**+Список дисциплин и вопросов междисциплинарного экзамена.**

**Информационные технологии**

1. **Определение и задачи информационной технологии. Структура проекта в области информационных технологий.**
2. **Определение и основные характеристики информационного общества.**

**Информационное общество** — это такое общество, в котором производство и потребление информации является важнейшим видом деятельности, а информация признается наиболее значимым ресурсом.

**Основными отличительными признаками информационного общества являются:**

• информационная экономика;

• высокий уровень информационных потребностей всех членов общества и фактическое их удовлетворение для основной массы населения;

• высокая информационная культура;

• свободный доступ каждого члена общества к информации, ограниченный только информационной безопасностью личности, общественных групп и всего общества.

**Информационному обществу присущи:**

• единое информационное пространство;

• доминирование в экономике новых технологических укладов;

• ведущая роль информационных ресурсов в развитии общества;

• возрастание роли инфраструктуры (телекоммуникационной, транспортной, организационной) в системе общественного производства;

• фактическое удовлетворение потребностей общества в информационных продуктах и услугах;

• высокий уровень образования;

• высокая значимость проблем обеспечения информационной безопасности и наличие эффективной системы обеспечения прав граждан и социальных институтов на свободное получение, распространение и использование информации.

**К стратегическим направлениям перехода к информационному обществу относятся:**

• формирование единого информационного пространства региона и интеграция его в информационное пространство России и единое мировое информационное пространство;

• подготовка населения региона к формированию информационного общества и к жизни в этом обществе;

• создание правовых основ перехода к информационному обществу и обеспечение условий их практического применения;

• создание информационной экономики региона;

• осуществление свободного доступа к информации каждому жителю региона при обеспечении информационной безопасности личности, общественных групп и всего населения региона в целом;

• научное сопровождение формирования информационного общества;

• участие в межрегиональном и международном сотрудничестве и разделении труда при формировании информационного общества города, регионов, стран и мира.

1. **Процесс передачи информации. Каналы связи.**
2. **Модели процессов обработки информации. Централизованная, децентрализованная и смешанная формы обработки.**

При **числовой** обработке используются такие объекты, как переменные, векторы, матрицы, многомерные массивы, константы и т.д. При **нечисловой** обработке объектами могут быть файлы, записи, поля, иерархии, сети, отношения и т.д. Другое отличие заключается в том, что при числовой обработке содержание данных не имеет большого значения, в то время как при нечисловой обработке нас интересуют непосредственные сведения об объектах, а не их совокупность в целом.

**С точки зрения реализации на основе современных достижений вычислительной техники выделяют следующие виды обработки информации:**

• последовательная обработка, применяемая в традиционной фоннеймановской архитектуре ЭВМ, располагающей одним процессором;

• параллельная обработка, применяемая при наличии нескольких процессоров в ЭВМ;

• конвейерная обработка, связанная с использованием в архитектуре ЭВМ одних и тех же ресурсов для решения разных задач, причем если эти задачи тождественны, то это последовательный конвейер, если задачи одинаковые — векторный конвейер.

Принято относить существующие архитектуры ЭВМ с точки зрения обработки информации к одному из следующих классов.

**Архитектуры с одиночным потоком команд и данных (SISD).** К этому классу относятся традиционные фоннеймановские однопроцессорные системы, где имеется центральный процессор, работающий с парами «атрибут — значение».

**Архитектуры с одиночными потоками команд и данных (SIMD).** Особенностью данного класса является наличие одного (центрального) контроллера, управляющего рядом одинаковых процессоров.

**Архитектуры с множественным потоком команд и одиночным потоком данных (MISD). К** этому классу могут быть отнесены конвейерные процессоры.

**Архитектуры с множественным потоком команд и множественным**

**потоком данных (MIMD).** К этому классу могут быть отнесены следующие конфигурации: мультипроцессорные системы, системы с мультобработкой, вычислительные системы из многих машин, вычислительные сети.

**Основные процедуры обработки данных представлены на рис. 4.5.**



**Процесс принятия решения протекает в различных условиях:**

1. Принятие решений в условиях определенности. В этой задаче модели объекта и системы управления считаются заданными, а влияние внешней среды — несущественным.

2. Принятие решений в условиях риска.Для принятия решений в условиях риска необходимо учитывать влияние внешней среды, которое не поддается точному прогнозу.

3. Принятие решений в условиях неопределенности.Каждой паре «стратегия — конечный результат» соответствует некоторая внешняя оценка в виде выигрыша.

4. Принятие решений в условиях многокритериальности.Появляется когда имеется больше одной цели.

В настоящее время принято выделять **два типа информационных систем поддержки принятия решений.**

Системы поддержки принятия решений DSS (Decision Support System)

Системы оперативной аналитической обработки OLAP (OnLine Analysis Processing).

Выделим в информационном приложении типовые функциональные компоненты, достаточные для формирования любого приложения на основе БД .

**PS (Presentation Services) —** средства представления.

**PL (Presentation Logic) —** логика представления.

**BL (Business or Application Logic) —** прикладная логика.

**DL (Data Logic) —** логика управления данными.Операции с базой данных, которые нужно выполнить для реализации прикладной логики управления данными.

**DS (Data Services) —** операции с базойданных. Действия СУБД, вызываемые для выполнения логики управления данными.

**FS (File Services) — файловые операции.**

Организационные формы использования информационных  
           технологий при обработке данных  
      Развитие организационных форм вычислительной техники строится на сочетании централизованной и децентрализованной (смешанной) форм, предпосылкой которой явилось создание сетей ЭВМ на основе различных средств связи.  
       Централизованные формы применения средств вычислительной техники, которые существовали до массового использования персональных электронно-вычислительных машин  
(ПЭВМ), предполагали их сосредоточение в одном месте и организацию информационновычислительных центров (ИВЦ) индивидуального и коллективного пользования (ИВЦКП).  
    Деятельность ИВЦ и ИВЦКП характеризовалась обработкой больших объемов информации, использованием нескольких средних и больших ЭВМ, квалификационным персона-лом для обслуживания техники и разработки программного обеспечения. Централизованное  
применение вычислительных и других технических средств позволяло организовать их на-дежную работу, планомерную загрузку и квалификационное обслуживание.  
     При централизованных формах, когда у пользователей нет непосредственного контакта с ЭВМ, его роль сводится к передаче исходных данных на обработку, получению результа-тов, выявлению и устранению ошибок.  
    Централизованная обработка информации наряду с рядом положительных сторон (вы-сокая степень загрузки и высокопроизводительное использование оборудования, квалифици-рованный кадровый состав операторов, программистов, инженеров, проектировщиков вычислительных систем и т.п.) имела ряд отрицательных черт, порожденных прежде всего от-рывом конечного пользователя (экономиста, плановика, нормировщика и т.п.) от технологи-ческого процесса обработки информации.   
     
    Децентрализованные формы использования вычислительных ресурсов начали формироваться со второй половины 80-х годов, когда сфера экономики получила возможность пе-рейти к массовому использованию персональных ЭВМ. Децентрализация предусматривает  
размещение ПЭВМ в местах возникновения и потребления информации, где создаются авто-номные пункты ее обработки. К ним относятся абонентские пункты и автоматизированные  
рабочие места (АРМ).  
    При непосредственном общении пользователя с ЭВМ его функции в информационной  
технологии расширяются. Он сам вводит данные, формирует информационную базу, решает  
задачи, получает результаты, оценивает их качество. У пользователя открываются реальные  
возможности решать задачи с альтернативными вариантами, анализировать и выбирать с по-  
мощью системы в конкретных условиях наиболее приемлемый вариант. Все это реализуется  
в пределах одного рабочего места. От пользователя при этом требуется знание основ приме-  
нения тех или иных информационных технологий.

Централизованная обработка:  
плюсы:  
- нет накладных расходов, связанных с согласованием информации в разных местах  
- полный контроль над системой в одном месте  
- проще разработка  
минусы:  
- ограничение производительности/пропускной способности - можность одной машины может расти не бесконечно  
- меньшая степень надёжности - одно звено определяет работоспособность всей системы  
Децентрализованная обработка:  
минусы:  
- необходимость синхронизации возможно противоречивых данных из разных источников, накладные расходы на это  
- сложнее разрабатывать такое ПО  
плюсы:  
- лучше масштабируемость - то есть можно наращивать пропускную способность, производительность увеличением кол-ва компонентов системы  
- больше надёжность - при отказе части компонентов вся система может продолжать работать, пусть и не в полной мере

1. **Технологии и методологии проектирования информационных систем. Характеристика CASE-средств.**

На данный момент в технологии разработки программного обеспечения существуют два основных подхода к разработке информационных систем, отличающиеся критериями декомпозиции: функционально-модульный (структурный) и объектно-ориентированный.

Функционально-модульный подход основан на принципе алгоритмической декомпозиции с выделением функциональных элементов и установлением строгого порядка выполняемых действий.

Объектно-ориентированный подход основан на объектной декомпозиции с описанием поведения системы в терминах взаимодействия объектов.

Главным недостатком функционально-модульного подхода является однонаправленность информационных потоков и недостаточная обратная связь. В случае изменения требований к системе это приводит к полному перепроектированию, поэтому ошибки, заложенные на ранних этапах, сильно сказываются на продолжительности и стоимости разработки. Другой важной проблемой является неоднородность информационных ресурсов, используемых в большинстве информационных систем. В силу этих причин в настоящее время наибольшее распространение получил объектно-ориентированный подход.

Под CASE-технологией будем понимать комплекс программных средств, поддерживающих процессы создания и сопровождения программного обеспечения, включая анализ и формулировку требований, проектирование, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом (CASE-средство может обеспечивать поддержку только в заданных функциональных областях или в широком диапазоне функциональных областей) [5].

В связи с наличием двух подходов к проектированию программного обеспечения существуют CASE-технологии ориентированные на структурный подход, объектно-ориентированный подход, а также комбинированные. Однако сейчас наблюдается тенденция переориентации инструментальных средств, созданных для структурных методов разработки, на объектно-ориентированные методы, что объясняется следующими причинами:

• возможностью сборки программной системы из готовых компонентов, которые можно использовать повторно;

• возможностью накопления проектных решений в виде библиотек классов на основе механизмов наследования;

• простотой внесения изменений в проекты за счет инкапсуляции данных в объектах;

• быстрой адаптацией приложений к изменяющимся условиям за счет использования свойств наследования и полиформизма;

• возможностью организации параллельной работы аналитиков, проектировщиков и программистов.

Основным направлением деятельности консорциума Object Management Group (OMG) является разработка спецификаций и стандартов для создания распределенных объектных систем в разнородных средах. Базисом стали спецификации под названием Object Management Architecture (ОМА).

ОМА состоит из четырех основных компонентов, представляющих спецификации различных уровней поддержки приложений (рис. 5.10):

• архитектура брокера запросов объектов (CORBA — Common Object Request Broker Architecture) определяет механизмы взаимодействия объектов в разнородной сети;

• объектные сервисы (Object Services) являются основными системными сервисами, используемыми разработчиками для создания приложений;

• универсальные средства (Common Facilities) являются высокоуровневыми системными сервисами, ориентированными на поддержку пользовательских приложений (электронная почта, средства печати и др.);

• прикладные объекты (Application Object) предназначены для решения конкретных прикладных задач.

Исходя из основных положений объектно-ориентированного подхода рассмотрим концепцию идеального объектно-ориентированного CASE-средства.

Классическая постановка задачи разработки программной системы (инжиниринг) представляет собой спиральный цикл итеративного чередования этапов объектно-ориентированного анализа, проектирования и реализации (программирования).

В реальной практике в большинстве случаев имеется предыстория в виде совокупности разработанных и внедренных программ, которые целесообразно использовать при разработке новой системы. Процесс проектирования в таком случае основан на реинжиниринге программных кодов, при котором путем анализа текстов программ восстанавливается исходная модель программной системы.

Современные CASE-средства поддерживают процессы инжиниринга и автоматизированного реинжиниринга.

Идеальное объектно-ориентированное CASE-средство (рис. 5.11) должно содержать четыре основных блока: анализ, проектирование, разработка и инфраструктура [34].

Основные требования к блоку анализа: 116

• возможность выбора выводимой на экран информации из всей совокупности данных, описывающих модели;

• согласованность диаграмм при хранении их в репозитарии;

• внесение комментариев в диаграммы и соответствующую документацию для фиксации проектных решений;

• возможность динамического моделирования в терминах событий;

• поддержка нескольких нотаций

Основные требования к блоку проектирования:

• поддержка всего процесса проектирования приложения;

• возможность работы с библиотеками, средствами поиска и выбора;

• возможность разработки пользовательского интерфейса;

• поддержка стандартов OLE, ActiveX и доступ к библиотекам HTML или Java;

• поддержка разработки распределенных или двух- и трех-звенных клиент-серверных систем (работа с CORBA, DCOM, Internet).

Основные требования к блоку реализации:

• генерация кода полностью из диаграмм;

• возможность доработки приложений в клиент- серверных CASE-средствах типа Power Builder;

• реинжиниринг кодов и внесение соответствующих изменений в модель системы;

• наличие средств контроля, которые позволяют выявлять несоответствие между диаграммами и генерируемыми кодами и обнаруживать ошибки как на стадии проектирования, так и на стадии реализации.

Основные требования к блоку инфраструктуры:

• наличие репозитория на основе базы данных, отвечающего за генерацию кода, реинжиниринг, отображение кода на диаграммах, а также обеспечивающего соответствие между моделями и программными кодами;

• обеспечение командной работы (многопользовательской работы и управление версиями) и реинжиниринга.

Выделим основные критерии оценки и выбора CASE-средств.

1. Функциональные характеристики:

• среда функционирования: проектная среда, программное обеспечение/технические средства, технологическая среда;

• функции, ориентированные на фазы жизненного цикла: моделирование, реализация, тестирование;

• общие функции: документирование, управление конфигурацией, управление проектом;

**2. Надежность; 3. Простота использования; 4. Эффективность; 5. Сопровождаемое; 6. Переносимость; 7. Общие критерии (стоимость, затраты, эффект внедрения, характеристики поставщика)**.

1. **Модели процессов накопления информации. Основные принципы поиска информации. Информационно-поисковые системы.**

Хранение и накопление являются одними из основных действий, осуществляемых над информацией и главным средством обеспечения ее доступности в течение некоторого промежутка времени. В настоящее время определяющим направлением реализации этой операции является концепция базы данных, склада (хранилища) данных.

