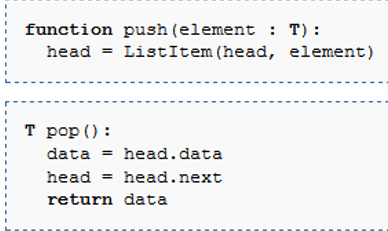


Стек можно реализовать и на списке. Для этого необходимо создать список и операции работы стека на созданном списке. Ниже представлен пример реализации стека на односвязном списке. Стек будем "держать" за голову. Добавляться новые элементы посредством операции push будут перед головой, сами при этом становясь новой головой, а элементом для изъятия из стека с помощью pop будет текущая голова. После вызова функции push текущая голова уже станет старой и будет являться следующим элементом за добавленным, то есть ссылка на следующий элемент нового элемента будет указывать на старую голову. После вызова функции pop будет получена и возвращена информация, хранящаяся в текущей голове. Сама голова будет изъята из стека, а новой головой станет элемент, который следовал за изъятой головой.

Ключевые поля:

· head.data — значение в верхушке стека,

· head.next — значение следующее за верхушкой стека.



# **5.** **Структуры данных типа очередь. Реализация очереди как отображение на массив и односвязный список. Примеры применения.**

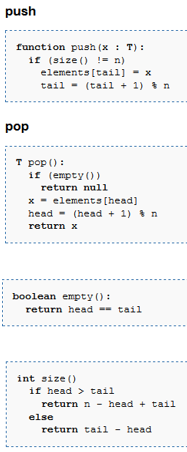
О́чередь — абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел». Очередь в программировании используется, как и в реальной жизни, когда нужно совершить какие-то действия в порядке их поступления, выполнив их последовательно. Примером может служить организация событий в Windows. Когда пользователь оказывает какое-то действие на приложение, то в приложении не вызывается соответствующая процедура (ведь в этот момент приложение может совершать другие действия), а ему присылается сообщение, содержащее информацию о совершенном действии, это сообщение ставится в очередь, и только когда будут обработаны сообщения, пришедшие ранее, приложение выполнит необходимое действие.

Очередь, способную вместить не более n элементов, можно реализовать с помощью **массива** [0 …n-1]. Она будет обладать следующими полями:

· head — голова очереди,

· tail — хвост очереди.

**ТУТ КОД ДЛЯ КОЛЬЦЕВОЙ ОЧЕРЕДИ**



**На списке**.

· ListItem(data : T, next : ListItem) — конструктор,

· x.value — поле, в котором хранится значение элемента,

· x.next — указатель на следующий элемент очереди.

# **6.** **Структуры данных типа таблица. Прямого доступа, хеш-таблица. Разрешение коллизий с помощью цепочек и открытой адресации и анализ их алгоритмов**

Таблица — набор элементов одинаковой организации, каждый из которых можно представить в виде двойки <K, V>, где K — ключ, а V — тело (информационная часть) элемента.

Ключ уникален для каждого элемента, то есть в таблице нет элементов с одинаковыми ключами. Ключ используется для доступа к элементам при выполнении операций.

В таблице прямого доступа ключ – адрес элемента.

Хеш-таблица — это таблица, в которой положение адреса элемента определяется с помощью некоторой функции H (хеш-функции), аргументом которой является значение ключа элемента.

**Метод цепочек.** Каждая ячейка массива H является указателем на связный список (цепочку) пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются цепочки длиной более одного элемента.

Операции поиска или удаления элемента требуют просмотра всех элементов соответствующей ему цепочки, чтобы найти в ней элемент с заданным ключом. Для добавления элемента нужно добавить элемент в конец или начало соответствующего списка, и, в случае, если коэффициент заполнения станет слишком велик, увеличить размер массива H и перестроить таблицу. Коэффициент заполнения высчитывается как отношение количества уже внесенных элементов к количеству возможных значений хэш-функции.

При предположении, что каждый элемент может попасть в любую позицию таблицы H с равной вероятностью и независимо от того, куда попал любой другой элемент, среднее время работы операции поиска элемента составляет Θ(1 + α), где α — коэффициент заполнения таблицы.

**Открытая адресация.** В массиве H хранятся сами пары ключ-значение. Алгоритм вставки элемента: сначала вычисляется хэш-функция, если возникает коллизия, то алгоритм проверяет ячейки массива H в некотором порядке до тех пор, пока не будет найдена первая свободная ячейка, в которую и будет записан новый элемент. Этот порядок вычисляется на лету, что позволяет сэкономить на памяти для указателей, требующихся в хеш-таблицах с цепочками.

Вставка - О( 1 / (1-α) ); поиск - О( (1/α) \* ln( 1/(1+α) ) ); удаление - О(1). 0 < α < 1. α опт. = 0,5.

# **7.** **Структуры данных бинарное дерево. Операции включения, исключения. Алгоритмы поиска и прохождения**

Бинарное дерево - иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками.

Существует два вида обхода графа: 1) в глубину; 2) в ширину.

Поиск в ширину (BFS) идет из начальной вершины, посещает сначала все вершины, находящиеся на расстоянии одного ребра от начальной, потом посещает все вершины на расстоянии два ребра от начальной и так далее.

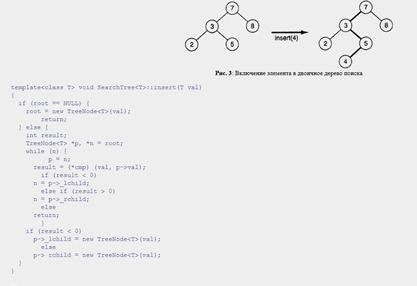
Поиск в глубину (DFS) идет из начальной вершины, посещая еще не посещенные вершины без оглядки на удаленность от начальной вершины.

Cложность обхода и в ширину и в глубину O(V + E), где V – количество вершин, E – количество ребер. То есть является линейной по количеству вершин и ребер.

Прямой обход идет в следующем порядке: корень, левый потомок, правый потомок. Симметричный — левый потомок, корень, правый потомок. Обратный – левый потомок, правый потомок, корень.

**Включение** **и удаление элемента в дерево поиска.**

**Включение.** Для включения нового элемента в двоичное дерево поиска вначале нужно определить его точное положение — а именно внешний узел, который должен быть заменен путем отслеживания пути поиска элемента, начиная с корня. Кроме сохранения указателя n на текущий узел мы будем хранить указатель р на предка узла n. Таким образом, когда n достигнет некоторого внешнего узла, р будет указывать на узел, который должен стать предком нового узла. Для осуществления включения узла мы создадим новый узел, содержащий новый элемент, и затем свяжем предок р с этим новым узлом (рис. 3).



**Удаление**. Чтобы удалить элемент из дерева поиска, вначале мы отслеживаем путь поиска элемента, начиная с корня и вниз до узла n, содержащего элемент. В этот момент могут возникнуть три ситуации:

1. Узел n имеет пустой левый потомок. В этом случае ссылка на n (записанная в предке n, если он есть) заменяется на ссылку на правого потомка n.

2. У узла n есть непустой левый потомок, но правый потомок пустой. В этом случае ссылка вниз на n заменяется ссылкой на левый потомок узла n.

3. Узел n имеет два непустых потомка. Найдем последователя для n (назовем его m), скопируем данные, хранящиеся в m, в узел n и затем рекурсивно удалим узел m из дерева поиска.

# 8. Программные продукты как сложные системы. Признаки сложных систем

Накопившийся к настоящему времени опыт разработки программного обеспечения, показывает, что большинство программных продуктов представляют собой сложные системы. Быть сложным, значит включать множество частей, аспектов, деталей, понятий, требующих для понимания серьёзного исследования или рассмотрения.

1. Сложные системы являются иерархическими, т. е. состоят из взаимосвязанных подсистем, которые в свою очередь могут быть разделены на подсистемы и т. д. вплоть до самого низкого уровня.

2. Выбор, какие компоненты в данной системе являются элементарными относительно произволен и в большей степени оставляется на усмотрение исследователя.

3. Внутрикомпонентные связи обычно сильнее, чем связи между компонентами системы. Это различие внутрикомпонентных и межкомпонентных взаимодействий обуславливает разделение функций между частями системы и даёт возможность относительно независимо изучать каждую часть.

4. Иерархические системы обычно состоят из немногих типов подсистем, по-разному скомбинированных и организованных.

Иными словами, разные сложные системы содержат одинаковые структурные части. Эти части могут использовать общие более мелкие компоненты, такие как клетки, или более крупные структуры, типа сосудистых систем, имеющиеся и у растений, и у животных.

5. Любая работающая сложная система является результатом развития работающей более простой системы. В процессе развития системы объекты первоначально рассматривающиеся как сложные становятся элементарными из них строят более сложные системы.

# Декомпозиция. Преимущества и недостатки объектно- ориентирован- ного подхода в программировании

Декомпозиция — разделение целого на части. Также декомпозиция — это научный метод, использующий структуру задачи и позволяющий заменить решение одной большой задачи решением серии меньших задач, пусть и взаимосвязанных, но более простых.

