

- [1. Работа с динамической памятью, на примере языков С и С++](#)
- [2. Понятие класса, типы классов, члены классов, на примере языка С#](#)
- [3. Стандартные типы данных языка С#, операции над ними. Стандартные типы данных](#)
- [4. Понятие интерфейса, на примере языков С++ и С#](#)
- [5. Наследование классов, на примере языков С++ и С#](#)
- [6. Теоретико-множественное описание и характеристические функции отношений. Аксиомы Армстронга. Унарные и бинарные операции реляционной алгебры](#)
- [7. Содержание процесса нормализации БД. Нормальные формы БД](#)
- [8. Процедурные расширения языка SQL – курсоры, подпрограммы, триггеры](#)
- [9. Виды архитектур ядер операционных систем. Монолитная, многослойная и микроядерная архитектура ядра ОС. Другие виды ядер ОС](#)
- [10. Планирование процессов. Критерии эффективности и свойства методов планирования процессов, параметры планирования процессов. Приоритетное и не приоритетное планирование, гарантированное планирование, многоуровневые очереди](#)
- [11. Проблемы взаимодействующих процессов. Алгоритмы реализации взаимоисключений. Семафоры Дейкстры. Решение проблемы «производитель-потребитель» с помощью семафоров](#)
- [12. Принципы управления памятью вычислительной системы. Виртуальная память и преобразование адресов. Методы распределения оперативной памяти без использования внешней памяти. Страничная и сегментно-страничная организация виртуальной памяти](#)
- [13. Организация распределенных вычислительных систем. Алгоритмы распределения памяти. Алгоритмы распределения доступа к файлам. Синхронизация в распределенных системах](#)
- [14. Стандартизация систем. ГОСТ Р МЭК 62264-1-2010 Интеграция систем управления предприятием. Функциональная модель управления предприятием. Контур управления. Функции управления](#)
- [15. Процессы проектирования. Проектирование программной архитектуры. Модели описания программной архитектуры. Шаблоны программной архитектуры](#)
- [16. Модели и их представления на UML – использования, поведения и структуры. Общие свойства модели и механизмы расширения – стереотипы, помеченные значения, ограничения](#)
- [17. Моделирование структуры на UML. Диаграмма классов. Компоненты и интерфейсы](#)
- [18. Диаграммы UML: диаграммы use case \(вариантов использования\), диаграммы состояний, диаграммы деятельности, диаграмма последовательности и диаграмма коммуникации](#)
- [19. Паттерны проектирования и каркасы на UML](#)
- [20. Общая характеристика стандартов управления ИТ инфраструктурой \(ITIL, ITSM, ISO20000, MOF\). Области применения, основное содержание, цели применения](#)
- [21. Локальные сети Ethernet. Активное оборудование. Коммутаторы L2, L3, их характерные особенности](#)

[22. Архитектура стека TCP/IP \(уровни, назначение, потоки данных, примеры протоколов\), адресная информация в TCP/IP. IP адреса, IP-сети, порты TCP\UDP. Соединение IP сетей. Маршрутизация в IP. Трансляция адресов \(NAT\). Проксирование](#)

[23. Надежность систем. Показатели надежности. Уровни надежности по классификации HP](#)

[24. Системы шифрования. Виды алгоритмов шифрования, общие характеристики. Общая архитектура PGP. Сертификаты. Назначение, устройство, области применения](#)

[26. Написание SQL запроса к реляционной базе данных в соответствии со стандартом SQL-92. В соответствии с описанием требуемой структуры данных и описанием таблиц и полей данных реляционной базы данных написать запрос, использующий основные конструкции языка SQL в соответствии со стандартом SQL-92, включая объединение таблиц, вложенные подзапросы, агрегирование и сортировку данных](#)

[27. Написание сценария на языке bash для автоматизации операций администрирования операционных систем семейства Linux. В соответствие с спецификацией требований написать сценарий на языке bash для автоматизации одной из следующих операций: Анализ строк текстовых журналов с использованием регулярных выражений и вывод результатов в заданном формате. Управление файлами, каталогами, ссылками, правами доступа к объектам системы. Мониторинг данных о процессах, управление приоритетами их выполнения. Работа выполняется в среде виртуализации Oracle Virtual Box 4 в виртуальной машине под управлением Linux CentOS 6.7](#)

[28. Настройка статической маршрутизации для IPv4 в программных маршрутизаторах Windows Server или Linux по выбору. При решении задачи используется схема клиент-маршрутизатор-маршрутизатор-клиент. Требуется подобрать адреса сетей по указанным требованиям \(количество хостов в сети\), настроить сетевые интерфейсы клиентов и маршрутизаторов, настроить таблицы маршрутизаторов. Работа выполняется в среде виртуализации Oracle Virtual Box 4 в ОС Windows 2003 или Linux Centos 6.7](#)

[29. Настройка правил файрвола в программном маршрутизаторе по заданным требованиям на ОС Linux или Windows \(до 7 правил\). Работа выполняется в среде виртуализации Oracle Virtual Box 4 в ОС Windows 2003 или Linux Centos 6.7](#)

[30. Создание и управление пользователями и группами в Active Directory через графический интерфейс и через командную строку на powershell или с помощью консольных утилит \(dsquery, dsadd и т.п.\). Настройка прав на файлы и каталоги по требованиям на платформе Windows в системе NTFS](#)

1. Работа с динамической памятью, на примере языков С и С++

Для работы с динамической памятью в языке С используются следующие функции:

`malloc, calloc, free, realloc.`

```
void *malloc(size_t size);
```

В качестве входного параметра функция принимает размер памяти, которую требуется выделить. Возвращаемым значением является указатель на выделенный в куче участок памяти.

```
int * p = malloc(1000000*sizeof(int));
```

```
void free(void *ptr);
```

В качестве входного параметра в `free` нужно передать указатель, значение которого получено из функции `malloc`.

```
void *calloc(size_t nmemb, size_t size);
```

Функция работает аналогично `malloc`, но отличается синтаксисом (вместо размера выделяемой памяти нужно задать количество элементов и размер одного элемента) и тем, что выделенная память будет обнулена.

```
int * q = (int *) calloc(1000000, sizeof(int));
```

```
void *realloc(void *ptr, size_t size);
```

Функция изменяет размер выделенной памяти (на которую указывает `ptr`, полученный из вызова `malloc, calloc` или `realloc`). Если размер указанный в параметре `size` больше, чем тот, который был выделен под указатель `ptr`, то проверяется, есть ли возможность выделить недостающие ячейки памяти подряд с уже выделенными. Если места недостаточно, то выделяется новый участок памяти размером `size` и данные по указателю `ptr` копируются в начало нового участка.

В С++ есть свой механизм выделения и освобождения памяти — это функции `new` и `delete`.

```
int * p = new int[1000000]; // выделение памяти под 1000000 int`ов
```

Освобождение выделенной при помощи `new` памяти осуществляется посредством следующего вызова:

```
delete [] p;
```

```
int * q = new int(10); // выделенный int проинициализируется значением 10  
delete q;
```

2. Понятие класса, типы классов, члены классов, на примере языка C#

Класс — это логическая структура, позволяющая создавать свои собственные пользовательские типы путем группирования переменных других типов, методов и событий.

Класс подобен чертежу. Он определяет данные и поведение типа. Если класс не объявлен статическим, то клиентский код может его использовать, создав объекты или экземпляры, назначенные переменной. Переменная остается в памяти, пока все ссылки на нее не выйдут из области видимости. В это время среда CLR помечает ее пригодной для сборщика мусора. Если класс объявляется статическим, то в памяти остается только одна копия и клиентский код может получить к ней доступ только посредством самого класса, а не переменной экземпляра.

Все значимые объекты и ссылки на ссылочные объекты хранятся в стеке. Ссылочные объекты хранятся в куче. При этом если значимый объект является частью ссылочного, то он тоже хранится в куче. Любой значимый тип можно явно привести к типу `Object`. Данная операция называется упаковкой и позволяет использовать значимый тип как ссылочный, например, для передачи параметра в метод. Обратное действие явного приведения `Object` к значимому типу называется распаковкой. Если в процессе распаковки выяснилось, что данная распаковка некорректна, вызывается `InvalidCastException`. Если значимый тип содержит ссылочный, то в стеке хранится только ссылка на ссылочный, а сам ссылочный хранится в куче.

Поля, свойства, методы и события в классе обозначаются термином члены класса.

3. Стандартные типы данных языка C#, операции над ними. Стандартные типы данных

Тип	Область значений	Размер
sbyte	-128 до 127	Знаковое 8-бит целое
byte	0 до 255	Беззнаковое 8-бит целое
char	U+0000 до U+ffff	16-битовый символ Unicode
bool	true или false	1 байт
short	-32768 до 32767	Знаковое 16-бит целое
ushort	0 до 65535	Беззнаковое 16-бит целое
int	-2147483648 до 2147483647	Знаковое 32-бит целое
uint	0 до 4294967295	Беззнаковое 32-бит целое
long	-9223372036854775808 до 9223372036854775807	Знаковое 64 -бит целое
ulong	0 до 18446744073709551615	Беззнаковое 64 -бит целое
float	$\pm 1,5 \cdot 10^{-45}$ до $\pm 3,4 \cdot 10^{33}$	4 байта, точность — 7 разрядов
double	$\pm 5 \cdot 10^{-324}$ до $\pm 1,7 \cdot 10^{306}$	8 байт, точность — 16 разрядов
decimal		12 байт, точность — 28 разрядов

Типом данных также является перечисление, структура, класс. Неявное преобразование возможно от менее мощного к более мощному типу, но не наоборот. Все типы данных явно или неявно унаследованы от Object.

Основные операторы

Выражение	Описание
x.y	Доступ к членам
f(x), x.f(y)	Вызов метода и делегата
a[x]	Доступ к массиву и индексатору
x++, ++x, x--, --x	Постфиксное/префиксное приращение/уменьшение
new T(...)	Создание объекта и делегата
new T(...){...}	Создание объекта с инициализатором
new {...}	Анонимный инициализатор объекта
new T[...]	Создание массива
typeof(T)	Получение объекта System.Type для T

checked(x)	Вычисление выражения в проверенном контексте
unchecked(x)	Вычисление выражения в непроверенном контексте
default(T)	Получение значения по умолчанию для типа
delegate {}	Анонимная функция (анонимный метод)
x is T	Возвращает значение true, если x относится к типу T, в противном случае возвращает значение false
x as T	Возвращает x типа T или нулевое значение, если x не относится к типу T
x < y ...	Операторы отношения
x op= y	Составные операторы присвоения
Логические	x & y, x ^ y, x y
Условные	x && y, x y
x ?? y	Равно y, если x — нулевое, в противном случае равно x
x ?: y : z	Равно y, если x имеет значение true, z если x имеет значение false
(T)x	Явное преобразование
Арифметические	*, +, -, /, %
x >> y	Сдвига

4. Понятие интерфейс, на примере языков C++ и C#

C#

Интерфейсы описывают группу связанных функциональных возможностей, которые могут принадлежать к любому классу или структуре. Интерфейсы могут содержать методы, свойства, события, индексаторы или любое сочетание этих перечисленных типов членов. Интерфейсы не могут содержать поля. Члены интерфейсов автоматически являются открытыми.

Когда говорят, что класс или структура наследует интерфейс, это означает, что класс или структура предоставляет реализацию для всех членов, определяемых интерфейсом. Сам интерфейс не предоставляет функциональных возможностей, которые класс или структура могут наследовать таким же образом, каким могут наследоваться функциональные возможности базового класса. Однако если базовый класс реализует интерфейс, производный класс наследует эту реализацию.

Классы и структуры могут быть унаследованы от интерфейса таким же образом, как классы могут быть унаследованы от базового класса или структуры, но есть два исключения:

- Класс или структура может наследовать несколько интерфейсов.
- Когда класс или структура наследует интерфейс, наследуются только имена и подписи методов, поскольку сам интерфейс не содержит реализаций.

Если два интерфейса имеют метод с одной и той же сигнатурой, то можно одним методом реализовать оба интерфейса. Если определить раздельную реализацию, то вызывать тот или иной метод просто по имени невозможно, нужно приводить экземпляр класса к типу интерфейса.

Интерфейс подобен абстрактному базовому классу: любой неабстрактный тип, наследующий интерфейс, должен реализовать все его члены. Невозможно создать экземпляр интерфейса напрямую. Интерфейсы могут содержать методы, свойства, индексаторы и события в качестве членов.

Интерфейсы не содержат реализации методов.

Как классы, так и структуры способны наследовать от нескольких интерфейсов.

Интерфейс может быть унаследован от нескольких интерфейсов.

```
interface ISampleInterface
{
    void SampleMethod();
}
class ImplementationClass : ISampleInterface
{
    // Explicit interface member implementation:
    void ISampleInterface.SampleMethod()
    {
        // Method implementation.
    }
    static void Main()
    {
        // Declare an interface instance.
        ISampleInterface obj = new ImplementationClass();
        // Call the member.
        obj.SampleMethod();
    }
}
```

```
}
```

С помощью интерфейсов можно, например, включить поведение из нескольких источников в классе. Эта возможность важна в C#, поскольку язык не поддерживает множественное наследование классов. Кроме того, необходимо использовать интерфейс, если требуется имитировать наследование для структур, поскольку они фактически не могут наследоваться из других структур или классов.

C++

C++ поддерживает множественное наследование и абстрактные классы, отдельная синтаксическая конструкция для интерфейсов отсутствует. Интерфейсы определяются при помощи абстрактных классов, а реализация интерфейса производится путём наследования этих классов.

```
class iOpenable
{
    public:
        virtual ~iOpenable(){}

        virtual void open()=0;
        virtual void close()=0;
};
class Door: public iOpenable
{
    public:
        Door(){std::cout << "Door object created" << std::endl;}
        virtual ~Door(){}

        //Конкретизация методов интерфейса iOpenable для класса Door
        virtual void open(){std::cout << "Door opened" << std::endl;}
        virtual void close(){std::cout << "Door closed" << std::endl;}

        //Специфические для класса Door свойства и методы
        std::string mMaterial;
        std::string mColor;
        //...
};
class Book: public iOpenable
{
    public:
        Book(){std::cout << "Book object created" << std::endl;}
        virtual ~Book(){}

        //Конкретизация методов интерфейса iOpenable для класса Book
        virtual void open(){std::cout << "Book opened" << std::endl;}
        virtual void close(){std::cout << "Book closed" << std::endl;}

        //Специфические для класса Book свойства и методы
        std::string mTitle;
        std::string mAuthor;
        //...
};
```


5. Наследование классов, на примере языков C++ и C#

Наследование, вместе с инкапсуляцией и полиморфизмом, является одной из трех основных характеристик (или базовых идей) объектно-ориентированного программирования. Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в других классах. Класс, члены которого наследуются, называется базовым классом, а класс, который наследует эти члены, называется производным классом.

При определении класса для наследования от другого класса, производный класс явно получает все члены базового класса, за исключением его конструкторов и деструкторов. Производный класс может таким образом повторно использовать код в базовом классе без необходимости в его повторной реализации. В производном классе можно добавить больше членов. Таким образом, производный класс расширяет функциональность базового класса.

Абстрактные и виртуальные методы

Когда базовый класс объявляет метод как виртуальный, производный класс может переопределить метод с помощью своей собственной реализации. Если базовый класс объявляет член как абстрактный, то этот метод должен быть переопределен в любом неабстрактном классе, который прямо наследует от этого класса. Если производный класс сам является абстрактным, то он наследует абстрактные члены, не реализуя их. Абстрактные и виртуальные члены являются основой для полиморфизма, который является второй основной характеристикой объектно-ориентированного программирования.

Интерфейсы

Интерфейс является ссылочным типом, в чем-то схожим с абстрактным базовым классом, который состоит только из абстрактных членов. Когда класс реализует интерфейс, он должен предоставить реализацию для всех членов интерфейса. В классе может быть реализовано несколько интерфейсов, хотя производным он может быть только от одного прямого базового класса.

Доступ производного класса к членам базового класса

Из производного класса можно получить доступ к открытым, защищенным, внутренним и защищенным внутренним членам базового класса. Хотя производный класс и наследует закрытые члены базового класса, он не может получить доступ к этим членам. Однако все эти закрытые члены все же присутствуют в производном классе и могут выполнять ту же работу, что и в самом базовом классе. Например, предположим, что защищенный метод базового класса имеет доступ к закрытому полю. Это поле должно присутствовать в производном классе для правильной работы унаследованного метода базового класса.

Множественное наследование C++

Если вы порождаете класс из нескольких базовых классов, то получаете преимущества множественного наследования.

При множественном наследовании производный класс получает атрибуты двух или более классов.

При использовании множественного наследования для порождения класса конструктор производного класса должен вызвать конструкторы всех базовых классов.

При порождении класса из производного класса вы создаете иерархию наследования (иерархию классов).

Пример:

```
class computer_screen
{
public:
    computer_screen(char *, long, int, int);
    void show_screen(void);
private:
    char type[32] ;
    long colors;
    int x_resolution;
    int y_resolution;
};
```

Предположим, что у вас есть также класс mother_board:

```
class mother_board
{
public:
    mother_board(int, int, int);
    void show_mother_board(void);
private:
    int processor;
    int speed;
    int RAM;
};
```

Используя эти два класса, можно породить класс computer, что показано ниже:

```
class computer : public computer_screen, public mother_board
{
public:
    computer(char *, int, float, char *, long, int, int, int, int, int);
    void show_computer(void);
private:
    char name[64];
    int hard_disk;
    float floppy;
};
```

Множественное наследование C#

Множественное наследование – это когда один класс сразу наследуется от нескольких классов. Но бывает так, что базовые классы содержат методы с одинаковыми именами, в результате чего возникают определенные неточности и ошибки. Множественное наследование есть в языке C++, а в C# от него отказались и внесли интерфейсы. В C# класс может реализовывать сразу несколько интерфейсов. Это и является главным отличием использования интерфейсов и абстрактных классов. Кроме того, конечно же, абстрактные классы могут содержать все остальные члены, которых не может быть в интерфейсе, и не все методы/свойства в абстрактном классе должны быть абстрактными. Если класс реализовывает несколько интерфейсов, они разделяются запятыми:

```
interface IDrawable
{
    void Draw();
}
interface IGeometrical
{
    void GetPerimeter();
    void GetArea ();
}
```

```

class Rectangle : IGeometrical, IDrawable
{
    public void GetPerimeter()
    {
        Console.WriteLine("(a+b)*2");
    }
    public void GetArea()
    {
        Console.WriteLine("a*b");
    }
    public void Draw()
    {
        Console.WriteLine("Rectangle");
    }
}
class Circle : IGeometrical, IDrawable
{
    public void GetPerimeter()
    {
        Console.WriteLine("2*pi*r");
    }
    public void GetArea()
    {
        Console.WriteLine("pi*r^2");
    }
    public void Draw()
    {
        Console.WriteLine("Circle");
    }
}

```

Здесь был объявлен интерфейс IDrawable, который предоставляет метод для рисования объекта. Этот интерфейс может реализовывать, например, класс Image. Классы Image и Circle совсем разные сущности, и они не имеют общего базового класса, но мы можем создать список указателей на интерфейс IDrawable, и работать с такими объектами, как с однотипными (с одинаковым интерфейсом). Этот пример с IDrawable более наглядно отображает то, что нам дают интерфейсы. На практике, IGeometrical стоило бы заменить на абстрактный класс.

Модификаторы наследования

- Public — доступ открыт всем, кто видит определение данного класса.
- Private — доступ открыт самому классу (т.е. функциям-членам данного класса) и друзьям (friend) данного класса, как функциям, так и классам.
- Protected — доступ открыт классам, производным от данного.

6. Теоретико-множественное описание и характеристические функции отношений. Аксиомы Армстронга. Унарные и бинарные операции реляционной алгебры

Желая получить математическую модель процесса функционирования системы, чтобы она охватывала широкий класс реальных объектов, в общей теории систем исходят из общих предположений о характере функционирования системы:

1. система функционирует во времени; в каждый момент времени система может находиться в одном из возможных состояний;
2. на вход системы могут поступать входные сигналы;
3. система способна выдавать выходные сигналы;
4. состояние системы в данный момент времени определяется предыдущими состояниями и входными сигналами, поступившими в данный момент времени и ранее;
5. выходной сигнал в данный момент времени определяется состояниями системы и входными сигналами, относящимися к данному и предшествующим моментам времени.

Характеристическая функция χ_A множества $A \subseteq U$ есть функция, отображающая универсальное множество U в двухэлементное множество $\{0;1\}$: (функцию χ_A можно представить как проверку принадлежности элемента к множеству A)

Функциональные зависимости

Пусть $R=(A_1...A_n)$ является функциональной схемой отношения и X, Y - некоторые подмножества атрибутов этой схемы. Говорят, что X функционально определяет Y ($X \rightarrow Y$), если в любом экземпляре отношения со схемой R не существует двух кортежей, совпадающих по подмножеству X и не совпадающих по подмножеству Y

Иначе говоря, если два кортежа совпадают по X , то они должны совпадать и по Y

Например, $R=(A_1,A_2,A_3,A_4)$, есть зависимости:

- $A_1 \rightarrow A_2$ (1)
- $A_1, A_3 \rightarrow A_4$ (2)

Предположим, что имеет место один экземпляр отношения со схемой RR

	A1	A2	A3	A4
RR	фирма X	улица Ленина, д.1	сахар	40
	фирма X	улица Ленина, д.1	карамель	50
	фирма X	улица Ленина, д.2	пастила	90

Вторая ФЗ (2) **имеет место быть**, так как нет двух кортежей, совпадающих по этой паре (по крайней мере, в приведённом экземпляре схемы отношения, а других мы не знаем).

А первая ФЗ (1) **не имеет место быть** (в третьей строке другой номер дома портит всю картину).

Аксиомы Армстронга

1. Аксиомы рефлексивности: Если $F \subseteq X \subseteq R$, то $X \rightarrow Y$
2. Аксиома пополнения: Пусть $X \rightarrow Y$, (Z может быть \emptyset) Тогда $(XZ \rightarrow YZ)$
3. Аксиома транзитивности: Если $X \rightarrow Y$, $Y \rightarrow Z$, то $X \rightarrow Z$

Основные восемь операций реляционной алгебры были предложены Э.Коддом.

- Объединение
- Пересечение (Результатом операции пересечения будет отношение, состоящее из кортежей, полностью входящих в состав обоих отношений.)
- Вычитание (Результатом вычитания будет отношение, состоящее из кортежей, которые являются кортежами первого отношения и не являются кортежами второго отношения.)
- Декартово произведение (Умножение или декартово произведение является операцией, производимой над двумя отношениями, в результате которой мы получаем отношение со всеми доменами из двух начальных отношений. Кортежи в этих доменах будут представлять из себя все возможные сочетания кортежей из начальных отношений.)
- Выборка (выделяет множество строк в таблице, удовлетворяющих заданным условиям)
- Проекция (из отношения выделяются атрибуты только из указанных доменов, то есть из таблицы выбираются только нужные столбцы, при этом, если получится несколько одинаковых кортежей, то в результирующем отношении остается только по одному экземпляру подобного кортежа)
- Соединение (Операция соединения обратна операции проекции и создает новое отношение из двух уже существующих. Новое отношение получается конкатенацией кортежей первого и второго отношений, при этом конкатенации подвергаются отношения, в которых совпадают значения заданных атрибутов.)
- Деление. Отношение с заголовком (X_1, X_2, \dots, X_n) и телом, содержащим множество кортежей (x_1, x_2, \dots, x_n) , таких, что для всех кортежей $(y_1, y_2, \dots, y_m) \in B$ в отношении $A(X_1, X_2, \dots, X_n, Y_1, Y_2, \dots, Y_m)$ найдется кортеж $(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m)$.

7. Содержание процесса нормализации БД. Нормальные формы БД

Нормализация - Приведение структуры БД к виду, отвечающему нормальным формам. (если не покатит то: Нормализация –это процесс последовательной замены отношения его полными декомпозициями до тех пор, пока все они не будут находиться в 5НФ.)

Выделяются шесть нормальных форм, пять из которых так и называются: первая, вторая, третья, четвертая, пятая нормальная форма, а также нормальная форма Бойса-Кодда, лежащая между третьей и четвертой.

База данных считается нормализованной, если ее таблицы представлены как минимум в третьей нормальной форме.

Нормализация отношений– это формальный аппарат ограничений на формирование отношений, который позволяет устранить дублирование и потенциальную противоречивость хранимых данных, уменьшает трудозатраты на ведение БД. Процесс нормализации заключается в декомпозиции исходных отношений на более простые отношения. Цель нормализации – получение такого проекта БД, в котором «каждый факт появляется лишь в одном месте».

Теория нормализации основана на наличии зависимостей между атрибутами отношения. Основными видами зависимостей являются:

- функциональные;
- многозначные;
- транзитивные.

Базовым является понятие функциональной зависимости, поскольку на его основе формируются определения всех остальных видов зависимостей. Атрибут В функционально зависит от атрибута А, если каждому значению А соответствует в точности одно значение В. Математически функциональную зависимость В от А обозначают $A \rightarrow B$. При этом А и В могут быть составными, то есть состоять из двух и более атрибутов.

Зависимость, при которой каждый неключевой атрибут зависит от всего составного ключа и не зависит от его частей, называется полной функциональной зависимостью. Если атрибут А зависит от атрибута В, а атрибут В зависит от атрибута С ($C \rightarrow B \rightarrow A$), но обратная зависимость отсутствует, то зависимость А от С называется транзитивной.

Многозначная зависимость. Говорят, что один атрибут отношения многозначно определяет другой атрибут того же отношения, если для каждого значения первого атрибута существует множество соответствующих значений второго атрибута. Многозначные зависимости могут быть:

- один-ко-многим (1:M);
- многие-к-одному (M:1);
- многие-ко-многим (M:M).

Каждая ступень процесса нормализации приводит схему отношений в последовательные нормальные формы. Для каждой ступени имеются наборы ограничений. Выделяют следующую последовательность нормальных форм:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);

- третья нормальная форма (3НФ);
- усиленная 3НФ или нормальная форма Бойса-Кодда (БКНФ);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма (5НФ).

Отношение находится в **первой нормальной форме (1НФ)**, когда каждая строка содержит только одно значение для каждого атрибута (столбца), то есть все атрибуты отношения имеют единственное значение (являются атомарными).

- запрещает повторяющиеся столбцы (содержащие одинаковую по смыслу информацию)
- запрещает множественные столбцы (содержащие значения типа списка и т.п.)
- требует определить первичный ключ для таблицы, то есть тот столбец или комбинацию столбцов, которые однозначно определяют каждую строку

Отношение находится во **второй нормальной форме (2НФ)**, если оно находится в 1НФ, и каждый неключевой атрибут полностью функционально зависит от всех составляющих первичного ключа. Если атрибут не зависит полностью от первичного ключа, то он внесен ошибочно и должен быть удален. Нормализация производится путем нахождения существующего отношения, к которому относится данный атрибут, или созданием нового отношения, в который атрибут должен быть помещен.

- Вторая нормальная форма требует, чтобы неключевые столбцы таблиц зависели от первичного ключа в целом, но не от его части.
- Если таблица находится в первой нормальной форме и первичный ключ у нее состоит из одного столбца, то она автоматически находится и во второй нормальной форме.

Отношение находится в **третьей нормальной форме (3НФ)**, если оно находится во 2НФ и ни один из его неключевых атрибутов не связан функциональной зависимостью с любым другим неключевым атрибутом. Атрибуты, зависящие от других неключевых атрибутов, нормализуются путем перемещения зависимого атрибута и атрибута, от которого он зависит, в новое отношение. Формально, для приведения схемы в 3НФ необходимо исключить все транзитивные зависимости.

Нормальная форма Бойса-Кодда(БКНФ) является развитием 3НФ и требует, чтобы в отношении были только такие функциональные зависимости, левая часть которых является потенциальным ключом отношения. Потенциальный ключ представляет собой атрибут (или множество атрибутов), который может быть использован для данного отношения в качестве первичного ключа. Фактически первичный ключ – это один из потенциальных ключей, назначенный в качестве первичного. Детерминантом называется левая часть функциональной зависимости. Отношение находится в БКНФ тогда и только тогда, когда каждый детерминант отношения является потенциальным ключом.

Нормальные формы высших порядков (4НФ и 5НФ) представляют больший интерес для теоретических исследований, чем для практики проектирования БД. В них учитываются многозначные зависимости между атрибутами. Полной декомпозицией отношения называют такую совокупность произвольного числа его проекций, соединение которых позволяет получить исходное отношение.

Отношение находится в **пятой нормальной форме (5НФ)**, когда в каждой его полной декомпозиции все проекции содержат возможный ключ. Отношение, не имеющее ни одной полной декомпозиции, также находится в 5НФ.

Четвертая нормальная форма (4НФ) является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. На практике непросто подобрать отношение, которое находится в 4НФ, не будучи в 5НФ.

8. Процедурные расширения языка SQL – курсоры, подпрограммы, триггеры

Курсоры

Иногда бывает необходимо просмотреть строки в прямом или обратном направлении один или несколько раз. Именно для этого используются курсоры. Курсор представляет собой запрос к базе данных, хранящийся на сервере СУБД, — это не оператор SELECT, но результирующее множество, выборка, полученная в результате действия оператора SELECT. После того как курсор сохранен, приложения могут "прокручивать" (просматривать) данные в прямом или обратном направлении, как только возникает такая потребность.

Работу с курсором можно разделить на несколько четко выраженных стадий.