База данных может быть определена как совокупность взаимосвязанных данных, используемых несколькими пользователями и хранящихся с регулируемой избыточностью. Хранимые данные не зависят от программ пользователей, для модификации и внесения изменений применяется общий управляющий метод.

Банк данных — система, представляющая определенные услуги по хранению и поиску данных определенной группе пользователей по определенной тематике.

Система баз данных — совокупность управляющей системы, прикладного программного обеспечения, базы данных, операционной системы и технических средств, обеспечивающих информационное обслуживание пользователей.

Хранилище данных (ХД — используют также термины Data Warehouse, «склад данных», «информационное хранилище») — это база, хранящая данные, агрегированные по многим измерениям. Основные отличия ХД от БД: агрегирование данных; данные из ХД никогда не удаляются; пополнение ХД происходит на периодической основе; формирование новых агрегатов данных, зависящих от старых — автоматическое; доступ к ХД осуществляется на основе многомерного куба или гиперкуба.

ИПС (информационно-поисковая система) - это система, обеспечивающая поиск и отбор необходимых данных в специальной базе с описаниями источников информации (индексе) на основе информационно-поискового языка и соответствующих правил поиска.

Главной задачей любой ИПС является поиск информации релевантной информационным потребностям пользователя. Очень важно в результате проведенного поиска ничего не потерять, то есть найти все документы, относящиеся к запросу, и не найти ничего лишнего. Поэтому вводится качественная характеристика процедуры поиска - релевантность.

Релевантность - это соответствие результатов поиска сформулированному запросу.

Основными показателями ИПС для WWW являются пространственный масштаб и специализация. По пространственному масштабу ИПС можно разделить на локальные, глобальные, региональные и специализированные. Локальные поисковые системы могут быть разработаны для быстрого поиска страниц в масштабе отдельного сервера. Региональные ИПС описывают информационные ресурсы определенного региона, например, русскоязычные страницы в Интернете. Глобальные поисковые системы в отличие от локальных стремятся объять необъятное - по возможности наиболее полно описать ресурсы всего информационного пространства сети Интернет.

В общем случае, можно выделить следующие поисковые инструменты для WWW: каталоги, поисковые системы, метапоисковые системы.

1. Каталог

Каталог - поисковая система с классифицированным по темам списком аннотаций со ссылками на web-ресурсы. Классификация, как правило, проводится людьми.

Поиск в каталоге очень удобен и проводится посредством последовательного уточнения тем. Тем не менее, каталоги поддерживают возможность быстрого поиска определенной категории или страницы по ключевым словам с помощью локальной поисковой машины. База данных ссылок (индекс) каталога обычно имеет ограниченный объем, заполняется вручную персоналом каталога. Некоторые каталоги используют автоматическое обновление индекса.

Результат поиска в каталоге представляется в виде списка, состоящего из краткого описания (аннотации) документов с гипертекстовой ссылкой на первоисточник.

Адреса популярных каталогов:

Зарубежные каталоги:

Yahoo - www.yahoo.com

Magellan - www.mckinley.com

Российские каталоги:

@Rus - www.aport.ru

Weblist - www.weblist.ru

Улитка - www.ulitka.ru

2. Поисковая машина

Поисковая машина - поисковая система с формируемой роботом базой данных, содержащей информацию об информационных ресурсах.

Отличительной чертой поисковых машин является тот факт, что база данных, содержащая информацию об Web-страницах, статьях Usenet и т.д., формируется программой-роботом.

Поиск в такой системе проводится по запросу, составляемому пользователем, состоящему из набора ключевых слов или фразы, заключенной в кавычки. Индекс формируется и поддерживается в актуальном состоянии роботами-индексировщиками.

В описании документа чаще всего содержится несколько первых предложений или выдержки из текста документа с выделением ключевых слов. Как правило, указана дата обновления (проверки) документа, его размер в килобайтах, некоторые системы определяют язык документа и его кодировку (для русскоязычных документов).

Адреса наиболее популярных поисковых машин за рубежом и в России.

Зарубежные поисковые машины:

Google - www.google.com

Altavista - www.altavista.com

Excite - www.excite.com

Российские поисковые машины:

Яndex - www.yandex.ru (или www.ya.ru)

Рэмблер - www.rambler.ru

Апорт - www.aport.ru

3. Метапоисковая машина

метапоисковые системы (поисковые службы) - системы, способные послать запросы пользователя одновременно нескольким поисковым серверам, затем объединить полученные результаты и представить их пользователю в виде документа со ссылками.

Адреса известных метапоисковых систем:

MetaCrawler - www.metacrawler.com

SavvySearch - www.savvysearch.com

1. **Модели, методы и средства реализации мультимедиа технологий.**

В настоящее время мультимедиа-технологии являются бурно развивающейся областью информационных технологий. В этом направлении активно работает значительное число крупных и мелких фирм, технических университетов и студий (в частности IBM, Apple, Motorola, Philips, Sony, Intel и др.). Области использования

чрезвычайно многообразны: интерактивные обучающие и информационные системы, САПР, развлечения и др.

**Основными характерными особенностями этих технологий являются:**

• объединение многокомпонентной информационной среды (текста, звука, графики, фото, видео) в однородном цифровом представлении;

• обеспечение надежного и долговечного хранения больших объемов информации;

• простота переработки информации.

Многокомпонентную мультимедиа-среду целесообразно разделить на три группы: аудиоряд, видеоряд, текстовая информация.

**Аудиоряд** может включать речь, музыку, эффекты. Главной проблемой при использовании этой группы мультисреды является информационная емкость. Для записи одной минуты WAVE-звука высшего качества необходима память порядка 10 Мбайт.

Другим направлением является использование в мультисреде звуков MIDI (Musical Instrument Digitale Interface).

**Видеоряд** характеризуется большим числом элементов.

Выделяют:

Статический видеоряд включает графику и фото.

Динамический видеоряд представляет собой последовательность статических элементов (кадров). Можно выделить три типовых группы:

• обычное видео (life video) — последовательность фотографий (около 24 кадров в секунду);

• квазивидео — разреженная последовательность фотографий (6—12 кадров в секунду);

• анимация — последовательность рисованных изображений.

Первая проблема при реализации видеорядов — разрешающая способность экрана и число цветов.

Вторая проблема — объем памяти.

При размещении **текстовой** информации на CD-ROM нет никаких сложностей и ограничений ввиду большого информационного объема оптического диска.

**Основные направления использования мультимедиа-технологий:**

• электронные издания для целей образования, развлечения и др.;

• в телекоммуникациях со спектром возможных применений от просмотра заказной телепередачи и выбора нужной книги до участия в мультимедиа-конференциях.

• мультимедийные информационные системы («мультимедиа-киоски»), выдающие по запросу пользователя наглядную информацию.

1. **Модели, методы и средства реализации геоинформационных технологий.**

Графическое представление какой-либо ситуации на экране компьютера подразумевает отображение различных графических образов. Сформированный на экране ЭВМ графический образ состоит из двух различных с точки зрения среды хранения частей — графической «подложки» или графического фона и других графических объектов.

Таким образом, геоинформационные технологии предназначены для широкого внедрения в практику методов и средств работы с пространственно-временными данными, представляемыми в виде системы электронных карт, и предметно-ориентированных сред обработки разнородной информации для различных категорий пользователей.

**Основным классом данных геоинформационных систем (ГИС)** являются координатные данные, содержащие геометрическую информацию и отражающие пространственный аспект. Основные типы координатных данных: точка (узлы, вершины), линия (незамкнутая), контур (замкнутая линия), полигон (ареал, район).

**Рассмотренные типы данных имеют большее число разнообразных связей, которые можно условно разделить на три группы:**

• взаимосвязи для построения сложных объектов из простых элементов;

• взаимосвязи, вычисляемые по координатам объектов;

• взаимосвязи, определяемые с помощью специального описания и семантики при вводе данных.

Основой визуального представления данных при использовании ГИС-технологий является графическая среда, основу которой составляют векторные и растровые (ячеистые) модели.

Векторные модели основаны на представлении геометрической информации с помощью векторов, занимающих часть пространства, что требует при реализации меньшего объема памяти.

В растровых моделях объект (территория) отображается в пространственные ячейки, образующие регулярную сеть. Каждой ячейке растровой модели соответствует одинаковый по размерам, но разный по характеристикам (цвет, плотность) участок поверхности. Ячейка модели характеризуется одним значением, являющимся средней характеристикой участка поверхности. Эта процедура называется пикселизацией.

Таким образом, векторная модель содержит информацию о местоположении объекта, а растровая о том, что расположено в той или иной точке объекта. Векторные модели относятся к бинарным или квазибинарным. Растровые позволяют отображать полутона.

**Важным моментом при проектировании ГИС является размерность модели.**

Применяют двухмерные модели координат (2D) и трехмерные (3D). Двухмерные модели используются при построении карт, а трехмерные — при моделировании геологических процессов, проектировании инженерных сооружений (плотин, водохранилищ, карьеров и др.), моделировании потоков газов и жидкостей.

**Большинство современных ГИС осуществляет комплексную обработку информации:**

• сбор первичных данных;

• накопление и хранение информации;

• различные виды моделирования (семантическое, имитационное, геометрическое, эвристическое);

• автоматизированное проектирование;

• документационное обеспечение.

Основные области использования ГИС:

• электронные карты;

• городское хозяйство;

• государственный земельный кадастр;

• экология;

• дистанционное зондирование;

• экономика;

• специальные системы военного назначения.

1. **Технологии защиты информации. Основные понятия о защите информации. Безопасность передачи данных.**
2. **Методы защиты информации. Криптография. Аутентификация. Цифровая подпись.**

Методы защиты:

1. Криптографические метода – шифрование инф-ии и эл.подпись
2. Прогр и технич методы – прогр.: ср-ва упр-я доступом и аудит, имеющиеся в ОС; технич.: замки в дверях, сис-ма видеонаблюдения, камеры, сигнализация
3. Методич.ср-ва – это различные законодательные нормативные док-т, кот определяют правила доступа и работы с инф-ей, а т же санкции за их нарушения.

Криптография

чтобы никто посторонний не мог ознакомиться с передаваемым сооб-ем М, А некот-м обр-м преобразует сооб-е в форму недоступную для прочтения посторонних. С=Ек(М). И преобрз-е так обр сооб-е направляется дальше. В выпо-ет обратное преобраз-е и восстанавливает из него исх.сооб-е М. М=Dк(С).

Исх соо-е наз-ся открытым текстом, нез-мо от типа его содержания. Преобразованное сооб-е С наз-ся шифр-текстом.

Процесс преобраз-ся открытого текста в шифртекст наз-ся зашифрование. Обратный процесс – расшифрование.

пар-р К – ключ для заш-я, расш-я.

Наука, изучающая методы защиты сооб-ий наз-мя криптографией, м-ды вскрытия текста – криптоанализом.

преобраз-е открытого текста в шифртекст заключается в выч-ии некот ф-ции от открытого текста.

проц-сс расш-я закл-ся в применении к шифр-тексту обратной ф-ции-ф-ции расш-я.

Сов-ть ф-ций расш-я и заш-я наз-ся криптографич алгоритмом или шифром.

Надеж-ть любого совр шифра основ-ся на исп-нии ключа. Знание ф-ции заш-я и рас-я явл.недостаточным для раскрытия сообщ-я - Это принцип Кирхгофа.

Если в шифре ключ рас-я совпадает с ключом заш-я или м\бфть из него получен, тогда шифр наз-ся симметричным.

Если ключ рас-я(личный) практически невозможно получить из ключа заш-я(открытый), то шифр наз ассимметричным или шифром с открытым ключом.

шифры:

1. Блочные(инф обраб-ся блоками фиксир длины, длина блока от 64 бит)
2. Потоковые(инф обраб-ся потоком, побитно или побайтно)

Установление подлинности объекта **(аутентификация)** - методика, которая позволяет одной стороне доказывать подлинность другой стороны. Объект может быть человеком, процессом, клиентом или сервером. **Объект, подлинность которого должна быть доказана, называется претендентом;** признаки подлинности ( идентификационный код) претендент а названы верификатором.

**Категории проверки**

В установлении подлинности объекта претендент должен идентифицировать себя для верификатора. Этот может быть сделано одним из трех видов свидетелей: нечто, известное только претенденту , нечто, чем обладает только претендент, или нечто, свойственное только претенденту.

- Нечто известное. Это - секретная информация известная только претенденту, что может быть проверено верификатором. Примеры: пароль, PIN-код, ключ засекречивания и секретный ключ.

**- Нечто, чем обладает**. Это то, что может доказать опознавательный код претендента. Примеры: паспорт, водительские права, удостоверение личности, кредитная карточка и карточка с интегральной схемой, включающей микропроцессор.

- Нечто свойственное. Это свойственные претенденту характеристики. Примеры: обычные подписи, отпечатки пальца, голос, характеристики лица, образец сетчатки глаза и почерк.

Аутентификация бывает **односторонней** (обычно клиент доказывает свою подлинность серверу) и**двусторонней** (**взаимной**). Пример односторонней аутентификации - процедура входа пользователя в систему.

В соответствии с рекомендациями стандарта Х.509 различают процедуры строгой аутентификации следующих типов:  
• односторонняя аутентификация;  
• двусторонняя аутентификация;  
• трехсторонняя аутентификация.

Односторонняя аутентификация предусматривает обмен информацией только в одном направлении.

Двусторонняя аутентификация по сравнению с односторонней содержит дополнительный ответ проверяющей стороны доказывающей стороне, который должен убедить ее, что связь устанавливается именно с той стороной, которой были предназначены аутентификационные данные;

Трехсторонняя аутентификация содержит дополнительную передачу данных от доказывающей стороны проверяющей. Этот подход позволяет отказаться от использования меток времени при проведении аутентификации.

В зависимости от используемых криптографических алгоритмов протоколы строгой аутентификации делятся на протоколы, основанные:  
• на симметричных алгоритмах шифрования;  
• однонаправленных ключевых хэш-функциях;  
• асимметричных алгоритмах шифрования;  
• алгоритмах электронной цифровой подписи.

**Электро́нная по́дпись** (ЭП) — [информация](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) в электронной форме, которая присоединена к другой информации в электронной форме (подписываемой информации) или иным образом связана с такой информацией и которая используется для определения [лица](http://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B0), подписывающего информацию[[1]](http://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C#cite_note-0).