Декомпозиция, как процесс расчленения, позволяет рассматривать любую исследуемую систему как сложную, состоящую из отдельных взаимосвязанных подсистем, которые, в свою очередь, также могут быть расчленены на части. В качестве систем могут выступать не только материальные объекты, но и процессы, явления и понятия.

Преимущества объектно-ориентированного подхода:

· Классы позволяют проводить конструирование из полезных компонент, обладающих простыми инструментами, что дает возможность абстрагироваться от деталей реализации.

· Данные и операции вместе образуют определенную сущность и они не «размазываются» по всей программе, как это нередко бывает в случае процедурного программирования.

· Локализация кода и данных улучшает наглядность и удобство сопровождения программного обеспечения.

· Инкапсуляция информации защищает наиболее критичные данные от несанкционированного доступа.

· Мы сокращаем время на разработку, которое с выгодой может быть отдано другим проектам.

· Компоненты многоразового использования обычно содержат гораздо меньше ошибок, чем вновь разработанные, ведь они уже не раз подвергались проверке.

· Когда некая компонента используется сразу несколькими клиентами, то улучшения, вносимые в ее код, одновременно оказывают свое положительное влияние и на множество работающих с ней программ.

· Если программа опирается на стандартные компоненты, то ее структура и пользовательский интерфейс становятся более унифицированными, что облегчает ее понимание и упрощает ее использование.

· Объектная модель вполне естественна, поскольку в первую очередь ориентирована на человеческое восприятие мира, а не на компьютерную реализацию.

Недостатки:

(1) Необходимо понимать базовые концепции, такие как классы, наследование и динамическое связывание. Для программистов, уже знакомых с понятием модуля и с абстрактными типами данных, это потребует минимальных усилий. Для тех же, кто никогда не использовал инкапсуляцию данных, это может означать изменения мировоззрения и может отнять на изучение значительное количество времени.

(2) Многоразовое использование требует от программиста познакомиться с большими библиотеками классов. А это может оказаться сложнее, чем даже изучение нового языка программирования. Библиотека классов фактически представляет собой виртуальный язык, который может включать в себя сотни типов и тысячи операций. В языке Smalltalk, к примеру, до того, как перейти к практическому программированию, нужно изучить значительную часть его библиотек и классов. А это тоже требует времени.

(3) Проектирование классов — задача куда более сложная, чем их использование. Проектирование класса, как и проектирование языка, требует большого опыта. Это итеративный процесс, где приходится учиться на своих же ошибках.

# 9. Основные понятия объектно-ориентированного программирования: объект, класс, виды отношений между классами

**Класс** – тип данных, описывающий структуру и поведение объектов.

**Объект** – экземпляр класса.

**Ассоциация** означает, что объекты двух классов могут ссылаться один на другой, иметь некоторую связь между друг другом.

**Агрегация** – отношение, когда один объект является частью другого. Объект, являющий частью может существовать отдельно.

**Композиция** - более сильная связь между классами, чем агрегация. Между агрегацией и композицией довольно тонкая грань. Особенностью композиции является то, что объекты, из которых создаётся композиция, могут принадлежать только классу, с которым они образуют композицию. При этом время жизни объекта и класса, в который встраивается объект, совпадает.

**Обобщение/расширение (наследование).** Диаграмма классов, показывающая наследование двух подклассов от одного суперкласса. Обобщение показывает, что один из двух связанных классов (подтип) является частной формой другого (надтипа), который называется обобщением первого. На практике это означает, что любой экземпляр подтипа является также экземпляром надтипа.

# 10. Основные принципы объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм

**Инкапсуляция** - это механизм, который объединяет данные и код, манипулирующий этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. Инкапсуляция используется для управления скрытием данных от неправильного использования.

**Полиморфизм** - это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Полиморфный метод будет иметь одинаковое имя, но разные типы параметров и будет осуществлять логически одинаковые действия над ними и достигать одной цели.

**Наследование** — механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса или интерфейса. Потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами. Позволяет строить иерархии.

# 11. Общая характеристика объектов в объектно-ориентированном программировании. Объектная декомпозиция. Виды отношений между объектами

**Общая характеристика объектов в объектно-ориентированном программировании.** Объект в ООП — это сущность, способная сохранять свое состояние (информацию) и обеспечивающая набор операций (поведение) для проверки и изменения этого состояния. Объект в объектно-ориентированном программировании — это модель или абстракция реальной сущности в программной системе. Предмет моделирования при построении объекта в ООП может быть любым.

Каждый объект в ООП характеризуется своим состоянием. Состояние объекта характеризуется текущим значением его атрибутов. Стоит заметить, что атрибутами объекта в ООП могут быть не только простейшие значения (число, логическое значение и т.д.), но и сложные величины или другие объекты.

Важнейшей характеристикой объекта в ООП является описание того, как он может взаимодействовать с окружающим миром. Это описание называется интерфейсом объекта.

Объекты в ООП взаимодействуют между собой с помощью сообщений. Принимая сообщение, объект выполняет соответствующее действие. Эти действия обычно называются методами.

**Объектная декомпозиция.** Объектной декомпозицией называют процесс представления предметной области задачи в виде совокупности функциональных элементов (объектов), обменивающихся в процессе выполнения программы входными воздействиями (сообщениями). Каждый выделяемый объект предметной области отвечает за выполнение некоторых действий, зависящих от полученных сообщений и параметров самого объекта. Совокупность значений параметров объекта называют его состоянием, а совокупность реакций на получаемые сообщения — поведением. Параметры состояния и элементы поведения объектов определяются условием задачи. В процессе решения задачи объект, получив некоторое сообщение, выполняет заранее определенные действия, например, может изменить собственное состояние, выполнить некоторые вычисления, нарисовать окно или график и, в свою очередь, сформировать сообщения другим объектам. Таким образом, процессом решения задачи управляет последовательность сообщений. Передавая эти сообщения от объекта к объекту, программа выполняет необходимые действия.

**Ассоциация.** Ассоциации обеспечивают взаимодействия объектов, принадлежащих разным классам. Ассоциация обозначает только семантическую связь. Она не указывает направление и точную реализацию отношения. Ассоциация пригодна для анализа проблемы, когда нам требуется лишь идентифицировать связи. 1 – 1. 1 – n. n – n.

**Агрегация** – связь часть – целое. Уничтожение владельца не приводит к уничтожению части.

**Композиция** – связь часть – целое. Уничтожение владельца приводит к уничтожению часть.

**Реализация, обобщение/наследование**

**Зависимость** — это слабая форма отношения использования, при котором изменение в спецификации одного влечёт за собой изменение другого, причём обратное не обязательно. Возникает, когда объект выступает, например, в форме параметра или локальной переменной.

# 13. Понятие «базы данных». Основные компоненты базы данных

БД – совокупность данных, организованных по определённым правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ. Эти данные относятся к определённой предметной области и организованы таким образом, что могут быть использованы для решения многих задач многими пользователями.

Широкое распространение получили БД, имеющие табличную структуру. Базы данных, имеющие связанные таблицы, называют *реляционными* базами данных. Координация осуществляется путем установления связей между таблицами. В объектно-ориентированных БД данные с различными отношениями рассматриваются как единый объект. Поэтому разработчик может не беспокоиться о связывании и разделении данных.

Основные компоненты:

Реляционная база данных состоит из множества таблиц. Структуру каждой таблицы разрабатывают отдельно. В этих таблицах столбцы называются *полями* или *атрибутами*, а строки - *записями* или *кортежами*. Поля образуют структуру базы данных, а записи составляют информацию, которая в ней содержится.

Для того чтобы связи между таблицами работали надежно, и по записи из одной таблицы можно было однозначно найти записи в другой таблице, надо предусмотреть в таблице *уникальные поля (идентификаторы)*.

Уникальное поле - это поле, значения в котором не могут повторяться. Если ни одно поле таблицы не приемлемо в качестве уникального, его можно создать искусственно. Кроме этого, существует понятие *ключевого поля*. При создании структуры таблиц одно поле (или одну комбинацию полей) можно назначить ключевым. С ключевыми полями СУБД работает особо. Она проверяет их уникальность и быстрее выполняет сортировку по таким полям. Ключевое поле - очевидный кандидат для создания связей. Иногда ключевое поле называют *первичным ключом*.

В реляционных базах данных важную роль играет еще один ключ - *внешний ключ*. Внешний ключ - это поле одной таблицы, которое ссылается на первичный ключ другой таблицы. Существует три разновидности связей между таблицами базы данных:

- «один-ко-многим»,

- «один-к-одному»,

- «многие-ко-многим».

В качестве первичного ключа в таблицах часто используют поле, имеющее тип "Счетчик". Ввести два одинаковых значения в такое поле нельзя по определению, поскольку приращение значения поля производится автоматически. Структура связей между таблицами называется *схемой данных*.