- **Объявите** переменные языка TransactSQL, которые будут содержать данные, возвращенные курсором. Объявите по одной переменной для каждого столбца результирующего набора. Объявите переменные достаточно большого размера для хранения значений, возвращаемых в столбце и имеющих тип данных, к которому могут быть неявно преобразованы данные столбца.
- **Связывание с переменной (слипается с предыдущим)** Воспользуйтесь инструкцией DECLARE CURSOR, чтобы связать курсор языка TransactSQL с инструкцией SELECT. Инструкция DECLARE CURSOR определяет также характеристики курсора, например, имя курсора и тип курсора (readonly или forwardonly).
- **Открытие** Воспользуйтесь инструкцией OPEN для выполнения инструкции SELECT и заполнения курсора.
- **Использование** Воспользуйтесь инструкцией FETCH INTO для выборки отдельных строк и перемещения данных для каждого столбца в указанную переменную. После этого другие инструкции языка TransactSQL могут ссылаться на эти переменные для доступа к выбранным значениям данных. Курсоры языка TransactSQL не поддерживают выборку группы строк.
- **Закрытие + высвобождение.** После завершения работы с курсором примените инструкцию CLOSE. Закрытие курсора освобождает некоторые ресурсы, например, результирующий набор курсора и его блокировки на текущей строке, однако структура курсора будет доступна для обработки, если снова выполнить инструкцию OPEN. Поскольку курсор все еще существует на этом этапе, повторно использовать его имя не удастся. Инструкция DEALLOCATE полностью освобождает все ресурсы, выделенные курсору, в том числе имя курсора. После освобождения курсора его необходимо перестроить заново с помощью инструкции DECLARE.

Курсоры создаются с помощью оператора DECLARE, синтаксис которого различен для разных СУБД. Оператор DECLARE дает курсору имя и принимает оператор SELECT, дополненный при необходимости предложением WHERE и другими. Чтобы показать, как это работает, мы создадим курсор, который будет делать выборку всех клиентов, не имеющих адресов электронной почты, в виде части приложения, позволяющего служащему вводить недостающие адреса.

```
DECLARE CustCursor CURSOR  
FOR  
SELECT * FROM Customers
```

```
WHERE cust_email IS NULL;
```

Оператор SELECT определяет курсор, содержащий имена всех клиентов, которые не имеют адреса электронной почты (соответствующее значение равно NULL).

Теперь, после того как курсор определен, его можно открыть.

Курсоры открываются с помощью оператора OPEN CURSOR.

При обработке оператора OPEN CURSOR выполняется запрос, и выборка данных сохраняется для последующих просмотра и прокрутки.

Теперь доступ к данным этого курсора может быть получен с помощью оператора FETCH. Оператор FETCH указывает строки, которые должны быть выбраны, откуда они должны быть выбраны и где их следует сохранить (имя переменной, например).

После использования курсоров их нужно закрывать. Кроме того, в некоторых СУБД (таких как SQL Server) требуется, чтобы ресурсы, занятые курсором, были освобождены явным образом

```
CLOSE CustCursor
```

Для закрытия курсора используется оператор CLOSE; после того как курсор закрыт, его нельзя использовать, не открыв перед этим вновь. Однако его не нужно объявлять заново при повторном использовании, достаточно оператора OPEN.

Хранимая процедура — объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Хранимые процедуры похожи на определяемые пользователем функции (UDF). Основное различие заключается в том, что пользовательские функции можно использовать как и любое другое выражение в SQL запросе, в то время как хранимые процедуры должны быть вызваны с помощью функции CALL:

CALL процедура (...) или
EXECUTE процедура(...)

Хранимые процедуры могут возвращать множества результатов, то есть результаты запроса SELECT. Такие множества результатов могут обрабатываться, используя курсоры, другими сохраненными процедурами, возвращая указатель результирующего множества, либо же приложениями. Хранимые процедуры могут также содержать объявленные переменные для обработки данных и курсоров, которые позволяют организовать цикл по нескольким строкам в таблице. Стандарт SQL предоставляет для работы выражения IF, LOOP, REPEAT, CASE и многие другие. Хранимые процедуры могут принимать переменные, возвращать результаты или изменять переменные и возвращать их, в зависимости от того, где переменная объявлена.

Реализация хранимых процедур варьируется от одной СУБД к другой. Большинство крупных поставщиков баз данных поддерживают их в той или иной форме. В зависимости от СУБД, хранимые процедуры могут быть реализованы на различных языках программирования, таких, как SQL, Java, C или C++. Хранимые процедуры, написанные не на SQL могут самостоятельно выполнять SQLзапросы, а могут и не выполнять. Все более широкое использование хранимых процедур привело к появлению процедурных элементов в языке SQL стандарта SQL:1999 и SQL:2003 в части SQL/PSM. Это сделало SQL императивным языком программирования. Большинство СУБД предлагают собственные проприетарные и расширения производителя, сверх SQL/PSM.

Назначение и преимущества хранимых процедур

Хранимые процедуры позволяют повысить производительность, расширяют возможности программирования и поддерживают функции безопасности данных.

Вместо хранения часто используемого запроса, клиенты могут ссылаться на соответствующую хранимую процедуру. При вызове хранимой процедуры её содержимое сразу же обрабатывается сервером.

Кроме собственно выполнения запроса, хранимые процедуры позволяют также производить вычисления и манипуляцию данными — изменение, удаление, выполнять DDLоператоры (не во всех СУБД!) и вызывать другие хранимые процедуры, выполнять сложную транзакционную логику. Одиноединственный оператор позволяет вызвать сложный сценарий, который содержится в хранимой процедуре, что позволяет избежать пересылки через сеть сотен команд и, в особенности, необходимости передачи больших объёмов данных с клиента на сервер.

В большинстве СУБД при первом запуске хранимой процедуры она компилируется (выполняется синтаксический анализ и генерируется план доступа к данным). В дальнейшем её обработка осуществляется быстрее.

Возможности программирования

Созданную хранимую процедуру можно вызвать в любой момент, что обеспечивает модульность и стимулирует повторное использование кода. Последнее облегчает сопровождение базы данных, так как она становится изолированной от меняющихся бизнесправил. Модифицировать хранимую процедуру в соответствии с новыми правилами можно в любой момент. После этого все приложения, использующие её, автоматически придут в соответствие с новыми бизнесправилами без непосредственной модификации.

Для упрощения тестирования, независимости бизнеслогики приложений от СУБД существует подход, в котором СУБД выступает лишь в роли хранилища, с минимальным количеством хранимых процедур или полным отказе от них. При этом используется отображение программных сущностей бизнеслогики на хранилище.

Безопасность

Использование хранимых процедур позволяет ограничить или вообще исключить непосредственный доступ пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение хранимых процедур, обеспечивающих косвенный и строго

регламентированный доступ к данным. Кроме того, некоторые СУБД поддерживают шифрование текста (wrapping) хранимой процедуры.

Эти функции безопасности позволяют изолировать от пользователя структуру базы данных, что обеспечивает целостность и надежность базы.

Снижается вероятность таких действий как «внедрение SQLкода», поскольку хорошо написанные хранимые процедуры дополнительно проверяют входные параметры перед тем, как передать запрос СУБД.

```
-- Procedure to insert a new city
```

```
CREATE OR REPLACE FUNCTION add_city (city VARCHAR (70 ), state CHAR (2))
RETURNS void AS $$
BEGIN
INSERT INTO cities VALUES (city , state );
END ;
$$ LANGUAGE plpgsql;
```

Вы можете использовать инструкцию Select для вызова процедуры add_city:

```
-- Add a new city
SELECT add_city ('St.Louis' , 'MO' );
```

Триггеры

Триггер — это особая разновидность хранимой процедуры, выполняемая автоматически при возникновении события на сервере базы данных. В основном используются для поддержания согласованности данных в БД. Триггеры языка обработки данных выполняются по событиям, вызванным попыткой пользователя изменить данные с помощью языка обработки данных. Событиями DML являются процедуры INSERT, UPDATE или DELETE, применяемые к таблице или представлению.

FOR | AFTER

Тип AFTER указывает, что триггер DML срабатывает только после успешного выполнения всех операций в инструкции SQL, запускаемой триггером. Все каскадные действия и проверки ограничений, на которые имеется ссылка, должны быть успешно завершены, прежде чем триггер сработает.

Если единственным заданным ключевым словом является FOR, аргумент AFTER используется по умолчанию.

INSTEAD OF

Указывает, что триггер DML срабатывает вместо инструкции SQL, используемой триггером, переопределяя таким образом действия выполняемой инструкции триггера. При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: inserted и deleted. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется триггер. Для каждого триггера создается свой комплект таблиц inserted и deleted, поэтому никакой другой триггер не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение триггера, содержимое таблиц inserted и deleted может быть разным: команда INSERT — в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни одной строки; после завершения триггера все строки из таблицы inserted переместятся в исходную таблицу; команда DELETE — в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; триггер может проверить каждую строку и определить, разрешено ли ее удаление; в таблице inserted не

окажется ни одной строки; команда UPDATE – при ее выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении триггера. Новые значения строк содержатся в таблице inserted. Эти строки добавятся в исходную таблицу после успешного выполнения триггера.

Для получения информации о количестве строк, которое будет изменено при успешном завершении триггера, можно использовать функцию @@ROWCOUNT; она возвращает количество строк, обработанных последней командой.

Если Триггер обнаружил, что из 100 вставляемых, изменяемых или удаляемых строк только одна не удовлетворяет тем или иным условиям, то никакая строка не будет вставлена, изменена или удалена. Такое поведение обусловлено требованиями транзакции – должны быть выполнены либо все модификации, либо ни одной. Триггер выполняется как неявно определенная транзакция, поэтому внутри триггера допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения триггера и отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду ROLLBACK TRANSACTION.

Нельзя:

1. Триггер не может использовать инструкцию CALL, чтобы вызвать сохраненные процедуры, которые возвращают данные пользователю или применяют динамический SQL.
2. Триггер не может использовать инструкции, которые явно или неявно начинают или заканчивают транзакцию, типа START TRANSACTION, COMMIT или ROLLBACK.
3. Вы не можете связывать триггер с VIEW или таблицей TEMPORARY.
4. Ключевые слова OLD и NEW дают возможность обратиться к столбцам в строках, на которые воздействует триггер. OLD и NEW не чувствительны к регистру. В триггере INSERT может использоваться только NEW.col_name : не имеется никакой старой строки. В триггере DELETE не ожидается никакой новой строки, так что может использоваться исключительно OLD.col_name. В триггере UPDATE Вы можете использовать OLD.col_name, чтобы обратиться к столбцам строки прежде, чем они изменятся, и NEW.col_name , чтобы обратиться к ним уже после внесения изменений.
5. Триггер не может возвращать результат, т.е. например, выводить SELECT.
6. Нельзя привязать к одной таблице два триггера с одинаковым временем и типом события.

Типы триггеров:

- **Триггеры DML** — Триггеры DML часто используются для применения бизнесправил и обеспечения целостности данных. В SQL Server декларативное ограничение ссылочной целостности обеспечивается инструкциями ALTER TABLE и CREATE TABLE. Однако декларативное ограничение ссылочной целостности не обеспечивает ссылочную целостность между базами данных. Ограничение ссылочной целостности подразумевает выполнение правил связи между первичными и внешними ключами таблиц. Для обеспечения ограничений ссылочной целостности используйте в инструкциях ALTER TABLE и CREATE TABLE ограничения PRIMARY KEY и FOREIGN KEY. Если ограничения распространяются на таблицу триггера, они проверяются после срабатывания триггера INSTEAD OF и до выполнения триггера AFTER. В случае нарушения ограничения выполняется откат действий триггера INSTEAD OF, и триггер AFTER не срабатывает.

- **Триггеры DDL** — как и стандартные триггеры, выполняют хранимые процедуры в ответ на какое-либо событие. В отличие от стандартных триггеров, они не срабатывают в ответ на выполнение инструкций UPDATE, INSERT или DELETE по отношению к таблице или представлению. Вместо этого триггеры срабатывают в первую очередь в ответ на инструкции языка определения данных (DDL). Это инструкции CREATE, ALTER, DROP, GRANT, DENY, REVOKE и UPDATE STATISTICS. Системные хранимые процедуры, выполняющие операции, подобные операциям DDL, также могут запускать триггеры DDL.
- **Триггеры входа** — выполняют хранимые процедуры в ответ на событие LOGON. Это событие вызывается при установке пользовательского сеанса с экземпляром SQL Server. Триггеры входа срабатывают после завершения этапа проверки подлинности при входе, но перед тем, как пользовательский сеанс реально устанавливается. Следовательно, все сообщения, которые возникают внутри триггера и обычно достигают пользователя, такие как сообщения об ошибках и сообщения от инструкции PRINT, перенаправляются в журнал ошибок SQL Server.

Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]

<method_specifier> ::=
    assembly_name.class_name.method_name
```

Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE, or UPDATE STATISTICS statement (DDL Trigger)

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier > [ ; ] }

<ddl_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

Trigger on a LOGON event (Logon Trigger)

```
CREATE TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR| AFTER } LOGON
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier > [ ; ] }

<logon_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

Пример (dml):

```
CREATE TRIGGER reminder2 ON Sales.Customer
AFTER INSERT, UPDATE, DELETE
AS
    EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'AdventureWorks2012 Administrator',
        @recipients = 'danw@Adventure-Works.com',
        @body = 'Don''t forget to print a report for the sales force.',
        @subject = 'Reminder';
```

9. Виды архитектур ядер операционных систем. Монолитная, многослойная и микроядерная архитектура ядра ОС. Другие виды ядер ОС

Виды архитектур ядер ОС:

- Монолитное ядро
- Многослойная монолитная архитектура
- Микроядерная архитектура
- Наноядерная архитектура
- Экзоядро
- Гибридные ядра

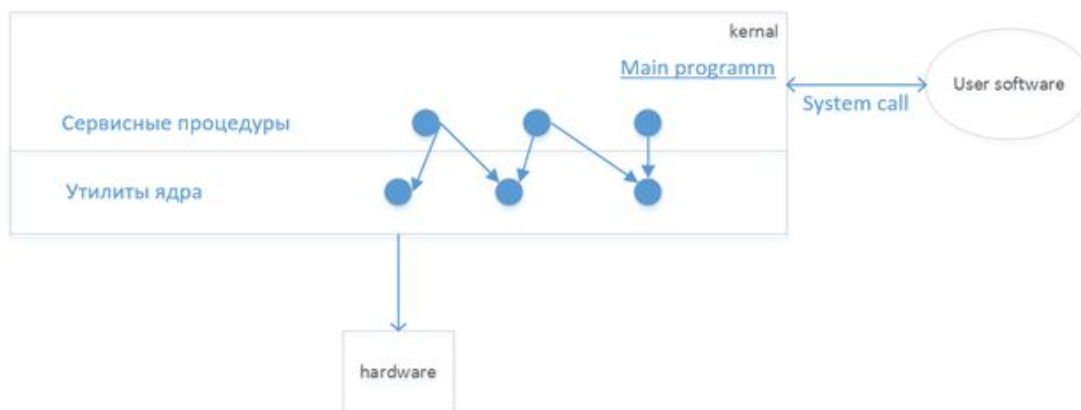
Ядро ОС: код ядра выполняется в привилегированном режиме.

Монолитное ядро

Все процедуры видят друг друга

Минимальные потребляемые ресурсы

Самые компактные ядра



Слои монолитного ядра

+ весь код уходит только на полезную нагрузку

+ высокая скорость работы

- сбой работы в одном из компонентов может нарушить работу всей системы

- монолитные ядра зависят от архитектуры (для каждой новой архитектуры ядро приходится перекомпилировать)

- ядра очень часто большие и сложные в обслуживании

- нет возможности динамического управления драйверами

Многослойная монолитная структура

Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем любого типа. В соответствии с этим подходом система состоит из иерархии слоев, каждый слой обслуживает вышележащий, выполняя для него набор функций, которые образуют межслойный интерфейс. На основе функций нижележащего слоя вышележащий слой строит свои более сложные функции. Между слоями существуют строгие правила взаимодействия, а между модулями внутри слоя связи могут быть произвольными. Отдельный модуль может выполнить свою работу либо самостоятельно, либо обратиться к другому модулю своего слоя, либо обратиться за помощью к нижележащему слою через межслойный интерфейс.

1 – средства аппаратной поддержки ядра:

Аппаратная поддержка прерываний,
Средства поддержки привилегированного режима,
Поддержка виртуальной памяти и ее защиты,
Реализация смены контекста.

2 – машинно-зависимые модули (HAL), драйвера:

Абстракция от железа.

3 – базовые механизмы ядра:

Исполнительный уровень.

4 – менеджер ресурсов

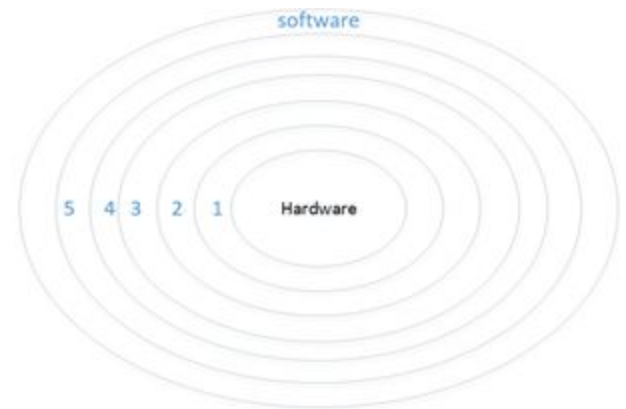
5 – слой системных вызовов

взаимодействие с software

+ минимизация издержек на переключение на режим ядра

+ высокая надежность

- ресурсоемкость



Микроядерная архитектура

Является альтернативой классическому многослойному способу построения операционных систем и состоит в перенесении значительной части системного кода на уровень пользователя и одновременной минимизацией ядра. Взаимодействие между программами операционной системы обеспечивает специальный модуль ядра – микроядро. Микроядро работает в привилегированном режиме и обеспечивает взаимодействие между программами, планирование использования процессора, первичную обработку прерываний, операции ввода-вывода и базовое управление памятью. Остальные компоненты системы взаимодействуют друг с другом путем передачи сообщений через микроядро.

Микроядро содержит следующие слои:

Hardware

Слой аппаратной поддержки ядра

HAL (драйверы)

Базовые механизмы ядра

+ экономия памяти

+ гибкость.

- снижение надежности

- большие накладные расходы на обращение к ядру.

Другие виды архитектур

Наноядерные – ядро только на обработку прерываний (встроенные системы)

Экзоядерные – в режиме ядра работают только менеджеры ресурсов

Гибридные – возможность менять количество слоев ядра

10. Планирование процессов. Критерии эффективности и свойства методов планирования процессов, параметры планирования процессов. Приоритетное и неприоритетное планирование, гарантированное планирование, многоуровневые очереди

Виды планирования:

1. Краткосрочное (между готовностью и исполнением - определение какому процессу сейчас дать выполняться)
2. Долгосрочное (между рождением и готовностью - если мы даем процессу родиться, то состояние операционной системой изменится)
3. Среднесрочное (ожидание разделяется на ожидание в оперативной памяти и вне ОП; готовность — на готовность в ОП и на готовность вне ОП)
4. Планирование отдельных очередей ввода/вывода (несколько процессов одновременно обращаются к жесткому диску? В очередь их!)

Критерии планирования

1. Критерий справедливости — каждому процессу гарантируется либо равный доступ к ресурсу, либо доступ пропорциональный какому-либо параметру
2. Эффективность использования ресурсов
3. Сокращение полного времени выполнения (времени между рождением процесса и его завершением; обычно вводят среднее время по всем процессам)
4. Сокращение времени ожидания
5. Сокращение времени отклика (Работает только для интерактивных процессов и в основном в системах реального времени. Реакция процесса на обращения должна быть минимальна по времени)

Свойства методов планирования

1. Предсказуемость — используя один и тот же алгоритм при одних и тех же условиях мы получим один и тот же результат.
2. Масштабируемость на максимально большое число входных данных (параметров)
3. Минимальные накладные расходы

Параметры планирования

	Параметры вычислительной системы	Параметры отдельного процесса
Статические	Не изменяются в рамках существующего сеанса ОС (до перезагрузки, завершения работы). Аппаратные характеристики, память	Исходный приоритет, права доступа
Динамические	Свободные ресурсы и текущее их состояние	CPU-burst — время непрерывного использования процессора I/O-burst —

		время непрерывного нахождения в ожидании
--	--	--

Дисциплина обслуживания — правило, по которому в некоторый момент времени выбирается процесс.

Без управления приоритетами:

1. FCFS — first come, first served — самый простой вариант, обычная очередь, нересурсоемкий

		CPU-burst	Пессимистичный сценарий
I	p ₀	13	uuuuuuuuuuuu
II	p ₂	4	uuuu
III	p ₃	1	u

$$t_{full} = 18 \text{ (полное время выполнения)}$$

$$\hat{t}_s = (0 + 13 + 17) / 3 = 10 \text{ (среднее время ожидания)}$$

$$\hat{t}_f = (13 + 17 + 18) / 3 = 16 \text{ (среднее полное время выполнения)}$$

		CPU-burst	Оптимистичный сценарий
I	p ₀	1	u
II	p ₂	4	uuuu
III	p ₃	13	uuuuuuuuuuuu

$$t_{full} = 18$$

$$\hat{t}_s = (0 + 1 + 5) / 3 = 2$$

$$\hat{t}_f = (1 + 5 + 18) / 3 = 8$$

2. RR (Round Robin (карусель такая)) — вытесняющий аналог FCFS Дается фиксированный квант времени (k), потом прерывание

		CPU-burst	k = 4
I	p ₀	13	uuuuuuuuuu
II	p ₂	4	uuuu
III	p ₃	1	u

$$t_{full} = 18$$

$$\hat{t}_s = (5 + 4 + 8) / 3 = 5,6$$

$$\hat{t}_f = (18 + 8 + 9) / 3 = 11,6$$

		CPU-burst	k = 1
I	p ₀	13	u u u u u u u u u u u u u
II	p ₂	4	u u u u
III	p ₃	1	u

$$t_{full} = 18$$

$$\hat{t}_s = (5 + 5 + 2) / 3 = 4$$

$$\hat{t}_f = (18 + 9 + 3) / 3 = 10$$

Чем меньше квант времени, тем оптимистичней сценарий, но тем больше накладные расходы на переключение между процессами. В среднем, этот алгоритм лучше, чем FCFS. (пессимистичный сценарий лучше, оптимистичный не хуже)

3. SJF — shortest job first (с внутренним вычислением приоритетов)

Невытесняющий SJF — выбирается процесс с меньшим CPU-burst

	t рождения	CPU-burst			
p0	0	6		uuuuuu	
p2	2	2		uu	
p3	6	7			uuuuuuuu
p4	0	5	uuuuu		

Вытесняющий SJF

	t рождения	CPU-burst	k = 2		
p0	0	6		uuuuuu	
p2	2	2	uu		
p3	6	7			uuuuuuuu
p4	0	5	uu	uuu	

Гарантированное планирование

Пусть, у нас есть N пользователей.

$T_i : i \in [1 .. N]$ — время нахождения i-го пользователя в системе

$t_i : i \in [1 .. N]$ — время исполнения программ i-го пользователя

Время будет распределяться справедливо, если $t_i = T_i / N$

$R = t_i * N / T_i$ — коэффициент справедливости.

Систему с таким планированием просто обдурить. Логинишься, развлекаешься пару часов, запускаешь свою программу, тебе дают много процессорного времени.

Приоритетное планирование

- Невытесняющее планирование, использующее модель приоритезации

	t рождения	CPU-burst	приоритет			
p0	0	6	4			uuuuuu
p2	2	2	3		uu	
p3	6	7	2			uuuuuuuu
p4	0	5	1	uuuuu		

- Вытесняющее планирование, использующее модель приоритезации.

	t рождения	CPU-burst	приоритет			
p0	0	6	4			uuuuuu
p2	2	2	3		u	u
p3	6	7	2		uuuuuuuu	
p4	0	5	1	uuuuu		

Вытеснение происходит и по прошествию кванта времени, и при появлении процесса с более высоким приоритетом '-' — низкоприоритетный процесс может вообще никогда не запуститься.

- Вертикальный лифт

Если какой-либо процесс не может выполняться из-за своего низкого приоритета, то тогда ему повышают приоритет на 1. Так происходит до тех пор, пока процесс-таки не отработает свой квант. После этого его приоритет приобретает первоначальное значение. Медленно, но верно такой процесс будет выполняться.

Многоуровневые очереди

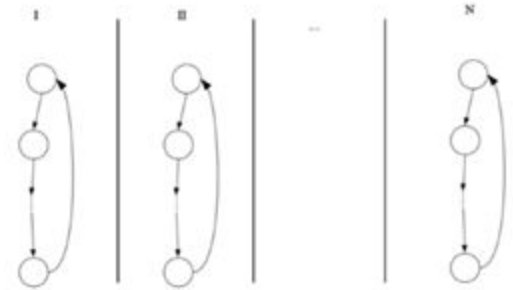
Несколько очередей, все процессы ставятся в эти очереди, и внутри этой очереди реализуется вытесняющее планирование (RR)

Если в первой очереди процессы завершились, начинается обработка 2-ой.

Многоуровневая очередь с обратной связью (Multi-level feedback queue)

Представим, что каждой очереди мы сопоставляем определенный квант времени. Изначально система ставит родившиеся процессы в самую приоритетную очередь. Там процесс выполняется один квант. Если этот процесс является хорошим (ему хватило времени выполниться), то мы оставляем его в этой очереди. Если он выполняется дольше этого времени

(является плохим), то его перемещают в очередь с более низким приоритетом. И так далее, пока он не уйдет в последнюю очередь. Если процесс попал в очередь 3 (например, там дается 32 мс) и 3 раза подряд уложился в 16 мс, то его переместят во 2-ю очередь (16 мс).



11. Проблемы взаимодействующих процессов. Алгоритмы реализации взаимоисключений. Семафоры Дейкстра. Решение проблемы «производитель- потребитель» с помощью семафоров

Проблемы:

1. Взаимоисключение — существует неразделяемый ресурс. Необходимо обеспечить невозможность одновременного доступа двумя разными процессами к нему. Неразделяемый ресурс — ресурс, который может быть использован только одним процессом в одно время.
2. Тупики (взаимоблокировки)

	R ₁	R ₂
P ₀	t ₁	t ₂
P ₁	t ₂	t ₁

3. 'Голодание' процессов — ситуация, когда для одного процесса долго не предоставляется ресурс, предоставляющийся другим.

Алгоритмы реализации взаимоисключений.

Допустим, что существует неразделяемый ресурс. Необходимо обеспечить невозможность одновременного доступа нескольких процессов к нему.

Критическая секция — фрагмент кода, в котором происходит обращение к неразделяемому ресурсу.

Race condition (соревнование) — несколько процессов гарантированно столкнутся у критической секции. Пролог критического входа — критическая секция — эпилог.

Пролог. В нем сообщается, что процесс вступает в критическую секцию и проверяет, чтобы в критической секции не было других процессов.

Эпилог. В нем сообщается о том, что процесс покинул критическую секцию.

Алгоритмы:

1. Запрещение прерываний.

Поскольку центральный процессор переключается с одного процесса на другой в результате таймерных или какие-нибудь других прерываний, то при выключенных прерываниях он не сможет переключиться на другой процесс. Поскольку процесс запретил прерывания, он может исследовать и обновлять общую память, не опасаясь вмешательства со стороны любого другого процесса.

Неразумно давать пользовательским процессам полномочия выключать все прерывания. Если один из них выключит и не включит прерывания, то система упадет. Если иметь дело с многопроцессорной системой, то запрет прерываний коснется только того процессора, на котором выполняется запретительная инструкция. Все остальные процессоры будут работать в штатном режиме и смогут обращаться к общей памяти.

Может использоваться ядром ОС для выполнения нескольких инструкций, например, при обновлении переменных или списков.

2. Использование блокирующих переменных.

Shared int lock = 0;

Process()

```
while (lock == 1);           \
lock = 1;                     / пролог
{ critical section }
lock = 0;                     > эпилог
```

При возникновении состязательной ситуации возможен случай, при котором два процесса могут одновременно оказаться в критической области. Предположим, что один процесс считывает значение блокирующей переменной и видит, что оно равно нулю. Перед тем, как он сможет установить значение в единицу, планировщик запускает другой процесс, который устанавливает значение в единицу. Когда возобновляется выполнение первого процесса, он также установит значение блокирующей переменной в единицу, и два процесса одновременно окажутся в своих критических областях.

3. Алгоритм строгого чередования

Shared int turn = 0;

Process()

```
while (turn != i);
{ critical section }
turn = 1 - i;
```

Алгоритм строгого чередования можно использовать в той ситуации, если два процесса будут входить в свои критические области, строго по очереди.

Возможна ситуация, при которой один процесс будет заблокирован другим процессом, который находится вне критической области. Допустим, первый процесс освободил критическую область, дав возможность другому процессу в нее войти. Но другой процесс уже больше не входит в эту критическую область. => первый процесс оказывается заблокирован. При возможности возникновения такой ситуации, использование этого алгоритма недопустимо.

4. Массив флагов (флаги готовности)

Shared int ready[2] = {0,0};

Process()

```
ready[i] = 1;
while (ready[1-i] == 1);
{ critical section }
ready[i] = 0;
```

Возможна ситуация, в которой оба процесса окажутся в вечном ожидании входа в свои критические области. Допустим, что первый процесс подошел к критической области, установил флаг в 1, а затем планировщик передал управление второму процессу, который также устанавливает свой флаг в единицу. Далее оба процесса не могут войти в критическую секцию, бесконечно ожидая установления флага процесса-конкурента в ноль.

Алгоритм Петтерсона (алгоритм взаимной вежливости)

```
Shared int ready[2] = {0,0};
```

```
Shared int turn;
```

```
Process()
```

```
    ready[i] = 1;
```

```
    turn = 1 - i;
```

```
    while (ready[1-i] == 1 && turn == 1 - i);
```

```
    { critical section }
```

```
    ready[i] = 0;
```

Допустим, ни один процесс не находится в критической секции. Затем 0-й процесс устанавливает `ready[0] = 1`; и `turn = 1`; . Поскольку `ready[1] = 1` 0-й, процесс условие в `while` ложно, и процесс уходит в критическую область. Затем, если 1-й процесс захочет войти в критическую область, то он будет ожидать до тех пор, пока `ready[1 - 1]` не станет нулем, то есть 0-й процесс не выйдет из критической секции.

Допустим, оба процесса практически одновременно подошли к критической секции. Они оба будут сохранять номер процесса-конкурента в переменной `turn`. В расчет будет браться последнее сохранение, поскольку первое будет затерто новым. Допустим, 1-й процесс сохранил 0 в `turn` последним. Когда оба процесса доберутся до оператора `while`, 0-й процесс не выполнит его ни одного раза и войдет в свою критическую область. 1-й процесс войдет в цикл и не будет входить в свою критическую область до тех пор, пока процесс 0 не выйдет из своей критической области.