По своему существу электронная подпись представляет собой [реквизит](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D0%B2%D0%B8%D0%B7%D0%B8%D1%82) [электронного документа](http://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82), позволяющий установить отсутствие искажения информации в электронном документе с момента формирования ЭП и проверить принадлежность подписи владельцу [сертификата ключа ЭП](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D1%8E%D1%87%D0%B0). Значение реквизита получается в результате криптографического преобразования информации с использованием закрытого ключа ЭП.

Электронная подпись предназначена для [идентификации](http://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) лица, подписавшего электронный документ и является полноценной заменой ([аналогом](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%BE%D0%B3)) собственноручной подписи в случаях, предусмотренных законом[[2]](http://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C#cite_note-1).

Использование электронной подписи позволяет осуществить:

- Контроль [целостности](http://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8) передаваемого документа: при любом случайном или преднамеренном изменении документа подпись станет недействительной, потому что вычислена она на основании исходного состояния документа и соответствует лишь ему.

- Защиту от изменений (подделки) документа: гарантия выявления подделки при контроле целостности делает подделывание нецелесообразным в большинстве случаев.

- Невозможность отказа от авторства. Так как создать корректную подпись можно, лишь зная закрытый ключ, а он должен быть известен только владельцу, то владелец не может отказаться от своей подписи под документом.

Доказательное подтверждение авторства документа: Так как создать корректную подпись можно, лишь зная закрытый ключ, а он должен быть известен только владельцу, то владелец пары ключей может доказать своё авторство подписи под документом. В зависимости от деталей определения документа могут быть подписаны такие поля, как «автор», «внесённые изменения», «метка времени» и т. д.

1. **Модели, методы и средства реализации технологий искусственного интеллекта.**

**Система называется интеллектуальной, если в ней реализованы следующие основные функции:**

• накапливать знания об окружающем систему мире, классифицировать и оценивать их с точки зрения прагматической полезности и непротиворечивости, инициировать процессы получения новых знаний, осуществлять соотнесение новых знаний с ранее хранимыми;

• пополнять поступившие знания с помощью логического вывода, получать обобщенные знания на основе более частных знаний и логически планировать свою деятельность;

• общаться с человеком на языке, максимально приближенном к естественному человеческому языку, и получать информацию от каналов, аналогичных тем, которые использует человек при восприятии окружающего мира.

**База знаний** представляет собой совокупность сред, хранящих знания различных типов. Рассмотрим кратко их назначение.

База фактов (данных) хранит конкретные данные, а база правил — элементарные выражения, называемые в теории искусственного интеллекта продукциями.

База процедур содержит прикладные программы, с помощью которых выполняются все необходимые преобразования и вычисления.

База закономерностей включает различные сведения, относящиеся к особенностям той среды, в которой действует система.

База метазнаний (база знаний о себе) содержит описание самой системы и способов ее функционирования.

База целей содержит целевые структуры, называемые сценариями, позволяющие организовать процессы движения от исходных фактов, правил, процедур к достижению той цели, которая поступила в систему от пользователя.

**Решатель** осуществляет ряд функций:

- дедуктивного вывода

- индуктивного и правдоподобного выводов.

- блок планирования

- система управления решениями.

**Рецепторы и эффекторы** осуществляют так называемое невербальное общение и используются в интеллектуальных роботах.

**Разновидности интеллектуальных систем:**

• интеллектуальные информационно-поисковые системы;

• экспертные системы (ЭС);

• расчетно-логические системы;

• гибридные экспертные системы.

**Интеллектуальные информационно-поисковые**. Для них характерно использование, помимо базы знаний, реализующей семантическую модель представления знаний о проблемной области, лингвистического процессора.

**Экспертные системы являются**. Для них характерна аккумуляция в системе знаний и правил рассуждений опытных специалистов в данной предметной области, а также наличие специальной системы объяснений.

**Расчетно-логические системы.** Реализуются благодаря наличию базы знаний в виде функциональной семантической сети и компонентов дедуктивного вывода и планирования.

В последнее время в специальный класс выделяются гибридные экспертные системы. Указанные системы должны вобрать в себя лучшие черты как экспертных, так и расчетно-логических и информационно-поисковых систем.

**В настоящее время выделяют следующие основные типы моделей представления знаний:**

Семантические сетиопределяют как граф общего вида, в котором можно выделить множество вершин и ребер.

Фреймом называют структуру данных для представления и описания стереотипных объектов, событий или ситуаций.

Существует два типа фреймов:

• образец (прототип) — интенсиональное описание некоторого множества экземпляров;

• экземпляр (пример) — экстенсиональное представление фрейм-образца.

Слоты — это некоторые незаполненные подструктуры фрейма, заполнение которых приводит к тому, что данный фрейм ставится в соответствие некоторой ситуации, явлению или объекту.

В качестве данных фрейм может содержать обращения к процедурам. Выделяют два вида процедур: процедуры-демоны и процедуры-слуги. Процедуры-демоны активизируются при каждой попытке добавления или удаления данных из слота. Процедуры-слуги активизируются только при выполнении условий, определенных пользователем при создании фрейма.

**Продукционные модели** — это набор правил вида «условия — действие», где условиями являются утверждения о содержимом базы данных, а действия представляют собой процедуры, которые могут изменять содержимое базы данных.

**Обычно выделяют следующие типы задач**:

• интерпретация символов или сигналов — составление смыслового описания по входным данным;

• диагностика — определение неисправностей (заболеваний) по симптомам;

• предсказание — определение последствий наблюдаемых ситуаций;

• конструирование — разработка объекта с заданными свойствами при соблюдении установленных ограничений;

• планирование — определение последовательности действий, приводящих к желаемому состоянию объекта;

• слежение — наблюдение за изменяющимся состоянием объекта и сравнение его показателей с установленными или желаемыми;

• управление — воздействие на объект для достижения желаемого поведения.

1. **Информационные технологии организационного управления Корпоративные информационные технологии. ERP, CRM, HRM, CPM, EAI**.

Можно выделить следующие три группы методов управления: ресурсами, процессами, корпоративными знаниями (коммуникациями).

Первоначально была разработана методология планирования материальных ресурсов предприятия **MRP** (Material Requirements Planning), которая использовалась с методологией объемно-календарного планирования MPS (Master Planning Shedule). Следующим шагом было создание методологии планирования производственных ресурсов (мощностей) — **CRP** (Capacitiy Requirements Planning). Эта методология была принципиально похожа на MRP, но ориентирована на расчет производственных мощностей, а не материалов и компонентов.

Объединение указанных выше методологий привело к появлению задачи MRP «второго уровня» — **MRP II** (Manufacturing Resource Planning) — интегрированной методологии планирования, включающей MRP/CRP и использующей MPS и FRP (Finance Resource/requirements Planning) — планирование финансовых ресурсов. Далее была предложена концепция **ERP** (Economic Requirements Planning) — интегрированное планирование всех «бизнес-ресурсов» предприятия.

Следующим шагом было создание концепции управления производственными ресурсами — **CSPP** (Customer Synchronized Resource Planning) — планирование ресурсов, синхронизированное с потреблением. Отличием данной концепции является учет вспомогательных ресурсов, связанных с маркетингом, продажей и послепродажным обслуживанием.

В связи с тем, что в современном производстве задействовано множество поставщиков и покупателей, появилась новая концепция логистических цепочек (Supply Chain). Суть этой концепции состоит в учете при анализе хозяйственной деятельности всей цепочки (сети) превращения товара из сырья в готовое изделие.

**При этом акцент сделан на следующие факторы:**

• стоимость товара формируется на протяжении всей логистической цепочки, но определяющей является стадия продажи конечному потребителю;

• на стоимости товара критическим образом сказывается общая эффективность всех операций;

• наиболее управляемыми являются начальные стадии производства товара, а наиболее чувствительными — конечные (продажные).

**ERP**

В основе ERP-систем лежит принцип создания единого хранилища (репозитория) данных, содержащего всю корпоративную бизнес-информацию: плановую и финансовую информацию, производственные данные, данные по персоналу и др. Наличие единого корпоративного репозитория устраняет необходимость в передаче данных от одной системы к другой (например, от производственной системы к финансовой или к кадровой), а также обеспечивает одновременную доступность информации для любого числа сотрудников предприятия, обладающих соответствующими полномочиями. Целью ERP-систем является не только улучшение управления производственной деятельностью предприятия, но и уменьшение затрат и усилий на поддержку его внутренних информационных потоков.

Существует немало определений ERP-систем. Одно из них, наиболее часто встречающихся, следующее:

ERP-система - это набор интегрированных приложений, позволяющих создать интегрированную информационную среду (ИИС) для автоматизации планирования, учета, контроля и анализа всех основных бизнес-операций предприятия. Основой ИИС предприятия являются именно ERP-системы.

Основные функции ERP-систем:

- ведение конструкторских и технологических спецификаций, которые определяют состав производимых изделий, а также материальные ресурсы и операции, необходимые для его изготовления;

- формирование планов продаж и производства;

- планирование потребностей в материалах и комплектующих, сроков и объемов поставок для выполнения плана производства продукции;

- управление запасами и закупками: ведение договоров, реализация централизованных закупок, обеспечение учета и оптимизации складских и цеховых запасов;

- планирование производственных мощностей: от стратегии всего предприятия до планов использования отдельных станков и оборудования;

- оперативное управление финансами, включая составление финансового плана и осуществление контроля его исполнения, финансовый и управленческий учет;

- управление проектами, включая планирование этапов и ресурсов, необходимых для их реализации.

**СRM-система** (Customer Relationship Management - Управление отношениями с клиентами) - корпоративная информационная система, незаменимый современный инструмент для ведения бизнеса. Дает возможность не просто автоматизировать взаимодействие с клиентами и процесс продаж, а выстроить их работу таким образом, чтобы получать максимальный результат.

**Возможности CRM-систем:**

- Быстрый доступ к актуальной информации о клиентах;

- Оперативность обслуживания клиентов и проведения сделок;

- Формализация схем взаимодействия с клиентами, автоматизация документооборота;

- Быстрое получение всех необходимых отчетных данных и аналитической информации;

- Снижение операционных затрат менеджеров;

- Контроль работы менеджеров;

- Согласованное взаимодействие между сотрудниками и подразделениями.

**HRM-системы** предназначены для управления персоналом, но их функциональность шире, чем у систем автоматизации кадровых операций. Продукты этого класса позволяют работать не только с количественными, но и с качественными показателями персонала. Основная их задача — привлечь и удержать ценных для компании специалистов.

Необходимость во внедрении HRM-систем определяется такими критичными для успешного развития бизнеса потребностями, как:

**- Управление расходами:** Расходы на оплату труда являются одной из крупнейших затратных статей,. По данным Forrester Research (HRMS Q4 2008) на их долю в США приходится в среднем 36,4% от общего объема расходов. HRM-системы используются для планирования и оптимизации расходов .

**- Эффективное управление бизнес-процессами.** HRM-системы поддерживает множество бизнес HR процессов: принятие кадровых решений, поддержание записей о сотрудниках в актуальном состоянии,расчет заработной платы, разработка схем мотивации и пр.Повышение эффективности при выполнении этих задач происходит за счет предоставления прямого доступа сотрудникам ([Employee self-service](http://www.tadviser.ru/index.php/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D1%8F:Employee_self-service)) и менеджерам ([Manager self-service](http://www.tadviser.ru/index.php/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D1%8F:Manager_self-service)) к нужной им информации.

**- Соблюдение всех правовых норм**, регламентирующих взаимоотношения работника и работодателя. Использование HRM-систем позволяет грамотно разрешать сложные вопросы, реализовывать гибкие схемы расчета заработной платы и кадрового документооборота.

**- Повышение ценности человеческого капитала.** Несмотря на экономический кризис, мало кто будет спорить о ценности человеческого капитала в качестве корпоративного актива. Компании с сильной функцией управления персоналом будут акцентировать внимание на «качественном» росте сотрудников, разрабатывая поощрительные программы и схемы мотивации.

Corporate Performance Management (CPM) — система управления эффективностью предприятия. Комплекс, объединяющий все процессы, методологии и метрики, необходимые для измерения показателей деятельности организации и управления этими показателями. Также можно встретить такие синонимы, как Enterprise Performance Management (EPM), Strategic Enterprise Management (SEM) и Business Performance Management (BPM).

Системы CPM предназначены для целостного мониторинга эффективности бизнес-процессов компании, для контроля и анализа соответствия оперативной деятельности подразделений стратегическим целям и задачам бизнеса. CPM-системы ориентированы на глубокий анализ всех аспектов хозяйственной деятельности предприятия.

В отличие от [ERP](http://www.tadviser.ru/index.php/ERP)-систем предоставляющих возможности оперативного управления бизнесом, в круг задач CPM входят стратегические аспекты деятельности. Именно поэтому системы CPM представляют собой набор мощных возможностей многомерного анализа [BI](http://www.tadviser.ru/index.php/BI)/[OLAP](http://www.tadviser.ru/index.php/OLAP), бизнес-планирования и бюджетирования в масштабах всей компании.

Внедрение CPM-системы в бизнес позволяет:

- создать качественную и адекватную бизнесу модель бюджетирования;

- оценить состояние дел в компании;

- выработать план для достижения поставленных целей.

1. **Системный подход к организации информационных процессов. Модель открытых систем OSI.**
2. **Признаки сложных систем. Сложность, присущая ПС.**

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

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

3. Внутрикомпонентная связь обычно сильнее, чем связь между компонентами. Это обстоятельство позволяет отделять «высокочастотные» взаимодействия внутри компонентов от «низкочастотной» динамики взаимодействия между компонентами.

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

5. Любая работающая сложная система является результатом развития работавшей более простой системы… Сложная система, спроектированная с «нуля», никогда не заработает. Следует начинать с работающей простой системы.

Сложные системы в программировании ничем не отличаются от других сложных систем, таких как биологические, социальные и т.п. Сложность системы можно выразить в уровнях управления, чем сложнее система, тем больше уровней управления она имеет. Однако большое количество уровней не означает, что те задачи, которые решает система, не могут быть решены более простой системой, с меньшим количеством уровней управления.