*Индекс* - это средство автоматической сортировки записей в таблице по значению индексируемого поля. Существует два вида индексов: допускающие и не допускающие повторение значений поля. Почти так же, как и по одному полю, индекс можно составить по двум и более полям. Применение индексов значительно ускоряет просмотр и выборку данных. Недостатки индексов - каждый из них занимает дополнительное место на жестком диске, а также в оперативной памяти. Кроме этого, увеличивается время добавления или удаления записи (особенно если в таблице содержится большое количество записей), поскольку индексы приходится создавать заново. Хотя эффект снижения быстродействия здесь не так важен, как эффект ускорения операций поиска и сортировки, все же следует следить за тем, чтобы не индексировать в таблице слишком много полей.

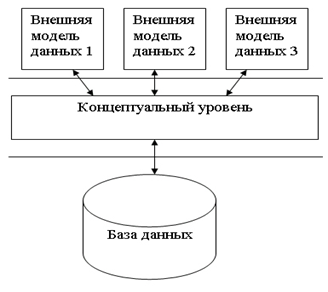
# 14. Системы баз данных. Архитектура систем баз данных. Современные СУБД.

Понятие **система баз данных** используется как в широком, так и в узком смысле.

В *широком смысле* система баз данных понимается фактически как синоним понятия [информационная система](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) и включает в себя [данные](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5), [аппаратное обеспечение](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5),[программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и [пользователей](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C).

В *узком смысле* система баз данных понимается как [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94) с управляемой ею [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), возможно, уже наполненной. Систе́ма управле́ния ба́зами да́нных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Трёхуровневая архитектура.



*Трехуровневая модель системы управления базой данных, предложенная ANSI*

**- Уровень внешних моделей** - самый верхний уровень, где каждая модель имеет свое «видение» данных. Этот уровень определяет точку зрения на БД отдельных приложений. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению.

**- Концептуальный уровень** - центральное управляющее звено, здесь база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.

**- Физический уровень** - собственно данные, расположенные в файлах или в страничных структурах, расположенных на внешних носителях информации.

Эта архитектура позволяет обеспечить логическую (между уровнями 1 и 2) и физическую (между уровнями 2 и 3) независимость при работе с данными. Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же базой данных. Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с данной базой данных.

**Современные СУБД.**

Обычно современная СУБД содержит следующие компоненты:

·**ядро**, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию,

·**процессор языка базы данных**, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода,

·**подсистему поддержки времени исполнения**, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД

· а также **сервисные программы** (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

К современным СУБД предъявляются следующие требования:

·масштабируемость — отсутствие существенного снижения скорости выполнения пользовательских запросов при пропорциональном росте количества запросов и аппаратных ресурсов используемых данной СУБД (таких как объем оперативной памяти, количество процессоров и серверов);

·доступность — возможность всегда выполнить запрос;

·надежность — минимальная вероятность сбоев, наличие средств восстановления данных после сбоев, инструментов резервного копирования и дублирования данных (в последнее время производители СУБД нередко предлагают инструменты, позволяющие осуществлять подобные операции, не прерывая работу пользователей);

· управляемость — простота администрирования, наличие средств автоматического конфигурирования (типичный современный набор средств администрирования включает средства создания баз данных и их объектов, инструменты описания правил репликации данных между различными серверами, утилиты управления пользователями, группами и их правами, средства мониторинга событий, средства просмотра планов выполнения запросов, утилиты миграции из других СУБД);

·наличие средств защиты данных от потери и несанкционированного доступа;

·поддержка доступа к данным с помощью Web-служб;

·поддержка стандартных механизмов доступа к данным (таких как ODBC, JDBC, OLE DB, ADO .NET), позволяющая создавать приложения для СУБД с помощью различных средств разработки.

Несоответствие СУБД какому-либо из этих требований приводит к тому, что даже у неплохой по другим потребительским свойствам СУБД область применения оказывается весьма ограниченной.

# 22. Процессы и потоки в многозадачной операционной системе. Алгоритмы планирования и управления.

# 23. Управление памятью в операционной системе. Физическая память. Виртуальная память. Адресное пространство. Алгоритмы замещения страниц памяти.

# 24. Межпроцессное взаимодействие. Механизмы взаимодействия процессов.

**Межпроцессное взаимодействие** (***i****nter-****p****rocess* ***c****ommunication*, **IPC**) — обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети.

Из механизмов, предоставляемых ОС и используемых для IPC, можно выделить:

- механизмы обмена сообщениями;

- механизмы синхронизации;

- механизмы разделения памяти;

- механизмы удалённых вызовов (RPC).

Для оценки производительности различных механизмов IPC используют следующие параметры:

- пропускная способность (количество сообщений в единицу времени, которое ядро ОС или процесс способна обработать);

- задержки (время между отправкой сообщения одним потоком и его получением другим потоком).

IPC может называться терминами ***межпотоковое взаимодействие*** (***i****nter-****t****hread* ***c****ommunication*) и ***межпрограммное взаимодействие*** (***i****nter-****a****pplication****c****ommunication*).

**Механизмы взаимодействия процессов:**

**1. Каналы**

Типичным механизмом обмена данными для процессов является использование потоков: стандартного потока ввода, вывода, ошибок, файловых потоков и т.п. Стандартные потоки ввода, вывода и ошибок, по сути, являются *каналами*, и такие каналы можно или создавать между родственными процессами (*безымянные каналы*), или привязывать к узлам файловой системы, позволяя ими пользоваться любым процессам, которые имеют к ним доступ (*именованные каналы*).

*Полудуплексным каналом* называется системный буфер, очередь, представляемая парой дескрипторов. Один из этих дескрипторов доступен на запись и позволяет помещать данные в канал, другой — доступен на чтение и позволяет получать данные из канала. Так как открытые дескрипторы сохраняются при использовании функции [fork](http://pubs.opengroup.org/onlinepubs/009695399/functions/fork.html), каналы можно использовать для обмена данными между родственными процессами.

**2. Сигналы**

Сигналы являются программными прерываниями, которые посылаются процессу, когда случается некоторое событие. Сигналы могут возникать синхронно с ошибкой в приложении. Сигналы могут посылаться процессу, если система обнаруживает программное событие, например, когда пользователь дает команду прервать или остановить выполнение, или получен сигнал на завершение от другого процесса. Сигналы могут прийти непосредственно от ядра ОС, когда возникает сбой аппаратных средств ЭВМ. Система определяет набор сигналов, которые могут быть отправлены процессу. При этом каждый сигнал имеет целочисленное значение и приводит к строго определенным действиям.

Известно три варианта реакции на сигналы:

1. вызов собственной функции обработки;

2. игнорирование сигнала;

3. использование предварительно установленной функции обработки по умолчанию.

Чтобы реагировать на разные сигналы, необходимо знать концепции их обработки. Процесс должен организовать так называемый обработчик сигнала в случае его прихода.

**3. Очереди сообщений**

Очереди сообщений как средство межпроцессной связи позволяют процессам взаимодействовать, обмениваясь данными. Данные передаются между процессами дискретными порциями, называемыми сообщениями. Процессы, использующие этот тип межпроцессной связи, могут выполнять две операции: *послать или принять сообщение*.

Процесс, прежде чем послать или принять какое-либо сообщение, должен запросить систему породить программные механизмы, необходимые для обработки данных операций.

Процессы, имеющие права на операции и пытающиеся послать или принять сообщение, *могут приостанавливаться*, если выполнение операции не было успешным. В частности это означает, что процесс, пытающийся послать сообщение, может ожидать, пока процесс-получатель не будет готов; и наоборот, получатель может ждать отправителя. Если указано, что процесс в таких ситуациях должен приостанавливаться, говорят о выполнении над сообщением «*операции с блокировкой*». Если приостанавливать процесс нельзя, говорят, что над сообщением выполняется ''*операция без блокировки*''.

**4. Семафоры**

Семафоры являются одним из классических примитивов синхронизации. Семафор (semaphore) - это целая переменная, значение которой можно опрашивать и менять только при помощи неделимых (атомарных) операций.

Двоичный семафор может принимать только значения 0 или 1. Считающий семафор может принимать целые неотрицательные значения.

В приложениях как правило требуется использование более одного семафора, ОС должна представлять возможность создавать множества семафоров.

Над каждым семафором, принадлежащим некоторому множеству, можно выполнить любую из трех операций:

- увеличить значение;

- уменьшить значение;

- дождаться обнуления.

Для выполнения первых двух операций у процесса должно быть право на изменение, для выполнения третьей достаточно права на чтение. Для третьей операции нужно передать 0; если текущее значение семафора отлично от нуля, операция не может быть успешно выполнена.

**5. Разделяемая память**

Разделяемая память может быть наилучшим образом описана как отображение участка (сегмента) памяти, которая будет разделена между более чем одним процессом. Это гораздо более быстрая форма IPC, потому что здесь нет никакого посредничества (т.е. каналов, очередей сообщений и т.п.). Вместо этого, информация отображается непосредственно из сегмента памяти в адресное пространство вызывающего процесса. Сегмент может быть создан одним процессом и впоследствии использован для чтения/записи любым количеством процессов.