5. Test and Set Lock (проверь и установи блокировку)

```
Shared int lock = 0;
```

```
Process()
```

```
    while (TestAndSet(&lock));
```

```
    { critical section }
```

```
    lock = 0;
```

TSL считывает содержимое слова памяти `lock` в регистр, а по адресу памяти, отведенному для `lock`, записывает ненулевое значение. При этом гарантируется неделимость операций чтения слова и сохранения в нем нового значения — никакой другой процесс не может получить доступ к слову в памяти, пока команда не завершит свою работу. Центральный процессор, выполняющий команду TSL, блокирует шину памяти, запрещая другим центральным процессорам доступ к памяти до тех пор, пока не будет выполнена эта команда

TSL — атомарная операция.

6. Семафоры Дейкстра

Дейкстра предложил использовать целочисленную переменную для подсчета количества активизация, отложенных на будущее. Он предложил учредить новый тип переменной — семафор. Значение семафора может быть равно 0, что будет свидетельствовать об отсутствии сохраненных

активизаций или иметь какое-нибудь положительное значение, если ожидается не менее одной активизации.

Также Дейкстра предложил использовать две операции: P и V.

P(S) (Proberen (пытаться), down) — атомарная операция, проверяющая значение семафора, при ненулевом значении уменьшает его на единицу, а при нулевом приостанавливает выполнение процесса, не завершая в этот раз операцию P.

V(S) (Verhogen (поднимать), up) — операция, увеличивающая значение, адресуемое семафором, на 1. Если с этим семафором связаны один или более приостановленных процессов, способных завершить ранее начатые операции P, то система выбирает один из них и позволяет ему завершить его операцию P. Операция увеличения значения семафора на 1 и активизации одного из процессов также является атомарной.

Мьютекс — семафор принимающий значения 0 и 1

7.Мониторы Хоара

Монитор представляет собой коллекцию переменных и структур данных, сгруппированных вместе в специальную разновидность модуля или пакета процедур. Процессы могут вызывать любые необходимые им процедуры, имеющиеся в мониторе, но они не могут получить непосредственный доступ к внутренним структурам данных монитора из процедур, объявленных за пределами монитора.

В любой момент времени в мониторе может быть активен только один процесс. Мониторы являются конструкцией языка программирования, поэтому компилятор осведомлен об их особенностях и способен обрабатывать вызовы процедур монитора не так, как вызовы всех остальных процедур. Обычно при вызове процессом процедуры монитора первые несколько команд процедуры осуществляют проверку на текущее присутствие активности других процессов внутри монитора. Если какой-нибудь процесс будет активен, вызывающий процесс будет приостановлен до тех пор, пока другой процесс не освободит монитор. Если монитор никаким другим процессом не используется, вызывающий процесс может в него войти.

Решение проблемы «производитель-потребитель» с помощью семафоров.

Пусть два процесса используют общий буфер фиксированного размера. Один из них, производитель, помещает информацию в буфер, а другой, потребитель, извлекает ее оттуда. Проблемы возникают в тот момент, когда производителю требуется поместить новую запись в уже заполненный буфер. Решение заключается в блокировании производителя до тех пор, пока потребитель не извлечет как минимум одну запись. Также, если потребителю нужно извлечь запись из буфера, и он видит, что буфер пуст, он блокируется до тех пор, пока производитель не поместит что-нибудь в буфер и не активизирует этого потребителя.

Семафор mutex используется для организации взаимного исключения. Семафоры full и empty используются для синхронизации. Они гарантируют, что производитель приостановит свою работу при заполненном буфере, а потребитель приостановит свою работу, если этот буфер опустеет.

semaphore mutex = 1;

semaphore empty = N;

```
semaphore full = 0;
```

```
producer()
```

```
    while (TRUE)
        item = produce_item();
        P(&empty);
        P(&mutex);
        put_item(item);
        V(&mutex);
        V(&full);
```

```
consumer()
```

```
    while (TRUE)
        P(&full);
        P(&mutex);
        item = get_item();
        V(&mutex);
        V(&empty);
```

```
consume_item(item);
```

12. Принципы управления памятью вычислительной системы. Виртуальная память и преобразование адресов. Методы распределения оперативной памяти без использования внешней памяти. Страничная и сегментно-страничная организация виртуальной памяти

Принципы управления памятью вычислительной системы

1. Абстрагирование памяти.

Адресное пространство — это набор адресов, который может быть использован процессом для обращения к памяти. У каждого процесса имеется свое собственное адресное пространство, независимое от того адресного пространства, которое принадлежит другим процессам. Понятие адресного пространства создает своеобразную абстрактную память, в которой существуют программы.

2. Функции ОС по управлению памятью:

1. Мониторинг свободной и занятой памяти
2. Первоначальное и динамическое выделение памяти процессам, освобождение памяти
3. Настройка адресов программы на конкретную область физической памяти
4. Полное или частичное вытеснение данных процессов во внешнюю память и возвращение их обратно
5. Защита памяти
6. Дефрагментация памяти

3. Использование внешних накопителей

1. swapping — выгрузка и загрузка процесса целиком на внешний накопитель и обратно.
 - + Не нужно решать проблему замены адресов?
 - Низкая производительность
 - Не факт, что процесс целиком влезет в RAM
 - + отсутствует проблема переадресации адресов (вычисление реальных адресов при загрузке в оперативную память, а не вычисление адреса при каждом обращении, используя системные вызовы)
2. Виртуальная память — часть процесса находится в RAM, а часть на внешнем накопителе
3. Способы организации
 1. Использование файла подкачки
 - + Неограниченность файла
 - Если диск забит, то все плохо
 - Производительность
 2. Использование раздела подкачки
 - Размер ограничен
 - + Производительность

4. Преобразование адресов.

Связывание логического адреса, порожденного оператором программы, с физическим должно быть осуществлено до начала выполнения оператора или в момент его выполнения. Таким

образом, привязка инструкций и данных к памяти в принципе может быть сделана на следующих шагах .

1. Этап компиляции (Compile time). Когда на стадии компиляции известно точное место размещения процесса в памяти, тогда непосредственно генерируются физические адреса. При изменении стартового адреса программы необходимо перекомпилировать ее код.

2. Этап загрузки (Load time). Если информация о размещении программы на стадии компиляции отсутствует, компилятор генерирует перемещаемый код. В этом случае окончательное связывание откладывается до момента загрузки. Если стартовый адрес меняется, нужно всего лишь перезагрузить код с учетом измененной величины.

3. Этап выполнения (Execution time). Если процесс может быть перемещен во время выполнения из одной области памяти в другую, связывание откладывается до стадии выполнения. Здесь желательно наличие специализированного оборудования, например, регистров перемещения. Их значение прибавляется к каждому адресу, сгенерированному процессом.



Методы распределения оперативной памяти без использования внешней памяти.

1. Фиксированные разделы

1. Равные

1. Оверлейное программирование (необходимо использовать, если программа не влезает в блок (ответственность на программисте))
2. Конечный набор процессов
3. Нерациональное использование памяти

2. Неравные

1. Не всегда известно количество необходимой памяти

2. Динамические разделы

3. Перемещаемые разделы

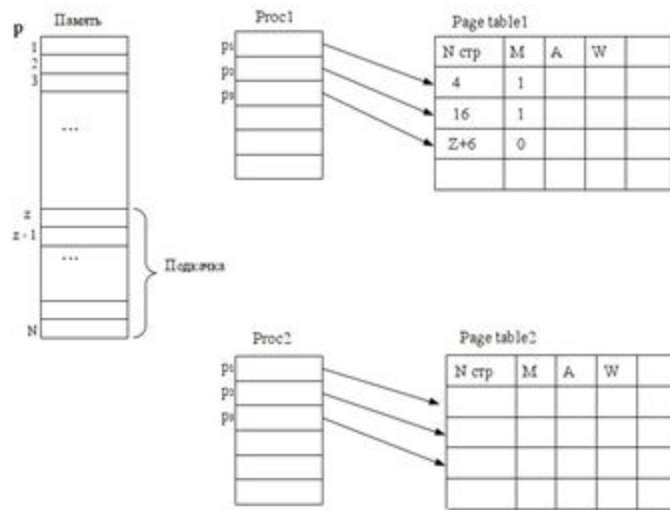
1. Для каждого процесса хранится 2 адреса: базовый и граничный
2. Когда перемещаем?
 1. Необходимость в месте для процесса (нужно один раз подождать)

2. Поддержка фрагментации на приемлемом уровне (система как бы подвисает) (раз в период времени выбирает процесс для перемещения и перемещает)



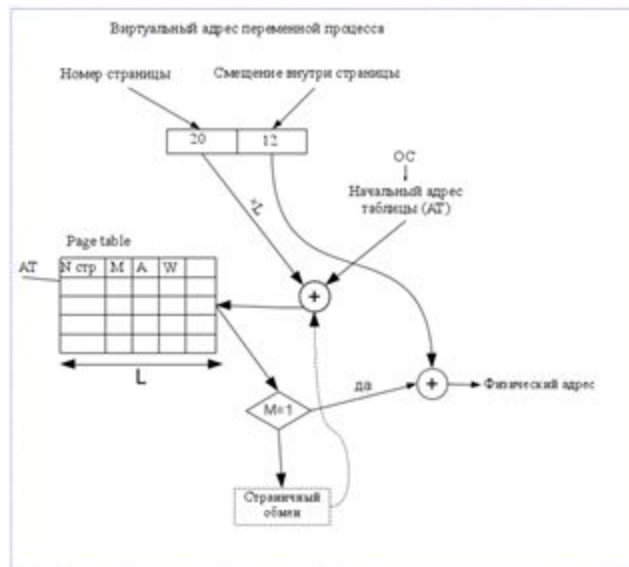
Страничная организация виртуальной памяти.

Память разбивается на блоки (страницы). Размер блока — 1КБ|2КБ|4КБ. Может достигать 128КБ.



Столбцы таблицы

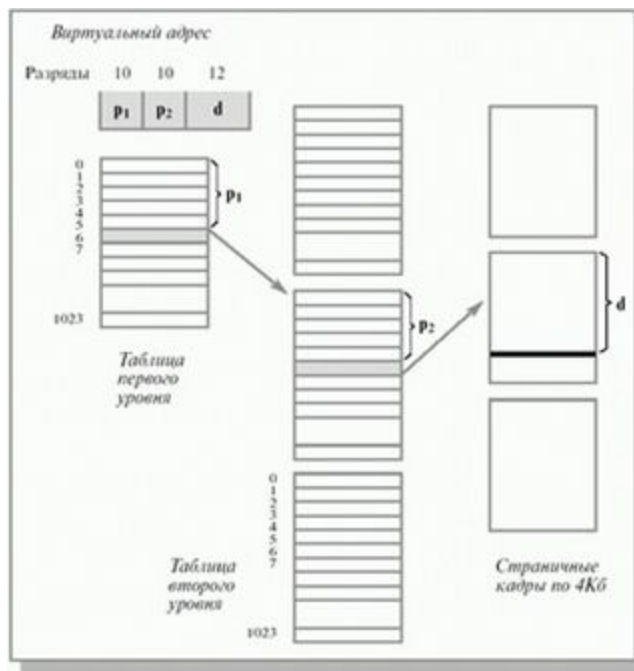
1. N ссылается на физическое пространство
2. M- в физической памяти / в подкачке?
3. A — обращались ли к фрагменту памяти после последней операции страничного обмена?
4. W — происходило ли изменение данных на этой странице?



Это происходит при каждом обращении к памяти => необходима оптимизация.

Используется кэширование (аппаратное решение (L3-кэш))

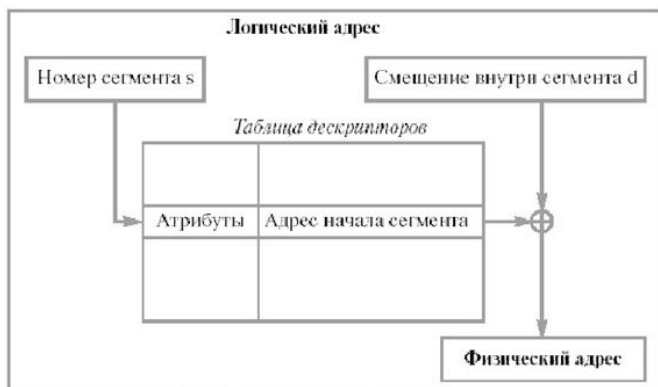
Используется иерархия страниц (разбиение таблицы страниц на таблицы размером в одну страницу, над которыми есть таблица первого уровня).



Сегментно-страничная организация виртуальной памяти.

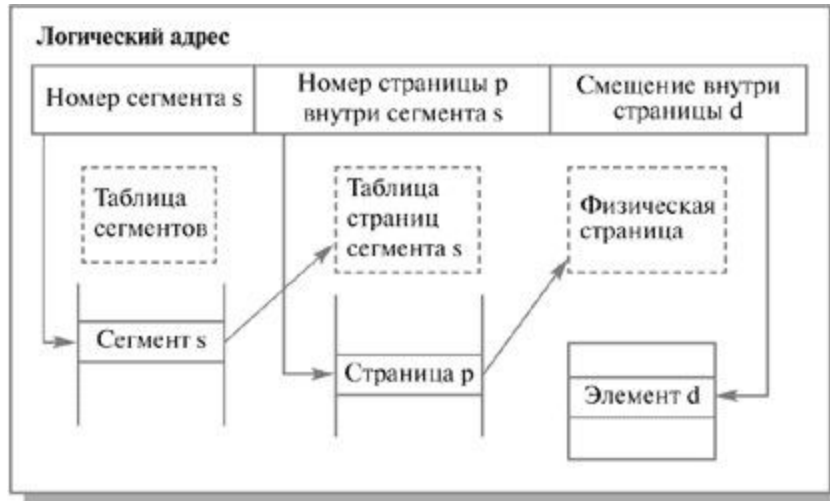
1. Сегменты, в отличие от страниц, могут иметь переменный размер. Идея сегментации изложена во введении. При сегментной организации виртуальный адрес является двумерным как для программиста, так и для операционной системы, и состоит из двух полей – номера сегмента и смещения внутри сегмента. Подчеркнем, что в отличие от страничной организации, где линейный адрес преобразован в двумерный операционной системой для удобства отображения, здесь двумерность адреса является следствием представления пользователя о процессе не в виде линейного массива байтов, а как набор сегментов переменного размера (данные, код, стек...).
2. Каждый сегмент – линейная последовательность адресов, начинающаяся с 0. Максимальный размер сегмента определяется разрядностью процессора (при 32-разрядной адресации это 232 байт или 4 Гбайт). Размер сегмента может меняться динамически (например, сегмент стека). В элементе таблицы сегментов помимо физического адреса начала сегмента обычно содержится и длина сегмента. Если размер смещения в виртуальном адресе выходит за пределы размера сегмента, возникает исключительная ситуация.

Логический адрес – упорядоченная пара $v=(s,d)$, номер сегмента и смещение внутри сегмента.



3.

4. При сегментно-страничной организации памяти происходит двухуровневая трансляция виртуального адреса в физический. В этом случае логический адрес состоит из трех полей: номера сегмента логической памяти, номера страницы внутри сегмента и смещения внутри страницы. Соответственно, используются две таблицы отображения – таблица сегментов, связывающая номер сегмента с таблицей страниц, и отдельная таблица страниц для каждого сегмента.



5.

13. Организация распределенных вычислительных систем. Алгоритмы распределения памяти. Алгоритмы распределения доступа к файлам. Синхронизация в распределенных системах

Распределенная система - совокупность независимых компьютеров, которая представляется пользователю единым компьютером. Примеры: сеть рабочих станций (выбор процессора для выполнения программы, единая файловая система), роботизированный завод (роботы связаны с разными компьютерами, но действуют как внешние устройства единого компьютера, банк со множеством филиалов, система резервирования авиабилетов.

Распределенные ОС - единый глобальный межпроцессный коммуникационный механизм, глобальная схема контроля доступа, одинаковое видение файловой системы. Вообще иллюзия единой ЭВМ.

Принципы построения распределенных ОС:

- Прозрачность для пользователей и программ:
 - Прозрачность расположения - Пользователь не должен знать, где расположены ресурсы;
 - Прозрачность миграции - Ресурсы могут перемещаться без изменения их имен;
 - Прозрачность конкуренции - Множество пользователей разделяет ресурсы автоматически;
 - Прозрачность параллелизма - Работа может выполняться параллельно без участия пользователя
- Гибкость (не все еще ясно потребуются менять решения) - использование в основе монолитного ядра ОС или микроядра;
- Надежность (доступность, устойчивость к ошибкам и одновременно секретность);
- Производительность (Грануллированность. Мелкозернистый и крупнозернистый параллелизм (finegrained parallelism, coarsegrained parallelism). Устойчивость к ошибкам требует дополнительных накладных расходов.);
- Масштабируемость
 - ни одна машина не имеет полной информации о состоянии системы;
 - машины принимают решения на основе только локальной информации;
 - выход из строя одной машины не должен приводить к отказу алгоритма;
 - не должно быть неявного предположения о существовании глобальных часов

В качестве памяти используется распределенная общая память (DSM - Distributed Shared Memory). Традиционно распределенные вычисления базируются на модели передачи сообщений, в которой данные передаются от процессора к процессору в виде сообщений. Удаленный вызов процедур фактически является той же самой моделью (или очень близкой).

DSM - виртуальное адресное пространство, разделяемое всеми узлами (процессорами) распределенной системы. Программы получают доступ к данным в DSM примерно так же, как они

работают с данными в виртуальной памяти традиционных ЭВМ. В системах с DSM данные перемещаются между локальными памятьми разных компьютеров аналогично тому, как они перемещаются между оперативной и внешней памятью одного компьютера.

Достоинства DSM:

- В модели передачи сообщений программист обеспечивает доступ к разделяемым данным посредством явных операций отправки и приема сообщений. При этом приходится квантовать алгоритм, обеспечивать своевременную смену информации в буферах, преобразовывать индексы массивов. Все это сильно усложняет программирование и отладку. DSM скрывает от программиста пересылку данных и обеспечивает ему абстракцию разделяемой памяти, к использованию которой он уже привык на мультипроцессорах. Программирование и отладка с использованием DSM гораздо проще.
- В модели передачи сообщений данные перемещаются между двумя различными адресными пространствами. Это делает очень трудным передачу сложных структур данных между процессами. Более того, передача данных по ссылке и передача структур данных, содержащих указатели, является в общем случае делом сложным и дорогостоящим. DSM же позволяет передавать данные по ссылке, что упрощает разработку распределенных приложений.
- Объем суммарной физической памяти всех узлов может быть огромным. Эта огромная память становится доступна приложению без издержек, связанных в традиционных системах с дисковыми обменами. Это достоинство становится все весомее в связи с тем, что скорости процессоров растут быстрее скоростей памяти и в то же время появляются очень быстрые коммуникации.
- DSM системы могут наращиваться практически беспредельно в отличие от систем с разделяемой памятью, т.е. являются масштабируемыми.
- Программы, написанные для мультипроцессоров с общей памятью, могут в принципе без каких-либо изменений выполняться на DSM системах (по крайней мере, они могут быть легко перенесены на DSM-системы). По существу, DSM-системы преодолевают архитектурные ограничения мультипроцессоров и сокращают усилия, необходимые для написания программ для распределенных систем. Обычно они реализуются программно-аппаратными средствами, но в последние годы появилось несколько коммерческих МПП с DSM, реализованной аппаратно (Convex SPP, KSR1).

При реализации DSM центральными являются следующие вопросы:

- как поддерживать информацию о расположении удаленных данных;
- как снизить при доступе к удаленным данным коммуникационные задержки и большие накладные расходы, связанные с выполнением коммуникационных протоколов;
- как сделать разделяемые данные доступными одновременно на нескольких узлах для того, чтобы повысить производительность системы.

Алгоритмы

Использование одного центрального сервера

Все разделяемые данные поддерживает центральный сервер. Он возвращает данные клиентам по их запросам на чтение, по запросам на запись он корректирует данные и посылает клиентам в ответ квитанции. Клиенты могут использовать таймаут для посылки повторных запросов при отсутствии ответа сервера. Дубликаты запросов на запись могут распознаваться путем нумерации запросов. Если несколько повторных обращений к серверу остались без ответа, приложение получит отрицательный код ответа (это обеспечит клиент).

Алгоритм прост в реализации, но сервер может стать узким местом. Чтобы избежать этого, разделяемые данные могут быть распределены между несколькими серверами. В этом случае клиент должен уметь определять, к какому серверу надо обращаться при каждом доступе к разделяемой переменной. Посылка запросов сразу всем серверам нежелательна, поскольку не снижает нагрузку на серверы. Лучшее решение распределить данные в зависимости от их адресов и использовать функцию отображения для определения нужного сервера.

Миграционный алгоритм

В отличие от предыдущего алгоритма, когда запрос к данным направлялся в место их расположения, в этом алгоритме меняется расположение данных они перемещаются в то место, где потребовались. Это позволяет последовательные обращения к данным осуществлять локально.

Миграционный алгоритм позволяет обращаться к одному элементу данных в любой момент времени только одному узлу. Обычно мигрирует целиком страницы или блоки данных, а не запрашиваемые единицы данных. Это позволяет воспользоваться присущей приложениям локальностью доступа к данным для снижения стоимости миграции. Однако, такой подход приводит к трэшингу, когда страницы очень часто мигрируют между узлами при малом количестве обслуживаемых запросов. Некоторые системы позволяют задать время, в течение которого страница насильно удерживается в узле для того, чтобы успеть выполнить несколько обращений к ней до миграции ее в другой узел.

Миграционный алгоритм позволяет интегрировать DSM с виртуальной памятью, обеспечивающейся операционной системой в отдельных узлах. Если размер страницы DSM совпадает с размером страницы виртуальной памяти (или кратен ей), то можно обращаться к разделяемой памяти обычными машинными командами, воспользовавшись аппаратными средствами проверки наличия в оперативной памяти требуемой страницы и замены виртуального адреса на физический. Конечно, для этого виртуальное адресное пространство процессоров должно быть достаточно, чтобы адресовать всю разделяемую память. При этом, несколько процессов в одном узле могут разделять одну и ту же страницу. Для определения места расположения блоков данных миграционный алгоритм может использовать сервер, отслеживающий перемещения блоков, либо воспользоваться механизмом подсказок в каждом узле. Возможна и широковещательная рассылка запросов.

Алгоритм размножения для чтения

Предыдущий алгоритм позволял обращаться к разделяемым данным в любой момент времени только процессам в одном узле (в котором эти данные находятся). Данный алгоритм расширяет миграционный алгоритм механизмом размножения блоков данных, позволяя либо многим узлам иметь возможность одновременного доступа по чтению, либо одному узлу иметь возможность читать и писать данные (протокол многих читателей и одного писателя).

Производительность повышается за счет возможности одновременного доступа по чтению, но запись требует серьезных затрат для уничтожения всех устаревших копий блока данных или их коррекции. При использовании такого алгоритма требуется отслеживать расположение всех блоков данных и их копий. Например, каждый собственник блока может отслеживать расположение его копий.

Данный алгоритм может снизить среднюю стоимость доступа по чтению тогда, когда количество чтений значительно превышает количество записей.

Алгоритм полного размножения

Этот алгоритм является расширением предыдущего алгоритма. Он позволяет многим узлам иметь одновременный доступ к разделяемым данным на чтение и запись (протокол многих читателей и многих писателей). Поскольку много узлов могут писать данные параллельно, требуется для поддержания согласованности данных контролировать доступ к ним. Одним из способов обеспечения консистентности данных является использование специального процесса для упорядочивания модификаций памяти. Все узлы, желающие модифицировать разделяемые данные должны посылать свои модификации этому процессу. Он будет присваивать каждой модификации очередной номер и рассылать его широкоэвентуально вместе с модификацией всем узлам, имеющим копию модифицируемого блока данных. Каждый узел будет осуществлять модификации в порядке возрастания их номеров. Разрыв в номерах полученных модификаций будет означать потерю одной или нескольких модификаций. В этом случае узел может запросить недостающие модификации.

Проблема распределения файлов

Распределение файлового сервера

Миграция файла целиком (копирование или перенос, в зависимости от алгоритма)

А давайте определенный хост будет интерфейсом управления определенного файла.

Тогда происходит перенаправление не файлов, а системных вызовов. »

Вопрос каталогов - что такое имя, если все может быть расположена на разных серверах

Единое пространство имен, однако обращение идет через единый узел.

Но можно сделать что бы в начале пути указывалось адрес хоста, на котором расположен файл

Что у нас тогда получается? Процессы хотят отправлять что-то друг другу и получать обратную информацию. Для этого существует набор команд

- Барьер - все кроме отправителя должны остановить свою работу
- Broadcast – один из набора процессов отправляет всем свои данные
- Scatter - если у одного из процессов есть набор команд, то он раскидывает их между процессами
- Gather - то же самое, но наоборот (соединяет процессы всех в одни)
- All Gather - собирает из всех ко всем
- All to All – все процессы ко всем

Синхронизация времени в распределенных системах

У нас в двух системах выполняется по одному процессу, но которые должны взаимодействовать между собой. Сложность в том, что время обмена может быть большим, а время выполнения малое, поэтому возникает вопрос кто отправил сообщение первый?

Идеи решения:

Отдельный сервер, который шлет всем одинаковые сигналы. Две проблемы - часы не должны ходить назад (надо ускорять или замедлять их для проведения коррекции) и ненулевое время прохождения сообщения о времени (можно многократно замерять время прохождения и брать среднее).

Многие распределенные алгоритмы требуют, чтобы один из процессов выполнял функции координатора, инициатора или некоторую другую специальную роль. Выбор такого специального процесса будем называть выбором координатора. При этом очень часто бывает не важно, какой именно процесс будет выбран. Можно считать, что обычно выбирается процесс с самым большим уникальным номером.

Могут применяться разные алгоритмы, имеющие одну цель - если процедура выборов началась, то она должна закончиться согласием всех процессов относительно нового координатора.

Алгоритм задиры

Если процесс обнаружит, что координатор очень долго не отвечает, то инициирует выборы. Процесс P проводит выборы следующим образом:

1. P посылает сообщение "ВЫБОРЫ" всем процессам с большими чем у него номерами.
2. Если нет ни одного ответа, то P считается победителем и становится координатором.
3. Если один из процессов с большим номером ответит, то он берет на себя проведение выборов. Участие процесса P в выборах заканчивается.

В любой момент процесс может получить сообщение "ВЫБОРЫ" от одного из коллег с меньшим номером. В этом случае он посылает ответ "ОК", чтобы сообщить, что он жив и берет проведение

выборов на себя, а затем начинает выборы (если к этому моменту он уже их не вел). Следовательно, все процессы прекратят выборы, кроме одного нового координатора. Он извещает всех о своей победе и вступлении в должность сообщением "КООРДИНАТОР". Если процесс выключился из работы, а затем захотел восстановить свое участие, то он проводит выборы (отсюда и название алгоритма).

Круговой алгоритм

Алгоритм основан на использовании кольца (физического или логического), но без маркера. Каждый процесс знает следующего за ним в круговом списке. Когда процесс обнаруживает отсутствие координатора, он посылает следующему за ним процессу сообщение "ВЫБОРЫ" со своим номером. Если следующий процесс не отвечает, то сообщение посылается процессу, следующему за ним, и т.д., пока не найдется работающий процесс. Каждый работающий процесс добавляет в список работающих свой номер и переправляет сообщение дальше по кругу. Когда процесс обнаружит в списке свой собственный номер (круг пройден), он меняет тип сообщения на "КООРДИНАТОР" и оно проходит по кругу, извещая всех о списке работающих и координаторе (процессе с наибольшим номером в списке). После прохождения круга сообщение удаляется.

14. Стандартизация систем. ГОСТ Р МЭК 62264-1-2010 Интеграция систем управления предприятием. Функциональная модель управления предприятием. Контур управления. Функции управления

Выпилен, т.к. Не будет спрашиваться

15. Процессы проектирования. Проектирование программной архитектуры. Модели описания программной архитектуры. Шаблоны программной архитектуры

Проектирование программной архитектуры:

Работу по проектированию программной архитектуры выполняет программный архитектор или их группа.

Любой программный продукт состоит из одного или нескольких компонентов, которые взаимодействуют между собой. В большинстве случаев системы являются именно многокомпонентными, в связи с чем приходится осуществлять проектирование не только на уровне внутреннего устройства каждой компоненты, но также учитывать и взаимодействие компонент друг с другом. Более формально, проектирование программной архитектуры системы начинается с проектирования отдельных ее компонентов с учетом межкомпонентного взаимодействия.

После того, как программная архитектура отдельных компонентов составлена, необходимо произвести оценивание результатов, путем сопоставления выделенной архитектуры системы техническому заданию. В случае выявления несоответствий архитектура отправляется на доработку. Если же решение удовлетворяет требованиям технического задания и является обоснованным, то оно отправляется на дальнейшую обработку.

Вслед за положительным оцениванием осуществляется создание единого документа «Описание программной архитектуры», который будет содержать информацию обо всех компонентах системы, требования к их разработке и использованию, а также обоснование выбора компонент и способов их взаимодействия.

Последним этапом является непосредственно разработка ПО. Если в процессе выяснилось, что разработанная архитектура нуждается в доработке, процесс возвращается на стадию проектирования отдельных компонент, дорабатывается и проходит все последующие стадии заново.

Описание программной архитектуры:

Языки описания архитектуры (ADLS) используются для описания архитектуры программного обеспечения. Различными организациями было разработано несколько различных ADLS, в том числе AADL (стандарт SAE), Wright, Acme, xADL, Darwin, DAOP-ADL, а также ByADL. Общими

элементами для всех этих языков являются понятия компонента, коннектора и конфигурации. Также, помимо специализированных языков, для описания архитектуры часто используется унифицированный язык моделирования UML. Диаграммы классов, развертывания, последовательностей.

Архитектура ПО обычно содержит несколько видов, которые аналогичны различным типам чертежей в строительстве зданий. В онтологии, установленной ANSI / IEEE 1471—2000, виды являются экземплярами точки зрения, где точка зрения существует для описания архитектуры с точки зрения заданного множества заинтересованных лиц.