Любая сложная система основана на процессах, которые определяют существо этой системы и которые остаются неизменными во времени. Однако реализация этих процессов может постоянно видоизменяться, принимая различные формы. Сложная система не может иметь детальной формализации в силу своей сложности. Попытка составить детальное формальное описание такой системы приводит к слишком высоким расходам человеческих, временных и финансовых ресурсов. За то время пока создаётся детальное формальное описание, предметная область или наши знания о ней могут существенно измениться, что сделает часть описания непригодным для использования.

*Сделаем выводы о том, что сложная система:*

1. основана на ограниченном числе базовых процессов, определяющих существо системы;

2. не имеет детального формального описания;

3. обладает выраженной иерархичностью;

4. находится в постоянном развитии и, как следствие, не имеет завершённого состояния;

5. разложима на более простые составляющие, вплоть до элементарных;

6. определяется сложностью связей между её составляющими;

7. выполняет только функции управления по отношению к составляющим;

1. **Классический жизненный цикл ПС.**

Очень часто классический жизненный цикл называют каскадной или водопадной моделью, подчеркивая, что разработка рассматривается как последовательность этапов, причем переход на следующий, иерархически нижний этап происходит только после полного завершения работ на текущем этапе (рис. 1.1).

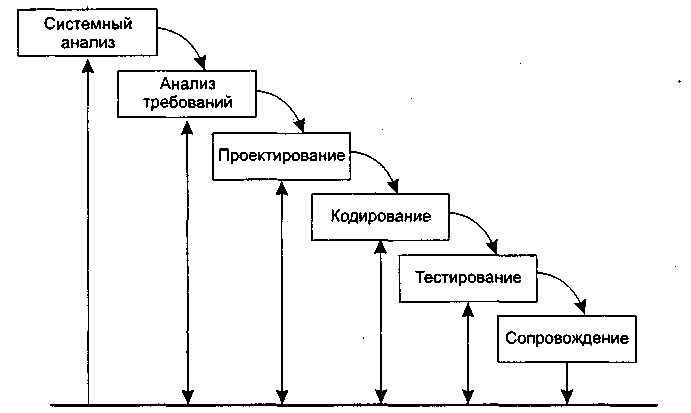
Охарактеризуем содержание основных этапов.

Подразумевается, что разработка начинается на системном уровне и проходит через анализ, проектирование, кодирование, тестирование и сопровождение. При этом моделируются действия стандартного инженерного цикла.

*Системный анализ* задает роль каждого элемента в компьютерной системе, взаимодействие элементов друг с другом. Поскольку ПО является лишь частью большой системы, то анализ начинается с определения требований ко всем системным элементам и назначения подмножества этих требований программному «элементу». Необходимость системного подхода явно проявляется, когда формируется интерфейс ПО с другими элементами (аппаратурой, людьми, базами данных). На этом же этапе начинается решение задачи планирования проекта ПО. В ходе планирования проекта определяются объем проектных работ и их риск, необходимые трудозатраты, формируются рабочие задачи и план-график работ.

*Анализ требований* относится к программному элементу — программному обеспечению. Уточняются и детализируются его функции, характеристики и интерфейс.

Все определения документируются в *спецификации анализа.* Здесь же завершается решение задачи планирования проекта.



**Рис. 1.1.** Классический жизненный цикл разработки ПО

Проектирование состоит в создании представлений:

* архитектуры ПО;
* модульной структуры ПО;
* алгоритмической структуры ПО;
* структуры данных;
* входного и выходного интерфейса (входных и выходных форм данных).

Исходные данные для проектирования содержатся в *спецификации анализа,* то есть в ходе проектирования выполняется трансляция требований к ПО во множество проектных представлений. При решении задач проектирования основное внимание уделяется качеству будущего программного продукта.

*Кодирование* состоит в переводе результатов проектирования в текст на языке программирования.

*Тестирование* — выполнение программы для выявления дефектов в функциях, логике и форме реализации программного продукта.

*Сопровождение —* это внесение изменений в эксплуатируемое ПО. Цели изменений:

* исправление ошибок;
* адаптация к изменениям внешней для ПО среды;
* усовершенствование ПО по требованиям заказчика.

Сопровождение ПО состоит в повторном применении каждого из предшествующих шагов (этапов) жизненного цикла к существующей программе но не в разработке новой программы.

Как и любая инженерная схема, классический жизненный цикл имеет достоинства и недостатки.

*Достоинства классического жизненного цикла:* дает план и временной график по всем этапам проекта, упорядочивает ход конструирования.

*Недостатки классического жизненного цикла:*

1) реальные проекты часто требуют отклонения от стандартной последовательности шагов;

2) цикл основан на точной формулировке исходных требований к ПО (реально в начале проекта требования заказчика определены лишь частично);

3) результаты проекта доступны заказчику только в конце работы.

1. **Итерационный и эволюционный жизненный цикл ПС.**

Общепринятая модель жизненного цикла является идеальной уже потому, что только очень простые задачи проходят все этапы без каких-либо итераций — возвратов на предыдущие шаги производственного процесса. При программировании, например, может обнаружиться, что реализация некоторой функции очень громоздка, неэффективна и вступает в противоречие с требуемой от системы производительностью. В этом случае необходимо перепроектирование, а может быть, и переделка спецификаций. При разработке больших нетрадиционных систем итеративность возникает регулярно на любом этапе жизненного цикла как из-за допущенных на предыдущих шагах ошибок и неточностей, так и из-за изменений внешних требований к условиям эксплуатации системы.

Существуют 3 стратегии конструирования ПО:

* *однократный проход* (водопадная стратегия) — линейная последовательность этапов конструирования;
* *инкрементная стратегия.* В начале процесса определяются все пользовательские и системные требования, оставшаяся часть конструирования выполняется в виде последовательности версий. Первая версия реализует часть запланированных возможностей, следующая версия реализует дополнительные возможности и т. д., пока не будет получена полная система;
* *эволюционная стратегия.* Система также строится в виде последовательности версий, но в начале процесса определены не все требования. Требования уточняются в результате разработки версий.

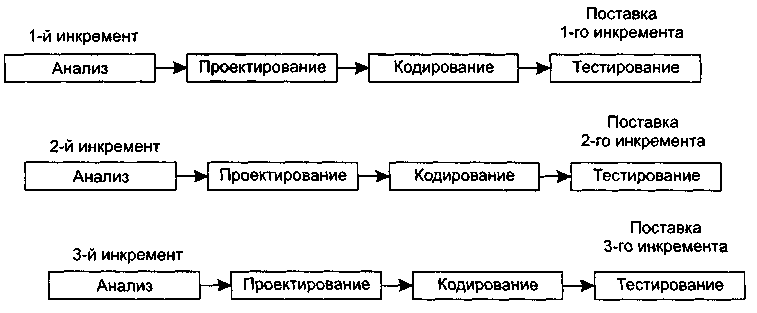
***Инкрементная модель***

Инкрементная модель является классическим примером инкрементной стратегии конструирования (рис. 1.4). Она объединяет элементы последовательной водопадной модели с итерационной философией макетирования.

Первый инкремент приводит к получению базового продукта, реализующего базовые требования (правда, многие вспомогательные требования остаются нереализованными).

План следующего инкремента предусматривает модификацию базового продукта, обеспечивающую дополнительные характеристики и функциональность.

По своей природе инкрементный процесс итеративен, но, в отличие от макетирования, инкрементная модель обеспечивает на каждом инкременте работающий продукт.



1. **Современные методы разработки ПП: ХР-процесс**

Экстр. программирование— облегченный (подвижный) процесс (или методология), (Кент Бек 1999) [11]. ХР-процесс ориентирован на группы малого и среднего размера, строящие программное обеспечение в условиях неопределенных или быстро изменяющихся требований. ХР-группу образуют до 10 сотрудников, которые размещаются в одном помещении. Динамизм обеспечивается с помощью четырех характеристик: непрерывной связи с заказчиком (и в пределах группы), простоты (всегда выбирается минимальное решение), быстрой обратной связи (с помощью модульного и функционального тестирования), смелости в проведении профилактики возможных проблем.

Основная идея ХР — устранить высокую стоимость изменения. ХР-процесс должен быть высокодинамичным процессом. ХР-группа имеет дело с изменениями требований на всем протяжении итерационного цикла разработки, причем цикл состоит из очень коротких итераций. 4 базовые действия в ХР-цикле: кодирование, тестирование, выслушивание заказчика и проектирование.

Базис ХР образуют перечисленные ниже двенадцать методов.

1. Игра планирования (Planning game) — быстрое определение области действия следующей реализации путем объединения деловых приоритетов и технических оценок. Заказчик формирует область действия, приоритетность и сроки с точки зрения бизнеса, а разработчики оценивают и прослеживают продвижение (прогресс).

2. Частая смена версий (Small releases) — быстрый запуск в производство простой системы. Новые версии реализуются в очень коротком (двухнедельном) цикле.

3. Метафора (Metaphor) — вся разработка проводится на основе простой, общедоступной истории о том, как работает вся система.

4. Простое проектирование (Simple design) — проектирование выполняется настолько просто, насколько это возможно в данный момент.

5. Тестирование (Testing) — непрерывное написание тестов для модулей, которые должны выполняться безупречно; заказчики пишут тесты для демонстрации законченности функций. «Тестируй, а затем кодируй» означает, что входным критерием для написания кода является «отказавший» тестовый вариант.

6. Реорганизация (Refactoring) — система реструктурируется, но ее поведение не изменяется; цель — устранить дублирование, улучшить взаимодействие, упростить систему или добавить в нее гибкость.

7. Парное программирование (Pair programming) — весь код пишется двумя программистами, работающими на одном компьютере.

8. Коллективное владение кодом (Collective ownership) — любой разработчик может улучшать любой код системы в любое время.

9. Непрерывная интеграция (Continuous integration) — система интегрируется и строится много раз в день, по мере завершения каждой задачи. Непрерывное регрессионное тестирование, то есть повторение предыдущих тестов, гарантирует, что изменения требований не приведут к регрессу функциональности.

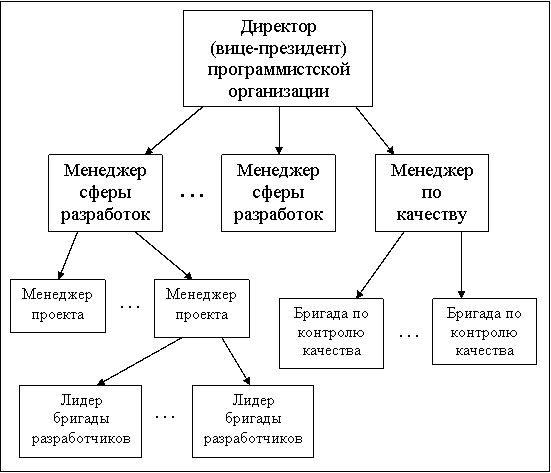
10. 40-часовая неделя (40-hour week) — как правило, работают не более 40 часов в неделю. Нельзя удваивать рабочую неделю за счет сверхурочных работ.

11. Локальный заказчик (On-site customer) — в группе все время должен находиться представитель заказчика, действительно готовый отвечать на вопросы разработчиков.

12. Стандарты кодирования (Coding standards) — должны выдерживаться правила, обеспечивающие одинаковое представление программного кода во всех частях программной системы

1. **Структура управления разработкой программного средства.**

Разработка ПС обычно производится в организации, в которой одновременно могут вестись разработки ряда других программных средств. Для управления всеми этими программными проектами используется иерархическая структура управления.



***директор***программистской организации, (упр. Всеми разработками ПС, решение о начале/прекращении проекта, обсуждение общих организ-х требований к проекту, решение возникающих проблем, требующих использование общих ресурсов программистской организации или изменения заказчиком общих требований)

***менеджер сферы разработок*** (упр-е разр-ками ПС опр. типа, формирование кол-ва исполнителей по проекту, осущ-ет обобщение опыта разр-к в своей сфере, накопление программных средств и документов для повторного использования.)

***менеджер проекта*** (назначается на каждый прогр. проект, осущ-ет планирование и составление расписаний работы бригад)

***бригада*** (большие проекты разбиваются а несколько относительно независимых подпроектов таким образом, чтобы каждый подпроект мог быть выполнен отдельной небольшой бригадой разработчиков)

     обычные бригады (лидер бригады руководит работой младших программистов. Ошибки одного программиста могут препятствовать работе других программистов.)

     неформальные демократические бригады (проект обсуждается всеми членами бригады, задания распределяются в зависимости от способностей и опыта )

     бригады ведущего программиста (Ядро бригады - ведущий программист , его дублер, администратор базы данных разработки).

***менеджер по качеству –***управление обеспечением качества. Каждый этап разр-ки заканчивается смотром по контролю качества - функцией управления разработкой, связан с оценкой того, насколько результаты этой работы согласуются с декларированными требованиями относительно качества ПС.

Существенную роль в управлении качеством ПС играют программные (софтверные) стандарты – они фиксируют удачный опыт высоко квалифицированных специалистов по разработке. Различают два вида таких стандартов:

     стандарты ПС (программного продукта),

     стандарты процесса создания и использования ПС.

*Стандарты ПС* определяют некоторые свойства, которыми должны обладать программы или документы ПС. ( стандарты на языки программирования, на состав документации, на структуру различных документов, на различные форматы и другие).

*Стандарты процесса создания и использования ПС* определяют, как должен проводится этот процесс, т.е. подход к разработке ПС, структуру жизненного цикла ПС и его технологические процессы.

***Бригада по контролю качества*** состоит из ассистентов по качеству ПС. Она проводит смотр ПС с целью поиска возникающих проблем в процессе его разработки, с учетом требований, сформулированных в спецификации качества ПС. Результат – замечания разработчикам (устные/письменные)

1. **Метрики и методы оценки объема работ.**
2. **Внешнее описание программного средства.**

Внешнее описание ПС играет роль точной постановки задачи, решение которой должно обеспечить разрабатываемое ПС. Более того, оно должно содержать всю информацию, которую необходимо знать пользователю для применения ПС. Оно является исходным документом для трех параллельно протекающих процессов: разработки текстов (конструированию и кодированию) программ, входящих в ПС, разработки документации по применению ПС и разработки существенной части комплекта тестов для тестирования ПС. Ошибки и неточности во внешнем описании, в конечном счете, трансформируются в ошибки самой ПС и обходятся особенно дорого, во-первых, потому, что они делаются на самом раннем этапе разработки ПС, и, во-вторых, потому, что они распространяются на три параллельных процесса. Это требует принятия особенно серьезных мер по их предупреждению.