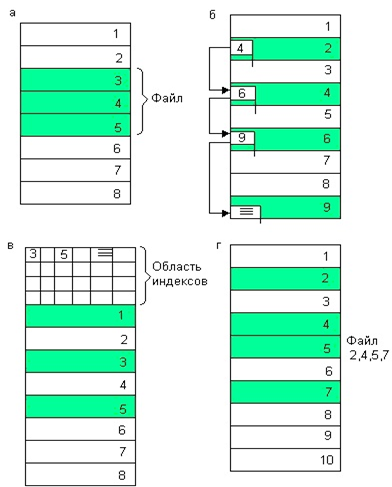
**6. Удалённый вызов процедур**, реже Вызов удалённых процедур (от *Remote Procedure Call, RPC*) — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве (как правило, на удалённых компьютерах). Обычно, реализация RPC технологии включает в себя два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов (или структур, для необъектных RPC). Различные реализации RPC имеют очень отличающуюся друг от друга архитектуру и разнятся в своих возможностях. На транспортном уровне RPC используют в основном протоколы TCP и UDP, однако, некоторые построены на основе HTTP (что нарушает архитектуру ISO/OSI, так как HTTP изначально не транспортный протокол).

# 25. Файловые системы. Основные особенности и физическая организация на примере одной из современных файловых систем.

**Фа́йловая систе́ма** (*file system*) — порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов и (каталогов), максимальный возможный размер файла и раздела, набор атрибутов файла. Некоторые файловые системы предоставляют сервисные возможности, например, разграничение доступа или шифрование файлов.

Файловая система связывает носитель информации с одной стороны и API для доступа к файлам— с другой. Именно файловая система устанавливает, где и как будет записан файл на физическом носителе (например, жёстком диске).

В общем случае жесткий диск состоит из набора пластин, покрытых магнитным слоем.



Непрерывное размещение — простейший вариант физической организации *(рисунок (а))*, при котором файлу предоставляется последовательность кластеров диска, образующих непрерывный участок дисковой памяти.

Следующий способ физической организации — размещение файла в виде связанного списка кластеров дисковой памяти *(рисунок (б))*. При таком способе в начале каждого кластера содержится указатель на следующий кластер. В этом случае адресная информация минимальна: расположение файла может быть задано одним числом — номером первого кластера.

Популярным способом, применяемым, например, в файловой системе FAT, является использование связанного списка индексов *(рисунок (в)).* Этот способ является некоторой модификацией предыдущего. Файлу также выделяется память в виде связанного списка кластеров. Номер первого кластера запоминается в записи каталога, где хранятся характеристики этого файла. Остальная адресная информация отделена от кластеров файла. С каждым кластером диска связывается некоторый элемент — индекс. Индексы располагаются в отдельной области диска.

Еще один способ задания физического расположения файла заключается в простом перечислении номеров кластеров, занимаемых этим файлом *(рисунок (г)).* Этот перечень и служит адресом файла. Недостаток данного способа очевиден: длина адреса зависит от размера файла и для большого файла может составить значительную величину. Достоинством же является высокая скорость доступа к произвольному кластеру файла, так как здесь применяется прямая адресация, которая исключает просмотр цепочки указателей при поиске адреса произвольного кластера файла. Фрагментация на уровне кластеров в этом способе также отсутствует.

**Пример:**

NTFS превосходно справляется с обработкой больших массивов данных и достаточно хорошо проявляет себя при работе с томами объемом от 300-400 Мбайт и выше. Максимально возможные размеры тома и файла составляют 16ЭБ (220ТБ). Количество файлов в каталогах не ограничено. В основу структуры каталогов NTFS заложено B-Tree.

Как и многие другие системы, NTFS делит все полезное дисковое пространство тома на кластеры - блоки данных, адресуемые как единицы данных. NTFS поддерживает размеры кластеров от 512Б до 64КБ; стандартом же считается кластер размером 2 или 4 КБ.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M  F  T | З  о  н  а  M  F  T | Зона для размещения  файлов и каталогов | К  о  п  и  я | Зона для размещения  файлов и каталогов |

**Структура тома NTFS**

Все дисковое пространство в NTFS делится на две неравные части. Первые 12% диска отводятся под так называемую MFT-зону - пространство, которое может занимать, увеличиваясь в размере, главный служебный *метафайл* MFT. MFT (matrer file table) - это специальный файл, главная системная структура данных, которая и позволяет определять местонахождение всех остальных файлов. MFT поделен на записи фиксированного размера 1КБ (минимум 1КБ и максимум 4КБ), и каждая запись соответствует какому-либо файлу (в общем смысле этого слова). Первые 16 файлов носят служебный характер и недоступны ОС - они называются *метафайлами*, причем самый первый метафайл - сам MFT. Эти 16 элементов MFT - единственная часть диска, имеющая строго фиксированное положение. Копия первых 16 записей хранится в середине тома для надежности. Каждый из упомянутых первых 16 файлов NTFS отвечает за какой-либо аспект работы системы и находится в корневом каталоге NTFS-тома. Все они начинаются с символа "$".

# 26. Каналы передачи данных. Физический канал. Логический канал. Понятие блока данных. Пример формата блока данных любого протокола

**Канал передачи данных** - это средства двухстороннего обмена данными, которые включают в себя линии связи и аппаратуру передачи (приема) данных. Каналы передачи данных связывают между собой источники информации и приемники информации.

Каналы связи (data link) создаются по линиям связи при помощи сетевого оборудования и физических средств связи. **Физические** средства связи построены на основе витых пар, коаксиальных кабелей, оптических каналов или эфира. Между взаимодействующими информационными системами через физические каналы коммуникационной сети и узлы коммутации устанавливаются логические каналы. **Логический канал** – это путь для передачи данных от одной системы к другой. Логический канал прокладывается по маршруту в одном или нескольких физических каналах. Логический канал можно охарактеризовать как маршрут, проложенный через физические каналы и узлы коммутации.

Информация в сети передается блоками (пакетами) данных по процедурам обмена между объектами. Эти процедуры называют протоколами передачи данных.

Пакет - основная единица информации в компьютерных сетях. При разбиении данных на пакеты скорость их передачи возрастает настолько, что каждый компьютер в сети получает возможность принимать и передавать данные практически одновременно с остальными компьютерами. На целевом компьютере (компьютере-получателе) пакеты накапливаются и выстраиваются в должном порядке для восстановления исходного вида данных.



Пример 2. Пакет HTTP:

Каждое HTTP-сообщение состоит из трех частей, которые передаются в следующем порядке:

1. Стартовая строка – определяет тип сообщения; Стартовые строки HTTP-сообщения различаются для запроса и ответа.

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

3. Тело сообщения – непосредственно данные сообщения.

# 27. Структуризация сетей. Понятие и характеристики основных сетевых топологий. Структурообразующие аппаратные средства и программное обеспечение

**Сетевая тополо́гия** — это конфигурация графа, вершинам которого соответствуют конечные узлы сети (компьютеры) и коммуникационное оборудование (маршрутизаторы), а рёбрам — физические или информационные связи между вершинами.

**Основные сетевые топологии**

**Полносвязная**

Сеть, в которой каждый компьютер непосредственно связан со всеми остальными. Однако этот вариант громоздкий и неэффективный, потому что каждый компьютер в сети должен иметь большое количество коммуникационных портов, достаточное для связи с каждым из остальных компьютеров.

**Неполносвязная**

Неполносвязных топологий существует несколько. В них, в отличие от полносвязных может применяться передача данных не напрямую между компьютерами, а через дополнительные узлы.

**Шина (Bus)**

Топология данного типа, представляет собой общий кабель (называемый шина или магистраль), к которому подсоединены все рабочие станции. На концах кабеля находятся терминаторы, для предотвращения отражения сигнала.

Преимущества сетей шинной топологии:

· расход кабеля существенно уменьшен

· отказ одного из узлов не влияет на работу сети в целом;

· сеть легко настраивать и конфигурировать;

· сеть устойчива к неисправностям отдельных узлов.

Недостатки сетей шинной топологии:

· разрыв кабеля может повлиять на работу всей сети;

· ограниченная длина кабеля и количество рабочих станций;

· недостаточная надежность сети из-за проблем с разъемами кабеля.

· низкая производительность, обусловлена разделением канала между всеми абонентами.

**Звезда**

В сети построенной по топологии типа «звезда» каждая рабочая станция подсоединяется кабелем (витой парой) к концентратору или хабу ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *hub*). Концентратор обеспечивает параллельное соединение ПК и, таким образом, все компьютеры, подключенные к сети, могут общаться друг с другом.

Преимущества сетей топологии звезда:

· легко подключить новый ПК;

· имеется возможность централизованного управления;

· сеть устойчива к неисправностям отдельных ПК и к разрывам соединения отдельных ПК.

Недостатки сетей топологии звезда:

· отказ хаба влияет на работу всей сети;

· большой расход кабеля.

Звезда – централизованная топология, так же встречаются и децентрализованные топологии. Децентрализация: в сетевой топологии существуют по крайней мере два узла с двумя или больше путями между ними, чтобы обеспечить дополнительные пути, которые будут использоваться в случае, если один из путей выйдет из строя. Эта децентрализация часто используется, чтобы компенсировать недостаток выхода из строя одного пункта, используя единственное устройство в качестве центрального узла (например, в звезде и сетях дерева). Специальный вид сети, ограничивающий число путей между двумя узлами, называется гиперкубом.