Архитектурный вид состоит из 2 компонентов:

- Элементы
- Отношения между элементами

Архитектурные виды можно поделить на 3 основных типа:

- Модульные виды (англ. module views) — показывают систему как структуру из различных программных блоков.
- Компоненты-и-коннекторы (англ. component-and-connector views) — показывают систему как структуру из параллельно запущенных элементов (компонентов) и способов их взаимодействия (коннекторов).
- Размещение (англ. allocation views) — показывает размещение элементов системы во внешних средах.

Примеры модульных видов:

- Декомпозиция (англ. decomposition view) — состоит из модулей в контексте отношения «является подмодулем»
- Использование (англ. uses view) — состоит из модулей в контексте отношения «использует» (т.е. один модуль использует сервисы другого модуля)
- Вид уровней (англ. layered view) — показывает структуру, в которой связанные по функциональности модули объединены в группы (уровни)
- Вид классов/обобщений (англ. class/generalization view) — состоит из классов, связанные через отношения «наследуется от» и «является экземпляром»

Примеры видов компонентов-и-коннекторов:

- Процессный вид (англ. process view) — состоит из процессов, соединённых операциями коммуникации, синхронизации и/или исключения
- Параллельный вид (англ. concurrency view) — состоит из компонентов и коннекторов, где коннекторы представляют собой «логические потоки»
- Вид обмена данными (англ. shared-data (repository) view) — состоит из компонентов и коннекторов, которые создают, сохраняют и получают постоянные данные
- Вид клиент-сервер (англ. client-server view) — состоит из взаимодействующих клиентов и серверов и коннектором между ними (например, протоколов и общих сообщений)

Примеры видов размещения:

- Развертывание (англ. deployment view) — состоит из программных элементов, их размещения на физических носителях и коммуникационных элементов
- Внедрение (англ. implementation view) — состоит из программных элементов и их соответствия файловым структурам в различных средах (разработческой, интеграционной и т.д.)
- Распределение работы (англ. work assignment view) — состоит из модулей и описания того, кто ответственен за внедрение каждого из них

Примеры архитектурных шаблонов:

- Многоуровневый шаблон (Layered pattern). Система разбивается на уровни, которые на диаграмме изображаются один над другим. Каждый уровень может вызывать только уровень на 1 ниже него. Таким образом разработку каждого уровня можно вести относительно независимо, что повышает модифицируемость системы. Недостатками данного подхода являются усложнение системы и снижение производительности.
- Шаблон посредника (Broker pattern). Когда в системе присутствует большое количество модулей, их прямое взаимодействие друг с другом становится слишком сложным. Для решения проблемы вводится посредник (например, шина данных), по которой модули общаются друг с другом. Таким образом, повышается функциональная совместимость модулей системы. Все недостатки вытекают из наличия посредника: он понижает производительность, его недоступность может сделать недоступной всю систему, он может стать объектом атак и узким местом системы.
- Шаблон «Модель-Представление-Контроллер» (Model-View-Controller pattern). Т.к. требования к интерфейсу меняются чаще всего, то возникает потребность часто его модифицировать, при этом сохраняя корректное взаимодействие с данными (чтение, сохранение). Для этого в шаблоне Model-View-Controller (MVC) интерфейс отделён от данных. Это позволяет менять интерфейсы, равно как и создавать их разные варианты. В MVC система разделена на:
 - а) Модель, хранящую данные
 - б) Представление, отображающее часть данных и взаимодействующее с пользователем
 - в) Контроллер, являющийся посредником между видами и моделью

Однако, концепция MVC имеет и свои недостатки. В частности, из-за усложнения взаимодействия падает скорость работы системы.

- Шаблон Model-View-ViewModel (MVVM). Делится на три части:
 - 1) *Модель* (англ. *Model*), так же, как в классической MVC, Модель представляет собой фундаментальные данные, необходимые для работы приложения.
 - 2) *Представление* (англ. *View*) — это графический интерфейс, то есть окно, кнопки и т. п. Представление является подписчиком на событие изменения значений свойств или команд, предоставляемых Моделью представления. В случае, если в Модели представления изменилось какое-либо свойство, то она оповещает всех подписчиков об этом, и Представление, в свою очередь, запрашивает обновленное значение свойства из Модели представления. В случае, если пользователь воздействует на какой-либо элемент

интерфейса, Представление вызывает соответствующую команду, предоставленную Моделью представления.

- 3) *Модель представления (англ. ViewModel)* является, с одной стороны, абстракцией Представления, а с другой, предоставляет обёртку данных из Модели, которые подлежат связыванию. То есть, она содержит Модель, которая преобразована к Представлению, а также содержит в себе команды, которыми может пользоваться Представление, чтобы влиять на Модель.
- Клиент-серверный шаблон (Client-Server pattern). Если есть ограниченное число ресурсов, к которым требуется ограниченный правами доступ большого числа потребителей, то удобно реализовать клиент-серверную архитектуру. Такой подход повышает масштабируемость и доступность системы. Но при этом сервер может стать узким местом системы, при его недоступности становится недоступна вся система.
 - Шаблон Naked objects определяется с помощью трех принципов:
 - 1) Вся бизнес-логика должна быть инкапсулирована в бизнес-объект domain objects. Данный принцип не является уникальной особенностью naked objects: это только строгое следование обязательствам, определенным инкапсуляцией.
 - 2) Интерфейс пользователя должен быть прямым представлением объектов предметной области (domain objects), со всеми действиями пользователя, явно содержащими создание или получение объектов предметной области и/или вызовы методов этих объектов. Данный принцип также не является уникальной особенностью naked objects: это только частная интерпретация объектно-ориентированного пользовательского интерфейса object-oriented user interface (ООУИ). Подлинная идея шаблона Naked objects возникает из комбинации обоих вышеперечисленных идей в форме третьего принципа:
 - 3) Пользовательский интерфейс может быть сформирован полностью автоматически из определения объектов предметной области (domain objects). Данный принцип может быть реализован путём использования нескольких технологий таких, как кодогенерация и рефлексия.

16. Модели и их представления на UML – использования, поведения и структуры. Общие свойства модели и механизмы расширения – стереотипы, помеченные значения, ограничения

Общие понятия UML

UML — это графический язык моделирования общего назначения, предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых при разработке программных систем.

Модель UML — это конечное множество сущностей и отношений между ними.

Сущности. Для удобства обзора сущности в UML можно подразделить на четыре группы:

- Структурные;
- Поведенческие;
- Группирующие;
- Аннотационные.

Структурные сущности:

- Класс – описание множества объектов с общими атрибутами и операциями.
- Интерфейс – множество операций, которое определяет набор услуг (службу), предоставляемых классом или компонентом;
- Действующее лицо – сущность, находящаяся вне моделируемой системы и непосредственно взаимодействующая с ней;
- Вариант использования – описание последовательности производимых системой действий, доставляющей значимый для некоторого действующего лица результат.
- Компонент – физически заменяемый артефакт, реализующий некоторый набор интерфейсов;
- Узел – физический вычислительный ресурс.

Поведенческие сущности:

- Состояние — период в жизненном цикле объекта, в котором объект удовлетворяет некоторому условию, выполняет деятельность или ожидает события.
- Деятельность — состояние, в котором выполняется работа, а не просто пассивно ожидается наступление события.

Группирующая сущность

- Пакет – группа элементов модели (в том числе пакетов).

Аннотационная сущность:

- Примечание – средство для внесения в модель дополнительной информации.

Отношения. В UML используются четыре основных типов отношений:

- Зависимость;
- Ассоциация;
- Обобщение;
- Реализация.

Зависимость — это наиболее общий тип отношения между двумя сущностями. Отношение зависимости указывает на то, что изменение независимой сущности каким-то образом влияет на

зависимую сущность. Графически отношение зависимости изображается в виде пунктирной стрелки, направленной от независимой сущности к зависимой.

Ассоциация — это наиболее часто используемый тип отношения между сущностями. Отношение ассоциации имеет место, если одна сущность непосредственно связана с другой (или с другими — ассоциация может быть не только бинарной). Графически ассоциация изображается в виде сплошной линии с различными дополнениями, соединяющей связанные сущности.

Обобщение — это отношение между двумя сущностями, одна из которых является частным (специализированным) случаем другой. Графически обобщение изображается в виде сплошной стрелки с треугольником на конце, направленной от частного к общему. Отношение наследования между классами в объектно-ориентированных языках программирования является типичным примером обобщения.

Реализации — отношение, указывающее, что одна сущность является реализацией другой. Например, класс является реализацией интерфейса. Графически реализация изображается в виде пунктирной стрелки с треугольником на конце, направленной от реализующей сущности к реализуемой.

Способы использования UML (сортировка по важности)

- Рисование картинок;
- Обмен информацией;
- Спецификация систем;
- Повторное использование архитектурных решений;
- Генерация кода;
- Имитационное моделирование;
- Верификация моделей.

Диаграммы UML — основная накладываемая на модель структура, которая облегчает создание и использование модели.

Диаграмма — это графическое представление некоторой части графа модели.



Модель UML — это конечное множество сущностей и отношений между ними. Можно сказать, что это граф (точнее, нагруженный мульти-псевдо-гипер-орграф), в котором вершины и ребра нагружены дополнительной информацией и могут иметь сложную внутреннюю структуру. Вершины этого графа называются сущностями, а ребра — отношениями.

Модель UML — это основной артефакт фазы проектирования.

Представления. Все аспекты моделируемой системы не удастся описать с единой точки зрения. Моделировать сложную систему следует с нескольких различных точек зрения, каждый раз принимая во внимание один аспект моделируемой системы и абстрагируясь от остальных. Выделяют три представления:

- **Представление использования (что делает система полезного?);**
- **Представление структуры (из чего состоит система?);**
- **Представление поведения (как работает система?).**

Представление использования.

Определяющим признаком для отнесения элементов модели к представлению использования является, явное сосредоточение внимания на факте наличия у системы внешних границ, то есть выделение внешних действующих лиц, взаимодействующих с системой, и внутренних вариантов использования, описывающих различные сценарии такого взаимодействия.

Наш язык и мышление устроены так, что самой простой, понятной и четкой формой изложения мыслей являются так называемые простые утверждения. Простое утверждение имеет следующую грамматическую форму: подлежащее — сказуемое — прямое дополнение. В логических терминах: субъект — предикат — объект. Например: начальник увольняет сотрудника, директор создает отдел.

По сути, именно простые утверждения и записаны на диаграмме использования. Действующее лицо — это субъект, а вариант использования — предикат (вместе с объектом).

Моделирование использования предполагает явное формулирование требований к системе на самом начальном этапе разработки.

Преимущества:

- *Простые утверждения.* Позволяет записать исходное техническое задание в строгой и формальной, но в тоже время очень простой и наглядной графической форме, как совокупность простых утверждений относительно того, что делает система для пользователей;
- *Абстрагирование от реализации.* Описывает только, что делает система, но не как это делается и не зачем это нужно делать;
- *Декларативное описание.* Описывает (именует) некоторое множество последовательностей действий, доставляющих значимый для пользователя результат. Никакого императивного описания представление использования не содержит, в модели нет указаний на то, какой вариант использования должен выполняться раньше, а какой позже, то есть нет описания алгоритма, а значит, нет алгоритмических ошибок;

- *Выявление границ.* определяет границы системы и постулирует существование во внешнем мире использующих её агентов.

Описывается диаграммой использования.

Представление структуры.

Определяющим признаком для отнесения элементов модели к представлению структуры является явное выделение структурных элементов — составных частей системы — и описания взаимосвязей между ними. Принципиальным является чисто статический характер описания, то есть отсутствие понятия времени в любой форме, в частности, в форме последовательности событий и/или действий.

В каждый конкретный момент функционирования системы можно указать конечный набор конкретных объектов и связей между ними, образующих систему. В настоящее время структуру из данных и процедур их обработки, существующую в памяти компьютера во время выполнения программы, чаще всего называют объектом, а описание множества однотипных объектов к тексту программы называют классом.

Можно смоделировать следующие структуры:

- Связей между объектами во время выполнения программы;
- Хранения данных;
- Программного кода;
- Компонентов в приложении;
- Используемых вычислительных ресурсов;
- Сложных объектов, состоящих из взаимодействующих частей;
- Артефактов в проекте.

Описывается диаграммами классов, а также, если нужно, диаграммами компонентов и размещения и, в редких случаях, диаграммами объектов.

Представление поведения.

Определяющим признаком для отнесения элементов модели к представлению поведения является явное использования понятия времени, в частности, в форме описания последовательности событий/действий, то есть в форме алгоритма.

- Модель поведения должна быть достаточно детальной для того, чтобы послужить основой для составления компьютерной программы;
- Модель поведения должна быть компактной и обозримой, чтобы служить средством общения между людьми в процессе разработки системы;
- Модель поведения не должна зависеть от особенностей реализации конкретных компьютеров, средств программирования и технологий;
- Средства моделирования поведения в UML должны быть знакомы и привычны большинству пользователей языка и не должны противоречить требованиям наиболее ходовых парадигм программирования.

Описывается диаграммами состояний и деятельности, а также диаграммами взаимодействия в форме диаграмм коммуникации и/или последовательности.

Общие свойства модели.

- Правильность;
- Непротиворечивость;
- Полнота;
- Вариации семантики.

Правильность.

Прежде всего, модель должна удовлетворять формальным требованиям к описанию сущностей, отношений и их комбинаций. Т.е. модель должна быть синтаксически правильной.

Например, отношение (ребро в графе модели) всегда определяется между сущностями, на диаграмме линия должна начинаться и заканчиваться в фигуре, иначе это синтаксическая ошибка.

Непротиворечивость.

В некоторых случаях даже синтаксически правильная модель может содержать такие конструкции, семантика которых не определена или неоднозначна. Такая модель называется противоречивой, а модель, в которой все в порядке и семантика всех конструкций определяется однозначно, называется непротиворечивой.

Например, пусть мы определим в модели, что класс А является подклассом класса В, класс В — подкласс С, а класс С — подкласс А. Каждое из этих отношений обобщения в отдельности допустимо и синтаксически правильно, а все вместе они образуют противоречие.

Полнота.

Модель не создается мгновенно — она появляется в результате многочисленных итераций и на каждой из них не полна. В некоторых случаях оказывается достаточно одной диаграммы использования, а в других необходимы диаграммы всех типов, прорисованные до мельчайших деталей. Все зависит от прагматики, т. е. от того, для чего составляется модель.

Вариации семантики.

В описании семантики UML определено некоторое количество точек вариации семантики. По сути авторы стандарта говорят: «мы понимаем это так-то и так-то, но допускаем, что другие могут это понимать иначе».

При реализации языка в конкретном инструменте разработчики в точке вариации семантики вправе выбрать альтернативный вариант, если он не противоречит семантике остальной части языка.

Механизмы расширения модели.

Механизмы расширения позволяют определять новые элементы модели на основе существующих управляемым и унифицированным способом.

- Помеченные значения;
- Ограничения;
- Стереотипы;

Помеченное значение — это пара: имя свойства и значение свойства, которую можно добавить к любому стандартному элементу модели. Помеченные значения записываются в модели в виде строки текста, имеющей следующий синтаксис: в фигурных скобках указывается пара: имя и

значение, разделенные знаком равенства. Можно указывать сразу список пар, разделяя элементы списка запятыми.

Ограничение — это логическое утверждение относительно значений свойств элементов модели. Логическое утверждение может иметь два значения: истина и ложь. Указывая ограничение для элемента модели, мы расширяем его семантику, требуя, чтобы ограничение выполнялось. Ограничение может относиться к отдельному элементу или к совокупности элементов модели или к совокупности элементов модели. Ограничения записываются в виде строки текста, заключенной в фигурные скобки.

Стереотип – это определение нового элемента моделирования в UML на основе существующего элемента моделирования. Взяв за основу некоторый существующий элемент модели, к нему добавляют новые *помеченные значения* (расширяя тем самым внутреннее представление), *новые ограничения* (расширяя семантику) и дополнения, то есть новые графические элементы (расширяя нотацию).

Стандартные стереотипы классов:

- actor - действующее лицо;
- enumeration - перечислимый тип данных;
- exception - сигнал, распространяемый по иерархии обобщений;
- implementation class - реализация класса;
- interface - нет атрибутов и все операции абстрактные;
- metaclass - экземпляры являются классами;
- power type - метакласс, экземплярами которого являются все наследники данного класса;
- process, thread - активные классы;
- signal - класс, экземплярами которого являются сообщения;
- stereotype – стереотип;
- type (datatype) - тип данных;
- utility - нет экземпляров (служба).

17. Моделирование структуры на UML. Диаграмма классов. Компоненты и интерфейсы

Моделируя структуру, мы описываем составные части системы и отношения между ними.

UML является объектно-ориентированным языком моделирования, поэтому не удивительно, что основным видом составных частей, из которых состоит система, являются объекты.

Процесс моделирования:



Моделирование структуры опирается на принципы ООП:

- **Инкапсуляция** – структурирование программы на структуры особого вида, объединяющего данные и процедуры их обработки, причем внутренние данные структуры не могут быть обработаны иначе, кроме как предусмотренными для этого процедурами;
- **Полиморфизм** – это способ идентификации процедур при вызове. В классических процедурных системах программирования процедуры идентифицируются просто по именам. В объектно-ориентированном программировании конкретный метод идентифицируется не только по имени, но и по объекту, которому метод принадлежит, типам и количеству аргументов (т. е. по полной сигнатуре);
- **Наследование** – это способ структуризации описаний многих классов, который позволяет сократить текст программы, сделав его тем самым более обзримым, а значит более надежным и удобным. Подкласс содержит все составляющие своего суперкласса и удовлетворяет принципу подстановочности.

Можно смоделировать следующие структуры:

- Связей между объектами во время выполнения программы;
- Хранилища данных;
- Программного кода;
- Компонентов в приложении;
- Используемых вычислительных ресурсов;
- Сложных объектов, состоящих из взаимодействующих частей;

- Артефактов в проекте.

Структура связей между объектами во время выполнения программы.

В парадигме ООП процесс выполнения программы состоит в том, что программные объекты взаимодействуют друг с другом, обмениваясь сообщениями. Наиболее распространенным типом сообщения является вызов метода объекта одного класса из метода объекта другого класса.

Для того чтобы вызвать метод объекта, нужно иметь доступ к этому объекту. Один объект "знает" другие объекты и, значит, может вызвать открытые методы, использовать и изменять значения открытых свойств и т.д.

В этом случае, мы говорим что объекты связаны. Для моделирования структуры связей в UML используются отношения ассоциации на диаграмме классов.

Структура хранения данных.

Программы обрабатывают данные, которые хранятся в памяти компьютера. В парадигме объектно-ориентированного программирования для хранения данных во время выполнения программы предназначены свойства объектов, которые моделируются в UML атрибутами классов.

Объекты, которые сохраняют (по меньшей мере) значения своих свойств даже после того, как завершился породивший их процесс, мы будем называть хранимыми.

В настоящее время самой распространенным способом хранения объектов является использование системы управления базами данных (СУБД). При этом хранимому классу соответствует таблица базы данных, а хранимый объект (точнее говоря, набор значений хранимых атрибутов) представляется записью в таблице.

Для моделирования структуры хранения данных в UML применяются ассоциации с указанием кратности полюсов.

Структура программного кода.

Для маленьких программ структура кода практически не имеет значения, для больших — наоборот, имеет едва ли не решающее значение. Поскольку UML не является языком программирования, модель не определяет структуру кода непосредственно, однако косвенным образом структура модели существенно влияет на структуру кода.

Структура классов и пакетов в модели UML фактически полностью моделирует структуру кода приложения.

Структура компонентов в приложении.

Приложение, состоящее из одной исполнимой компоненты, имеет тривиальную структуру компонентов, моделировать которую нет нужды. Большинство современных приложений состоят из многих компонентов, даже если и не являются распределенными.

Компонентная структура предполагает описание двух аспектов: во-первых, как классы распределены по компонентам, во-вторых, как (через какие интерфейсы) компоненты взаимодействуют друг с другом.

Оба эти аспекта моделируются диаграммами компонентов UML.

Структура используемых вычислительных ресурсов.

Многокомпонентное приложение, как правило, бывает распределенным, т. е. различные компоненты выполняются на разных компьютерах.

Диаграммы размещения и компонентов позволяют включить в модель описание и этой структуры.

Структура сложных объектов, состоящих из взаимодействующих частей.

Для моделирования этой структуры применяется диаграмма внутренней структуры классификатора (UML 2). В курсе не рассматриваем. (Видимо, не нужно)

Диаграммы классов.

Диаграмма классов является основным средством моделирования структуры UML. Диаграммы классов наиболее информационно насыщены по сравнению с другими типами канонических диаграмм UML.

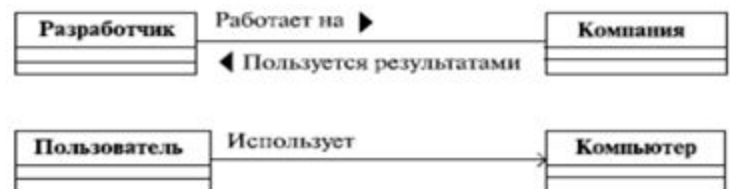
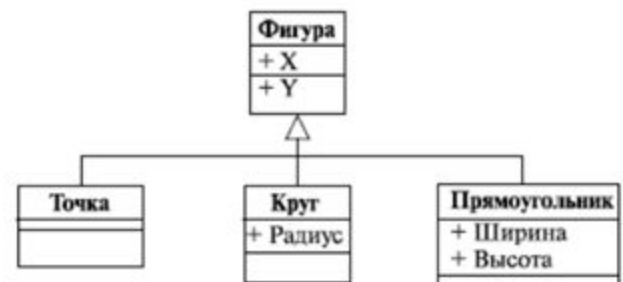
На диаграммах классов в качестве сущностей применяются прежде всего классы, как в своей наиболее общей форме, так и в форме многочисленных стереотипов и частных случаев: интерфейсы, типы данных, процессы и др. Кроме того, в диаграмме классов могут использоваться (как и везде) пакеты и примечания.

Сущности на диаграммах классов связываются главным образом отношениями ассоциации (в том числе агрегирования и композиции) и обобщения. Отношения зависимости и реализации на диаграммах классов применяются реже.

Описание класса может включать множество различных элементов, и чтобы они не путались, в языке предусмотрено группирование элементов описания класса по разделам.

Стандартных разделов три:

- Раздел имени — наряду с обязательным именем может содержать также стереотип, кратность и список свойств;
- Раздел атрибутов — содержит список описаний атрибутов класса;
- Раздел операций — содержит список описаний операций класса.



Атрибут — это свойство класса, которое может принимать множество значений.

В общем случае описание атрибута имеет следующий синтаксис: видимость ИМЯ кратность : тип = начальное_значение {свойства}. Видимость, обозначается знаками +, -, #. Если видимость не указана, то никакого значения видимости по умолчанию не подразумевается.

- o «+» public (открытый доступ);

- «#» protected (только из операций этого же класса и классов, создаваемых на его основе);
- «-» private (только из операций того же класса).

Как и любой другой элемент модели, атрибут может быть наделен дополнительными свойствами в форме ограничений и именованных значений.

Операция – реализация функции, которую можно запросить у любого объекта класса.

Описания операций класса перечисляются в разделе операций и имеют следующий синтаксис: видимость ИМЯ (параметры) : тип {свойства}. Здесь слово параметры обозначает последовательность описаний параметров операции.

Описания параметров в списке разделяются запятой. Для каждого параметра обязательно указывается имя, а также могут быть указаны направление передачи параметра, его тип и значение аргумента по умолчанию.

Отношения на диаграмме классов.

Обобщение - это отношение между более общей сущностью, называемой суперклассом, и ее конкретным воплощением, называемым подклассом.

Ассоциация – связь между объектами, по которой можно между ними перемещаться. Ассоциация может иметь имя, показывающее природу отношений между объектами, при этом в имени может указываться направление чтения связи при помощи треугольного маркера. Однонаправленная ассоциация изображается стрелкой.

Ассоциацией с агрегированием – более сложное отношение между классами, связь типа «часть-целое». Один класс имеет более высокий статус(целое) и состоит из низших по статусу классов (частей). При этом выделяют простое и композитное агрегирование (агрегация и композиция).

Агрегация предполагает, что части, отделенные от целого, могут продолжать свое существование независимо от него.

Композиция – целое владеет своими частями и их время жизни соответствует времени жизни целого, т. е. независимо от целого части существовать не могут.

Для ассоциации определены следующие дополнения:

- имя ассоциации (возможно, вместе с направлением чтения);
- кратность полюса ассоциации (полюсом называется конец линии ассоциации).
- вид агрегации полюса ассоциации;
- роль полюса ассоциации;
- направление навигации полюса ассоциации;
- упорядоченность объектов на полюсе ассоциации;
- изменяемость множества объектов на полюсе ассоциации;
- квалификатор полюса ассоциации;
- класс ассоциации;
- видимость полюса ассоциации;
- многополюсные ассоциации.

Компоненты.

Компонент — это физически существующий и заменяемый артефакт системы.

При выделении компонентов применяются следующие *неформальные критерии*.

- Компонент нетривиален. Это нечто более сложное и объемное, чем фрагмент кода или одиночный класс;
- Компонент независим, но не самодостаточен. Он содержит все, что нужно для функционирования, но предназначен для работы во взаимодействии с другими компонентами;
- Компонент однороден. Он выполняет несколько взаимосвязанных функций, которые могут быть естественным образом охарактеризованы как единое целое в контексте более сложной системы;
- Компонент заменяем. Он поддерживает строго определенный набор интерфейсов и может быть без ущерба для функционирования системы заменен другим компонентом, поддерживающим те же интерфейсы.

Диаграмма компонентов предназначена для перечисления и указания взаимосвязей артефактов моделируемой системы.

Основные *типы сущностей*:

- Компоненты;
- Интерфейсы;
- Классы;
- Объекты.

Основные *типы отношений*:

- Зависимость;
- Ассоциация (главным образом в форме композиции);
- Реализация.

Интерфейсы.

Интерфейс — это абстрактный класс, в котором нет атрибутов и все операции абстрактны. Поскольку интерфейс — это абстрактный класс, он не может иметь непосредственных экземпляров. Между интерфейсами и другими классификаторами, в частности классами, на диаграмме классов применяются два отношения:

- классификатор (в частности, класс) использует интерфейс — это показывается с помощью зависимости со стереотипом «call»;
- классификатор (в частности, класс) реализует интерфейс — это показывается с помощью отношения реализации.

Роль — это интерфейс, который предоставляет классификатор в данной ассоциации.

18. Диаграммы UML: диаграммы use case (вариантов использования), диаграммы состояний, диаграммы деятельности, диаграмма последовательности и диаграмма коммуникации

Диаграммы вариантов использования.

Диаграмма использования — это наиболее общее представление функционального назначения системы. Диаграмма использования призвана ответить на главный вопрос моделирования: что делает система во внешнем мире?

Основные типы сущностей:

- Действующие лица;
- Варианты использования;
- Примечания;
- Пакеты.

Основные типы отношений:

- Ассоциация между действующим лицом и вариантом использования;
- Обобщение между действующими лицами;
- Обобщение между вариантами использования;
- Зависимости между вариантами использования:

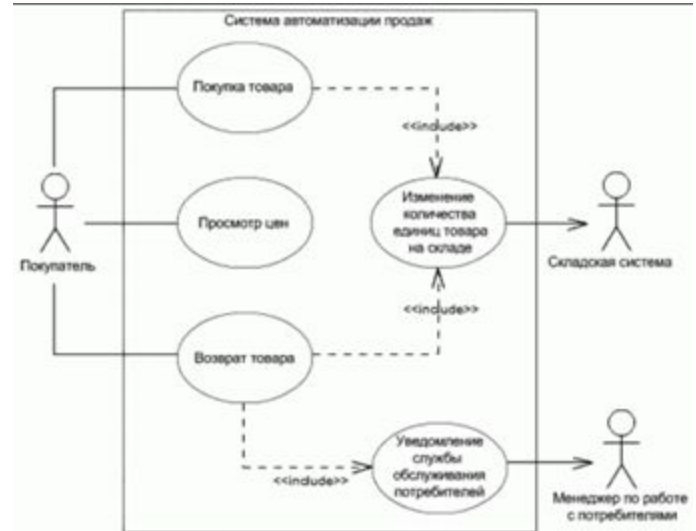
§ Include – показывает, что сценарий независимого варианта использования включает в себя в качестве подпоследовательности действий сценарий зависимого варианта использования;

§ Extend – показывает, что в сценарий зависимого варианта использования может быть в определенном месте вставлен в качестве подпоследовательности действий сценарий независимого варианта использования.

- Зависимости между пакетами.

С синтаксической точки зрения **действующее лицо** — это стереотип классификатора, который обозначается специальным значком. Для действующего лица указывается только имя, идентифицирующее его в системе. Семантически действующее лицо — это множество логически взаимосвязанных ролей.

Роль в UML — это интерфейс, поддерживаемый данным классификатором в данной ассоциации.





С прагматической точки зрения главным является то, что действующие лица находятся вне проектируемой системы (или рассматриваемой части системы).

Выделение вариантов использования — ключ ко всему дальнейшему моделированию. На этом этапе определяется функциональность системы, то есть, что она должна делать. Нотация для варианта использования — это просто имя, помещенное в овал (или помещенное под овалом — такой вариант тоже допустим).

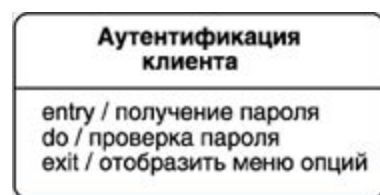
Семантически вариант использования — это описание множества возможных последовательностей действий (событий), приводящих к значимому для действующего лица результату.

Прагматика варианта использования состоит в том, что среди всех последовательностей действий, могущих произойти при работе приложения, выделяются такие, в результате которых получается явно видимый и достаточно важный для действующего лица результат.

Действующие лица находятся вне системы — с ними ничего делать не нужно. Таким образом, переход от моделирования использования к другим видам моделирования состоит в уточнении, детализации и конкретизации вариантов использования.

Диаграммы состояний.

Диаграмма состояний — это основной способ детального описания поведения в UML. В сущности, диаграммы состояний представляют собой граф состояний и переходов конечного автомата, нагруженный множеством дополнительных деталей и подробностей.



Конечный автомат — модель для спецификации поведения объекта в форме последовательности его состояний, которые описывают реакцию объекта на внешние события, выполнение объектом действий, а также изменение его отдельных свойств.