Исходным документом для разработки внешнего описания ПС являются *определение требований* к ПС. Но так как через этот документ передается от заказчика (пользователя) к разработчику основная информация относительно требуемого ПС, то формирование этого документа представляет собой довольно длительный и трудный итерационный процесс взаимодействия между заказчиком и разработчиком, с которого и начинается этап разработки требований к ПС.

В определении внешнего описания легко бросаются в глаза две самостоятельные его части. Описание поведения ПС определяет функции, которые должна выполнять ПС, и потому его называют *функциональной спецификацией* ПС. Функциональная спецификация определяет допустимые фрагменты программ, реализующих декларированные функции. Требования к качеству ПС должны быть сформулированы так, чтобы разработчику были ясны цели, которые он должен стремиться достигнуть при разработке этого ПС. Эту часть внешнего описания будем называть *спецификацией качества* ПС (в литературе ее часто называют *нефункциональной спецификацией*, но она, как правило, включает и требования к технологическим процессам). Она, в отличие от функциональной спецификации, представляется в неформализованном виде и играет роль тех ориентиров, которые в значительной степени определяют выбор подходящих альтернатив при реализации функций ПС, а также определяет стиль всех документов и программ требуемого ПС. Тем самым, спецификация качества играет решающую роль в обеспечении требуемого качества ПС.

Обычно разработка спецификации качества предшествует разработке функциональной спецификации ПС, так как некоторые требования к качеству ПС могут предопределять включение в функциональную спецификацию специальных функций, например, функции защиты от несанкционированного доступа к некоторым объектам информационной среды. Таким образом, структуру внешнего описания ПС можно выразить формулой:

Внешнее описание ПС = определение требований + спецификация качества ПС  + функциональная спецификация ПС

Таким образом, внешнее описание определяет, что должно делать ПС и какими внешними свойствами оно должно обладать. Оно должно достаточно точно и полно определять задачи, которые должны решить разработчики требуемого ПС. В то же время оно должно быть понято представителем пользователем - на его основании заказчиком достаточно часто принимается окончательное решение на заключение договора на разработку ПС. Внешнее описание играет большую роль в обеспечении требуемого качества ПС, так как спецификация качества ставит для разработчиков ПС конкретные ориентиры, управляющие выбором приемлемых решений при реализации специфицированных функций.

1. **Основные классы архитектур программных средств.**

**Архитектура ПС** - это представление ПС как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы.

Разработка архитектуры является первым этапом борьбы со сложностью ПС, на котором реализуется принцип выделения относительно независимых компонент.

**Основные задачи разработки архитектуры ПС:**

1)выделение программных подсистем и отображение на них внешних функций (заданных во внешнем описании) ПС;

2)определение способов взаимодействия между выделенными программными подсистемами.

**Основные классы архитектур программных средств:**

1)цельная программа;

2)комплекс автономно выполняемых программ;

3)слоистая программная система;

4)коллектив параллельно выполняемых программ.

**Цельная программа** представляет вырожденный случай архитектуры ПС: в состав ПС входит только одна программа. Такую архитектуру выбирают обычно в том случае, когда ПС должно выполнять одну какую-либо ярко выраженную функцию и ее реализация не представляется слишком сложной. Естественно, что такая архитектура не требует какого-либо описания (кроме фиксации класса архитектуры), так как отображение внешних функций на эту программу тривиально, а определять способ взаимодействия не требуется (в силу отсутствия какого-либо внешнего взаимодействия программы, кроме как взаимодействия ее с пользователем, а последнее описывается в документации по применению ПС).

**Комплекс автономно выполняемых программ** состоит из набора программ, такого, что:

1)любая из этих программ может быть активизирована (запущена) пользователем;

2)при выполнении активизированной программы другие программы этого набора не могут быть активизированы до тех пор, пока не закончит выполнение активизированная программа;

3)все программы этого набора применятся к одной и той же информационной среде.

Таким образом, программы этого набора по управлению никак не взаимодействуют - взаимодействие между ними осуществляется только через общую информационную среду.

**Слоистая программная система** состоит из некоторой упорядоченной совокупности программных подсистем, называемых слоями, такой, что:

1)на каждом слое ничего не известно о свойствах (и даже существовании) последующих (более высоких) слоев;

2)каждый слой может взаимодействовать по управлению (обращаться к компонентам) с непосредственно предшествующим (более низким) слоем через заранее определенный интерфейс, ничего не зная о внутреннем строении всех предшествующих слоев;

3)каждый слой располагает определенными ресурсами, которые он либо скрывает от других слоев, либо предоставляет непосредственно последующему слою (через указанный интерфейс) некоторые их абстракции.

**Коллектив параллельно действующих программ** представляет собой набор программ, способных взаимодействовать между собой, находясь одновременно в стадии выполнения. Это означает, что такие программы, во-первых, вызваны в оперативную память, активизированы и могут попеременно разделять по времени один или несколько центральных процессоров, а во-вторых, осуществлять между собой динамические (в процессе выполнения) взаимодействия, на базе которых производиться их синхронизация. Обычно взаимодействие между такими процессами производится путем передачи друг другу некоторых сообщений.

1. **Понятие программного модуля и его основные характеристики.**

Приступая к разработке каждой программы ПС, следует иметь ввиду, что она, как правило, является большой системой, поэтому мы должны принять меры для ее упрощения. Для этого такую программу разрабатывают по частям, которые называются программными модулями. Программный модуль - это любой фрагмент описания процесса, оформляемый как самостоятельный программный продукт, пригодный для использования в описаниях процесса. Это означает, что каждый программный модуль программируется, компилируется и отлаживается отдельно от других модулей программы, и тем самым, физически разделен с другими модулями программы. Более того, каждый разработанный программный модуль может включаться в состав разных программ, если выполнены условия его использования, декларированные в документации по этому модулю.

**Основные характеристики программного модуля:**

хороший модуль снаружи проще, чем внутри;

хороший модуль проще использовать, чем построить.

Размер модуля измеряется числом содержащихся в нем операторов (строк). Модуль не должен быть слишком маленьким или слишком большим.

**Прочность модуля** - это мера его внутренних связей. Чем выше прочность модуля, тем больше связей он может спрятать от внешнейпо отношению к нему части программы и, следовательно, тем больший вклад в упрощение программы он может внести

**Функционально прочный модуль** - это модуль, выполняющий (реализующий) одну какую-либо определенную функцию. При реализации этой функции такой модуль может использовать и другие модули.

**Информационно прочный модуль** - это модуль, выполняющий (реализующий) несколько операций (функций) над одной и той же структурой данных (информационным объектом), которая считается неизвестной вне этого модуля. Для каждой из этих операций в таком модуле имеется свой вход со своей формой обращения к нему.

**Сцепление модуля** - это мера его зависимости по данным от других модулей. Характеризуется способом передачи данных. Чем слабее сцепление модуля с другими модулями, тем сильнее его независимость от других модулей.

**Рутинность модуля** - это его независимость от предыстории обращений к нему.

1. **Методы разработки структуры программ.**

В процессе разработки программы ее модульная структура может по-разному формироваться и использоваться для определения порядка программирования и отладки модулей, указанных в этой структуре. Поэтому можно говорить о разных методах разработки структуры программы. Обычно в литературе обсуждаются два метода: **метод восходящей разработки и метод нисходящей разработки.**

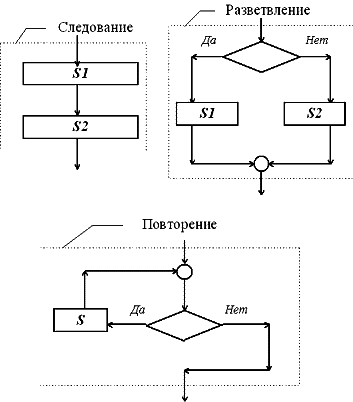
**Метод восходящей разработки** заключается в следующем. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня (листья дерева модульной структуры программы), в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в принципе в таком же (восходящем) порядке, в каком велось их программирование.

**Метод *нисходящей разработки*** заключается в следующем. Как и в предыдущем методе сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модуля самого верхнего уровня (головного), переходя к программированию какого-либо другого модуля только в том случае, если уже запрограммирован модуль, который к нему обращается. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же (нисходящем) порядке. При этом первым тестируется головной модуль программы, который представляет всю тестируемую программу и поэтому тестируется при естественном состоянии информационной среды, при котором начинает выполняться эта программа. При этом те модули, к которым может обращаться головной, заменяются их имитаторами (так называемыми заглушками). Каждый *имитатор* модуля представляется весьма простым программным фрагментом, который, в основном, сигнализирует о самом факте обращения к имитируемому модулю, производит необходимую для правильной работы программы обработку значений его входных параметров (иногда с их распечаткой) и выдает, если это необходимо, заранее запасенный подходящий результат.

1. **Структурное программирование и пошаговая детализация. Псевдокод.**

***Структурное программирование.***

Построение программы как композиции из нескольких типов управляющих конструкций (структур), которые позволяют сильно повысить понимаемость логики работы программы. Программирование с использованием только таких конструкций назвали *структурным*.



Основными конструкциями структурного программирования являются: следование, разветвление и повторение (см. Рис.5). Компонентами этих конструкций являются обобщенные операторы (узлы обработки) S, S1, S2 и условие (предикат) P. В качестве обобщенного оператора может быть либо простой оператор используемого языка программирования (операторы присваивания, ввода, вывода, обращения к процедуре), либо фрагмент программы, являющийся композицией основных управляющих конструкций структурного программирования. Существенно, что каждая из этих конструкций имеет по управлению только один вход и один выход. Тем самым, и обобщенный оператор имеет только один вход и один выход.

Весьма важно также, что эти конструкции являются уже математическими объектами (что, по существу, и объясняет причину успеха структурного программирования). Доказано, что для каждой неструктурированной программы можно построить функционально эквивалентную (т.е. решающую ту же задачу) структурированную программу. Для структурированных программ можно математически доказывать некоторые свойства, что позволяет обнаруживать в программе некоторые ошибки.

***Пошаговая детализация и понятие о псевдокоде.***

В качестве основного метода построения текста модуля современная технология программирования рекомендует *пошаговую детализацию.* Сущность этого метода заключается в разбиении процесса разработки текста модуля на ряд шагов.

На первом шаге описывается общая схема работы модуля в обозримой линейной текстовой форме (т.е. с использованием очень крупных понятий), причем это описание не является полностью формализованным и ориентировано на восприятие его человеком. На каждом следующем шаге производится уточнение и детализация одного из понятий (будем называть его *уточняемым*), в каком либо описании, разработанном на одном из предыдущих шагов. В результате такого шага создается описание выбранного уточняемого понятия либо в терминах базового языка программирования (т.е. выбранного для представления модуля), либо в такой же форме, что и на первом шаге с использованием новых уточняемых понятий. Этот процесс завершается, когда все уточняемые понятия будут *уточнения* (т.е. в конечном счете будут выражены на базовом языке программирования). Последним шагом является получение текста модуля на базовом языке программирования путем замены всех вхождений уточняемых понятий заданными их описаниями и выражение всех вхождений конструкций структурного программирования средствами этого языка программирования.

Пошаговая детализация связана с использованием частично формализованного языка для представления указанных описаний, который получил название *псевдокода*. Этот язык позволяет использовать все конструкции структурного программирования, которые оформляются формализовано, вместе с неформальными фрагментами на естественном языке для представления обобщенных операторов и условий.

1. **Тестирование и отладка программных средств.**

*Отладка* ПС − это деятельность, направленная на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ. *Тестирование* ПС − это процесс выполнения его программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется *тестовым* или просто *тестом*. Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в ПС ошибки, поиска места ошибки в программах и документации ПС и редактирования программ и документации с целью устранения обнаруженной ошибки. Другими словами:

Отладка = Тестирование + Поиск ошибок + Редактирование.

*1)Автономная отладка программного средства:*

При автономной отладке ПС каждый модуль на самом деле тестируется в некотором программном окружении, кроме случая, когда отлаживаемая программа состоит только из одного модуля. Это окружение состоит из других модулей, часть которых является модулями отлаживаемой программы, которые уже отлажены, а часть − модулями, управляющими отладкой (отладочными модулями, см. ниже). Таким образом, при автономной отладке тестируется всегда некоторая программа (тестируемая программа), построенная специально для тестирования отлаживаемого модуля.

*2)Комплексная отладка программного средства:*

**Тестирование архитектуры ПС***.* Целью тестирования является поиск несоответствия между описанием архитектуры и совокупностью программ ПС.

**Тестирование внешних функций**. Целью тестирования является поиск расхождений между функциональной спецификацией и совокупностью программ ПС

**Тестирование качества ПС***.* Целью тестирования является поиск нарушений требований качества, сформулированных в спецификации качества ПС.

**Тестирование документации по применению ПС***.* Целью тестирования является поиск несогласованности документации по применению и совокупностью программ ПС, а также выявление неудобств, возникающих при применении ПС.

**Тестирование определения требований к ПС.** Целью тестирования является выяснение, в какой мере ПС не соответствует предъявленному определению требований к нему.

1. **Объектно-ориентированный подход к разработке ПС.**

Объектно-ориентированное представление ПС основывается на принципах абстрагирования, инкапсуляции, модульности и иерархической организации.

*Абстрагирование* – это отвлечение от несущественных характеристик конкретных объектов, выделение только существенных характеристик. (часы – время без учета формы , цвета.. )

*Инкапсуляция* является процессом разделения элементов абстракции на секции с различной видимостью. Инкапсуляция служит для отделения интерфейса абстракции от ее реализации.

Инкапсуляция и абстракция — взаимодополняющие понятия: абстракция выделяет внешнее поведение объекта, а инкапсуляция содержит и скрывает реализацию, которая обеспечивает это поведение.

Модульность определяет способность системы подвергаться декомпозиции на ряд сильно связанных и слабо сцепленных модулей. Общая цель декомпозиции на модули: уменьшение сроков разработки и стоимости ПС за счет выделения модулей, которые проектируются и изменяются независимо.

Иерархическая организация — формирование из абстракций иерархической структуры. Определением иерархии в проекте упрощаются понимание проблем заказчика и их реализация — сложная система становится обозримой человеком.

Двумя важными инструментами иерархической организации в объектно-ориентированных системах являются:

* структура из классов *(«is a»*-иерархия);
* структура из объектов *(«part of»*-иерархия).