**Кольцо**

В сети с топологией кольцо все узлы соединены каналами связи в неразрывное кольцо (необязательно окружность), по которому передаются данные. Выход одного ПК соединяется со входом другого ПК. Начав движение из одной точки, данные, в конечном счете, попадают на его начало. Данные в кольце всегда движутся в одном и том же направлении.

Принимающая рабочая станция распознает и получает только адресованное ей сообщение. В сети с топологией типа физическое кольцо используется маркерный доступ, который предоставляет станции право на использование кольца в определенном порядке. Логическая топология данной сети — логическое кольцо. Данную сеть очень легко создавать и настраивать.

К основному недостатку сетей топологии кольцо является то, что повреждение линии связи в одном месте или отказ ПК приводит к неработоспособности всей сети.

Как правило, в чистом виде топология «кольцо» не применяется из-за своей ненадёжности, поэтому на практике применяются различные модификации кольцевой топологии.

**Ячеистая топология**

Получается из полносвязной топологии путём удаления некоторых связей. Допускает соединения большого количества компьютеров и характерна для крупных сетей.

Дополнительные способы являются комбинациями базовых. В общем случае такие топологии называются смешанными или гибридными, но некоторые из них имеют собственные названия, например «Дерево».

**Смешанная топология** — сетевая топология, преобладающая в крупных сетях с произвольными связями между компьютерами. В таких сетях можно выделить отдельные произвольно связанные фрагменты (*подсети*), имеющие типовую топологию, поэтому их называют сетями со смешанной топологией.

**ТЕХНИЧЕСКИЕ СРЕДСТВА СЕТЕЙ**

Технические средства локальных сетей включают в себя следующие функциональные группы оборудования:

1) Средства линий передачи данных (кабель, витая пара, оптоволокно) – реализуют собственно перенос сигналов;

2) Средства увеличения дистанции передачи данных (репитер, усилитель, модемы) – осуществляют усиление сигнала или преобразование в форму, удобную для дальнейшей передачи;

3) Средства повышения ёмкости линий передачи (мульплексоры) – позволяют реализовывать несколько логических каналов в рамках одного физического соединения путём разделения частот передачи, чередования пакетов во времени и т.д.

4) Средства управления информационными потоками в сети (коммутация каналов, коммутация пакетов, разветвление линий передачи) – осуществляют адресацию сообщений;

5) Средства соединения линий передачи с сетевым оборудованием узлов (сетевые платы, адаптеры) Предназначены для: взаимодействия с другими устройствами в локальной сети; повышения производительности, назначают приоритеты для ответственного трафика, поддерживают удалённую активизацию связи с центральной рабочей станцией, поддерживают удалённое изменение конфигурации, что значительно экономит время и силы администраторов постоянно растущих сетей.

КОНЦЕНТРАТОРЫ

Концентраторы (многопортовые повторители, hub) осуществляют функции повторителей сигналов на всех участках передачи данных. Обнаруживают коллизии в сегменте и посылают джам-последовательности на все свои выходы.

В зависимости от числа рабочих станций применяют пассивные и активные концентраторы. Активные hub содержат усилитель для подключения от 4 до 32 рабочих станций, пассивный являются исключительно разветвительным устройством (максимум на три рабочие станции).

ПОВТОРИТЕЛЬ

Повторитель (репитер) принимает сигналы от одного из компьютеров, а передаёт их синхронно на все остальные порты, кроме того, с которого поступил сигнал. Если длина сети превышает максимальную длину сегмента сети, необходимо разбить сеть на несколько (до 5) сегментов), соединив их через репитер.

Функции репитера заключаются в физическом разделении сегментов сети и обеспечении восстановления пакетов, передаваемых из одного сегмента сети в другой

КОММУТАТОР

Коммутатор (Switch) – многопортовое устройство, обеспечивающее высокочастотную коммутацию пакетов между портами.

Он позволяет передавать пакеты сообщений одновременно между парами портов в сетях Ethernet. Использует алгоритм, «прозрачного сегмента». В начальный момент времени коммутатор ничего не знает о чужых подключаемых узлах или сегмента сети его портах.

По мере того, как подключённые к портам коммутаторы узлы начинают передачу, он начинает анализировать содержимое адресов отправителя, что позволяет делать выводы о принадлежности того или иного узла к тому или иному порту коммутатора.

МАРШРУТИЗАТОР

Маршрутизатор имеет несколько портов, к которым подсоединяются подсети. Каждый порт маршрутизатора можно рассматривать как отдельный узел сети. Он имеет собственный сетевой и локальный адреса к той подсети, которая к нему подключена. Маршрутизатор можно рассматривать как совокупность нескольких узлов, каждый из которых входит в свою подсеть.

Но его нельзя рассматривать как единое устройство, так как оно не имеет отдельного сетевого и локального адресов.

Маршрут – последовательность маршрутизаторов, которые должен пройти от отправителя до пункта назначения. Задачу выбора маршрута из нескольких возможных маршрутов решают маршрутизаторы.

Чтобы маршрут был рациональным, каждый конечный предел и маршрутизатор составной сети анализируют специальную структуру, таблицу маршрутов

Основные функции маршрутизаторов:

1) Чтение заголовков пакетов, сетевых протоколов, которые принимаются в буфер по каждому порту маршрутизатора;

2) Принятие решений о дальнейшем маршруте следования;

3) Подключение локальных сетей к территориально-распределённым сетям.

Операционная система сети включает в себя набор управляющих и обслуживающих программ, обеспечивающих:

· межпрограммный метод доступа (возможность организации связи между отдельными прикладными программами комплекса, реализуемыми в различных узлах сети);

· доступ отдельных прикладных программ к ресурсам сети (и в первую очередь к устройствам ввода-вывода);

· синхронизацию работы прикладных программных средств в условиях их обращения к одному и тому же вычислительному ресурсу;

· обмен информацией между программами с использованием сетевых "почтовых ящиков";

· выполнение команд оператора с терминала, подключенного к одному из узлов сети, на каком-либо устройстве, подключенном к другому удаленному узлу вычислительной сети;

· удаленный ввод заданий, вводимых с любого терминала, и их выполнение на любой ЭВМ в пакетном или оперативном режиме;

· обмен наборами данных (файлами) между ЭВМ сети;

· доступ к файлам, хранимым в удаленных ЭВМ, и обработку этих файлов;

·защиту данных и вычислительных ресурсов сети от несанкционированного доступа;

·выдачу различного рода справок об использовании информационных, программных и технических ресурсов сети;

·передачу текстовых сообщений с одного терминала пользователя на другие (электронная почта).

Отдельным видом сетевого программного обеспечения является драйвер:

Драйвер (англ. driver) (множественное число драйверы) - это компьютерная программа, с помощью которой другая программа (обычно операционная система) получает доступ к аппаратному обеспечению некоторого устройства. В общем случае, для использования любого сетевого устройства (как внешнего, так и внутреннего) необходим драйвер.

# 28. Понятие адресации в сетях. Типы адресов. Адресация в IP-сетях

Для того, чтобы компьютеры могли идентифицировать друг друга в информационно-вычислительной сети, им присваиваются явные адреса. Основными типами адресов являются следующие:

* MAC-адрес (физический адрес);
* IP-адрес (сетевой адрес);
* доменный адрес (символьные имена, DNS-имена);

Для обмена данными в Интернете (между различными локальными сетями) узлу необходим IP-адрес. Это логический сетевой адрес конкретного узла. Для обмена данными с другими устройствами, подключенными к Интернету, необходим правильно настроенный, уникальный IP-адрес. IP-адрес присваивается сетевому интерфейсу узла.

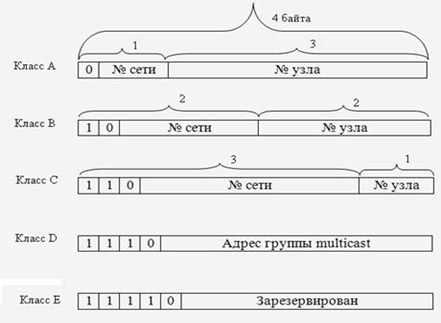
IP-адрес – представляет собой основной тип адресов, на основании которых сетевой уровень передаёт пакеты между сетями. Эти адреса состоят из 4 байт. Назначаются администратором во время конфигурирования компьютеров и маршрутизатора. Он состоит из двух частей:

а) Номер сети – выбирается администратором произвольно или назначается службой InterNic;

б) Номер узла в сети – назначается независимо от локального адреса узла.

Маршрутизатор имеет столько адресов, сколько сетевых связей.

Классы IP-адресов:



Если адрес начинается с 0, то сеть относят к классу А и номер сети занимает 1 байт, номер узла 3 байта. Сети класса А имеют номера в диапазоне от 1 до 126. Таких сетей немного, зато количество узлов в них может достигать 224 .

Если первые два бита равны 10, то сеть относится к классу В. Является сетью средних размеров, максимальное количество узлов в которой равняется 216 .

Если адрес начинается последовательностью 110, то он относится к классу С, количество узлов в котором равняется28 .