Вершинами графа конечного автомата являются состояния. Дуги графа служат для обозначения переходов из состояния в состояние.

На диаграммах состояний применяется всего один *тип сущностей* — состояния, и всего один *тип отношений* — переходы. Совокупность состояний и переходов между ними образует машину состояний (т.е. конечный автомат).

Состояния бывают: простые, составные, специальные. Каждый тип состояний имеет дополнительные подтипы и различные составляющие элементы.

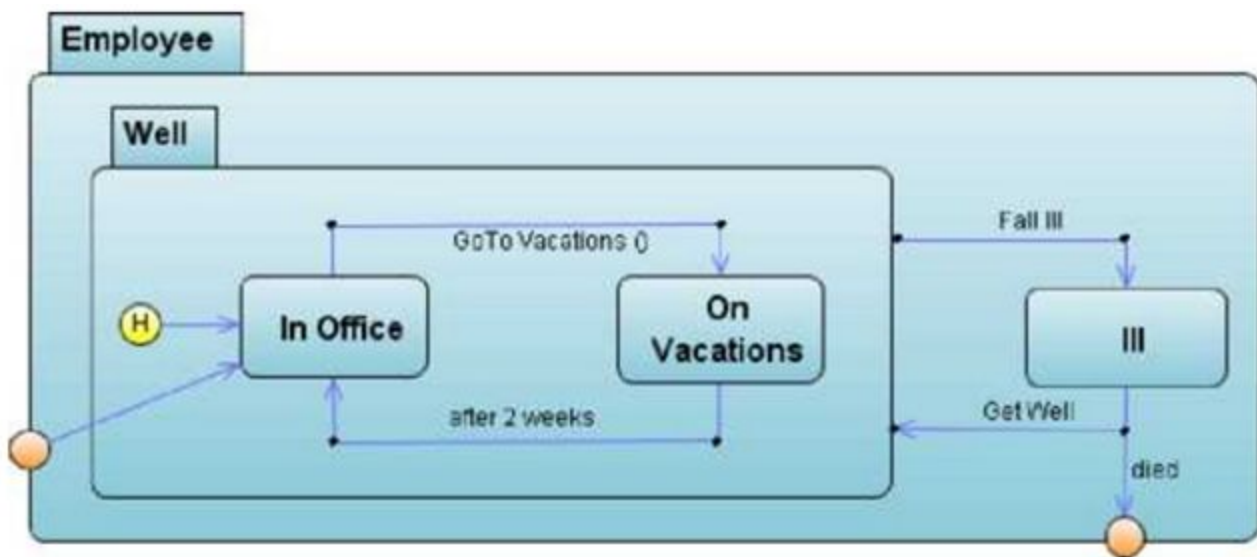
Переходы бывают простые и составные, и каждый переход может содержать:

- Исходное состояние (обязательно);
- Событие перехода – это тот входной символ (стимул), который вкупе с текущим состоянием автомата определяет следующее состояние;
- Сторожевое условие – это логическое выражение, которое должно оказаться истинным для того, чтобы возбужденный переход сработал:
 - § Сегментированные переходы;
 - § Символы ветвления;
 - § Переходные состояния;
 - § Предикат else.
- Действие на переходе;
- Целевое состояние (обязательно).

Простое состояние имеет следующую структуру:

- Имя (обязательно);
- Действие при входе (entry) — это указание атомарного действия, которое должно выполняться при переходе автомата в данное состояние;
- Действие при выходе (exit) — это указание атомарного действия, которое должно выполняться при переходе автомата из данного состояния. Действие при выходе выполняется до всех других действий, предписанных переходом, выводящим автомат из данного состояния;
- Внутренняя активность (do) — это указание деятельности, которая начинает выполняться при переходе в данное состояние после выполнения всех действий, предписанных переходом, включая действие на входе. Внутренняя активность либо заканчивается по завершении, либо прерывается в случае выполнения перехода (в том числе и внутреннего перехода). В классической модели конечный автомат, находясь в некотором состоянии, ничего не делает: он находится в состоянии ожидания перехода.

Составное состояние — это состояние, в которое вложена машина состояний. Глубина вложенности в UML не ограничена.

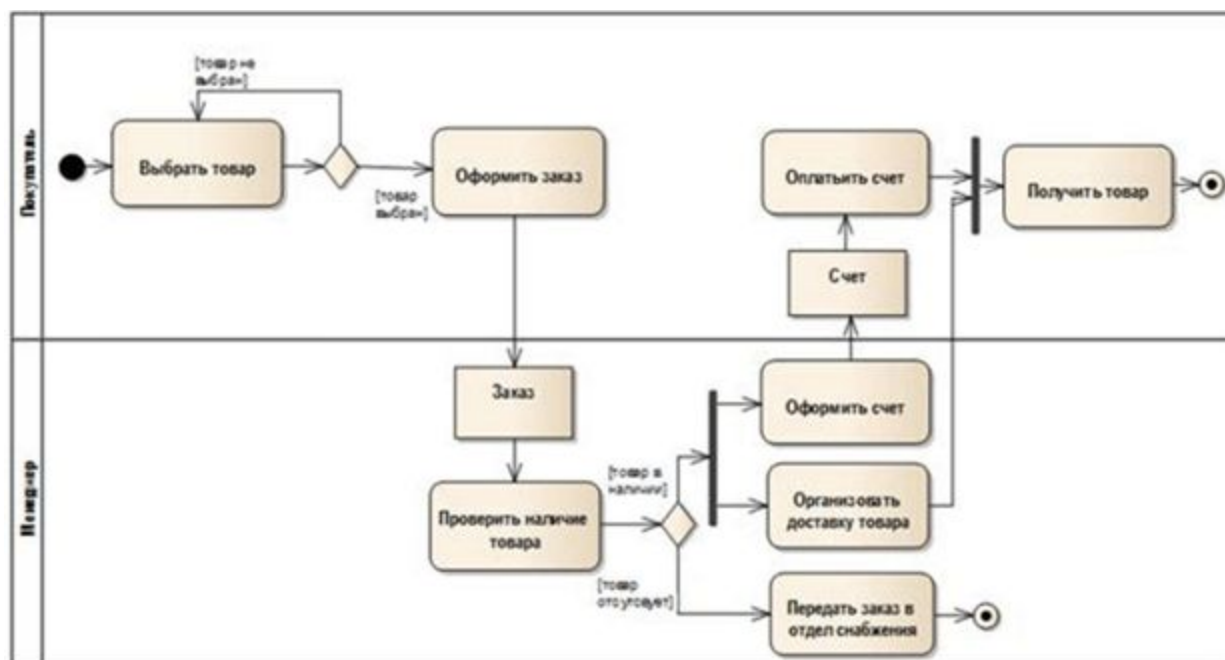


Диаграммы деятельности.

Диаграмма деятельности — это, фактически, блок-схема алгоритма, в которой модернизированы обозначения, а семантика согласована с современным объектно-ориентированным подходом.

Применение диаграмм деятельности для реализации вариантов использования не слишком приближает к появлению целевого артефакта — программного кода, однако может привести к более глубокому пониманию существа задачи и даже открыть неожиданные возможности улучшения приложения, которые было трудно усмотреть в первоначальной постановке задачи.

Дорожка — это графический комментарий, позволяющий классифицировать по некоторому признаку сущности на диаграмме деятельности. Применять не обязательно.



Диаграммы взаимодействия: последовательности и коммуникации.

Диаграмма последовательности — это способ описать поведение системы "на примерах". Фактически, диаграмма последовательности — это запись протокола конкретного сеанса работы

системы (или фрагмента такого протокола). В объектно-ориентированном программировании самым существенным во время выполнения является посылка сообщений взаимодействующими объектами.

Диаграммы коммуникации и диаграммы последовательности семантически эквиваленты, хотя графически выглядят совсем по-разному. Семантически эти диаграммы эквиваленты потому, что описывают одно и то же: последовательность передачи сообщений между объектами в процессе взаимодействия объектов. А выглядят по-разному они потому, что в диаграмме последовательности графически подчеркивается упорядоченность во времени передаваемых сообщений, в то время как в диаграмме коммуникации на передний графический план выдвигается структура связей между объектами, по которым передаются сообщения.





Сильная сторона диаграмм взаимодействия состоит в том, что в объектно-ориентированной парадигме обмен сообщениями — это и есть само выполнение программы, поэтому протокол передачи сообщений является наиболее точной моделью поведения. Диаграммы взаимодействия находятся "ближе" к реальному выполнению программы, чем другие средства описания поведения.

Слабость диаграмм взаимодействия состоит в том, что это диаграммы описывают поведение на уровне объектов, а не классов, на уровне протоколов выполнения алгоритма, а не самого алгоритма. Диаграммы взаимодействия менее "алгоритмичны", чем машины состояний и диаграммы деятельности.

Основными сущностями являются объекты: экземпляры классификаторов — классов и действующих лиц. *Отношениями* же являются связи, т. е. экземпляры ассоциаций, по которым передаются сообщения.

Сообщение — это передача управления и информации от одного объекта (отправителя) к другому (получателю). Отправка сообщения является *действием*, а получение сообщения — *событием*.

Не все действия связаны с передачей информации и отправкой сообщений. В UML таковыми считаются:

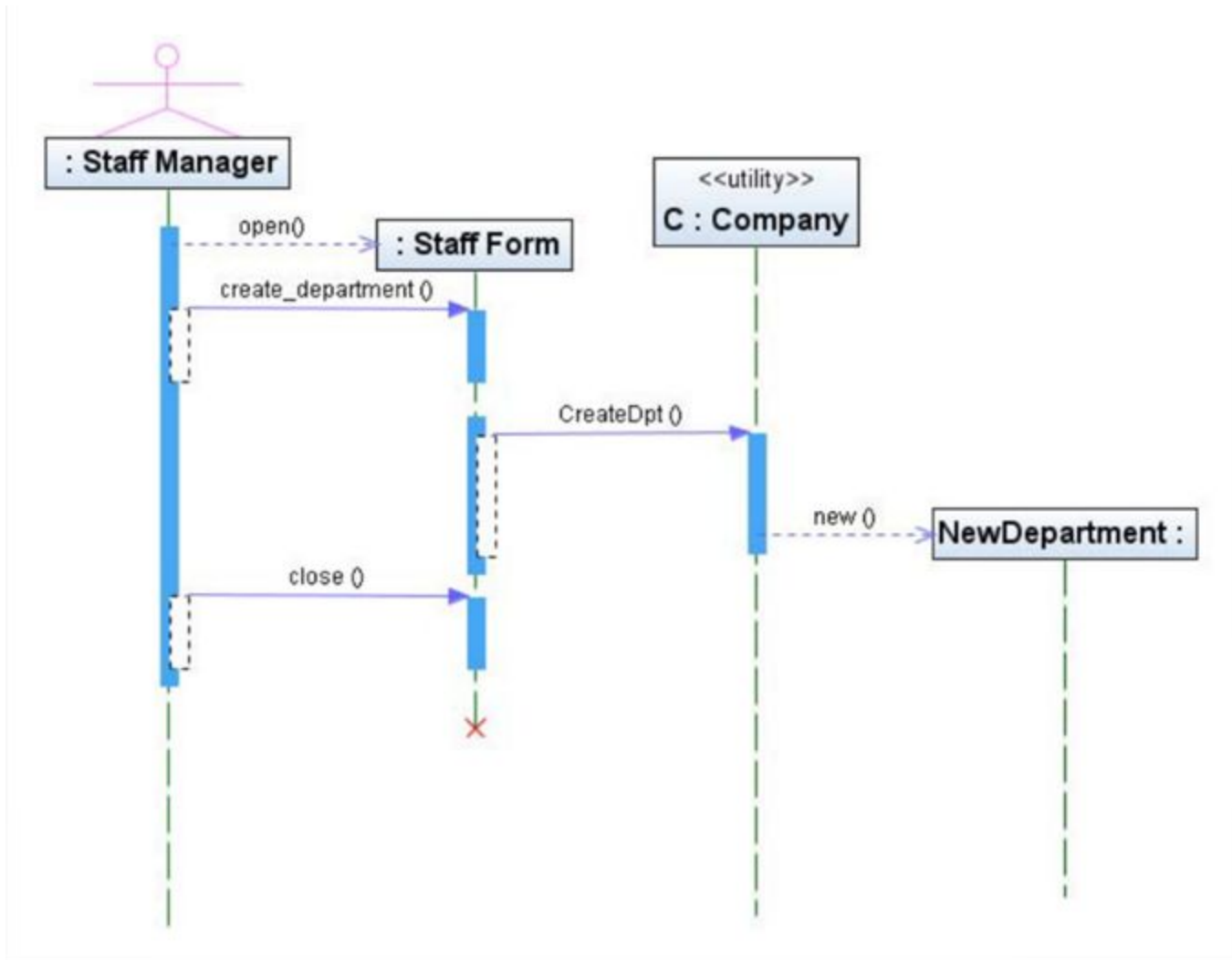
- Вызов операции; 
- Создание объекта; 
- Уничтожение  объекта;
- Возврат значения; 
- Посылка сигнала.

Типы сообщений:



- Вложенный поток управления. Отправитель может отправить следующее сообщение только после того, как завершится выполнение всех действий, инициированных данным сообщением;
- Простой поток управления. Управление передается от отправителя сообщения получателю. Обычно применяется при моделировании поведения на уровне действующих лиц и вариантов использования;

- Асинхронный поток управления. Сообщение асинхронно передается от отправителя получателю, при этом у отправителя сохраняется свой поток управления, независимый от потока управления получателя;
- Возврат управления. Возврат управления после выполнения всех действий, инициированных передачей сообщения с вложенным поток управления. Можно не отображать на диаграмме, поскольку он подразумевается по умолчанию при вызове операций.



Сообщение имеет отправителя и получателя. *Получателей* может быть несколько, такое сообщение называется *широковещательным*. Все получатели широковещательного сообщения получают одно и то же сообщение.

Поскольку получение сообщения является событием, то получатель сообщения вместе с информацией получает и управление (для того, чтобы иметь возможность выполнить действия, инициируемые полученным сообщением).

Для того, чтобы сообщение могло быть передано от отправителя к получателю, отправитель должен "знать" получателя. Другими словами, должна существовать ассоциация между классами отправителя и получателя, экземпляр которой (связь) и служит тем путем, по которому передается сообщение. На диаграмме коммуникации эта связь всегда изображается в явном виде, на диаграмме последовательности она подразумевается.

Способы задания *порядка сообщений*:

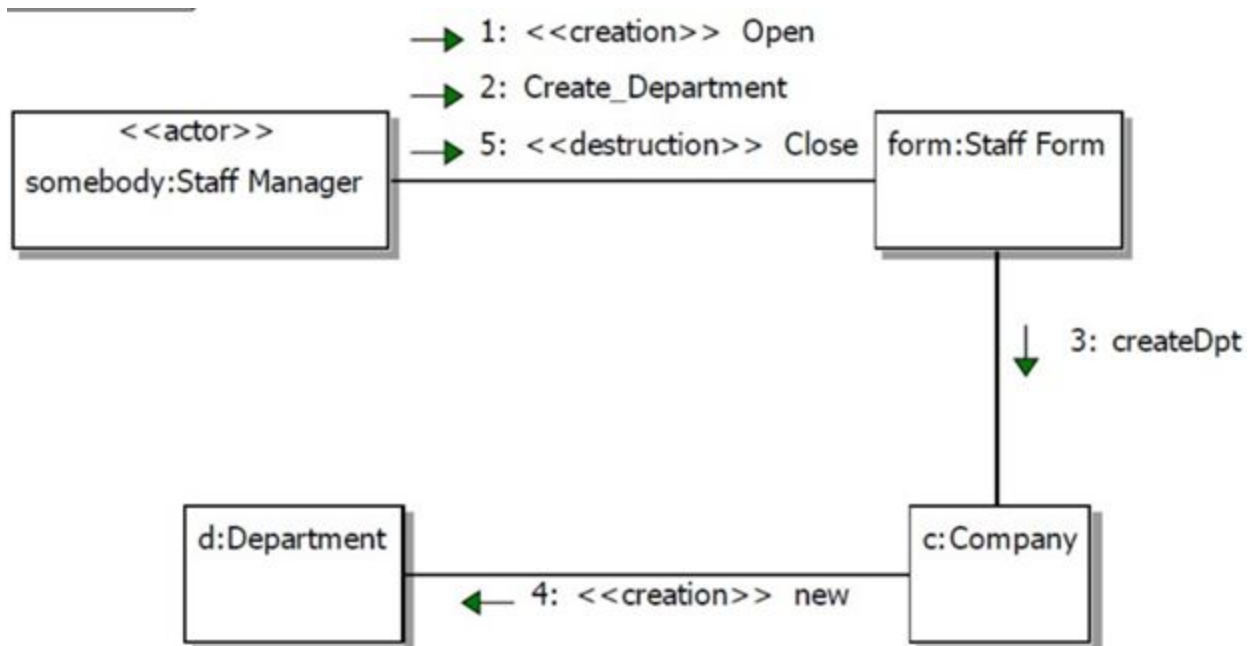
- Определяется временем отправки сообщений, а время считается текущим на диаграмме сверху вниз (основной способ);
- С помощью последовательного номера сообщения;
- Перечислив (через запятую) номера сообщений, предшествующих данному.

В направлении оси времени от всех участвующих во взаимодействии объектов отходит прямая пунктирная линия, которая называется линией жизни.

Линия жизни представляет объект во взаимодействии: если стрелка отходит от линии жизни объекта, то это означает, что данный объект отправляет сообщение, а если стрелка сообщения входит в линию жизни, то это означает, что данный объект получает сообщение. Если же стрелка пересекает линию жизни объекта, то это ничего не значит — сообщение пролетело мимо. Если в процессе взаимодействия объект заканчивает свое существование, то линия жизни обрывается и в этом месте ставится жирный косой крест.

На диаграмме коммуникации определены следующие ключевые слова:

- create (стереотип операции в сообщении). Операция создает объект, т. е. данное сообщение является вызовом конструктора;
- destroy (стереотип операции в сообщении). Операция уничтожает объект, т. е. данное сообщение является вызовом деструктора;
- destroyed (ограничение роли классификатора). Объект уничтожается в процессе описываемого взаимодействия;
- New (ограничение роли классификатора). Объект создается в процессе описываемого взаимодействия;
- transient (ограничение роли классификатора). Объект создается и уничтожается в процессе описываемого взаимодействия. Такой объект называется временным. Данное ограничение эквивалентно одновременному указанию ограничений new и destroyed.



19. Паттерны проектирования и каркасы на UML

Неформально говоря, образец проектирования — это типичное решение типичной проблемы в данном контексте.

Описание образца состоит из четырех элементов:

- **Имя.** Ссылаясь на имя образца, мы можем кратко описать проблему проектирования, ее решения и их последствия. Это позволяет проектировать на более высоком уровне абстракции. Словарь общеизвестных имен образцов позволяет эффективно вести обсуждение с коллегами, лаконично документировать принимаемые архитектурные решения. Подбор хорошего имени — одна из важнейших задач при составлении описания образца;
- **Задача.** Описание контекста применения образца проектирования, т. е. описание конкретной проблемы проектирования и перечня условий, при выполнении которых имеет смысл применять данный образец;
- **Решение.** Описание элементов проектирования, отношений между ними, функции каждого элемента. Дается абстрактное описание задачи проектирования и ее обобщенное решение;
- **Результат.** Здесь описываются следствия применения образца: влияние на степень эффективности, гибкости, расширяемости и переносимости системы.

Используя UML, архитектор программной системы может сообщить свои идеи (именно идеи, а не примеры готовых решений на языке программирования) в лаконичной и понятной форме, доступной для восприятия подавляющему большинству разработчиков.

Синтаксически в UML образец проектирования — это параметрическая кооперация классов (т. е. шаблон коммуникации). Классы, входящие в кооперацию, можно рассматривать как параметры, а всю кооперацию в целом, как шаблон взаимодействия.

Таким образом, чтобы применить некоторый образец проектирования в определенном контексте, достаточно связать параметры шаблона с конкретными значениями, т. е. указать, какие конкретные классы модели играют роли классификаторов, участвующих в данной коммуникации (образце).

Для этого в UML предусмотрен специальный синтаксис: применяемый образец изображается в виде пунктирного овала, внутри которого написано имя коммуникации. Этот овал соединяется пунктирными линиями с классами, которые являются фактическими аргументами, причем на линии указывается имя роли, которую класс играет в применяемой коммуникации.

Пример.

Классический образец проектирования Observer (он же Publish-Subscribe).

Задача.

Поведение некоторых объектов системы (подписчиков — экземпляров класса Subscriber) должно зависеть от изменения состояния (события) другого объекта (издателя — экземпляра класса Publisher). Однако издатель не должен прямо взаимодействовать с подписчиками.

Решение.

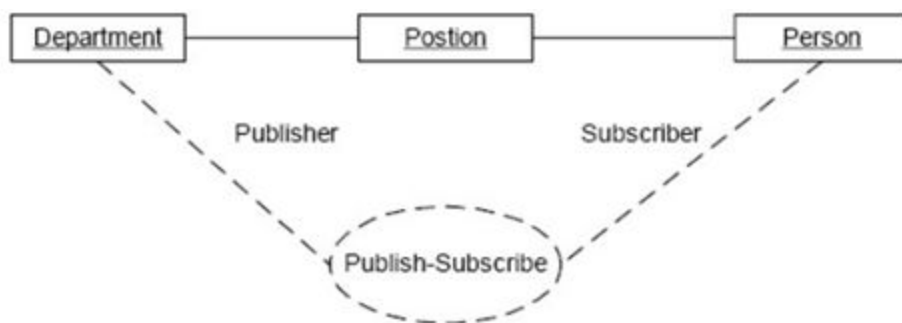
Ввести службу уведомления о событиях, с тем чтобы издатель мог опосредованно уведомлять подписчиков о наступлении события. Для этого вводится (единственный) объект класса EventManager, реализующий данную службу.

Класс EventManager имеет метод subscribe, вызывая который подписчик подписывается на уведомления о наступлении события, и метод signalEvent, посредством которого издатель уведомляет о наступлении события. При вызове метода signalEvent объект EventManager посылает уведомления о событии всем подписчикам, вызывая метод notify, переданный в качестве параметра при подписке.



Пример.

Пусть в системе требуется уведомлять сотрудников об изменении состояния подразделения. В этом случае мы можем на диаграмме коммуникации показать, что к классам Department и Person следует применить образец проектирования Publish-Subscribe, причем класс Department играет роль Publisher, а класс Person играет роль Subscriber.



Приведенная диаграмма определяет коммуникацию, причем в этой коммуникации определено гораздо больше элементов, чем нарисовано на диаграмме. В частности, подразумевается, что в модели определен класс EventManager (хотя он и не присутствует на диаграмме), между классами определены соответствующие ассоциации для вызова методов, сами классы обладают нужными методами для применения образца и т. д.

Каркас.

В UML определено еще одно понятие, тесно связанное с образцами проектирования. Это каркас — совокупность логически связанных образцов. Синтаксически каркас в UML — это пакет со стереотипом «framework». Такой стереотип означает, что в пакете собраны образцы проектирования согласованные таким образом, чтобы их можно было применять совместно и систематически.

Польза.

UML позволяет наглядно описывать образцы проектирования. Применение образцов проектирования повышает качество проектирования архитектуры, что, по общему мнению, ускоряет реализацию.

20. Общая характеристика стандартов управления ИТ инфраструктурой (ITIL, ITSM, ISO20000, MOF). Области применения, основное содержание, цели применения

Одним из направлений повышения эффективности функционирования промышленных предприятий является совершенствование управления их ИТ-инфраструктурой. Основные функции управления ИТ-инфраструктурой обобщены в рамках определенных моделей (стандартов, лучших практик, «методологий»), основными из которых являются следующие: специализированные (ITIL\ITSM, CobIT, MOF, IT Service CMM и др.), так и адаптированные для ИТ-менеджмента универсальные модели управления проектами и ресурсами.

ITIL

Среди названных моделей в первую очередь необходимо выделить Information Technology Infrastructure Library (ITIL).

Модель ITIL – библиотека передового опыта в настоящее время фактически стала международным стандартом в сфере организации и управления информационными технологиями. ITIL включает в себя описание различных видов деятельности в сфере информационных технологий (например, Управление Проектами, Управление Закупками и ИТ-сервис-менеджмент), и состоит из ряда отдельных практических руководств, предоставляющих информацию о эффективном и рациональном использовании различных ИТ – сервисов и предоставлении их потребителям. Философия библиотеки ITIL основана на общих схемах обеспечения качества (Total Quality frameworks), предлагаемых Европейской организацией Управления Качеством EFQM (European Foundation of Quality Management) и стандартах серии ISO 9000. Эти системы качества поддерживаются за счет предоставления стандартизированного описания процессов с учетом передового опыта ИТ-сервис-менеджмента. Библиотека ITIL предлагает структурированное описание наиболее часто используемых ИТ-процессов, их целей и параметров, а также связей между отдельными ИТ – процессами, однако, целью библиотеки ITIL не является предоставление описания конкретного способа внедрения этих процессов.

Первоначально библиотека ITIL состояла из нескольких комплектов книг, в каждом из которых описывалась отдельное направление в сфере организации и управления информационной инфраструктурой. Основой ITIL считались десять книг, в которых описывались поддержка и предоставление услуг. Также в состав библиотеки были включены книги по вспомогательным предметам, относящимся к ИТ-сервис-менеджменту и серия книг, рассматривающая бизнес-аспекты ИТ-сервис-менеджмента. Управление качеством информационных услуг происходит в соответствии с теми же основными принципами, что заложены и в стандарт ISO 9001, а именно: использование процессного подхода к организации предоставления услуг; измеримость показателей качества процессов; контроль процессов в соответствии с определенными критериями и постоянное их совершенствование. Каждый процесс имеет ряд обязательных атрибутов: цель, описание взаимодействия с другими процессами, определенный набор внутренних действий, а также параметры контроля процесса (куда входят отчеты и ключевые индикаторы

производительности). «Информационная услуга» – базовое понятие ITIL, обладает измеряемым качеством и предназначена для удовлетворения потребности пользователей ИТ-услуг в информации. На предприятии такими пользователями являются все сотрудники, которым для выполнения своих обязанностей необходима информационная поддержка.

Обязательное условие управления качеством предоставления ИТ-услуг – разработка и принятие исчерпывающего перечня информационных услуг – «Соглашения об уровне качества предоставляемых услуг» (Service Level Agreement – SLA). Это соглашение появляется на свет в результате договорного процесса между ИТ-подразделением и другими подразделениями предприятия. SLA содержит описание количественных характеристик каждой из услуг, позволяющих измерить уровень их качества, а также подробное описание регламента измерения данных количественных характеристик, необходимое для того, чтобы качество услуги можно было измерить и управлять им.

ITSM

ITSM (IT Service Management, управление ИТ-услугами) — подход к управлению и организации ИТ-услуг, направленный на удовлетворение потребностей бизнеса. Управление ИТ-услугами реализуется поставщиками ИТ-услуг путём использования оптимального сочетания людей, процессов и информационных технологий. Для содействия реализации подхода к управлению ИТ-услугами используется серия документов ITIL.

В отличие от более традиционного технологического подхода, ITSM рекомендует сосредоточиться на клиенте и его потребностях, на услугах, предоставляемых пользователю информационными технологиями, а не на самих технологиях. При этом процессная организация предоставления услуг и наличие заранее оговоренных в соглашениях об уровне услуг параметров эффективности (KPI) позволяет ИТ-отделам предоставлять качественные услуги, измерять и улучшать их качество.

Важным моментом при изложении принципов ITSM является системность. При изложении каждого составного элемента ITSM (управление инцидентами, управление конфигурациями, управление безопасностью и т. д.) в обязательном порядке прослеживается его взаимосвязь и координация с остальными элементами (службами, процессами) и при этом даются необходимые практические рекомендации.

ITIL не является конкретным алгоритмом или руководством к действию, но она описывает передовой опыт (best practices) и предлагает рекомендации по организации процессного подхода и управления качеством предоставления услуг. Это позволяет оторваться от особенностей данного конкретного предприятия в данной конкретной отрасли. Вместе с тем, несмотря на определённую абстрактность, ITIL всячески нацелено на практическое использование. В каждом разделе библиотеки приводятся ключевые факторы успеха внедрения того или иного процесса, практические рекомендации при этом превалируют над чисто теоретическими рассуждениями.

ISO 20000

ISO 20000 — международный стандарт для управления и обслуживания ИТ сервисов. ISO 20000-1 представляет собой подробное описание требований к системе менеджмента ИТ сервисов и

ответственность за инициирование, выполнение и поддержку в организациях. Эта часть состоит из 10 разделов, 13 процессов, собранных в пять ключевых групп:

- Процессы предоставления сервисов (Service delivery process): в группу входят управление уровнем сервисов, управление непрерывностью и доступностью, управление мощностями, отчётность по предоставлению сервисов, управление информационной безопасностью, бюджетирование и учёт затрат.
- Процессы управления взаимодействием (Relationship processes): эта область включает в себя управление взаимодействием с бизнесом, управление поставщиками.
- Процессы разрешения (Resolution processes): разработчики стандарта фокусируются на инцидентах, которые удалось предотвратить или успешно разрешить – управление проблемами, управление инцидентами.
- Процессы контроля (Control processes): в данном разделе рассматриваются процессы управления изменениями и конфигурациями.
- Процессы управления релизами (Release process): речь идёт о выработке новых и коррекции уже имеющихся решений.

Стандарт ISO 20000:2005 предлагает модель улучшения процессов - цикл PDCA (более известен как цикл Деминга):

- PLAN: Спроектируйте или измените процесс для улучшения результатов
- DO: Осуществляйте выполнение
- CHECK: Проведите измерение и подготовьте отчет о работе
- ACT: Решите, какие изменения необходимы для улучшения процесса

Кроме того, в стандарте выдвигаются требования к мере ответственности руководителей компании, предоставляющей ИТ сервисы, а также к управлению документацией, компетенции, осведомлённости и подготовке персонала.