**объектно-ориентированный подход помогает справиться с такими сложными проблемами, как**

1)уменьшение сложности программного обеспечения;

2)повышение надежности программного обеспечения;

3)обеспечение возможности модификации отдельных компонентов программного обеспечения без изменения остальных его компонентов;

4)обеспечение возможности повторного использования отдельных компонентов программного обеспечения.

Систематическое применение объектно-ориентированного подхода позволяет разрабатывать хорошо структурированные, надежные в эксплуатации, достаточно просто модифицируемые программные системы. Этим объясняется интерес программистов к объектно-ориентированному подходу и объектно-ориентированным языкам программирования. Объектно-ориентированный подход является одним из наиболее интенсивно развивающихся направлений теоретического и прикладного программирования.

1. **Унифицированный язык моделирования UML.**

UML — стандартный язык для написания моделей анализа, проектирования и реализации объектно-ориентированных программных систем. UML может использоваться для визуализации, спецификации, конструирования и документирования результатов программных проектов. UML — это не визуальный язык программирования, но его модели прямо транслируются в текст на языках программирования (Java, C++, Visual Basic, Ada 95, Object Pascal) и даже в таблицы для реляционной БД.

Словарь UML образуют три разновидности строительных блоков: предметы, отношения, диаграммы.

Предметы — это абстракции, которые являются основными элементами в модели, отношения связывают эти предметы, диаграммы группируют коллекции предметов.

**Язык UML предназначен для решения следующих задач:**

1)Предоставить в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, специально предназначенный для разработки и документирования моделей сложных систем самого различного целевого назначения.

2)Снабдить исходные понятия языка UML возможностью расширения и специализации для более точного представления моделей систем в конкретной предметной области.

3)Описание языка UML должно поддерживать такую спецификацию моделей, которая не зависит от конкретных языков программирования и инструментальных средств проектирования программных систем.

4)Описание языка UML должно включать в себя семантический базис для понимания общих особенностей объектно-ориентированного анализа и проектирования.

5)Поощрять развитие рынка объектных инструментальных средств.

6)Способствовать распространению объектных технологий и соответствующих понятий объектно-ориентированного анализа и проектирования.

7)Интегрировать в себя новейшие и наилучшие достижения практики объектно-ориентированного анализа и проектирования.

**Общая структура языка UML:**

С самой общей точки зрения описание языка UML состоит из двух взаимодействующих частей, таких как:

1)Семантика языка UML. Представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на языке UML.

2)Нотация языка UML. Представляет собой графическую нотацию для визуального представления семантики языка UML.

В UML имеются четыре разновидности предметов:

***Структурные предметы***являются существительными в UML-моделях. Представляют статические части модели — понятийные или физические элементы. Перечислим восемь разновидностей структурных предметов.

1. *Класс* — описание множества объектов, которые разделяют одинаковые свойства, операции, отношения и семантику (смысл). Класс реализует один или несколько интерфейсов.
2. *Интерфейс —* набор операций, которые определяют услуги класса или компонента. Интерфейс описывает поведение элемента, видимое извне.
3. *Кооперация* (сотрудничество) определяет взаимодействие и является совокупностью ролей и других элементов, которые работают вместе для обеспечения коллективного поведения более сложного, чем простая сумма всех элементов.
4. *Актер —* набор согласованных ролей, которые могут играть пользователи при взаимодействии с системой. Каждая роль требует от системы определенного поведения.
5. *Элемент Use Case* (Прецедент) — описание последовательности действий (или нескольких последовательностей), выполняемых системой в интересах отдельного актера и производящих видимый для актера результат.
6. *Активный класс —* класс, чьи объекты имеют один или несколько процессов (или потоков) и поэтому могут инициировать управляющую деятельность
7. *Компонент —* физическая и заменяемая часть системы, которая соответствует набору интерфейсов и обеспечивает реализацию этого набора интерфейсов
8. *Узел —* физический элемент, который существует в период работы системы и представляет ресурс, обычно имеющий память и возможности обработки

***Предметы поведения*** *—* динамические части UML-моделей. Они являются глаголами моделей, представлением поведения во времени и пространстве. Существует две основные разновидности предметов поведения.

1. *Взаимодействие* — поведение, заключающее в себе набор сообщений, которыми обменивается набор объектов в конкретном контексте для достижения определенной цели.
2. *Конечный автомат —* поведение, которое определяет последовательность состояний объекта или взаимодействия, выполняемые в ходе его существования в ответ на события .С помощью конечного автомата может определяться поведение индивидуального класса или кооперации классов. Элементами конечного автомата являются состояния, переходы (от состояния к состоянию), события (предметы, вызывающие переходы) и действия (реакции на переход).

***Группирующие предметы***— организационные части UML-моделей. Это ящики, по которым может быть разложена модель. Предусмотрена одна разновидность группирующего предмета — *пакет —* общий механизм для распределения элементов по группам. В пакет могут помещаться структурные предметы, предметы поведения и даже другие группировки предметов. В отличие от компонента (который существует в период выполнения), пакет существует только в период разработки.

***Поясняющие предметы*** *—* разъясняющие части UML-моделей. Они являются замечаниями, которые можно применить для описания, объяснения и комментирования любого элемента модели. Предусмотрена одна разновидность поясняющего предмета — *примечание —* символ для отображения ограничений и замечаний, присоединяемых к элементу или совокупности элементов

В UML имеются четыре разновидности отношений:

*Зависимость* — семантическое отношение между двумя предметами, в котором изменение в одном предмете (независимом предмете) может влиять на семантику другого предмета (зависимого предмета).

1. *Ассоциация —* структурное отношение, которое описывает набор связей, являющихся соединением между объектами.
2. *Обобщение* — отношение специализации/обобщения, в котором объекты специализированного элемента (потомка, ребенка) могут заменять объекты обобщенного элемента (предка, родителя). Иначе говоря, потомок разделяет структуру и поведение родителя.
3. *Реализация —* семантическое отношение между классификаторами, где один классификатор определяет контракт, который другой классификатор обязуется выполнять (к классификаторам относят классы, интерфейсы, компоненты, элементы Use Case, кооперации).

*Диаграмма* — графическое представление множества элементов, наиболее часто изображается как связный граф из вершин (предметов) и дуг (отношений). UML включает девять видов диаграмм:

1) диаграммы классов;

2) диаграммы объектов;

**Архитектура ЭВМ и систем.**

1. **Поколения архитектур ЭВМ. Понятие архитектура ЭМВ в узком и широком смысле. Основные характеристики ЭВМ.**

Термин архитектура системы часто употребляется как в узком так и в широком смысле. В узком смысле под арх понимается арх набора команд. Арх набора команд служит границей между аппаратурой и ПО и представляет ту часть системы, которая видна программисту или разработчику компиляторов. В широком смысле арх охватывает понятие организации системы (сист памяти, структуру сист шины, организацию ввода-вывода и тд.). Применительно к выч системам термин арх может быть определён как распределение функций, реализуемых системой между её уровнями, тоесть предполагается многоуровневая организация.

Арх ЭВМ принято считать совокупность принципов системной, функциональной, логической и физической организации аппаратных и программных средств ЭВМ.

ЭВМ – комплекс технических средств, предназначенных для автоматической обработки информации в процессе решения вычислительных информационных задач.

ЭВМ классифицируется по следующим признакам:

1 по физическому представлению обрабатываемой информации

2 по поколениям(этапам создания и элементной базе)

3 по сферам применения и методам использования, а также размерам и вычислительной мощности.

Поколения ЭВМ:

Идея делить машины на поколения вызвана стремительной эволюцией ЭВМ как в смысле элементной базы, так и в смысле изменения её структуры, появления новых возможностей, расширения областей применения и характеристика использования

К 1му поколению относятся машины, созданные на рубеже 50-х годов(элементная база – электронные лампы). Такие ЭВМ потребляли большое количество тепла. Набор команд был ограничен, устройство управления и схема АЛУ были простыми, ПО практически отсутствовало. Показатели объёма оперативной памяти были низкими (10-20 тыс операций в секунду). Для ввода – вывода использовались перфоленты и перфокарты, магнитные ленты и печатающие устройства. Программы писали на языке конкретной машины. Отечественные машины этого поколения: урал, м-20, малоэлектронная счётная машина (МЭСМ).

2ое поколение (1955-1965) характеризуется использованием как ламповых так и транзисторных логических элементов. Оперативная память построена на магнитных сердечниках. Появились высокопроизводительные устройства для работы с магнитными лентами, магнитные барабаны и первые магнитные диски. Быстродействие достигло сотен тыс операций, ёмкость памяти 10 десятков тысяч слов. Появились языки высокого уровня. Для перевода их на машинный язык были созданы особые программы, которые получили называние трансляторы. Появился широкий набор библиотечных программ, которые в дальнейшем привели к созданию оперативных систем. Недостаток машин 2ого поколения это программная несовместимость.

3е поколение . Совместимость машин программной архитектуры. Имеют разные ОС, обладают возможностями мультипрограммирования. Быстродействие до млн операций в сек. Ёмкость ОП несколько сотен тыс слов. Примеры: IBM 360, 370 и единая система ЭВМ.

4ое поколение (после 1970г.). Эти машины проектировались в расчёте на эффективное использование современных высокоуровневых языков и упрощение процесса программирования для конечного пользователя. Быстродействие несколько десятков млн слов. ОП 512Мб. Для них характерно: применение ПК, телекоммуникационная обработка данных, комп сети, широкое применение СУБД, элементы интеллектуального поведения СУБД и устройств.

5ое поколение. В них должен произойти качественный переход от обработки данных к обработке знаний. Архитектура таких компов будет содержать 2 основных блока: традиционный комп + интеллектуальный интерфейс.

Важнейшие традиционные характеристики ЭВМ:

Э=Р/(Сэвм+Сэкс)

Э – общий коэф эффективности машины, Р – производительность машины, Сэвм – стоимость ЭВМ, Сэкс – эксплуатация, затраты на эксплуатацию.

К наиболее распространённым хар-кам ЭВМ относятся: 1.число разрядов в машинном слове, 2. Скорость выполнения основных видов команд, 3. Ёмкость ОП, 4. Макс скорость передачи инф между ядром ЭВМ и периферийным оборудованием, 5. Эксплуатационная надёжность машины.

1. **Принципы (архитектура) фон Неймана. Состав ЭВМ, основные устройства.**

В основу построения большинства компов положены следующие общие принципы, которые сформированы Фон Нейманом в 1945г.

Принципы Фон Неймана:

1 принцип программного управления. Из него следует, что программа состоит из набора команд, которые выполняются процессором автоматически по определённой последовательности. Выборка программы из памяти осуществляется с помощью счётчика команд. Если для выполнения команд следует перейти не к следующей, а к какой то другой команде используются команды условного и безусловного перехода(ветвления). Таким образом процессор выполняет переход автоматически без вмешательства человека.

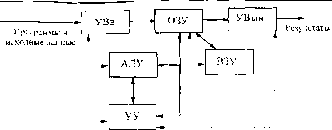
2 принцип однородности памяти. Проги и данные хранятся в одной и той же памяти, комп не различает хранится ли в данной ячейке памяти число, текст или команда. Это открывает целый ряд возможностей. Например, прога в процессе выполнения может подвергаться переработке, что позволяет создавать самой программе правила получения некоторых составных её частей. Команды одной её проги могут быть получены как результат выполнения другой проги. На этом принципе основаны методы трансляции – перевода текста программ с языка программирования высокого уровня на язык конкретной машины.

3 принцип адресности. Структурно основная память состоит из пронумерованных ячеек. Процесс в произвольный момент времени может обратиться к любой ячейке.

Центральное устройство состоит из центрального процессора и оперативной памяти. ЦП реализует операции обработки информации и управления выч процессом, осуществляет выборку машинных команд и данных из ОП и запись в ОП. Включение и отключение внешних устройств. Процессор состоит: 1.устройство управления с интерфейсом процессора (системы сопряжения связей процессора с другими узлами машины). 2. АЛУ. 3. Процессорная память (КЭШ).

Центральное устройство описывается следующими характеристиками : 1 длина машинного слова(разрядность, адресность), 2 система команд, 3 объём ОП, 4 быстродействие(тактовая частота процессора, цикл записи/считывания ОП).

Внешние устройства обеспечивают взаимодействие компа с окружающей средой – пользователями, объектами управления др выч машинами. Внешние устройства делятся на след группы: 1. Устройства ввода-вывода. 2. Устройства хранения(массовые накопители). 3. Устройство массового ввода/вывода информации.

Классическая архитектура (Фон Неймана) – это однопроцессорный комп, который включает в себя следующие типы устройств: 1. Центральный процессор(АЛУ, устройство управления). 2. Запоминающее устройство(ОП и внешнее ЗУ).устройства ввода-вывода.

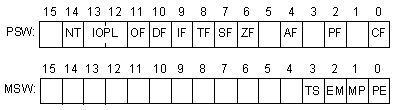
1. **Регистры АЛУ микропроцессора. РОН, сегментные регистры, регистр флагов.**

Данные, с которыми работает процессор, должны находиться в *регистрах*. Регистры – это устройства, предназначенные для временного хранения данных ограниченного размера. Р-р состоит из разрядов, которые можно записывать, запоминать и считывать слово, команду, двоичное число и т.д. Регистр, обладающий способностью перемещать содержимое своих разрядов, наз-ся сдвиговым В этом регистре за 1 такт хранимое слово сдвигается на 1 разряд. Такие регистры исп-ся для кодирования и декодирования. Пользовательские регистры: такие регистры программист может исп-ть для написания программ. К ним относятся РОН(8), регистры сегментов(6), регистры состояния и управления(2).

Регистры общего назначения. Они физически расположены внутри АЛУ процессора. Все регистры общего назначения могут использоваться для адресных вычислений и для получения результатов большинства арифметических и логических операций. Тем не менее, некоторые команды используют фиксированные регистры для хранения операндов. Например, команды обработки строк используют в качестве операндов содержимое регистров ECX, ESI и EDI. Использование фиксированных регистров для некоторых операций позволяет более компактно кодировать набор команд. Следующие команды используют фиксированные регистры: умножение и деление, ввод/вывод, обработка строк, перекодирование, цикл, сдвиговые операции, операции со стеком. Eax– аккумулятор, применяется для хранения промежуточных данных. Ebx– базовый регистр, применяется для хранения базового адреса некот объекта в памяти. Ecx– регистр – счетчик, применяется в командах, произв-х нек-е повтор-ся действие. Esi– индекс источника, этот регистр в цепочных операциях содержит текущий адрес эл-та цепочки устр-ва. Edi– индекс приемника, в цепочных операциях сод-т регистр. Esp – регистр – указатель стека, содержит указатель вершины стека в текущем эл-те. Ebp – регистр базы кадра стека, предназначен для орг-ии произв-го доступа к данным внутри стека. Стек – структура данных, кот поддерживает на программно-аппаратном уровне в арх микропр-ра.