Если адрес начинается последовательностью 1110, то это сеть класса D. Она означает групповой адрес—Multicast. Если в пакете в качестве адреса назначения указан адрес класса D, то такой пакет получают все узлы, которым присвоен данный адрес.

Если адрес начинается с 11110, то эта сеть относится к классу Е. Адреса этого класса зарезервированы для будущего применения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Первые биты** | **Наименьший номер сети** | **Наибольший номер сети** | **Максимальное число узлов в сети** |
| A | 0 | 1.0.0.0 | 126.0.0.0 | 224 |
| B | 10 | 128.0.0.0 | 191.255.0.0 | 216 |
| C | 110 | 192.0.1.0 | 223.255.255.0 | 28 |
| D | 1110 | 224.0.0.0 | 239.255.255.255 | Multicast |
| E | 11110 | 240.0.0.0 | 247.255.555.555 | Зарезервирован |

ИСПОЛЬЗОВАНИЕ МАСОК В IP-АДРЕСАЦИИ

Маска – число, которое используется в паре с IP адресом, двоичная запись маски содержит единицы в тех разделах, которые должны в IP адресе интерпретироваться как номер сети. Поскольку номер сети является цельной частью адреса, 1 в маске представляют непрерывную последовательность.

Для стандартных классов сетей маски имеют след. значения:

1) Класс А – 11111111.00000000.00000000.00000000 (255.0.0.0)

2) Класс В – 11111111.11111111.00000000.00000000 (255.255.0.0)

ОПРЕДЕЛЕНИЕ IP-АДРЕСА

Для определения локального адреса по IР адресу используется протокол разрешения (ARP). Существует также протокол, решающий обратную задачу – нахождение IP адреса по известному локальному адресу. Он называется реверсивный ARP (RARP).

Работа протокола ARP начинается с просмотра ARP-таблицы. Каждая строка таблицы устанавливает соответствие между IP адресом и локальным адресом. Для каждой сети, подключённой к сетевому адаптеру или к порту маршрутизатора, строится отдельная таблица.

Статические записи создаются вручную с помощью утилиты ARP и не имеет срока устаревания (существует до тех пор, пока компьютер или маршрутизатор не будут выключены). Динамические записи создаются модулем протокола ARP и периодически обновляются. Если в течение нескольких минут запись не обновляется, то она исключается из таблицы (ARP-кэш).

# 29. Характеристика протоколов IP, TCP, ARP, ICMP, POP3, SMTP

**Протокол IP**

IP компонента TCP/IP определяет, куда пакеты данных будут направлены, основываясь на адресе источника назначения, у IP есть определенные характеристики, связанные с тем, как работает эта функция.

IP использует пакеты для передачи информации через сеть. Пакет - независимый контейнер, содержащий достаточное количество информации, для передачи его от источника до точки назначения без необходимости предварительной обработки.

IP характеризуется следующим:

·IP работает на третьем уровне модели OSI и стека TCP/IP.

·IP - протокол, работающий без установки соединения, в котором данные отправляются к точке назначения без дополнительного уведомления точки назначения. Принимающие устройство получает данные и не возвращает информацию о статусе полученного пакета, отправляющему устройству.

·IP использует иерархическую адресацию, в которой сетевую часть адреса можно представить как название улицы, а хостовую часть адреса, как номер дома на этой улице.

·IP предоставляет доставку пакетов без подтверждения и не гарантирует доставку пакета. Поэтому пакет может быть неправильно направлен, получен дважды или потерян на пути к месту назначения.

·IP не предоставляет специальных возможностей, которые позволили бы восстановить поврежденный пакет. Это обеспечивается оконечными система в сети.

В реальном масштабе времени голосовая или видео передача, без проблем переносит потерю нескольких пакетов; в данном случае наиболее важна скорость, чем гарантированное получение пакетов, потому что восстановление пакетов приведет к задержке по времени.

**Протокол TCP**

Transmission Control Protocol (TCP) (протокол управления передачей) - является обязательным протоколом стандарт TCP/IP, определенный в стандарте RFC 793, "Transmission Control Protocol (TCP)".

TCP — это протокол транспортного уровня, предоставляющий транспортировку (передачу) потока данных, с необходимостью предварительного установления соединения, благодаря чему гарантирует уверенность в целостности получаемых данных, также выполняет повторный запрос данных в случае потери данных или искажения. Помимо этого протокол TCP отслеживает дублирование пакетов и в случае обнаружения - уничтожает дублирующиеся пакеты.

Протокол гарантирует целостность передаваемых данных и подтверждения отправителя о результатах передачи. Используется при передаче файлов, где потеря одного пакета может привести к искажению всего файла.

TCP обеспечивает свою надежность благодаря следующему:

· Данные от приложения разбиваются на блоки определенного размера, которые будут отправлены.

· Когда TCP посылает сегмент, он устанавливает таймер, ожидая, что с удаленного конца придет подтверждение на этот сегмент. Если подтверждение не получено по истечении времени, сегмент передается повторно.

· Когда TCP принимает данные от удаленной стороны соединения, он отправляет подтверждение. Это подтверждение не отправляется немедленно, а обычно задерживается на доли секунды

· TCP осуществляет расчет контрольной суммы для своего заголовка и данных. Это контрольная сумма, рассчитываемая на концах соединения, целью которой является выявить любое изменение данных в процессе передачи. Если сегмент прибывает с неверной контрольной суммой, TCP отбрасывает его и подтверждение не генерируется. (Ожидается, что отправитель отработает тайм-аут и осуществит повторную передачу.)

· Так как TCP сегменты передаются в виде IP датаграмм, а IP датаграммы могут прибывать беспорядочно, также беспорядочно могут прибывать и TCP сегменты. После получения данных TCP может по необходимости изменить их последовательность, в результате приложение получает данные в правильном порядке.

· Так как IP датаграмма может быть продублирована, принимающий TCP должен отбрасывать продублированные данные.

·TCP осуществляет контроль потока данных. Каждая сторона TCP соединения имеет определенное пространство буфера. TCP на принимающей стороне позволяет удаленной стороне посылать данные только в том случае, если получатель может поместить их в буфер. Это предотвращает от переполнения буферов медленных хостов быстрыми хостами.

Перед началом передачи каких-либо данных, согласно протоколу TCP, стороны должны установить соединение.

**Протокол ICMP**

Протокол ICMP - это протокол *сообщения об ошибках*, а не протокол *коррекции ошибок*. Конечный узел может предпринять некоторые действия для того, чтобы ошибка больше не возникала, но эти действия протоколом ICMP не регламентируются.

Каждое сообщение протокола ICMP передается по сети внутри пакета IP. Пакеты IP с сообщениями ICMP маршрутизируются точно так же, как и любые другие пакеты, без приоритетов, поэтому они также могут теряться. Кроме того, в загруженной сети они могут вызывать дополнительную загрузку маршрутизаторов. Для того, чтобы не вызывать лавины сообщения об ошибках, потери пакетов IP, переносящие сообщения ICMP об ошибках, не могут порождать новые сообщения ICMP. Сообщение ICMP всегда содержит заголовок и первые 64 бита данных пакета IP, который вызвал ошибку.

Сообщение, прибывшее в узел-источник, может быть обработано там либо ядром операционной системы, либо протоколами транспортного и прикладного уровней, либо приложениями, либо просто проигнорированы. Важно, что обработка ICMP-сообщений не входит в обязанности протоколов IP и ICMP.

Заметим, что некоторые из пакетов могут исчезнуть в сети, не вызвав при этом никаких оповещений. В частности, протокол ICMP не предусматривает передачу сообщений о проблемах, возникающих при обработке IP-пакетов, несущих ICMP-сообщения об ошибках. Такое решение было принято разработчиками протокола, чтобы не порождать «штормы» в сетях, когда количество сообщений об ошибках лавинообразно возрастает.

**Протокол POP3**

POP3 (англ. Post Office Protocol Version 3 — протокол почтового отделения, версия 3) — стандартный интернет-протокол прикладного уровня, используемый клиентами электронной почты для получения почты с удаленного сервера по TCP/IP-соединению.

POP и IMAP (Internet Message Access Protocol) — наиболее распространённые интернет-протоколы для извлечения почты. Практически все современные клиенты и серверы электронной почты поддерживают оба стандарта. Протокол POP был разработан в нескольких версиях, нынешним стандартом является третья версия (POP3). Большинство поставщиков услуг электронной почты (такие как Hotmail, Gmail и Yahoo! Mail) также поддерживают IMAP и POP3. Предыдущие версии протокола (POP, POP2) устарели.

POP3 поддерживает простые требования «загрузи-и-удали» для доступа к удаленным почтовым ящикам. Хотя большая часть POP-клиентов предоставляют возможность оставить почту на сервере после загрузки, использующие POP клиенты обычно соединяются, извлекают все письма, сохраняют их на пользовательском компьютере как новые сообщения, удаляют их с сервера, после чего разъединяются.