Чтобы обеспечить интегрированный подход к оказанию высококачественных и эффективных ИТ сервисов, стандарт ISO 20000:2005 предлагает переход к сервисной модели ИТ, который реализуется путём разработки и внедрения формальной системы управления ИТ сервисами (СУИС). СУИС позволяет повысить уровень капитализации предприятия и способствует его признанию на международном уровне.

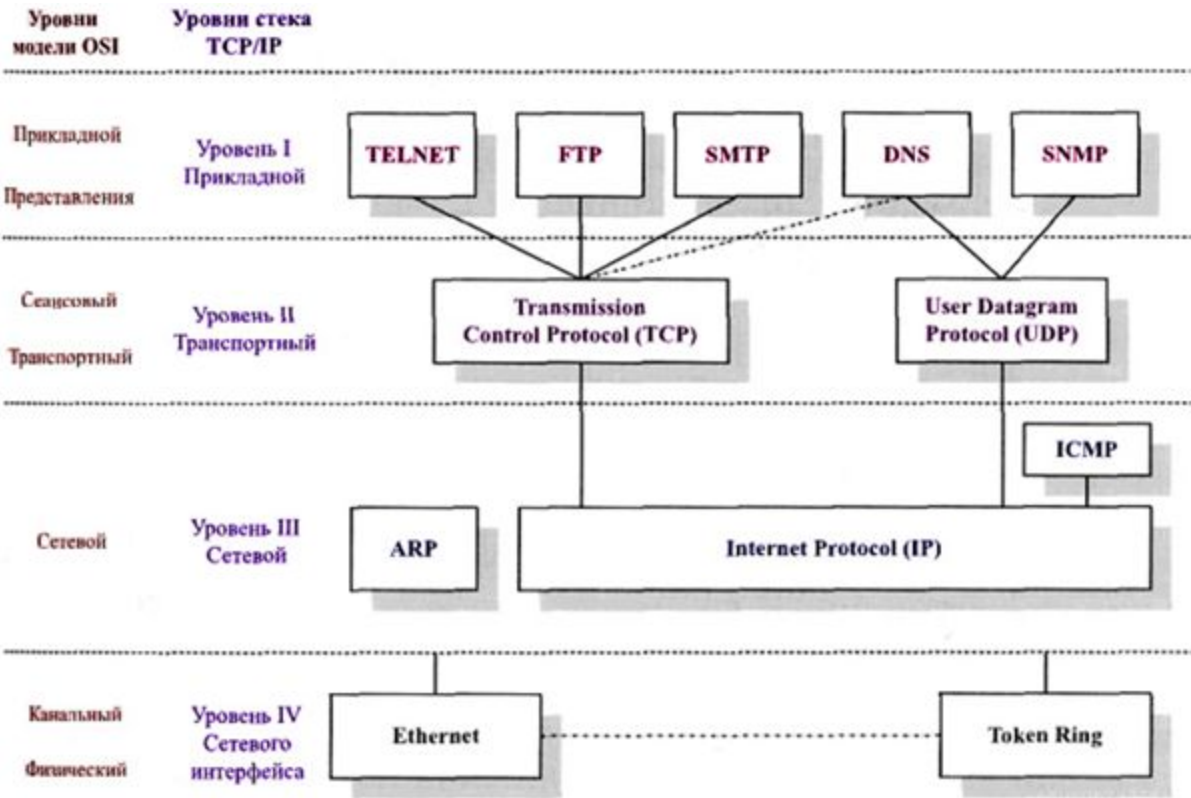
MOF

MOF - предоставляет практическое руководство для ежедневной ИТ практики и видов деятельности, помогая установить и внедрить надёжные, эффективные по цене ИТ сервисы. Оно охватывает полный жизненный цикл ИТ, интегрируя:

- процессы для планирования, предоставления, эксплуатации и управления ИТ;
- ИТ руководство, риски и соответствующую деятельность;
- управленческую отчётность;
- лучшие практики Microsoft Solutions Framework (MSF).

Рекомендации MOF касаются вопросов персонала, процессов, технологий и стратегии управления в сложных, распределённых, гетерогенных ИТ средах.

21. Локальные сети Ethernet. Активное оборудование. Коммутаторы L2, L3, их характерные особенности



Ethernet (Канальный, физический уровень):

Ethernet — семейство технологий пакетной передачи данных для компьютерных сетей. Стандарты Ethernet определяют проводные соединения и электрические сигналы на физическом уровне, формат кадров и протоколы управления доступом к среде — на канальном уровне модели OSI. Ethernet в основном описывается стандартами IEEE группы 802.3. Ethernet стал самой распространённой технологией ЛВС в середине 1990-х годов, вытеснив такие устаревшие технологии, как ARCNET и Token ring.

Название «Ethernet» (буквально «эфирная сеть» или «среда сети») отражает первоначальный принцип работы этой технологии: всё, передаваемое одним узлом, одновременно принимается всеми остальными (то есть имеется некое сходство с радиовещанием). В настоящее время практически всегда подключение происходит через коммутаторы (switch), так что кадры, отправляемые одним узлом, доходят лишь до адресата (исключение составляют передачи на широковещательный адрес) — это повышает скорость работы и безопасность сети.

Пассивное оборудование

ГОСТ Р 51513-99 определяет пассивное оборудование, как оборудование, не получающее питание от электрической сети или других источников, и выполняющее функции распределения или снижения уровня сигналов. Например, кабельная система: кабель (коаксиальный и витая пара), вилка/розетка (RG58, RJ45, RJ11, GG45), патч-панель, балун для коаксиальных кабелей (RG-58) и т.

д. Также, к пассивному оборудованию иногда относят оборудование трассы для кабелей: кабельные лотки, монтажные шкафы и стойки, телекоммуникационные шкафы.

Активное оборудование

В соответствии с ГОСТ Р 51513-99, активное оборудование — это оборудование, содержащее электронные схемы, получающее питание от электрической сети или других источников и выполняющее функции усиления, преобразования сигналов и иные. Это означает способность такого оборудования обрабатывать сигнал по специальным алгоритмам. В сетях происходит пакетная передача данных, каждый пакет данных содержит также техническую информацию: сведения о его источнике, цели, целостности информации и другие, позволяющие доставить пакет по назначению. Активное сетевое оборудование не только улавливает и передает сигнал, но и обрабатывает эту техническую информацию, перенаправляя и распределяя поступающие потоки в соответствии со встроенными в память устройства алгоритмами. Эта «интеллектуальная» особенность, наряду с питанием от сети, является признаком активного оборудования. Например, в состав активного оборудования включаются следующие типы приборов:

- сетевой адаптер — плата, которая устанавливается в компьютер и обеспечивает его подключение к ЛВС;

- репитер — прибор, как правило, с двумя портами, предназначенный для повторения сигнала с целью увеличения длины сетевого сегмента;

- концентратор (активный хаб, многопортовый репитер) — прибор с 4-32 портами, применяемый для объединения пользователей в сеть;

- мост — прибор с 2 портами, обычно используемый для объединения нескольких рабочих групп ЛВС, позволяет осуществлять фильтрацию сетевого трафика, разбирая сетевые (MAC) адреса;

- коммутатор (свитч) — прибор с несколькими (4-32) портами, обычно используемый для объединения нескольких рабочих групп ЛВС (иначе называется многопортовый мост);

- маршрутизатор (роутер) — используется для объединения нескольких рабочих групп ЛВС, позволяет осуществлять фильтрацию сетевого трафика, разбирая сетевые (IP) адреса;

- ретранслятор - для создания усовершенствованной беспроводной сети с большей площадью покрытия и представляет собой альтернативу проводной сети. По умолчанию устройство работает в режиме усиления сигнала и выступает в роли ретрансляционной станции, которая улавливает радиосигнал от базового маршрутизатора сети или точки доступа и передает его на ранее недоступные участки.

- медиаконвертер — прибор, как правило, с двумя портами, обычно используемый для преобразования среды передачи данных (коаксиал-витая пара, витая пара-оптоволокно);

- сетевой трансивер — прибор, как правило, с двумя портами, обычно используемый для преобразования интерфейса передачи данных (RS232-V35, AUI-UTP).

Отметим, что некоторые специалисты не включают в состав активного оборудования повторитель (репитер) и концентратор (хаб), так как эти устройства просто повторяют сигнал для увеличения расстояния соединения или топологического разветвления и обработки его по каким-либо

алгоритмам не проводят. Но управляемые хабы и при этом подходе относятся к активному сетевому оборудованию, так как могут быть наделены некой «интеллектуальной особенностью».

Коммутатор (switch) L2/L3:

Коммутатор

Сетевой коммутатор — устройство, предназначенное для соединения нескольких узлов компьютерной сети в пределах одного или нескольких сегментов сети. Коммутатор работает на канальном (втором) уровне модели OSI. Коммутаторы были разработаны с использованием мостовых технологий и часто рассматриваются как многопортовые мосты. Для соединения нескольких сетей на основе сетевого уровня служат маршрутизаторы (3 уровень OSI).

Коммутатор уровня 2 (Layer 2 или L2) предназначен для соединения нескольких устройств локальной вычислительной сети (LAN) или нескольких сегментов данной сети. Коммутатор уровня 2 обрабатывает и регистрирует MAC-адреса поступающих фреймов, осуществляет физическую адресацию и управления потоком данных (VLAN, QoS).

В отличие от концентратора (1 уровень OSI), который распространяет трафик от одного подключённого устройства ко всем остальным, коммутатор передаёт данные только непосредственно получателю (исключение составляет широковещательный трафик всем узлам сети и трафик для устройств, для которых неизвестен исходящий порт коммутатора). Это повышает производительность и безопасность сети, избавляя остальные сегменты сети от необходимости (и возможности) обрабатывать данные, которые им не предназначались.

Коммутатор хранит в памяти (т.н. ассоциативной памяти) таблицу коммутации, в которой указывается соответствие MAC-адреса узла порту коммутатора. При включении коммутатора эта таблица пуста и он работает в режиме обучения. В этом режиме поступающие на какой-либо порт данные передаются на все остальные порты коммутатора. При этом коммутатор анализирует фреймы (кадры) и, определив MAC-адрес хоста-отправителя, заносит его в таблицу на некоторое время. Впоследствии, если на один из портов коммутатора поступит кадр, предназначенный для хоста, MAC-адрес которого уже есть в таблице, то этот кадр будет передан только через порт, указанный в таблице. Если MAC-адрес хоста-получателя не ассоциирован с каким-либо портом коммутатора, то кадр будет отправлен на все порты, за исключением того порта, с которого он был получен. Со временем коммутатор строит таблицу для всех активных MAC-адресов, в результате трафик локализуется. Стоит отметить малую латентность (задержку) и высокую скорость пересылки на каждом порту интерфейса.

Коммутатор L2 способен использовать третий уровень лишь в немногих случаях, например в построении VLAN'ов. Коммутатор не воспринимает IP-маршруты.

Коммутаторы уровня 3 (Layer 3 или L3) фактически являются маршрутизаторами, которые реализуют механизмы маршрутизации (логическая адресация и выбор пути доставки данных (маршрута) с использованием протоколов маршрутизации (RIPv1 и v.2, OSPF, BGP, проприетарные протоколы маршрутизации и др.)) не в программном обеспечении устройства, а с помощью специализированных аппаратных средств (микросхем). В результате устройство становится менее

гибким, поскольку для модернизации средств реализации применяемых протоколов маршрутизации требуется замена аппаратного обеспечения, а не просто обновление программного обеспечения.

L3-коммутатор умеет обмениваться маршрутами с другими устройствами и перенаправлять ("форвардить") трафик. Так как аппаратные ресурсы дороги, то по некоторым численным характеристикам (количество маршрутов, например) L3-коммутатор может отставать от маршрутизатора. Кроме того, L3-коммутатор поддерживает технологии из сферы коммутации (xSTP, etherchannel и т.д.).

Таким образом, маршрутизатор обладает богатой L3-функциональностью, L3-коммутатор - более узкой, но оптимизированной, L3-функциональностью и, вдобавок, функциональностью "обычного" коммутатора.

22. Архитектура стека TCP/IP (уровни, назначение, потоки данных, примеры протоколов), адресная информация в TCP/IP. IP адреса, IP-сети, порты TCP\UDP. Соединение IP сетей. Маршрутизация в IP. Трансляция адресов (NAT). Проксирование

Уровни OSI

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. (Работа со средой передачи, сигналами и двоичными данными - USB, витая пара, коаксиальный кабель, оптический кабель)

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 общей 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. (Физическая адресация - PPP, IEEE 802.2, Ethernet, DSL, ARP)

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. (Определение маршрута и логическая адресация - IPv4, IPv6, IPsec, AppleTalk)

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. (Прямая связь между конечными пунктами и надежность - TCP, UDP, SCTP)

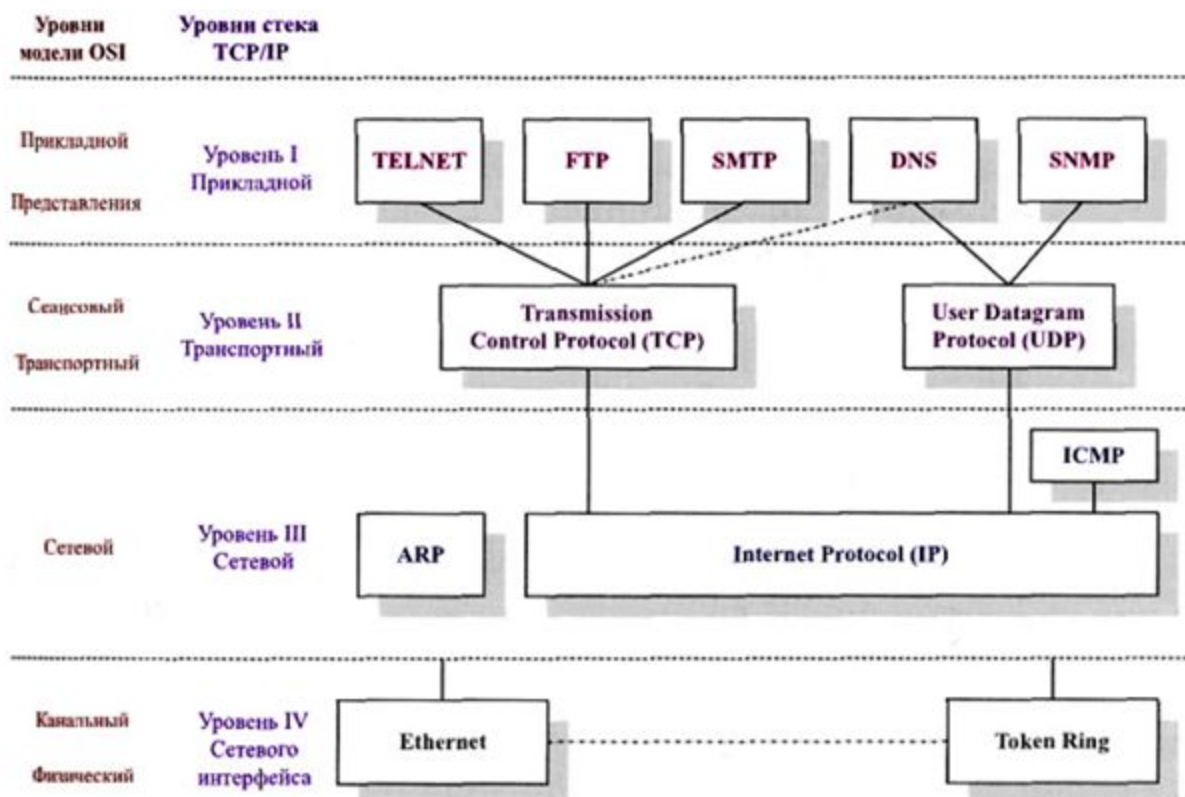
Сессионный (или сеансовый) уровень (Session layer) позволяет двум программам поддерживать продолжительное взаимодействие по сети, называемое сессией (session) или сеансом. Этот уровень управляет установлением сеанса, обменом информацией и завершением сеанса. Он также отвечает за идентификацию, позволяя тем самым только определенным абонентам принимать участие в сеансе, и обеспечивает работу служб безопасности с целью упорядочивания доступа к информации сессии. (Управление сеансом связи - RPC, PAP, L2TP)

Уровень представления (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. (Представление и шифрование данных - ASCII, EBCDIC, JPG)

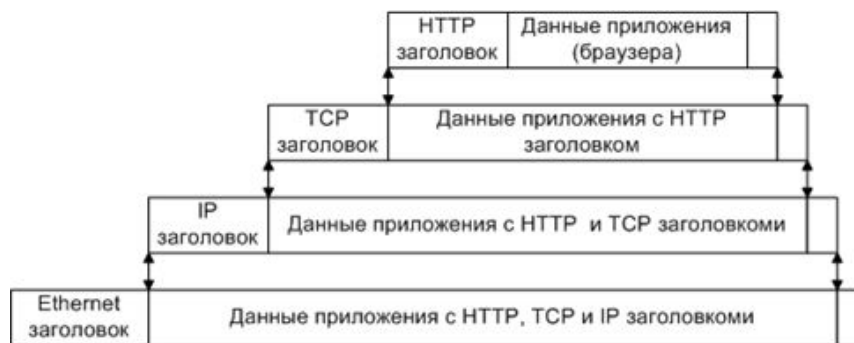
Прикладной уровень (Application layer) предоставляет высокоуровневые функции сетевого взаимодействия, такие, как передача файлов, отправка сообщений по электронной почте и т.п. (Доступ к сетевым службам - HTTP, FTP, SMTP)

TCP/IP



TCP/IP - это аббревиатура термина Transmission Control Protocol/Internet Protocol (Протокол управления передачей/Межсетевой протокол). Фактически TCP/IP не один протокол, а множество, стек протоколов.

В сети, работающей на основе TCP/IP, информация передается в виде дискретных блоков, называемых IP-пакетами (IP packets) или IP-дейтаграммами (IP datagrams). По существу TCP/IP скрывает маршрутизаторы и базовую архитектуру сетей от пользователей, так что всё это выглядит как одна большая сеть. Точно так же, как подключения к сети Ethernet распознаются по 48-разрядным идентификаторам Ethernet, подключения к интрасети идентифицируются 32-разрядными IP-адресами, которые мы выражаем в форме десятичных чисел, разделенных точками (например, 128.10.2.3). Взяв IP-адрес удаленного компьютера, компьютер в интрасети или в Internet может отправить данные на него, как будто они составляют часть одной и той же физической сети.



Данные передаются в пакетах. Пакеты имеют заголовок и окончание, которые содержат служебную информацию. Данные, более верхних уровней вставляются (инкапсулируются), как письмо в конверт, в пакеты нижних уровней.

TCP/IP дает решение проблемы обмена данными между двумя компьютерами, подключенными к одной и той же интрасети, но принадлежащими различным физическим сетям. Решение состоит из нескольких частей, причем каждый уровень семейства протоколов TCP/IP вносит свою лепту в общее дело. IP - самый фундаментальный протокол из комплекта TCP/IP - передает IP-дейтаграммы и обеспечивает выбор маршрута, по которому дейтаграмма будет следовать из пункта А в пункт В и использование маршрутизаторов для "прыжков" между сетями.

Ethernet (Канальный, физический уровень):

Ethernet — семейство технологий пакетной передачи данных для компьютерных сетей.

Название «Ethernet» (буквально «эфирная сеть» или «среда сети») отражает первоначальный принцип работы этой технологии: всё, передаваемое одним узлом, одновременно принимается всеми остальными (то есть имеется некое сходство с радиовещанием). В настоящее время практически всегда подключение происходит через коммутаторы (switch), так что кадры, отправляемые одним узлом, доходят лишь до адресата (исключение составляют передачи на широковещательный адрес) — это повышает скорость работы и безопасность сети.

ARP (Сетевой уровень):

ARP — протокол, предназначенный для определения MAC-адреса по известному IP-адресу.

Рассмотрим суть функционирования ARP на простом примере. Компьютер А (IP-адрес 10.0.0.1) и компьютер Б (IP-адрес 10.22.22.2) соединены сетью Ethernet. Компьютер А желает переслать пакет данных на компьютер Б, IP-адрес компьютера Б ему известен. Однако сеть Ethernet, которой они соединены, не работает с IP-адресами. Поэтому компьютеру А для осуществления передачи через Ethernet требуется узнать адрес компьютера Б в сети Ethernet (MAC-адрес в терминах Ethernet). Для этой задачи и используется протокол ARP. По этому протоколу компьютер А отправляет широковещательный запрос, адресованный всем компьютерам в одном с ним широковещательном домене. Суть запроса: «компьютер с IP-адресом 10.22.22.2, сообщите свой MAC-адрес компьютеру с MAC-адресом (напр. a0:ea:d1:11:f1:01)». Сеть Ethernet доставляет этот запрос всем устройствам в том же сегменте Ethernet, в том числе и компьютеру Б. Компьютер Б отвечает компьютеру А на

запрос и сообщает свой MAC-адрес (напр. 00:ea:d1:11:f1:11) Теперь, получив MAC-адрес компьютера Б, компьютер А может передавать ему любые данные через сеть Ethernet.

Наибольшее распространение ARP получил благодаря повсеместности сетей IP, построенных поверх Ethernet, поскольку практически в 100 % случаев при таком сочетании используется ARP. В семействе протоколов IPv6 ARP не существует, его функции возложены на ICMPv6.

IP (Сетевой уровень):

Internet Protocol («межсетевой протокол») — маршрутизируемый протокол сетевого уровня стека TCP/IP. Именно IP стал тем протоколом, который объединил отдельные компьютерные сети во всемирную сеть Интернет.

IP объединяет сегменты сети в единую сеть, обеспечивая доставку пакетов данных между любыми узлами сети через произвольное число промежуточных узлов (маршрутизаторов). Он классифицируется как протокол третьего уровня по сетевой модели OSI. IP не гарантирует надёжной доставки пакета до адресата — в частности, пакеты могут прийти не в том порядке, в котором были отправлены, продублироваться (приходят две копии одного пакета), оказаться повреждёнными (обычно повреждённые пакеты уничтожаются) или не прийти вовсе. Гарантию безошибочной доставки пакетов дают некоторые протоколы более высокого уровня — транспортного уровня сетевой модели OSI, — например, TCP, которые используют IP в качестве транспорта.

В современной сети Интернет используется IP четвёртой версии, также известный как IPv4. В протоколе IP этой версии каждому узлу сети ставится в соответствие IP-адрес длиной 4 октета (4 байта). При этом компьютеры в подсетях объединяются общими начальными битами адреса. Количество этих бит, общее для данной подсети, называется маской подсети (ранее использовалось деление пространства адресов по классам — А, В, С; класс сети определялся диапазоном значений старшего октета и определял число адресуемых узлов в данной сети, сейчас используется бесклассовая адресация).

В настоящее время вводится в эксплуатацию шестая версия протокола — IPv6, которая позволяет адресовать значительно большее количество узлов, чем IPv4. Эта версия отличается повышенной разрядностью адреса, встроенной возможностью шифрования и некоторыми другими особенностями. Переход с IPv4 на IPv6 связан с трудоёмкой работой операторов связи и производителей программного обеспечения и не может быть выполнен одномоментно. На середину 2010 года в Интернете присутствовало более 3000 сетей, работающих по протоколу IPv6. Для сравнения, на то же время в адресном пространстве IPv4 присутствовало более 320 тысяч сетей, но в IPv6 сети гораздо более крупные, нежели в IPv4.

ICMP (Сетевой уровень):

ICMP (ang. Internet Control Message Protocol) - это один из протоколов сетевого уровня в модели ISO/OSI. Его задачей является обслуживание функции контроля правильности работы сети. С его помощью передаются всякого рода, низкоуровневые сводки, с раскрытыми неправильностями во время сетевых связей. Практически целая коммуникация между данными компьютерами или

другими устройствами при употреблении протокола ICMP происходит незаметным для конечного пользователя образом. Единичными исключениями являются здесь инструменты ping и traceroute. Коммуникация, использующая протокол ICMP, состоит в пересылке подходящих информации об ошибках, раскрытых во время связи между двумя устройствами. Одиночная информация существует в виде пакета, сформированного надлежащим образом (англ. Datagram), который, следовательно, будет подвергнут инкапсуляции в рамке протокола IP. Протокол ICMP, вопреки всеобщему мнению, не использует в своей работе протоколы TCP, ни UDP.

Примеры работы протокола ICMP:

Ping - один из инструментов, выступающих практически в каждой операционной системе, обслуживающей протокол TCP/IP. С его помощью пакеты ICMP ECHO_REQUEST отправляются в целевой компьютер. Дистанционная машина, после получения такого сообщения должна ответить при помощи ECHO_REPLY. Поэтому можно определить следующее: Конфигурация сети делает возможной связь с дистанционной машиной, и оценку её нагрузки на основании информации, касающихся количества потерянных пакетов и времени ответа.

Traceroute – инструмент, делающий возможным определение, через какие маршрутизаторы проходит пакет по дороге к дистанционному компьютеру. Сначала, локальный компьютер посылает пакет ECHO_REQUEST в дистанционное устройство, с параметром TTL (TTL -Time to Live), установленным на 1. Первый роутер уменьшает TTL на один, значит до нуля, удаляет пакет и отсылает адресату сообщение ICMP TIME_EXCEEDED. Целевой компьютер, после получения такой информации, возобновляет высылку ECHO_REQUEST, но с TTL установленным на стоимость 2. Первый роутер уменьшает TTL на 1, второй сделает то же самое, устанавливая 0, и вновь удалит пакет, и отошлёт сообщение TIME_EXCEEDED. Такая ситуация повторяется так долго, что даже пакет доберётся до дистанционного компьютера, который тогда отошлёт отправителю сообщение ECHO_REPLY

RARP (Сетевой уровень):

RARP (Reverse Address Resolution Protocol — Обратный протокол преобразования адресов) —выполняет обратное отображение адресов, то есть преобразует физический адрес в IP-адрес.

Протокол применяется во время загрузки узла (например компьютера), когда он посылает групповое сообщение-запрос со своим физическим адресом. Сервер принимает это сообщение и просматривает свои таблицы (либо перенаправляет запрос куда-либо ещё) в поисках соответствующего физическому, IP-адреса. После обнаружения найденный адрес отсылается обратно на запросивший его узел. Другие станции также могут «слышать» этот диалог и локально сохранить эту информацию в своих ARP-таблицах.

RARP позволяет разделять IP-адреса между не часто используемыми хост-узлами. После использования каким-либо узлом IP-адреса он может быть освобождён и выдан другому узлу.

TCP (Транспортный, сеансовый уровень):

TCP (англ. transmission control protocol — протокол управления передачей) — один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных. Сети и подсети, в которых совместно используются протоколы TCP и IP называются сетями TCP/IP.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым, в отличие от UDP, целостность передаваемых данных и уведомление отправителя о результатах передачи.

Реализации TCP обычно встроены в ядра ОС. Существуют реализации TCP, работающие в пространстве пользователя.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере (например, программы для электронной почты, для обмена файлами). TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик.

UDP (Транспортный, сеансовый уровень):

UDP (англ. User Datagram Protocol — протокол пользовательских датаграмм) — один из ключевых элементов TCP/IP, набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

UDP использует простую модель передачи, без неявных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Таким образом, UDP предоставляет ненадёжный сервис, и датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты, что может оказаться невозможным в системах реального времени.

Природа UDP как протокола без сохранения состояния также полезна для серверов, отвечающих на небольшие запросы от огромного числа клиентов, например DNS и потоковые мультимедийные приложения вроде IPTV, Voice over IP, протоколы туннелирования IP и многие онлайн-игры.

IP адреса

IP-адрес — уникальный сетевой адрес узла в компьютерной сети, построенной по протоколу IP. В сети Интернет требуется глобальная уникальность адреса; в случае работы в локальной сети требуется уникальность адреса в пределах сети. В версии протокола IPv4 IP-адрес имеет длину 4 байта, а в версии протокола IPv6 IP-адрес имеет длину 16 байт.

В 4-й версии IP-адрес представляет собой 32-битовое число. Удобной формой записи IP-адреса (IPv4) является запись в виде четырёх десятичных чисел значением от 0 до 255, разделённых точками, например, 192.168.0.3.

В 6-й версии IP-адрес (IPv6) является 128-битовым. Внутри адреса разделителем является двоеточие (напр. 2001:0db8:85a3:0000:0000:8a2e:0370:7334). Ведущие нули допускается в записи опускать. Нулевые группы, идущие подряд, могут быть опущены, вместо них ставится двойное двоеточие (fe80:0:0:0:0:0:1 можно записать как fe80::1). Более одного такого пропуска в адресе не допускается.

IP-адрес состоит из двух частей: номера сети и номера узла. В случае изолированной сети её адрес может быть выбран администратором из специально зарезервированных для таких сетей блоков адресов (10.0.0.0/8, 172.16.0.0/12 или 192.168.0.0/16). Если же сеть должна работать как составная часть Интернета, то адрес сети выдаётся провайдером либо региональным интернет-регистратором (Regional Internet Registry, RIR). Согласно данным на сайте IANA, существует пять RIR: ARIN, обслуживающий Северную Америку, а также Багамы, Пуэрто-Рико и Ямайку; APNIC, обслуживающий страны Южной, Восточной и Юго-Восточной Азии, а также Австралии и Океании; AfriNIC, обслуживающий страны Африки; LACNIC, обслуживающий страны Южной Америки и бассейна Карибского моря; и RIPE NCC, обслуживающий Европу, Центральную Азию, Ближний Восток. Региональные регистраторы получают номера автономных систем и большие блоки адресов у IANA, а затем выдают номера автономных систем и блоки адресов меньшего размера локальным интернет-регистраторам (Local Internet Registries, LIR), обычно являющимся крупными провайдерами. Номер узла в протоколе IP назначается независимо от локального адреса узла. Маршрутизатор по определению входит сразу в несколько сетей. Поэтому каждый порт маршрутизатора имеет собственный IP-адрес. Конечный узел также может входить в несколько IP-сетей. В этом случае компьютер должен иметь несколько IP-адресов, по числу сетевых связей. Таким образом, IP-адрес характеризует не отдельный компьютер или маршрутизатор, а одно сетевое соединение.