Сегментные регистры*.* Микропр-р аппаратно поддерживает структ орг-ю программы в виде 3-х частей, кот наз-ют сегментами, соот-но память наз-ся сегментная. МП поддерж-ет след. Типы сегментов: 1)Cs-сегм регистр кода(содержит команды программы). 2) Ds - сегм регистр данных(сод-т обрабатываемые данные). 3) Ss-сегм регистр стека. (область памяти, кот наз-ют стеком), 4) Es, gs, fc- доп сегмент данных. *Регистры состояния и назначения.* В микропр-р вкл нескоко регистров, кот постоянно содержат информ о состоянии самого микропр-ра, программы, команды, кот в данный момент производится работа.Делятся на регистр флагов и регистр указателя команды. Регистр флагов – определяет и позволяет изменять состояние процессора. Каждый его бит называется «*флагом»* и может быть установлен или сброшен в зависимости от результата выполнения очередной команды. Также некоторые команды могут выполнять какие-то действия только в том случае, если установлен определенный флаг.

**Регистр флагов**

Этот регистр, называется eflags или 16-битная его часть flags. Этот регистр содержит информацию, которая используется побитно, а не в качестве числа. Каждый бит этого регистра называется флагом и имеет определенное значение. Программа пользуется этими флагами для управления своим выполнением. Так же есть управляющие флаги, которые задают работу программе. По выполнению каждой команды, анализируя определенные флаги можно судить о выполнении команды. Например, команда add(сложение двух чисел). По ее выполнению можно определить было ли перенос разрядов. Соответсвенно если было, то определенный флаг(СF) установится в 1, если нет, то останется 0. Дальше, анализируя флаг с помощью команд условного перехода можно к примеру обработать перенос.

mov al, 240d|mov ah, 16d|add al, ah|jc \_label

al= 11110000|ah=10000|

Дополнительные биты слова состояния процессора PSW имеют следующее назначение

CF флаг переноса

PF флаг четного

AF флаг дополнительного переноса

ZF флаг нуля

SF флаг знака

OF флаг переполнения

IOPL два бита, определяющие уровень привилегий ввода/вывода;

NT флаг вложенной задачи.

3 группы флагов: 1) 8 флагов состояния. Эти флаги изменяются после выполнения машинных команд. Отражают особенности рез0та выполнения логических или арифметических команд. Это даёт возможность анализировать состояние вычисл процесса и реагировать на него спомощью команд условных переходов и вызова попрограмм. 2) флаг управления DF, нах-ся в 10-ом виде р-ра и исп-ся цепочными командами. Значение DF определяет направления по элементной обработке в этой операции: от начала строки к концу(DF = 0) или от конца к началу (DF = 1). Для работы с DF сущ-ют спец-ые команды: cld – снять флаг DF и std – установить DF. Их использование позволяет увеличить или уменьшить значение счётчика при выполнении операций со строками. 3) 5 системных флагов. Управляют вводом, выводом, прерыванием, отладкой, переключением между режимами проц-ра.

1. **Основные характеристики памяти. Критерии классификации запоминающих устройств. Иерархия памяти современного ПК.**

памятью ЭВМ наз-ся сов-ть устройств, кот служат для запоминания, хран-я, и выдачи инф-ии.

отдельные уср-ва, входящие в эту сов-ть наз-ся запоминающие устр-ва.

Класс-ция ЗУ:

1)по типу ЗУ: - полупроводниковые, - магнитные, - конденсаторные, - оптоэлектронные, - голографические, - креагенные

2) по функц. назначения: - ОЗУ, - СОЗУ(свероперативные), - внешние зу, - промежуточные ЗУ

3) по способу орг-ции обр-я: - с последовательным поиском, - с прямым доступом, - с непосредственным доступом. – ассоциативные, - стековые, - магазинные

4) по хар-ру считывания: - с разрушением инф-ции, - без разрушения инф-ции

5) по способу хран-я: - статические, - динамические

6) по способу орг-ции: - однокоординатные, - двух-коорд, - 3-коорд, - 2-3-коорд

Хар-ки памяти:

Емкость памяти опред-ся максимальным кол-вом данных, кот могут иметь границы

Удельная емкость – отношение емкости ЗУ к её физич объему

Быстродействие памяти опр-ся продолжительностью операции обращения, т е временем, затрачиваемым на поиск нужной единицы инф-ции в памяти на её считывание или на поиск места в памяти, предназнач.для хран-я данный ед-цы инф-и и на её запись в память.

продолжит-ть обращения к памяти при считывании опр-ся по формуле:

, где - время доступа, определяющееся промежутком времени м/ду моментом начала операции обращения при счит-нии до мом-та когда стан-ся возможным доступ к данной ед-це инф-ии, - продолж-ть самого физич.процесса обнаружения и фиксации состояния соответствующих запом.эл-в.

В некот устр-вах считывание инф-ии сопров-ся её разрушением. В этом сл-е:, где – время доступа при записи, - время подготовки, расходуемое на поведение в исх.сост-е и запом.эл-в им участка пов-ти носителя инф-ии для записи опред.ед-цы инф-ии, -время занесения инф-ии.

**Иерархическая структура памяти.**

Идеальная память должна обеспечивать процессор командами и данными так, чтобы не вызывать простоев процессора. При этом память должна иметь большую емкость. В современных условиях уменьшение времени доступа достигается введе­нием многоуровневой иерархии памяти. Время доступа зависит от объема и типа используемой памяти.

Типовая современная иерархия памяти имеет следующую структуру:

* регистры 64 - 256 слов с временем доступа 1 такт процес­сора;
* кэш 1 уровня - 8к слов с временем доступа 2 такта;
* кэш 2 уровня - 256к слов с временем доступа 3-5 тактов;
* основная память - до 4 Гигаслов с временем доступа 12-55 тактов.

Используя помимо основной памяти небольшую и более быструю бу­ферную память, можно значительно сократить количество обращений к основной памяти, за счет аккумуляции текущего фрагмента программного кода в буферной памяти. Создание иерархической многоуровневой памяти, пересылающей блоки программ и данных между уровнями памяти за вре­мя, пока предшествующие блоки обрабатываются процессором, позволяет существенно сократить простои процессора в ожидании данных. При этом эффект уменьшения времени доступа в память будет тем больше, чем боль­ше время обработки данных в буферной памяти по сравнению с временем пересылки между буферной и основной памятью. Это достигается при ло­кальности обрабатываемых данных, когда процессор многократно исполь­зует одни и те же данные для выработки некоторого результата. Например, такая ситуация имеет место при решении систем уравнений в научных и инженерных расчетах, когда короткие участки программного кода с боль­шим количеством вложенных и зацепленных друг с другом циклов обраба­тывают поочередно, переходя от точки к точке, небольшие порции данных, многократно используя одни и те же данные и внутренние результаты.

В связи с тем, что локально обрабатываемые данные могут возникать в динамике вычислений и не обязательно сконцентрированы в одной об­ласти при статическом размещении в основной памяти, буферную память организуют как ассоциативную, в которой данные содержатся в совокуп­ности с их адресом в основной памяти. Такая буферная память получила название кэш-памяти. Кэш-память позволяет гибко согласовывать струк­туры данных, требуемые в динамике вычислений, со статическими струк­турами данных основной памяти.

1. **Адресная, ассоциативная и стековая организация памяти. Кэш – память.**

По способу обращения ЗУ делятся:

- с последов.поиском

- с прямым доступом

- с несосредств.доступом или адресные

- ассоциативные

- стековые

- магазинные

Организация опер.памяти

В адресном ЗУ каждый эл-т памяти имеет адрес, соответствующий его пространств.расположению в запоминающей среде. Поэтому обращение к соответств.эл-ту производится в соответствии с кодом его адреса. В ОЗУ после приема кода осущ-ся его дешифрация, после чего следует выборка из эл-та конкретной группы битов или слов.

*Ассоциативная память.* В памяти этого типа поиск нужной информации производится не по адресу, а по ее содержанию (по ассоциативному признаку). При этом поиск по ассоциативному признаку (или последовательно по отдельным разрядам этого признака) происходит параллельно во времени для всех ячеек запоминающего массива. Во многих случаях ассоциативный поиск позволяет существенно упростить и ускорить обработку данных.

Стек – область памяти, предназначенная для временного хран-я произвольных данных. Стек традиционно исп-ся для сохранения сожержимого регистров, использ=х пр-мой, перед вызовом подпрограммы. Др. распростр-й прием – прием-передача продпрограмме требуемых ею пар-ров через стек.

Отличит-й особ-тью явл. Своеобразный порядок в выборке содержащихс в нем данных. В любой момент времени в стеке доступен только верхний эл-т , т е эл-т загруженный в стек последним. Выгрузка из стека ерхнего эл-та делает доступным след. Эл-т. Эл-ты стека располагаются в обл.памяти отведенной под стек со дна стека, т е с его максимального адреса. Адрес верхнего доступного эл-та хранится в регистре указателя стека SP. Как любая др.область памяти стек должен входить в какой-то сегмент или образовывать отдельный сегмент. Сегментный адрес этого сегмента помещается в сегм.регистр стека SS. Пара регистров SS:SP описывает адрес доступной ячейки стека.

Кэш память представляет собой буферное ЗУ, работающее со скоростью обеспечивающей функц-е центрального процесса без режима ожидания. Создание иерархической многоуровневой памяти, пересылающей блоки программ и данных между уровнями памяти за вре­мя, пока предшествующие блоки обрабатываются процессором, позволяет существенно сократить простои процессора в ожидании данных. Успешные или неуспешные обращения к более высокому уровню памяти наз-ся соот-но попаданием или промахом. Частота попадания и промаха явл-ся одной из важнейших хар-к памяти: время обращения и время доступа. Кэш имеет совокупность строк, каждая из которых состо­ит из фиксированного количества адресуемых единиц памяти (байтов, слов) с последовательными адресами. Типичный размер строки: 16, 64, 128, 256 байтов. Наиболее часто используются три способа организации кэш-памяти: 1)кэш-память с прямым отображением; 2)частично ассоциативная кэш-память; 3)ассоциатив­ная кэш-память. Кэш память наз-ся полностью ассоц-ой, если некот блок осн памяти может располагаться в любом месте кэш памяти. И наз-ся частично ассоц-ной, если блок памяти может располагаться на ограниченном кол-ве мест. Кэш память наз-ся памятью с прямым отображением. При исп-ии кэш-памяти с прямым отображением адрес пред­ставляется как набор трех компонент, составляющих группы старших, средних и младших разрядов адреса, соответственно тега, номера строки, смещения. Ассоц-я кэш-память использует 2-хкомпонентное представле­ние адреса: группа старших разрядов трактуется как тег, а группа млад­ших разрядов - как смещение в строке. Нахождение строки в кэше определяется совпадением тега-строки со зна­чением тега адреса. По­этому при определении нахождения требуемой строки в кэш-памяти необ­ходимо сравнение тега адреса с тегами всех строк кэша. При отсутствии необходимой строки в кэш-памяти одна из его строк должна быть заменена на требуемую. Кэш-память состоит из набора ассоц-х блоков кэш-памяти. Средняя компонента адреса задает в отличие от прямо адресуе­мой кэш-памяти не номер строки, а номер одного из ассоциативных бло­ков. При поиске данных ассоциативное сравнение тегов выполняется толь­ко для набора блоков, номер, которого совпадает со средней компонентой адреса. По количеству *n* строк в наборе кэш-память называется *n*-входовой. Соответствие между данными в оперативной памяти и кэш-памяти обес­печивается внесением изменений в те области оперативной памяти, для которых данные в кэш-памяти подверглись модификации. Суще­ствует несколько способов такой модификации: 1)внесение изменений в оперативную память сразу после изменения данных в кэше. При этом процессор простаивает в ожидании завершения записи в основную память. В основной памяти под­держивается правильная копия данных кэша, и при замене строк не требу­ется никаких дополнительных действий. Кэш-память, работающая в та­ком режиме, называется памятью со сквозной записью. 2)Предполагает отображение изменений в основной па­мяти только в момент вытеснения строки данных из кэша. Если данные по адресу памяти, в кот необходимо произвести запись, находятся в кэш-памяти, то идет запись только в кэш-память. При отсутствии данных в кэш-памяти производится запись в основную память. Такой режим рабо­ты кэша получил название обратной записи. Существуют также промежуточные варианты, при которых запросы на изменение в основной памяти буферизуются и не за­держивают процессор на время операции записи в память. Эта запись вы­полняется по мере возможности доступа контроллера кэш-памяти к ос­новной памяти.

1. **Назначение и структура процессора. Система команд. Динамическая и статическая память.**

В больш-ве ЭВМ принципы Фон-Неймана реализованы в след.виде:

1)ОП организована как сов-ть машинных слов фиксир.доины, например первые ЭВМ имели разрядность 8, 16, 32, 64. Существовали 45-разрядные, 35 –разр.ЭВМ

2)ОП образует единое адресное простр-во. Адреса машин.слов возрастают от младших к старшим.

3)В ОП размещаются как данные, так и пр-мы

4) Команды выполняются в естественной посл-ти по возрастанию адресов в ОП, пока не встретится команда ветвления

5) Центр.процессор может произвольно обращаться к любым адресам в ОП для выборки или записи в машин.слова чисел и команд

Команда – описание операции, кот необх-мо выполнить.

Каждая команда начинается с кода операции(КОП), содержит необходимые адреса, хар-ся форматом, кот опр-ет структуру команды, её орг-цию, код, длину, метод расположения адресов. Длина различ.команд может быть одинаковой или разной. Команды подразделяются на арифметич., логич, ввода-выв., передачи данных.