POP3-сервер прослушивает порт 110. Шифрование связи для POP3 запрашивается после запуска протокола. Доступные сообщения клиента фиксируются при открытии почтового ящика POP-сессией и определяются количеством сообщений для сессии, или, по желанию, с помощью уникального идентификатора, присваиваемого сообщению POP-сервером. Этот уникальный идентификатор является постоянным и уникальным для почтового ящика и позволяет клиенту получить доступ к одному и тому же сообщению в разных POP-сессиях. Почта извлекается и помечается для удаления с помощью номера сообщения. При выходе клиента из сессии помеченные сообщения удаляются из почтового ящика.

Команды РОРЗ состоят из ключевых слов, состоящих из ASCII-символов, и одним или несколькими параметрами, отделяемыми друг от друга символом "пробела" - <SP>. Все команды заканчиваются символами "возврата каретки" и "перевода строки" - <CRLF>. Длина ключевых слов не превышает четырех символов, а каждого из аргументов может быть до 40 символов.

Ответы РОРЗ-сервера на команды состоят из строки статус-индикатора, ключевого слова, строки дополнительной информации и символов завершения строки. Длина строки ответа может достигать 512 символов. Строка статус-индикатора принимает два значения: положительное ("+ОК") и отрицательное ("-ERR"). Любой сервер РОРЗ обязан отправлять строки статус-индикатора в верхнем регистре, тогда как другие команды и данные могут приниматься или отправляться как в нижнем, так и в верхнем регистрах.

Ответы РОРЗ-сервера на отдельные команды могут составлять несколько строк. В этом случае строки разделены символами <CRLF>. Последнюю строку информационной группы завершает строка, состоящая из символа "." (код — 046) и <CRLF>, т. е. последовательность "CRLF.CRLF"

Команды протокола POP3:

· USER - идентифицирует пользователя с указанным именем;

· PASS - указывает пароль для пары клиент-сервер;

· QUIT - закрывает TCP-соединение;

· STAT - сервер возвращает количество сообщений в почтовом ящике плюс размер почтового ящика;

· LIST - сервер возвращает идентификаторы сообщений вместе с размерами сообщений;

· RETR - извлекает сообщение из почтового ящика;

· DELE - отмечает сообщение для удаления;

· NOOP - Сервер возвращает положительный ответ, но не совершает никаких действий;

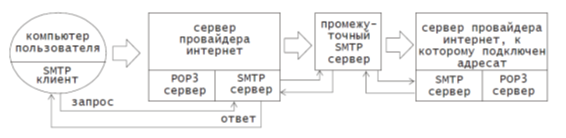
· LAST - Сервер возвращает наибольший номер сообщения из тех, к которым ранее уже обращались;

· RSET - Отменяет удаление сообщения, отмеченного ранее командой DELE.

**Протокол SMTP**

**SMTP (Simple Mail Transfer Protocol)** – широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP.

Упрощенно схема взаимодействия представлена на рисунке ниже (объемными стрелками показано направление движения почтовых сообщений).



Со стороны пользователя обычно одна и та же программа выступает в роли и POP3 клиента, и SMTP клиента отправителя.

На схеме у пользователя есть клиентское ПО, а у провайдера – серверная часть приложения. На самом деле это немного не так. Протокол SMTP делает возможным смену сторон даже в ходе одного сеанса. Условно принято считать клиентом ту сторону, которая начинает взаимодействие и хочет отослать почту, а сервером ту, что принимает запросы. После того, как клиент посылает серверу несколько служебных команд и получает положительные ответы на них, он отправляет SMTP серверу собственно тело сообщения. SMTP сервер получает сообщение, вносит в него дополнительные заголовки, указывающие на то, что он обработал данное послание, устанавливает связь со следующим SMTP сервером по пути следования письма.

Общение между любыми SMTP серверами происходит по той же схеме. Инициирует переговоры клиент, сервер на них отвечает, а затем получает корреспонденцию и "ставит штампик" в теле письма (в его заголовочной части).

Как только почта достигнет конечного пункта (SMTP сервера адресата сообщения), она будет сложена в почтовый ящик абонента, который всегда сможет в удобное для него время изъять ее по протоколам POP3 или IMAP, в зависимости от того, какой из них поддерживается провайдером. В арсенал SMTP клиента, равно как и сервера, входит около 10 команд, но, воспользовавшись только пятью из них, уже можно легально послать почтовое сообщение. Это HELO, MAIL, RCPT, DATA, QUIT. Их использование подразумевается именно в такой последовательности. HELO предназначена для идентификации отправителя, MAIL указывает адрес отправителя, RCPT – адрес назначения. После команды DATA и ответа на нее, клиент посылает серверу тело сообщения, которое должно заканчиваться строкой, содержащей лишь одну точку.

# 30. Эталонная модель взаимодействия открытых систем. Понятие уровня и уровневых сетевых сервисов. Характеристика уровней

Эталонная модель взаимодействия открытых систем (БЭМВОС) – это концептуальная основа, определяющая характеристики и средства открытых систем. Она обеспечивает работу в одной сети систем, выпускаемых различными производителями. Разработана ISO (международной организацией стандартов) и широко используется во всём мире как основа концепций информационных сетей и их ассоциаций. На базе этой модели описываются правила и процедуры передачи данных между открытыми системами. Она также описывает структуру открытой системы и комплекс стандартов, которым она должна удовлетворять.

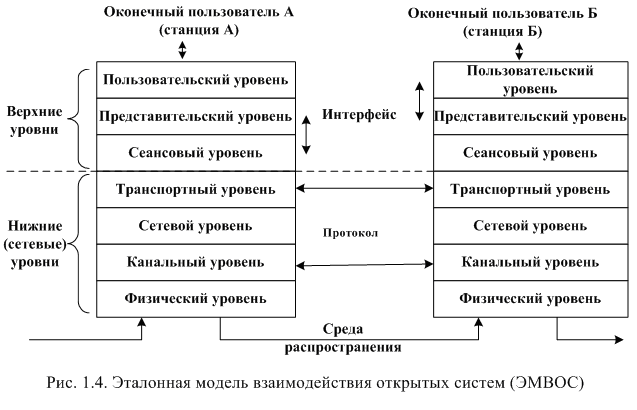
Основными элементами модели являются: уровни, объекты, соединения, физические средства соединений.

В соответствии с этой моделью процесс передачи сообщений в системах связи последовательно разбивается на принципиально различающиеся операции. Каждую ив этих операций относят к своему уровню.

Каждый уровень определяется сервисом, который он предоставляет вышестоящему уровню, и протоколом - набором правил и форматов данных для взаимодействия между собой объектов одного уровня, работающих на разных компьютерах.

Уровни строятся по принципу строгой иерархии: на высшем уровне находятся источник и получатель информации - пользователи системы связи, на нижнем - среда распространения электромагнитных волн. Высший уровень управляет работой низшего. Каждому уровню соответствует свое техническое устройство или организационная единица системы связи пользователь или должностное лицо, обеспечивающее функционирование системы связи. В некоторых системах связи часть этих устройств может отсутствовать либо выполнять не все функции некоторого уровня.

В ЭМВОС выделяют 7 уровней: пользовательский, представительский, сеансовый, транспортный, сетевой, канальный, физический (рис. 1.4). Полную совокупность средств у одного пользователя, выполняющих операции различных уровней, называют станцией.



Характеристика уровней:

· **Физический уровень** (Physical layer) определяет способ физического соединения компьютеров в сети. Функциями средств, относящихся к данному уровню, являются побитовое преобразование цифровых данных в сигналы, передаваемые по физической среде (например, по кабелю), а также собственно передача сигналов.

· **Канальный уровень**(Data Link layer) отвечает за организацию передачи данных между абонентами через физический уровень, поэтому на данном уровне предусмотрены средства адресации, позволяющие однозначно идентифицировать отправителя и получателя во всем множестве абонентов, подключенных к обще линии связи. В функции данного уровня также входит упорядочивание передачи с целью параллельного использования одной линии связи несколькими парами абонентов. Кроме того, средства канального уровня обеспечивают проверку ошибок, которые могут возникать при передаче данных физическим уровнем.

· **Сетевой уровень**(Network layer) обеспечивает доставку данных между компьютерами сети, представляющей собой объединение различных физических сетей. Данный уровень предполагает наличие средств логической адресации, позволяющих однозначно идентифицировать компьютер в объединенной сети. Одной из главных функций, выполняемых средствами данного уровня, является целенаправленная передача данных конкретному получателю.

· **Транспортный уровень**(Transport layer) реализует передачу данных между двумя программами, функционирующими на разных компьютерах, обеспечивая при этом отсутствие потерь и дублирования информации, которые могут возникать в результате ошибок передачи нижних уровней. В случае, если данные, передаваемые через транспортный уровень, подвергаются фрагментации, то средства данного уровня гарантируют сборку фрагментов в правильном порядке.

· **Сессионный (или сеансовый) уровень** (Session layer) позволяет двум программам поддерживать продолжительное взаимодействие по сети, называемое **сессией** (session) или **сеансом**. Этот уровень управляет установлением сеанса, обменом информацией и завершением сеанса. Он также отвечает за идентификацию, позволяя тем самым только определенным абонентам принимать участие в сеансе, и обеспечивает работу служб безопасности с целью упорядочивания доступа к информации сессии.