Типы адресации / Маски / Адрес сети

Иногда встречается запись IP-адресов вида 192.168.5.0/24. Данный вид записи заменяет собой указание диапазона IP-адресов. Число после косой черты означает количество единичных разрядов в маске подсети. Для приведённого примера маска подсети будет иметь двоичный вид 11111111 11111111 00000000 или то же самое в десятичном виде: 255.255.255.0. 24 разряда IP-адреса отводятся под номер сети, а остальные $32-24=8$ разрядов полного адреса — под адреса хостов этой сети, адрес этой сети и широковещательный адрес этой сети. Итого, 192.168.5.0/24 означает диапазон адресов хостов от 192.168.5.1 до 192.168.5.254, а также 192.168.5.0 — адрес сети и 192.168.5.255 — широковещательный адрес сети.

Первый IP адрес в диапазоне (со всеми нулями в адресе хоста) – **адрес сети**.

Последний (со всеми единицами в адресе хоста) - **широковещательный адрес** (воспринимается всеми компьютерами сети как дополнительный свой адрес, то есть пакет на этот адрес получают все

хосты сети как адресованные лично им. Если на сетевой интерфейс хоста, который не является маршрутизатором пакетов, попадет пакет, адресованный не ему, то он будет отброшен).

Запись IP-адресов с указанием через слэш маски подсети переменной длины также называют CIDR-адресом в противоположность обычной записи без указания маски, в операционных системах типа UNIX также именуемой INET-адресом.

Особые адреса

В протоколе IP существует несколько соглашений об особой интерпретации IP-адресов: если все двоичные разряды IP-адреса равны 1, то пакет с таким адресом назначения должен рассылаться всем узлам, находящимся в той же сети, что и источник этого пакета. Такая рассылка называется ограниченным широковещательным сообщением (limited broadcast). Если в поле номера узла назначения стоят только единицы, то пакет, имеющий такой адрес, рассылается всем узлам сети с заданным номером сети. Например, в сети 192.168.5.0 с маской 255.255.255.0 пакет с адресом 192.168.5.255 доставляется всем узлам этой сети. Такая рассылка называется широковещательным сообщением (direct broadcast).

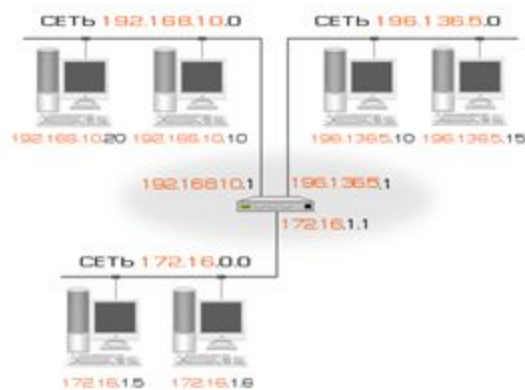
Установление соединения / Шлюзы / Маршрутизация

Связь между TCP/IP-компьютером и узлом из другой сети обычно осуществляется через устройство, называемое маршрутизатором. С точки зрения TCP/IP маршрутизатор, указанный на узле, связывающем подсеть узла с другими сетями, называется основным шлюзом. В этом разделе рассказывается, каким образом протокол TCP/IP определяет, отправлять или нет пакеты данных на основной шлюз, чтобы связаться с другим компьютером или устройством в сети.

При попытке установления связи между узлом и другим устройством с помощью протокола TCP/IP узел сопоставляет определенную маску подсети и IP-адрес назначения с маской подсети и своим собственным IP-адресом. В результате этого сопоставления компьютер узнает, для какого из узлов предназначен данный пакет – локального или удаленного.

Если в результате этого процесса назначением является локальный узел, то компьютер просто отправляет пакет в локальную подсеть. Если в результате сопоставления выясняется, что назначением является удаленный узел, компьютер направляет пакет на основной шлюз, определенный в свойствах TCP/IP. Таким образом, именно маршрутизатор отвечает за отправку пакета в правильную подсеть.

Маршрутизация осуществляется на узле-отправителе в момент отправки IP-пакета, а затем на IP-маршрутизаторах.



Выбор пути на маршрутизаторе осуществляется на основе информации, представленной в таблице маршрутизации. Таблица маршрутизации – это специальная таблица, сопоставляющая IP-адресам сетей адреса следующих маршрутизаторов, на которые следует отправлять пакеты с целью их доставки в эти сети. Обязательной записью в таблице маршрутизации является так называемый маршрут по умолчанию, содержащий информацию о том, как направлять пакеты в сети, адреса которых не присутствуют в таблице, поэтому нет необходимости описывать в таблице маршруты для всех сетей. Таблицы маршрутизации могут строиться «вручную» администратором или динамически, на основе обмена информацией, который осуществляют маршрутизаторы с помощью специальных протоколов.

Трансляция адресов (NAT)

Преобразование адреса методом NAT может производиться почти любым маршрутизирующим устройством — маршрутизатором, сервером доступа, межсетевым экраном. Наиболее популярным является SNAT, суть механизма которого состоит в замене адреса источника (англ. source) при прохождении пакета в одну сторону и обратной замене адреса назначения (англ. destination) в ответном пакете. Наряду с адресами источник/назначение могут также заменяться номера портов источника и назначения.

Принимая пакет от локального компьютера, роутер смотрит на IP-адрес назначения. Если это локальный адрес, то пакет пересылается другому локальному компьютеру. Если нет, то пакет надо переслать наружу в интернет. Но ведь обратным адресом в пакете указан локальный адрес компьютера, который из интернета будет недоступен. Поэтому роутер «на лету» транслирует (подменяет) обратный IP-адрес пакета на свой внешний (видимый из интернета) IP-адрес и меняет номер порта (чтобы различать ответные пакеты, адресованные разным локальным компьютерам). Комбинацию, нужную для обратной подстановки, роутер сохраняет у себя во временной таблице. Через некоторое время после того, как клиент и сервер закончат обмениваться пакетами, роутер сотрет у себя в таблице запись о n-ом порте за сроком давности.

Помимо source NAT (предоставления пользователям локальной сети с внутренними адресами доступа к сети Интернет) часто применяется также destination NAT, когда обращения извне транслируются межсетевым экраном на компьютер пользователя в локальной сети, имеющий внутренний адрес и потому недоступный извне сети непосредственно (без NAT).

Примеры протоколов:

HTTP (TCP)

HTTP (HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных). Основой HTTP является технология «клиент-сервер», то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

Текущая версия протокола - HTTP/1.1. Новым в этой версии был режим «постоянного соединения»: TCP-соединение может оставаться открытым после отправки ответа на запрос, что позволяет посылать несколько запросов за одно соединение. Клиент теперь обязан посылать информацию об имени хоста, к которому он обращается, что сделало возможной более простую организацию виртуального хостинга.

Каждое HTTP-сообщение состоит из трёх частей: стартовая строка — определяет тип сообщения; заголовки — характеризуют тело сообщения, параметры передачи и прочие сведения, тело сообщения — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

23. Надежность систем. Показатели надежности. Уровни надежности по классификации НР

Надежность – свойство объекта выполнять заданные функции, сохраняя во времени и в заданных пределах значения установленных эксплуатационных показателей.

Объект – техническое изделие определенного целевого назначения, рассматриваемое в периоды проектирования, производства, испытаний и эксплуатации. Объектами могут быть различные системы и их элементы.

Элемент – простейшая составная часть изделия, в задачах надежности может состоять из многих деталей.

Система – совокупность совместно действующих элементов, предназначенная для самостоятельного выполнения заданных функций. Понятия элемента и системы трансформируются в зависимости от поставленной задачи. Например, станок, при установлении его собственной надежности рассматривается как система, состоящая из отдельных элементов – механизмов, деталей и т.п., а при изучении надежности технологической линии – как элемент. Надежность объекта характеризуется следующими основными состояниями и событиями.

Применение (использование) объекта по назначению прекращается в следующих случаях:

- при неустранимом нарушении безопасности;
- при неустранимом отклонении величин заданных параметров;
- при недопустимом увеличении эксплуатационных расходов.

К числу невосстанавливаемых объектов можно отнести, например: подшипники качения, полупроводниковые изделия, зубчатые колеса и т.п. Объекты, состоящие из многих элементов, например, станок, автомобиль, электронная аппаратура, являются восстанавливаемыми, поскольку их отказы связаны с повреждениями одного или немногих элементов, которые могут быть заменены.

В ряде случаев один и тот же объект в зависимости от особенностей, этапов эксплуатации или назначения может считаться восстанавливаемым или невосстанавливаемым.

Отказ – событие, заключающееся в нарушении работоспособного состояния объекта.

Критерий отказа – отличительный признак или совокупность признаков, согласно которым устанавливается факт возникновения отказа.

По типу отказы подразделяются на:

- отказы функционирования (выполнение основных функций объектом прекращается, например, поломка зубьев шестерни);
- отказы параметрические (некоторые параметры объекта изменяются в недопустимых пределах, например, потеря точности станка).

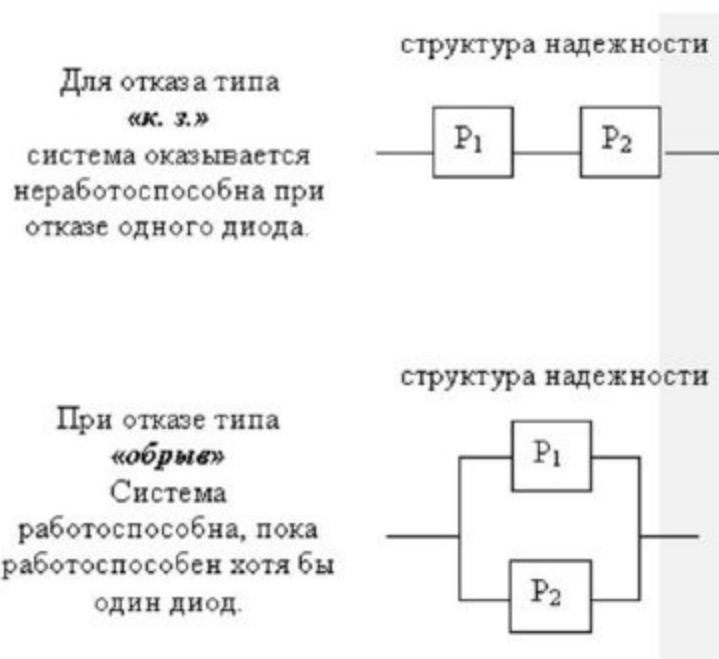
По своей природе отказы могут быть:

- случайные, обусловленные непредусмотренными перегрузками, дефектами материала, ошибками персонала или сбоями системы управления и т. п.;
- систематические, обусловленные закономерными и неизбежными явлениями, вызывающими постепенное накопление повреждений: усталость, износ, старение, коррозия и т. П.

По структуре системы могут быть:

- система без резервирования (основная система);
- системы с резервированием.

Для одних и тех же систем могут быть составлены различные структурные схемы надежности в зависимости от вида отказов элементов.

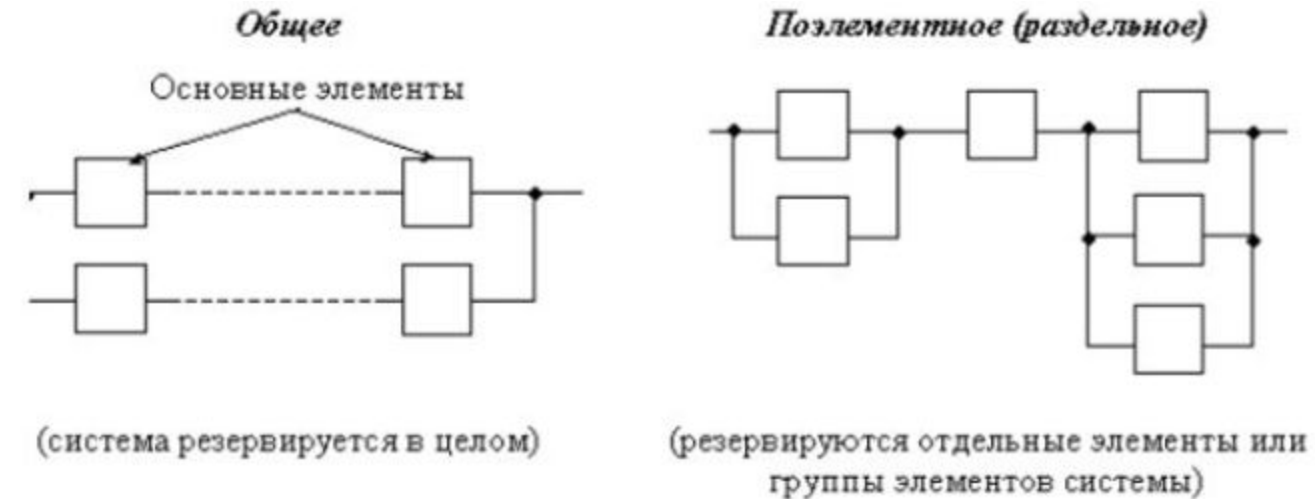


Работоспособность систем без резервирования требует работоспособности всех элементов системы. В сложных технических устройствах без резервирования никогда не удастся достичь высокой надежности даже, если использовать элементы с высокими показателями безотказности.

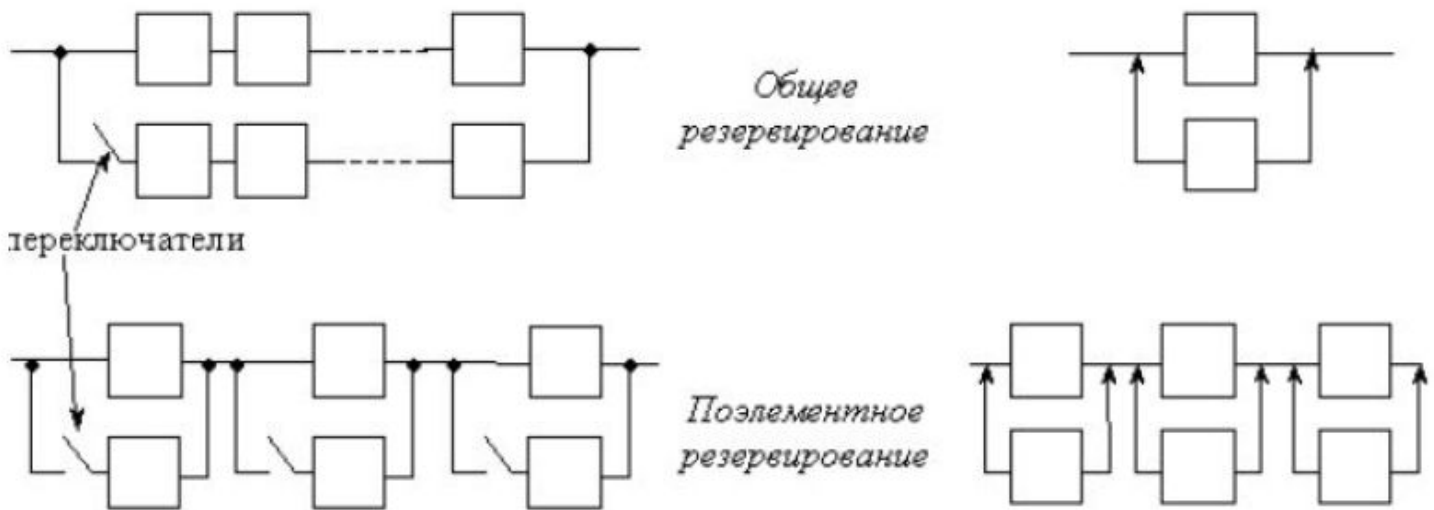
Система с резервированием – это система с избыточностью элементов, т. е. с резервными составляющими, избыточными по отношению к минимально необходимой (основной) структуре и выполняющими те же функции, что и основные элементы. В системах с резервированием работоспособность обеспечивается до тех пор, пока для замены отказавших основных элементов имеются в наличии резервные.

Структурное резервирование может быть:

- пассивное (нагруженное) – резервные элементы функционируют наравне с основными (постоянно включены в работу);
- активное (ненагруженное) – резервные элементы вводятся в работу только после отказа основных элементов (резервирование замещением).



Примеры ненагруженного резервирования:



Показатели надёжности количественно характеризуют, в какой степени данному объекту присущи определенные свойства, обуславливающие надёжность.

Показатели надёжности (например, технический ресурс, срок службы) могут иметь размерность, ряд других (например, вероятность безотказной работы, коэффициент готовности), являются безразмерными.

* Единичные показатели надёжности

○ Показатели безотказности

1. вероятность безотказной работы ($P(t)$) - это вероятность того, что в пределах заданной наработки или заданном интервале времени отказ объекта не возникает;

2. средняя наработка до отказа ($T_{ср}$) - эквивалентный параметр для неремонтопригодного устройства. Поскольку устройство не восстанавливается, то это просто среднее время, которое проработает устройство до того момента, как сломается;
 3. средняя наработка на отказ (T_o) - технический параметр, характеризующий надёжность восстанавливаемого прибора, устройства или технической системы;
 4. интенсивность отказов ($\lambda(t)$) - отношение числа отказавших объектов (образцов аппаратуры, изделий, деталей, механизмов, устройств, узлов и т. п.) в единицу времени к среднему числу объектов, исправно работающих в данный отрезок времени при условии, что отказавшие объекты не восстанавливаются и не заменяются исправными. Другими словами, интенсивность отказов численно равна числу отказов в единицу времени, отнесенное к числу узлов, безотказно проработавших до этого времени;
- Показатели долговечности
 1. средний ресурс - это математическое ожидание ресурса;
 2. гамма-процентный ресурс - это наработка, в течение которой объект не достигнет предельного состояния с заданной вероятностью, выраженной в процентах;
 3. назначенный ресурс - суммарная наработка, при достижении которой эксплуатация объекта должна быть прекращена независимо от его технического состояния;
 4. средний срок службы - математическое ожидание срока службы;
 5. гамма-процентный срок службы - календарная продолжительность от начала эксплуатации объекта, в течение которой он не достигнет предельного состояния с заданной вероятностью γ , выраженной в процентах.
 6. назначенный срок службы - календарная продолжительность эксплуатации объекта, при достижении которой применение по назначению должно быть прекращено.
 - Показатели ремонтпригодности
 1. вероятность восстановления работоспособного состояния - вероятность того, что время восстановления работоспособного состояния не превысит заданного.
 2. среднее время восстановления работоспособного состояния - математическое ожидание времени восстановления.
 3. интенсивность восстановления
 - Показатели сохраняемости
 1. средний срок сохраняемости - календарная продолжительность хранения и (или) транспортирования объекта, в течение которой сохраняются в заданных пределах значения параметров, характеризующих способность объекта выполнять заданные функции;

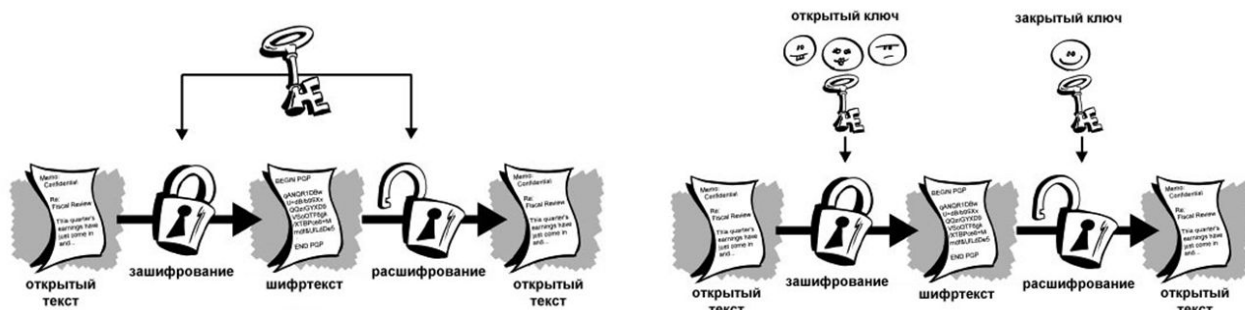
2. гамма-процентный срок сохраняемости - срок сохраняемости, достигаемый объектом с заданной вероятностью γ , выраженной в процентах.

* Комплексные показатели надёжности

- коэффициент готовности (K_g) — вероятность того, что объект окажется в работоспособном состоянии в произвольный момент времени, кроме планируемых периодов, в течение которых применение объекта по назначению не предусматривается.
- коэффициент оперативной готовности ($K_{ог}$)
- коэффициент технического использования ($K_{ти}$) — отношение математического ожидания интервалов времени, пребывания объекта в работоспособном состоянии за некоторый период эксплуатации к сумме математических ожиданий интервалов времени пребывания объекта в работоспособном состоянии, простоев, обусловленных техническим обслуживанием (ТО), и ремонтов за тот же период эксплуатации.
- коэффициент планируемого применения (K_p) — доля периода эксплуатации, в течение которой объект не должен находиться в плановом ТО или ремонте.
- коэффициент сохранения эффективности ($K_{эф}$)

24. Системы шифрования. Виды алгоритмов шифрования, общие характеристики. Общая архитектура PGP. Сертификаты. Назначение, устройство, области применения

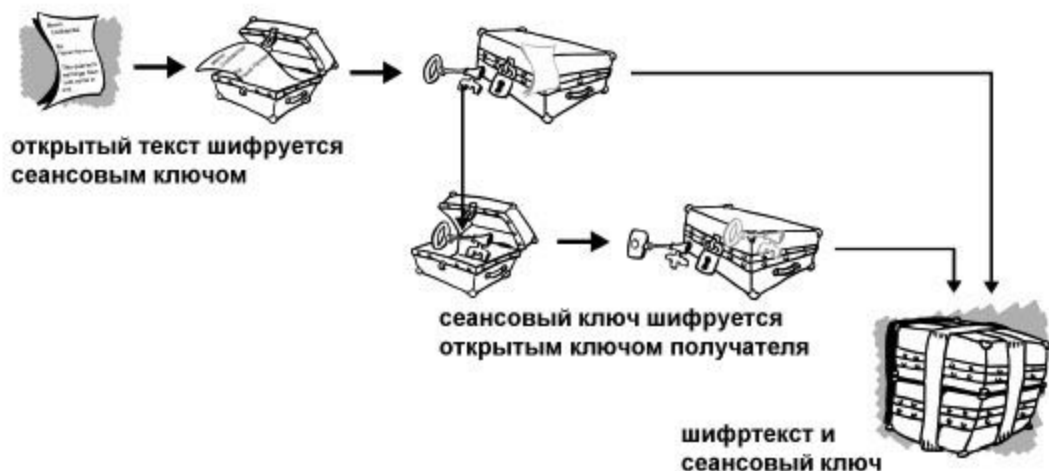
- Симметричные (с секретным, единым ключом, одноключевые, single-key)
 - Поточковые (шифрование потока данных):
 - с одноразовым или бесконечным ключом (infinite-key cipher);
 - с конечным ключом (система Вернама - Vernam);
 - на основе генератора псевдослучайных чисел (ПСЧ).
 - Блочные (шифрование данных поблочно):
 - Шифры перестановки (permutation, P-блоки);
 - Шифры замены (подстановки, substitution, S-блоки):
 - моноалфавитные (код Цезаря);
 - полиалфавитные (шифр Видженера, цилиндр Джефферсона, диск Уэтстоуна, Enigma);
 - Составные (таблица 1):
 - Lucifer (фирма IBM, США);
 - DES (Data Encryption Standard, США);
 - FEAL-1 (Fast Enciphering Algorithm, Япония);
 - IDEA/IPES (International Data Encryption Algorithm/Improved Proposed Encryption Standard, фирма Ascom-Tech AG, Швейцария);
 - B-Crypt (фирма British Telecom, Великобритания);
 - ГОСТ 28147-89 (СССР); * Skipjack (США).
- Асимметричные (с открытым ключом, public-key):
 - Диффи-Хеллман DH (Diffie, Hellman);
 - Райвест-Шамир-Адлеман RSA (Rivest, Shamir, Adleman);
 - Эль-Гамаль ElGamal.



Общая архитектура PGP (Pretty Good Privacy)

Пакет обеспечивает конфиденциальность переписки и пересылки файлов. Конфиденциальность заключается в том, что только лица, кому адресовано послание или файл, смогут его прочесть. PGP поддерживает технологию электронной подписи, гарантирующей получателю, что сообщение пришло именно от вас.

Шифрование PGP осуществляется последовательно хешированием, сжатием данных, шифрованием с симметричным ключом, и, наконец, шифрованием с открытым ключом, причём каждый этап может осуществляться одним из нескольких поддерживаемых алгоритмов. Симметричное шифрование производится с использованием одного из семи симметричных алгоритмов (AES, CAST5, 3DES, IDEA, Twofish, Blowfish, Camellia) на сеансовом ключе. Сеансовый ключ генерируется с использованием криптографически стойкого генератора псевдослучайных чисел. Сеансовый ключ зашифровывается открытым ключом получателя с использованием алгоритмов RSA или Elgamal. Каждый открытый ключ соответствует имени пользователя или адресу электронной почты.



Ключи

Пользователь PGP создаёт ключевую пару: открытый и закрытый ключ. При генерации ключей задаются их владелец (имя и адрес электронной почты), тип ключа, длина ключа и срок его действия. Открытый ключ используется для шифрования и проверки цифровой подписи. Закрытый ключ — для декодирования и создания цифровой подписи.

Цифровая подпись

PGP поддерживает аутентификацию и проверку целостности посредством цифровой подписи. По умолчанию она используется совместно с шифрованием, но также может быть применена и к открытому тексту. Отправитель использует PGP для создания подписи алгоритмом RSA или DSA.

При этом сначала создаётся хеш открытого текста (также известный как дайджест), затем — цифровая подпись хеша при помощи закрытого ключа отправителя.



Хэши

Среди множества существующих хеш-функций принято выделять криптографически стойкие, применяемые в криптографии. Для того чтобы хеш-функция H считалась криптографически стойкой, она должна удовлетворять трем основным требованиям, на которых основано большинство применений хеш-функций в криптографии:

- **Необратимость:** для заданного значения хеш-функции m должно быть вычислительно неосуществимо найти блок данных X , для которого $H(X)=m$.
- **Стойкость к коллизиям первого рода:** для заданного сообщения M должно быть вычислительно неосуществимо подобрать другое сообщение N , для которого $H(N)=H(M)$.
- **Стойкость к коллизиям второго рода:** должно быть вычислительно неосуществимо подобрать пару сообщений, имеющих одинаковый хеш.

Сжатие данных

В целях уменьшения объёма сообщений и файлов и, возможно, для затруднения криптоанализа PGP производит сжатие данных перед шифрованием. Сжатие производится по одному из алгоритмов ZIP, ZLIB, BZIP2. Для сжатых, коротких и слабосжимаемых файлов сжатие не выполняется.

Сертификаты

Цифровой сертификат состоит из трёх компонентов:

- открытого ключа;
- сведений сертификата (информация о личности пользователя, как то: имя, ID, и т.п.);
- одной или более цифровых подписей.

Цель ЭЦП на сертификате — указать, что сведения сертификата были заверены третьим лицом или организацией. В то же время цифровая подпись не подтверждает достоверность сертификата как целого; она является поручительством только того, что подписанный ID связан с данным открытым ключом.

Таким образом, сертификат, обычно, — это открытый ключ с прикрепленными к нему одной или несколькими формами ID плюс отметка подтверждения от доверенного лица.

Сертификаты применяются, когда нужно обменяться с кем-нибудь ключами.

Способы распространения сертификатов

- Ручной
- В виде хранилища-депозитария (сервера сертификатов)

Ручной способ характерен для небольших групп людей, устанавливающих криптографированную связь (не составит труда просто передать друг другу дискеты или отправить электронные письма, содержащие копии их ключей).

При необходимости обеспечить достаточную надёжность и безопасность, предоставления возможности хранения и обмена ключами, используют системный подход.

Такая система может реализоваться в форму простого хранилища-депозитария или иметь более сложную и комплексную структуру, предполагающую дополнительные возможности администрирования ключей, и называемую инфраструктурой открытых ключей (Public Key Infrastructure, PKI).

Сервер-депозитарий, также называемый сервером сертификатов, или сервером ключей, — это база данных, позволяющая пользователям оставлять и извлекать из неё цифровые сертификаты.

Сервер ключей также может предоставлять некоторые административные функции, помогающие компании поддерживать свою политику безопасности. Например, на хранение могут оставаться только ключи, удовлетворяющие определённым критериям.

PKI, как и сервер-депозитарий, имеет базу хранения сертификатов, но, в то же время, предоставляет сервисы и протоколы по управлению открытыми ключами. В них входят возможности выпуска, отзыва и системы доверия сертификатов. Главной же возможностью PKI является введение компонентов, известных как Центр сертификации (Certification Authority, CA) и Центр регистрации (Registration Authority, RA).

Центр сертификации (ЦС) создаёт цифровые сертификаты и подписывает их своим закрытым ключом. Из-за важности своей роли, ЦС является центральным компонентом инфраструктуры PKI. Используя открытый ключ ЦС, любой пользователь, желающий проверить достоверность (подлинность) конкретного сертификата, сверяет подпись Центра сертификации и, следовательно, удостоверяется в целостности содержащейся в сертификате информации.

Как правило, Центром регистрации (ЦР) называется система лиц, процессов и устройств, служащая целям регистрации новых пользователей в структуре PKI (зачислению) и дальнейшему администрированию постоянных пользователей системы. Также, ЦР может производить «веттинг» — процедуру проверки того, принадлежит ли конкретный открытый ключ предполагаемому владельцу.

ЦР — это человеческое сообщество: лицо, группа, департамент, компания или иная ассоциация. С другой стороны, ЦС — обычно, программа, выдающая сертификаты своим зарегистрированным пользователям.

Роль ЦР-ЦС аналогична той, что выполняет государственный паспортный отдел.

26. Написание SQL запроса к реляционной базе данных в соответствии со стандартом SQL-92. В соответствии с описанием требуемой структуры данных и описанием таблиц и полей данных реляционной базы данных написать запрос, использующий основные конструкции языка SQL в соответствии со стандартом SQL-92, включая объединение таблиц, вложенные подзапросы, агрегирование и сортировку данных

Структура запроса:

```
SELECT [DISTINCT | DISTINCTROW | ALL]
       select_expression,...
FROM table_references
     [WHERE where_definition]
     [GROUP BY {unsigned_integer | col_name | formula}]
     [HAVING where_definition]
     [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
```

Выбор всех столбцов таблицы:

```
SELECT * FROM Salespeople
```

Выбор определенных столбцов таблицы:

```
SELECT sname, city FROM Salespeople
```

Получение списка без дубликатов:

```
SELECT DISTINCT sname FROM Orders
```

Простое условие:

```
SELECT sname, city FROM Salespeople WHERE city = 'London'
```

Сортировка:

ORDER BY осуществляет упорядочение строк результата по значению одного или нескольких столбцов с указанием порядка сортировки или без такого указания. При включении в список ORDER BY нескольких столбцов СУБД сортирует строки результата по значениям первого столбца списка пока не появится несколько строк с одинаковыми значениями данных в этом столбце.

Кроме того, в список ORDER BY можно включать не только имя столбца, а его порядковую позицию в перечне SELECT. Благодаря этому возможно упорядочение результатов на основе вычисляемых столбцов, не имеющих имен.

Условия:

В синтаксисе фразы WHERE показано, что для отбора нужных строк таблицы можно использовать операторы сравнения = (равно), <> (не равно), < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), которые могут предваряться оператором NOT, создавая, например, отношения "не меньше" и "не больше".

Возможность использования нескольких условий, соединенных логическими операторами AND, OR, AND NOT и OR NOT, позволяет осуществить более детальный отбор строк.

С помощью BETWEEN ... AND ... (находится в интервале от ... до ...) можно отобрать строки, в которых значение какого-либо столбца находится в заданном диапазоне.

BETWEEN особенно удобен при работе с данными, задаваемыми интервалами, начало и конец которых расположен в разных столбцах.

```
WHERE Начало BETWEEN '1-9-1993' AND '31-8-1994'
```

```
WHERE '15-05-1994' BETWEEN Начало AND Конец
```

Рассмотренная форма IN является в действительности просто краткой записью последовательности отдельных сравнений, соединенных операторами OR. Можно задать и NOT IN (не принадлежит), а также возможность использования IN (NOT IN) с подзапросом.

```
WHERE Основа IN (Яйца Крупа Овощи);
```

Обычная форма "имя_столбца LIKE текстовая_константа" для столбца текстового типа позволяет отыскать все значения указанного столбца, соответствующие образцу, заданному "текстовой_константой". Символы этой константы интерпретируются следующим образом:

- символ _ (подчеркивание) – заменяет любой одиночный символ
- символ % (процент) – заменяет любую последовательность из N символов (где N может быть нулем)
- все другие символы означают просто сами себя

```
WHERE Блюдо LIKE '%салат%'
```

Если при загрузке данных не введено значение в какое-либо поле таблицы, то СУБД поместит в него NULL-значение. Аналогичное значение можно ввести в поле таблицы, выполняя операцию изменения данных. Так, при отсутствии сведений о наличии у поставщиков судака и моркови в столбцы Цена и К_во соответствующих строк таблицы Поставки вводится NULL и там будет храниться код NULL-значения, а не 0 или пробел.

Группирование:

Фраза GROUP BY инициирует перекомпоновку указанной во FROM таблицы по группам, каждая из которых имеет одинаковые значения в столбце, указанном в GROUP BY. Далее к каждой группе применяется фраза SELECT. Отметим, что фраза GROUP BY не предполагает ORDER BY по умолчанию.

Если в запросе используются фразы WHERE и GROUP BY, то строки, не удовлетворяющие фразе WHERE, исключаются до выполнения группирования.

Фраза HAVING играет такую же роль для групп, что и фраза WHERE для строк: она используется для исключения групп, точно так же, как WHERE используется для исключения строк.

Агрегационные функции:

COUNT - число значений в столбце,

SUM - сумма значений в столбце,

AVG - среднее значение в столбце,

MAX - самое большое значение в столбце,

MIN - самое малое значение в столбце.

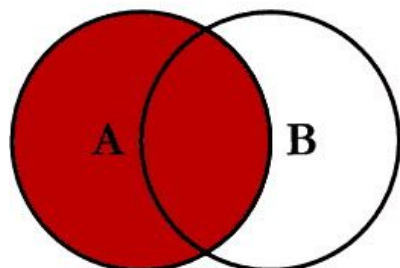
Для функций SUM и AVG рассматриваемый столбец должен содержать числовые значения.

Аргументу всех функций, кроме COUNT(*), может предшествовать ключевое слово DISTINCT (различный), указывающее, что избыточные дублирующие значения должны быть исключены перед тем, как будет применяться функция. Специальная же функция COUNT(*) служит для подсчета всех без исключения строк в таблице (включая дубликаты).

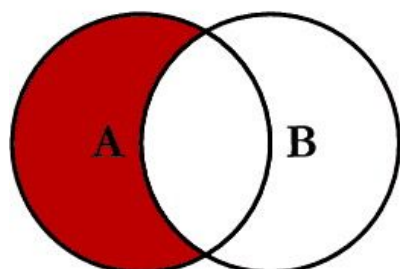
```
SELECT SUM(Цена) , AVG(Цена) , COUNT(Цена) , COUNT(DISTINCT Цена) , COUNT(*) FROM Поставки  
WHERE ПС = 5
```

Пересечение таблиц:

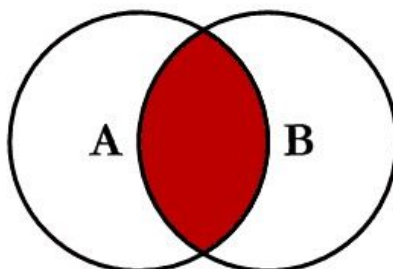
SQL JOINS



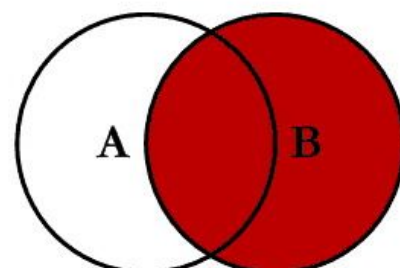
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



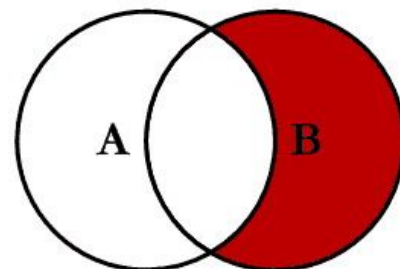
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



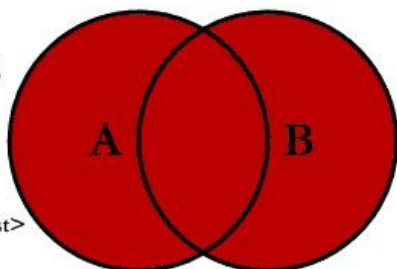
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



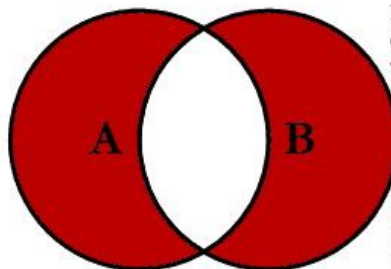
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

Вложенные подзапросы:

Простые вложенные подзапросы используются для представления множества значений, исследование которых должно осуществляться в каком-либо предикате (IN, EXISTS, ...)

```
SELECT Название, Статус
FROM Поставщики
WHERE ПС IN (SELECT ПС FROM Поставки WHERE ПП = 11);
```

Объединение таблиц:

Операция "Объединение", позволяющая получить отношение, состоящее из всех строк, входящих в одно или оба объединяемых отношения. Но при этом исходные отношения или их объединяемые проекции должны быть совместимыми по объединению. Для SQL это означает, что две таблицы можно объединять тогда и только тогда, когда:

- они имеют одинаковое число столбцов
- для всех i ($i = 1, 2, \dots, m$) i -й столбец первой таблицы и i -й столбец второй таблицы имеют в точности одинаковый тип данных

```
SELECT Продукт FROM Продукты WHERE Жиры = 0
UNION SELECT Продукт FROM Состав WHERE БЛ = 1
```

27. Написание сценария на языке bash для автоматизации операций администрирования операционных систем семейства Linux. В соответствии с спецификацией требований написать сценарий на языке bash для автоматизации одной из следующих операций: Анализ строк текстовых журналов с использованием регулярных выражений и вывод результатов в заданном формате. Управление файлами, каталогами, ссылками, правами доступа к объектам системы. Мониторинг данных о процессах, управление приоритетами их выполнения. Работа выполняется в среде виртуализации Oracle Virtual Box 4 в виртуальной машине под управлением Linux CentOS 6.7

команда > файл – перенаправление стандартного вывода в файл, содержимое существующего файла удаляется

команда >> файл – перенаправление стандартного вывода в файл, поток дописывается в конец файла

команда1 | команда2 – перенаправление стандартного вывода первой команды на стандартный ввод второй команды = образование конвейера команд

команда1 \$(команда2) – передача вывода команды 2 в качестве параметров при запуске команды 1. Внутри скрипта конструкция \$(команда2) может использоваться, например, для передачи результатов работы команды 2 в параметры цикла for ... in.

Работа со строками (внутренние команды bash)

\${#string} – выводит длину строки (string – имя переменной);

\${string:position:length} – извлекает \$length символов из \$string, начиная с позиции \$position. Частный случай: \${string:position} извлекает подстроку из \$string, начиная с позиции \$position.

\${string#substring} – удаляет самой короткой из найденных подстроки \$substring в строке \$string. Поиск ведется с начала строки. \$substring – регулярное выражение (см. ниже).

\${string##substring} – удаляет самую длинную из найденных подстроки \$substring в строке \$string. Поиск ведется с начала строки. \$substring – регулярное выражение.

\${string/substring/replacement} – замещает первое вхождение \$substring строкой \$replacement. \$substring – регулярное выражение.

\${string//substring/replacement} – замещает все вхождения \$substring строкой \$replacement. \$substring – регулярное выражение.

Работа со строками (внешние команды)

Для каждой команды доступно управление с помощью передаваемых команде параметров. Рекомендуем ознакомиться с документацией по этим командам с помощью команды man.

sort – сортирует поток текста в порядке убывания или возрастания, в зависимости от заданных опций.

uniq – удаляет повторяющиеся строки из отсортированного файла.

cut – извлекает отдельные поля из текстовых файлов (поле – последовательность символов в строке до разделителя).

head – выводит начальные строки из файла на stdout.

tail – выводит последние строки из файла на stdout.

wc – подсчитывает количество слов/строк/символов в файле или в потоке

tr – заменяет одни символы на другие.

Полнофункциональные многоцелевые утилиты

grep – многоцелевая поисковая утилита, использующая регулярные выражения.

sed – неинтерактивный "поточковый редактор".

awk – утилита контекстного поиска и преобразования текста в формате полей.

Регулярные выражения

Их основное назначение – поиск текста по шаблону и работа со строками. При построении регулярных выражений используются нижеследующие конструкции (в порядке убывания приоритета), некоторые из которых могут быть использованы только в расширенных версиях соответствующих команд (например, при запуске **grep** с ключом **-E**).

c Любой неспециальный символ соответствует самому себе

\c Указание убрать любое специальное значение символа **c** (экранирование)

^ Начало строки

\$ Конец строки; выражение **^\$** соответствует пустой строке.

. Любой одиночный символ, за исключением символа перевода строки

[...] Любой символ из ...; допустимы диапазоны типа **a-z**; возможно объединение диапазонов, например **[a-z0-9]**

[^...] Любой символ не из ...; допустимы диапазоны

\n Строка, соответствующая **n**-му выражению **\(...\)**

r* Ноль или более вхождений символа **r**

r+ Одно или более вхождений символа **r**

r? Ноль или одно вхождение символа **r**

\<...\> Границы слова

\{ \} Число вхождений предыдущего выражения. Например, выражение **"[0-9]\{5\}"** соответствует подстроке из пяти десятичных цифр

r1r2 За **r1** следует **r2**

r1|r2 **r1** или **r2**

(r) Регулярное выражение **r**; может быть вложенным

Работа с процессами

/proc – псевдо-файловая система, которая используется в качестве интерфейса к структурам данных в ядре.

ps – получение общих сведений о запущенных процессах

pstree – выводит процессы в форме дерева

top – программа, используемая для наблюдения за процессами в режиме реального времени

stat – детальная информация о процессе в виде набора полей;

status – предоставляет бо́льшую часть информации из **stat** в более лёгком для прочтения формате.

statm – предоставляет информацию о состоянии памяти в страницах как единицах измерения.

kill – передает сигнал процессу.

killall – работает аналогично команде **kill**, но использует его символьное имя, а не PID;

pidof – определяет PID процесса по его имени;

pgrep - определяет PID процессов с заданными характеристиками

pkill - позволяет отправить сигнал группе процессов с заданными характеристиками;

nice - запускает процесс с заданным значением приоритета.

renice - изменяет значения приоритета для запущенного процесса.

at - осуществляет однократный отсроченный запуск команды.

cron - демон, который занимается планированием и выполнением команд

sleep - задает паузу в выполнении скрипта.

Сигналы

Генератор	Обработчик
<pre>#!/bin/bash while true; do read LINE case \$LINE in STOP) kill -USR1 \$(cat .pid) ;; *) : ;; esac done</pre>	<pre>#!/bin/bash echo \$\$ > .pid A=1 MODE="rabota" usr1() { MODE="ostanov" } trap 'usr1' USR1 while true; do case \$MODE in "rabota") let A=\$A+1 echo \$A ;; "ostanov") echo "Stopped by SIGUSR1" exit ;; esac sleep 1 done</pre>

Файловая система

cd - смена каталога

cp - копирование файлов

ls - выводит список файлов и каталогов текущей директории

file - указывает тип указанного файла

find - поиск файлов

ln - создание ссылок

mkdir - создание каталога

mv - перемещение файла или каталога

pwd - вывод имени текущего каталога

rm - удаления файла

rmdir - удаление каталога

cat - слияние и вывод файлов

Пользователи и права

groupadd, useradd, groupdel, userdel, groupmod, usermod, openssl, chage, chmod, chgrp, chown, groups, id.

28. Настройка статической маршрутизации для IPv4 в программных маршрутизаторах Windows Server или Linux по выбору. При решении задачи используется схема клиент-маршрутизатор-маршрутизатор-клиент. Требуется подобрать адреса сетей по указанным требованиям (количество хостов в сети), настроить сетевые интерфейсы клиентов и маршрутизаторов, настроить таблицы маршрутизаторов. Работа выполняется в среде виртуализации Oracle Virtual Box 4 в ОС Windows 2003 или Linux Centos 6.7

Linux (делайте в windows, глупцы)

ip link - посмотреть список и имена сетевых интерфейсов

В директории /etc/sysconfig/network-scripts/ нас интересуют файлы ifcfg-[имя интерфейса].

Следует переименовать (скопировать) имеющийся там файл настройки для каждого из настраиваемого интерфейса. Далее в каждом файле нужно исправить часть параметров на указанные ниже в соответствии с конфигурацией сети:

```
DEVICE=[имя интерфейса]
HWADDR - стереть во избежание конфликтов на физическом уровне
UUID - тоже стереть
ONBOOT=yes (чтобы не делать ifup каждый раз после перезагрузки)
BOOTPROTO=none
IPADDR=[адрес машины]
NETMASK=[маска сети]
NETWORK=[адрес сети]
```

После чего необходимо активировать все настроенные интерфейсы с помощью **ifup [имя интерфейса]**

Для надежности можно перезагрузить машины.

Для активации переадресации пакетов в файле /etc/sysctl.conf

Необходимо установить значение параметра: net.ipv4.ip_forward = 1, после чего перезагрузить систему

Для очистки и разрешения файерволом (делать при каждом перезапуске):

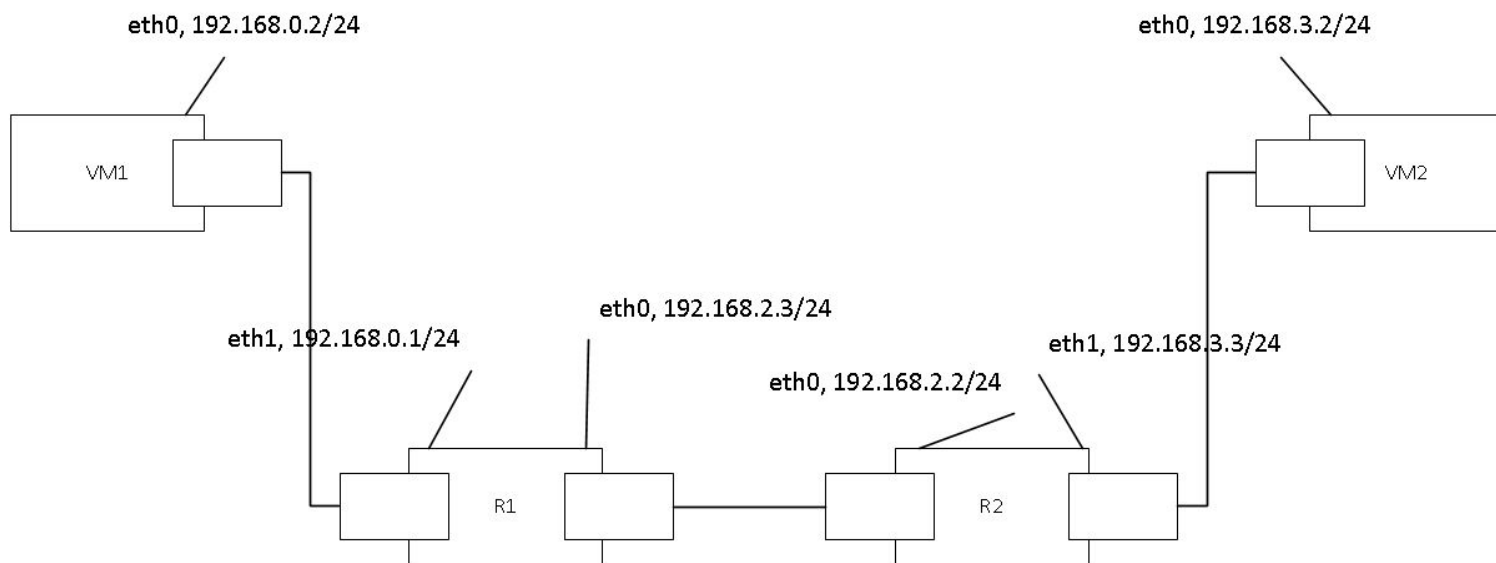
```
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
```

```
iptables -P OUTPUT ACCEPT
```

Для добавления статического маршрута используется команда (вводить после каждой перезагрузки)
`ip route add [адрес сети]/[маска один числом] via [адрес машины, через которую достигается сеть]`

На каждой машине необходимо добавить статические маршруты сетей, к которым данная машина не подключена напрямую.

Пример:



1. На каждой машине настраиваем адреса согласно рисунку
2. На R1 и R2 активируем переадресацию пакетов, перезагружаем их
3. На каждой машине чистим iptables
4. На VM1:

```
ip route add 192.168.2.0/24 via 192.168.0.1  
ip route add 192.168.3.0/24 via 192.168.0.1
```

5. На R1:

```
ip route add 192.168.3.0/24 via 192.168.2.2
```

6. На R2:

7. `ip route add 192.168.0.0/24 via 192.168.2.3`

8. На VM2:

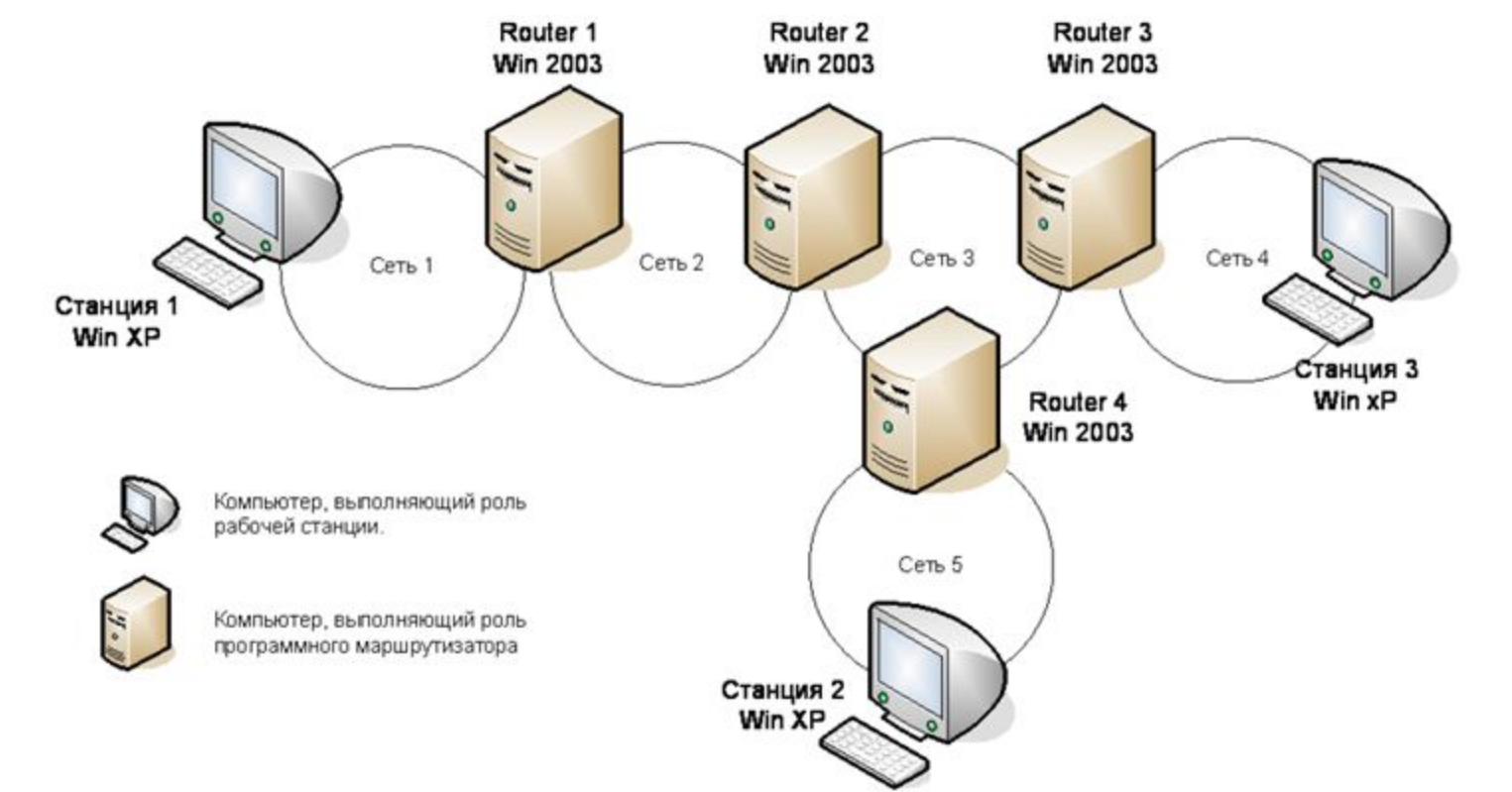
```
ip route add 192.168.0.0/24 via 192.168.3.3  
ip route add 192.168.2.0/24 via 192.168.3.3
```

Windows

Все адреса и настройки сети делаются мышкой (клик по иконке сетевого соединения с трее).

Для активации переадресации:

Панель управления - Администрирование - Маршрутизация и удаленный доступ. Правой кнопкой мыши по серверу, настроить маршрутизацию. Следовать по пунктам, выбрать “Особая конфигурация”, далее “Маршрутизация ЛВС”. Статические маршруты добавлять либо прямо тут, либо из консоли.



Определение адресов сети

Сеть	Сеть 1	Сеть 2	Сеть 3	Сеть 4	Сеть 5
IP-адрес сети	120.13.120.10 4/29	120.13.120.11 2/29	120.13.120.12 0/29	120.13.120.9 6/30	120.13.120.10 0/30
Количество	8	8	8	4	4
Начальный и конечный адреса сети	120.13.120.10 5 120.13.120.11 0	120.13.120.11 3 120.13.120.11 8	120.13.120.12 1 120.13.120.12 6	120.13.120.9 7 120.13.120.9 8	120.13.120.10 1 120.13.120.10 2

Router 1	route -p add 120.13.120.120 mask 255.255.255.248 120.13.120.118
	route -p add 120.13.120.96 mask 255.255.255.252 120.13.120.118
	route -p add 120.13.120.100 mask 255.255.255.252 120.13.120.118
Router 2	route -p add 120.13.120.104 mask 255.255.255.248 120.13.120.113

	route -p add 120.13.120.96 mask 255.255.255.252 120.13.120.126
	route -p add 120.13.120.100 mask 255.255.255.252 120.13.120.124
Router 3	route -p add 120.13.120.104 mask 255.255.255.248 120.13.120.121
	route -p add 120.13.120.112 mask 255.255.255.248 120.13.120.121
	route -p add 120.13.120.100 mask 255.255.255.252 120.13.120.124
Router 4	route -p add 120.13.120.104 mask 255.255.255.248 120.13.120.121
	route -p add 120.13.120.112 mask 255.255.255.248 120.13.120.121
	route -p add 120.13.120.96 mask 255.255.255.252 120.13.120.126

29. Настройка правил файрвола в программном маршрутизаторе по заданным требованиям на ОС Linux или Windows (до 7 правил). Работа выполняется в среде виртуализации Oracle Virtual Box 4 в ОС Windows 2003 или Linux Centos 6.7

Linux

Включает прием всех пакетов с хоста ws.mytrust.ru

```
iptables -t filter -A INPUT -s ws.mytrust.ru -j ACCEPT
```

Запрещает отправку всех пакетов на хост mail.ifmo.ru на порт 25

```
iptables -t filter -A OUTPUT -d mail.ifmo.ru --dport 25 -j DROP
```

Запрещает прием всех сообщений

```
iptables -t filter -A INPUT -j DROP
```

Примеры из лаб:

```
iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 2222 -j ACCEPT
```

```
iptables -A INPUT -i lo -j ACCEPT
```

```
iptables -A INPUT -d 192.168.248.21/32 -i eth3 -p udp -m udp --sport 53 -j ACCEPT
```

```
iptables -A INPUT -i eth3 -p icmp -m icmp --icmp-type 8 -j DROP
```

```
iptables -A INPUT -d 194.85.160.50/32 -p tcp -m tcp --sport 143 -m state --state ESTABLISHED -j ACCEPT
```

```
iptables -A INPUT -p tcp -m tcp --sport 110 -m state --state ESTABLISHED -j ACCEPT
```

```
iptables -A INPUT -d 194.85.160.55/32 -p tcp -m tcp --sport 8080 -m state --state ESTABLISHED -j ACCEPT
```

```
iptables -A INPUT -d 194.85.160.60/32 -p tcp -m tcp --sport 21 -m state --state ESTABLISHED -j ACCEPT
```

```
iptables -A INPUT -d 10.10.11.173/32 -i eth2 -j DROP
```

```
iptables -A INPUT -d 10.10.11.173/32 -i eth3 -j DROP
```

```
iptables -A INPUT -d 83.0.0.0/16 -i eth2 -p tcp -m tcp --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -p tcp -m tcp --dport 22 -j ACCEPT
```

```
iptables -A OUTPUT -d 192.168.248.21/32 -o eth3 -p udp -m udp --dport 53 -j ACCEPT
```

```
iptables -A OUTPUT -d 194.85.160.50/32 -p tcp -m tcp --dport 143 -m state --state NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -p tcp -m tcp --dport 110 -m state --state NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -d 194.85.160.55/32 -p tcp -m tcp --dport 8080 -m state --state NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -d 194.85.160.60/32 -p tcp -m tcp --dport 21 -m state --state NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -d 10.10.11.173/32 -j DROP
```

```
iptables -A OUTPUT -d 83.0.0.0/16 -o eth2 -p tcp -m tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT
```

Windows

Все настраивается через файервол мышкой, потом все экспортируется через netsh dump.

включает автоматическую загрузку Брандмауэра Windows

```
sc config SharedAccess start= auto
```

запускает брандмауэр

```
netsh firewall set opmode ENABLE ENABLE
```

включает протоколирование входящих соединений

```
netsh firewall set logging c:\log.txt 8192 ENABLE
```

настраивает службу Telnet на ручной запуск

```
sc config TlntSvr start= demand
```

разрешает системе отвечать на запросы echo-request ICMP

```
netsh firewall set icmpsetting 8 ENABLE
```

добавляет правило, разрешающее доступ с IP адресов сети компьютерного класса к службе Telnet

```
netsh firewall set portopening TCP 23 Telnet ENABLE ALL
```

запускает службу Telnet

```
net start telnet
```


30. Создание и управление пользователями и группами в Active Directory через графический интерфейс и через командную строку на powershell или с помощью консольных утилит (dsquery, dsadd и т.п.). Настройка прав на файлы и каталоги по требованиям на платформе Windows в системе NTFS

Установка и настройка AD

- Для виртуальной машины следует назначить статический ip и имя компьютера ADServ. В качестве типа подключения сетевого адаптера виртуальной машины использовать режим «Внутренняя сеть»
- Панель управление - администрирование - мастер настройки сервера - первоначальная настройка сервера (добавить роль). AD и DHCP настроятся сами (домен COMP.local). DNS трансляцию не включать. Если будет спрашивать вставить диск - отмена, при этом может обвалиться несколько этапов настройки DNS - не страшно. Перезагрузить.
- Создать папку C:\UsersHome. Расшарить папку с именем UsersHome, дать полные права всем.

Создать organization unit

```
dsadd ou "ou=testou,dc=COMP,dc=local"
```

Создать group

```
dsadd group "cn=testgr,ou=testou,dc=COMP,dc=local"
```

Скрипт добавления пользователя и раздачи прав

```
set name=%1
set password=%2
dsadd user "cn=%name%,ou=testou,dc=COMP,dc=local" -pwd %password% -mustchpwd yes -memberof
"cn=testgr,ou=testou,dc=COMP,dc=local" -profile \\ADServ\UsersHome\%name%\_profile -hmdrv
W: -hmdir \\ADServ\UsersHome\%name%
net share %name%=C:\UsersHome\%name% /grant:%name%,change
cacls C:\UsersHome\%name% /g %name%:F
```

Разрешение локального входа

Панель управления - администрирование - политики безопасности домена - параметры безопасности - локальные политики - назначение прав пользователей - локальный вход в систему. Добавляем группу testgr и Администраторы. Аналогичные операции проводятся для Панель управления - администрирование - политики безопасности контроллера домена