· **Уровень представления (или представительский)**(Presentation layer) осуществляет промежуточное преобразование данных исходящего сообщения в общий формат, который предусмотрен средствами нижних уровней, а также обратное преобразование входящих данных из общего формата в формат, понятный получающей программе.

· **Прикладной (или пользовательский) уровень** (Application layer) предоставляет высокоуровневые функции сетевого взаимодействия, такие, как передача файлов, отправка сообщений по электронной почте и т.п.

# 31. Этапы сбора и анализа требований к программной системе, требований заказчика и детальных требований

**Сбор требований.**

На этапе сбора требований основная работа ведется с заказчиком системы и еѐ будущими пользователями. Цель этапа — точно определить функции продукта и способы его интеграции в существующие процессы.

Для того чтобы продукт был удобен пользователям и делал «то, что надо», сначала надо определить, кто же им будет пользоваться. Для этого определяются профили пользователей.

Когда определены профили пользователей продукта, следует найти людей, соответствующих этим профилям и желающих помочь вам в разработке продукта. Для проектов под заказ это достаточно простая задача — нужно выбрать одного или двух наиболее грамотных и инициативных людей для каждого профиля.

Далее, необходимо начать общение с конечными пользователями и узнать для достижения каких целей будет использоваться будущий продукт. Лучшим методом для достижения этой цели является сбор пользовательских историй. Пользовательская история — это вариант использования будущего продукта в конкретной ситуации с целью достижения измеримого результата. Каждая пользовательская история должна приносить пользу — не должно быть историй, которые выполняют действия ради действий. Пользовательские истории могут содержать как сложные инструкции с ответвлениями, так и конкретные примеры.

Каждая пользовательская история имеет одно или больше родительских бизнес требований и ее главная цель описать наиболее удачные способы их удовлетворения. Именно на основе родительского требования устанавливается приоритет истории.

Нужно очень четко контролировать поступающую от заказчика ин- формацию. Требования должны быть продиктованы бизнесом и быть нацелены на достижение реальных и измеряемых результатов. Одним словом реализация требования должна приносить пользу, а не являться самоцелью. Если пользователь говорит, что неплохо было бы сделать «мастер того-то и того-то» нужно мгновенно среагировать и задать вопрос — «зачем этот мастер ему нужен». Как правило, он с радостью расскажет о реальных мотивах, которые и интересуют!

Качественное выполнение работ на этом этапе гарантирует то, что будущий продукт будет соответствовать ожиданиям заказчика. Четкая расстановка приоритетов обеспечивает реализацию наиболее востребованной функциональности и исключение второстепенной/невостребованной функциональности, что сэкономит бюджет и сроки.

В процессе этапа сбора требований формируются первичные требования (или требования заказчика).

**Анализ требований**

На этапе анализа требований проходит структуризация уже собранных ранее требований. Цель этапа — предоставить четкий список не дублируемых требований к системе, которые должны быть выделены из избыточных и частично дублирующихся сценариев и пользовательских историй, которые были полученных на предыдущем этапе. Правильно сгруппированные требования помогут обойтись минимальным количеством функционала для удовлетворения максимально большего количества целей, а это, в свою очередь, поможет сэкономить бюджет и не даст расползтись рамкам проекта.

По окончании этапа анализа требований, многостраничный документ, содержащий сотни пользовательских историй, будет разбит на части. Каждая часть будет освещать только необходимую функциональность, а в еѐ основе будет стоять диаграмма вариантов использования, на основе которой можно будет легко увидеть все требуемые функции системы. Описание вариантов использования не будут дублироваться, а лишь дополнять друг друга.

Работу над этапом анализа можно считать законченной, когда вы выделили все пользовательские истории в отдельные пакеты. Каждый пакет имеет понятное название, а описание содержит критерии, на основе которых в него попали или не попали пользовательские истории.

**Существует два основных метода проектирования** — проектирование на основе вариантов использования и проектирование на основе требований.

Проектирование на базе вариантов использования считается более эффективным, так как этот метод позволяет не терять связь с пользовательскими историями и прекрасно иллюстрирует требуемое поведение системы в целом, а, следовательно, гарантирует востребованность всего функционала, который будет создан (очень расточительно и болезненно для разработчиков писать код, которым никогда не удастся воспользоваться). Но все же есть условия, при которых аналитик может пренебречь этим методом в пользу проектирования на базе требований:

· Проектирование на основе требований следует предпочесть, если необходимо сократить затраты на стадию анализа до минимума, а количество пользовательских историй в пакете не велико (их содержимое можно просмотреть не дольше чем за 10 минут)

· Команда разработки не умеет или не хочет работать с вариантами использования и требует предоставление требований к системе в классическом виде.

**Проектирование на базе вариантов использования:**

Как правило, у всех пользовательских историй в пакете есть одна или несколько главных целей, и есть история, которая описывает наиболее простой способ их достижения. Такая история называется — базовой, а описание еѐ действия — базовый путь. Главным критерием определения базового варианта использования является наличие общих со всеми остальными вариантами использования действий. Остальные истории описывают альтернативный способ достижения результата или содержат дополнительные действия для достижения специфического результата и по большому счету являются дополнениями к базовой истории.

Пользовательские истории это разновидность стандартных вариантов использования, с той лишь особенностью, что они описывают взаимодействие не с реальной, а с гипотетической системой.

Таким образом, в этом методе выделяются базовые пользовательские истории, а так же альтернативные способы достижение результатов.

**Проектирование на основе требований:** Для нормальной разработки нужен список требований, который однозначно идентифицирует потребности пользователей и не имеет множественных повторений, которые присутствуют с избытком в пользовательских историях (если не произвести структурирование историй, описанное в предыдущем подходе). Кроме этого, для более гибкого проектирования необходим как можно более детализированный (раздробленный) список.

Благодаря дроблению списка, приобретается возможность запланировать реализацию наиболее значимых требований на начало разработки, а низкоприоритетных - на конец. Как результат, в случае отставания от графика выполнения в продукте уже будут реализованы наиболее приоритетные требования, и вы будете иметь возможность пожертвовать низкоприоритетными требованиями с целью сокращения отставания.

В процессе анализа требований формируются детальные требования.

# 32.Требования к программному обеспечению. Типы требований. Уровни описания требований

**Требования к ПО**

Требования к ПО – это возможности или условия, которым должна соответствовать система или проект, совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации.

**Классификация требований ПО**

**1) Функциональные.**

Это требования, которые описывают поведение системы и ее функции. Если функциональные требования оформлены как пользовательские, они, как правило описывают систему в обобщенном виде. В противоположность этому функциональные требования, оформленные как системные описывают систему максимально подробно включая ее функции, исключения и т.д.

Спецификация функциональных требований должна быть комплексной и непротиворечивой, но на практике этого добиться сложно. Причем зачастую выявить несогласованность на начальных этапах очень трудно.

Это перечень сервисов, которые должна выполнять система, причем должно быть указано, как система реагирует на те или иные входные данные, как она ведет себя в определенных ситуациях и т.д. В некоторых случаях указывается, что система не должна делать.

2) **Нефункциональные.**

Описывают характеристики системы и ее окружения, а не поведение системы. В них обычно приводится перечень ограничений накладываемых на действия, функции выполняемых в системе (загруженность процессора и т.д.)

Многие нефункциональные требования относятся к системе в целом, а не к отдельным ее функциям. Это означает, что они более значимы и критичны, чем отдельные функциональные требования. Ошибка допущенная в функциональных требованиях может снизить качество системы, ошибка допущенная в нефункциональных может привести к неработоспособности системы.

Нефункциональные требования можно разбить на следующие группы:

a. Требования к продукту

b. Организационные требования

c. Внешние требования

Здесь также может быть приведен перечень ограничений, накладываемых на действия и функции, выполняемые системой. Они включают временные ограничения, ограничения на процесс разработки системы, стандарты и тд.

**3)** **Требования предметной области**

Характеризуют ту предметную область, где будет эксплуатироваться система, эти требования могут быть функциональными и нефункциональными.

**Уровни описания требований**

1) **Пользовательские требования.** Должны описывать функциональные и нефункциональные требования так, чтобы они были понятны даже пользователю не имеющему специальных технических знаний. Пользовательские требования - описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на неё.

2) **Системные требования** - детализированное описание системных функций и ограничений, которое иногда называют функциональной спецификацией. Она служит основой для заключения контракта между покупателем системы и разработчиками ПО.

3) **Проектная системная спецификация** - обобщённое описание структуры программной системы, которое будет основой для детализованного проектирования системы и её последующей реализации. Эта спецификация дополняет и детализирует спецификацию системных требований.

Пользовательские требования пишутся для заказчика ПО и для лица, заключающего контракт на разработку программной системы, причём они могут не иметь детальных технических знаний по разрабатываемой системе. Спецификация системных требований предназначена для руководящего технического состава компании-разработчика и для менеджеров проекта. Она также необходима заказчику ПО и субподрядчикам по разработке. Эти оба документа также предназначены для конечных пользователей программной системы. Наконец, проектная системная спецификация является документом, который ориентирован на разработчиков ПО.