Классы команд:

1. Класс обр-ки д-х:

1.1 короткие операции

1.1.1 логические: логич сложение, лог умножение, инверсия, лог сравление

1.1.2 арифм.: слодение операндов, вычитание, арифм сравнение

1.2 длинные операции: сложение, умн-е

2 операции управления: безусловный переход, условный пер-д

3 операции обращения к вне шустр-вам: требование на запись или чтение

типовая стр-ра 3-адресной команды: КОП-А1-А2-А3. А2 и А3 – адреса ячеек где расположены 1 и 2 числа, А1 – адрес ячейки регистра, куда необходимо поместить результат, КОП-код оп-ции

типовая стр-ра 2-адресной команды: А1 – ад-с ячейки, где хр-ся первое из чисел, участ-х в оп-ции и куда будет записан рез-т, А2-//- 2 число

типовая стр-ра 1-адресной команды: А1 – может обозн-ть адрес ячейки где хр-ся 1 из чисел, уч-х в оп-ции либо адрес ячейки для рез-та

безадресная команда сод-т КОП

Каждая команда сод-т эл-ты, определяющие:

1. Что делать(код)
2. Объекты, над кот надо что то сделать
3. Как делать(типы операндов)

Максим.длина команды 15байт

Префиксы(необязат поле – 1 байт)

- префикс замены сегмента

- префикс разрядности адреса

- префикс разрядности операнда

- пр-с повторения

Каждая команда вып-ся за 1 или неск тактов.

Послед-ть взаимосвяз-х команд именуется макрокомандой. Исп-е макрокоманд упрощает программир-е и обеспеч-ет механизм вставки добавления пр-мы.

Цикл пр-ра – это период времени, за кот осущ-ся вып-е команды исход.пр-мы в машинном виде.

Состоит из неск.тактов.

Такт работы пр-ра – период времени между соседними импульсами генер-ра тактовых импульсов.

Процедура, соответ. Такту реализуется определенной логич.цепью проц-ра и наз-ся микропрог-й.

Существуют две стратегии распределения оперативной памяти, как и любого ресурса: статическое и динамическое распределение.

При ***статическом распределении*** вся необходимая оперативная память выделяется процессу в момент его порождения. При этом память выделяется единым блоком необходимой длины, начало которого определяется базовым адресом. Программа пишется в адресах относительно начала блока, а физический адрес команды или операнда при выполнении программы формируется как сумма базового адреса блока и относительного адреса в блоке. Значение базового адреса устанавливается при загрузке программы в оперативную память. Так как в разных программах используются блоки разной длины, то при таком подходе возникает проблема фрагментации памяти, то есть возникают свободные участки памяти, которые невозможно без предварительного преобразования использовать для вычислительного процесса.

В кч-ве запоминающ эл-та исп-ся статич триггер, состояий из 4-6 транзисторов

«+» обладает большим быстродействием

«-»Плотность упаковки таких микросхем меньше пл-ти уп-ки динамич памяти, стоимость выше

При ***динамическом распределении памяти*** каждой программе в начальный момент выделяется лишь часть от всей необходимой ей памяти, а остальная часть выделяется по мере возникновения реальной потребности в ней.Такой подход базируется на следующих предпосылках.

В кач запомин эл-та исп-ся простейшая сборка, состоящая из 1 транзистора и 1 конденсатора

«+»высокая плотность интеграции(кол-во числа запом.эл-в на ед-цу пов-ти), малое потр-е энергии

«-»для того, чтобы сохранить записанную инф-ю запом.эл-т должен постоянно регенерироваться

Во-первых, при каждом конкретном исполнении в зависимости от исходных данных некоторые части программы (до 25% ее длины) вообще не используются. Следует стремиться к тому, чтобы эти фрагменты кода не загружались в ОП.

Во-вторых, исполнение программы характеризуется так называемым принципом локальности ссылок. Он подразумевает, что при исполнении программы в течение некоторого относительно малого интервала времени происходит обращение к памяти в пределах ограниченного диапазона адресов (как по коду программы, так и по данным). Следовательно, на протяжении этого времени нет необходимости хранить в ОП другие блоки программы.

При этом системные средства должны отслеживать возникновение требований на обращение к тем частям программы, которые в данный момент отсутствуют в ОЗУ, выделять этой программе необходимый блок памяти и помещать туда из внешнего ЗУ требуемую часть программы. Для этого может потребоваться предварительное перемещение некоторых блоков информации из ОЗУ во внешнюю память. Данные перемещения должны быть скрыты от пользователя и в наименьшей степени замедлять работу его программы.

Перемещение блоков информации из ОЗУ во внешнюю память с целью освобождения места для новой информации происходит обычно по одному из следующих алгоритмов:

LRU (least recently used) - наиболее давно не использовавшийся;

FIFO - самый давний по пребыванию в ОЗУ;

Random - случайным образом.

1. **Реальный режим работы процессора типа Intel-8086 и старше. Адресация памяти в реальном режиме. Виртуальный режим.**

**Реальный режим**

Первоночально IBM PC исп-ся пр-р 8086, кот имел 16-разрядные команды и мог адресовать один Мб памяти используя 20 разрядов для адреса. ПО – DOS, 1-е варианты Windows. Более поздние процессоры – Intel 80286 также могли выполнять те же самые 16-разр команды, но намного быстрее. 16-разр режим, в котором выполнялись команды пр-ра Intel 8086 и Intel 80286 был назван реальным режимом. Для ПО этого типа обычно используется однозадачный режим, т.е. выполняется только 1 программа, нет никакой встроенной защиты для предотвращения перезаписи ячеек памяти одной программы или даже ОС. Это означает что при выполнении в реальном режиме нескольких программ вполне м.б. испорчены данные или код одной из них, соот-но это может привести к краху систему или остановку.

Реальный режим процессора Intel 8086

П0 при работе в этом режиме можно разбить на лог блоки по 64 Кб (сегменты), причем каждый сегмент может и начинаться с адреса кратного 16 байтам т.о. 1-сегмент имеет нач адрес 0, 2-й адрес 16 и т.д. Это удобно при организации совместного доступа к командам и данным разл программ. Доступ к каждой ячейке памяти происходит путем указания знач регистра сегмента, кот определяет лог блок размером 64 Кб и положение этого адреса внутри лог блока(смещение). Микропроцессор исп-ет 4 сегментных регистра. Каждый регистр при этом имеет размер, равный 1-му слову(16 разр)

1-регистр сегмента команд cs code segment, указ-ий на сегмент созд-ий текущую исп-мую пр-му

2-регистр сегмента данных ds data segment, указ-ий на данные

3-регистр доп сегмента es extra segment, указ-ий на доп данные

4-регистр сегмента стека ss steck segment, указ-ий на стэк

Содержание каждого из этих регистров однозначно связано с местом в памяти соот-го сегмента. Его адрес получается приписыванием справа 0000 (4-х двоичных 0) знач-го сегмента, что соот-ет умножению на 16. Полученное 20 битовое значение, представляет собой адрес начала(базовый адрес) сегмента физ памяти. Для определения реального адреса команды или данных процессор добавляет значение смещения к базовому адресу. В реальном режиме не существует никакого механизма защиты, поэтому любая прог-ма может обратиться к любой ячейке в пределах 1 Мб, включая область экрана и область расположения ОС

Виртуальный реальный режим

В проц-х Intel 386 и старше реал-н так наз-емый режим вирт.проц-ра Intel 8086. Режим реал-ся в рамках защищенного режима, т е проц-р может переключаться в вирт режим только из защищенного. В вирт режиме пр-р способен выполнять программы составленные для проц-ра 8086, находясь в защищенном режиме и используя аппарат ср-ва защищ.режима: мультизадачность, изолирование адресных простр-в отдельных задач др от друга, страничную вирт память.

1. **Защищенный режим работы процессора типа Intel-80286 и старше.**

Процессор Intel 80286 может работать в новом защищенном режиме и соответственно выполнять ряд задач. Адресная шина этого процессора была увеличена с 20 до 24 разрядов. Это привело к расширению адресного пространства с 1 Мб до 16Мб. Новый метод адресации позволил изолировать адресные пространства отдельных задач друг от друга. При этом прикладная программа, работающая в среде ОС, использующей защищенный режим не может случайно или намерено разрушить целостность самой ОС. Основным защищаемым ресурсом является память, в которой хранятся коды, данные, различные системные таблицы. Защита памяти основывается на сегментации. Защищать требуется и совместно используемую аппаратуру, обращение к которой обычно происходит через операции ввода\вывода и прерывания. В ЗР процессор аппаратно реализует многие функции защиты, необходимые для построения многозадачной ОС, в том числе и механизма виртуальной памяти. В ЗР программа может записывать данные только в те области памяти, которые выделяет ей ОС. Изолирование адресных пространств задач исключает ситуацию, когда после запуска одним пользователем на недостаточно отлаженной программе приходится перезапускать всю систему. Следующие модели процессоров Intel -386, -486, -586 (Pentium) были 32х разрядными. Адресное пространство было увеличено до 4Гб и в них была реализована концепция строчной виртуальной памяти, возможной только в ЗР. Механизм строчной виртуальной памяти позволяет разместить часть ОП на диске, при этом размер виртуальной памяти, предоставляемой программам ограничивается размером свободного пространства на диске.

Осн преимущества:

1. Возм-ть непосредственной адресациипамяти за пределами первого Мб
2. Реализован механизм страничной вирт памяти, размер кот больше физич.ОП установленной на компе
3. Аппаратная подд-ка мультизадачности
4. **Таблица дескрипторов.**

Содержимое каждого регистра сегмента не соот-ет непосредственно какому либо участку ОП. Регистр сегмента, кот в защищ режиме наз-ют селектором становится индексом указывающим на эл-т таблицы, называемой таблицей дескрипторов. Каждый эл-т этой т-цы хар-ет 1 сегмент команд или 1 сегмент д-х. Сегмент стека отн-ся при этом к сегментам д-х, т к он не содержит исполняемых команд.

Дескриптор содержит физич адрес начала сегмента памяти, его размер, а т же др инф-цию. Базовый адрес размещается в 24 битах, поэтому сегменты теперь не обязательно должны быть выровнены на адрес кратный 16.

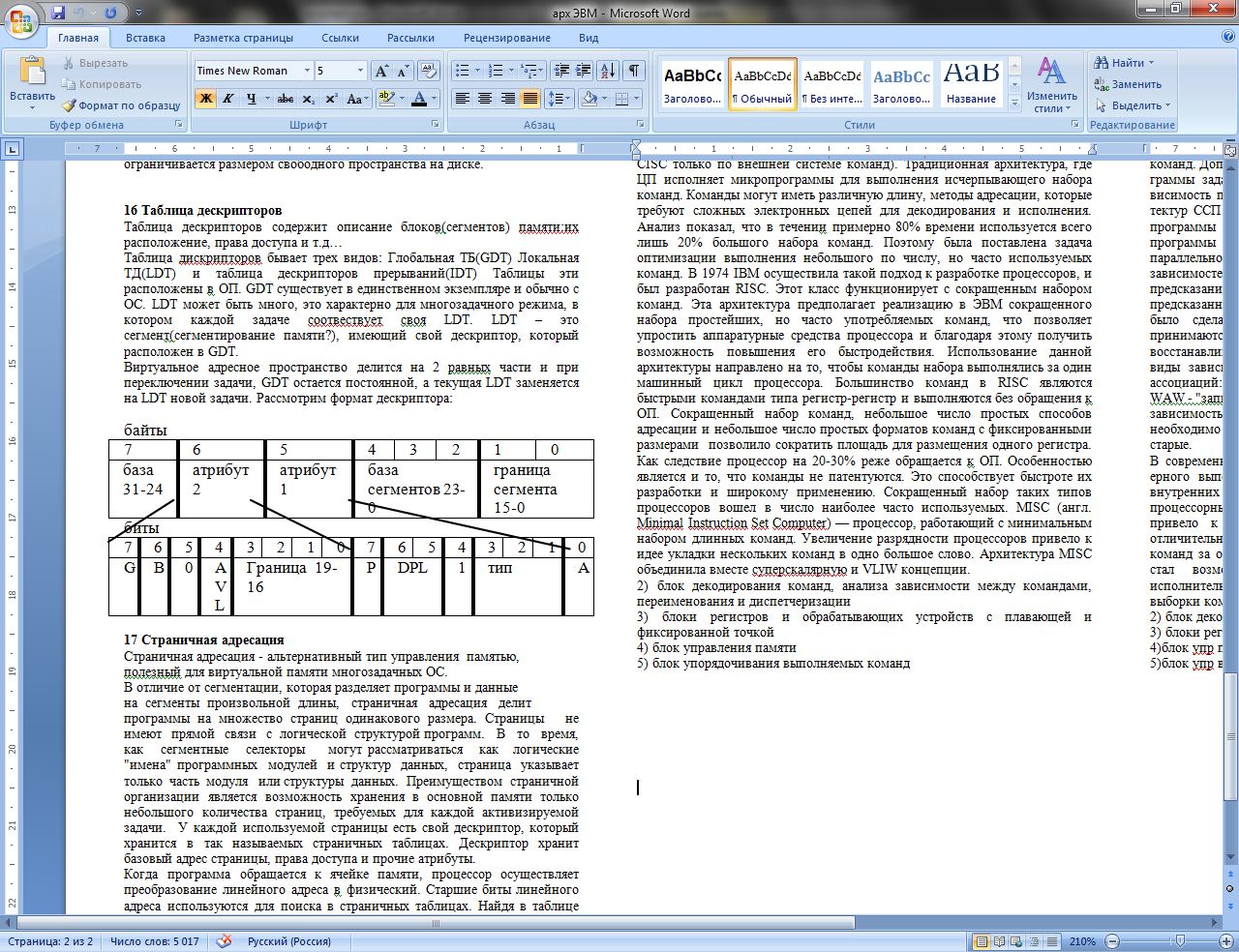
Размер сегмента указ-ся в 16 битах и может принимать любые значения от 64 кб и меньше. В этом состоит фундаментальное отличие от реального режима, где каждый сегмент по умолчанию имеет размер 64 кб. После опред-я базового адреса к нему прибавляется значение смещения, размещаемого в 24 битах и проц-р обращ-ся к соответствующей ячейке памяти.

Очевидно, что данный метод сложнее чем для реального режима, но он остается «прозрачным» для прогр-та, т к мех-м адресации в целом выглядит неизменно.

Таблица дескрипторов содержит описание блоков(сегментов) памяти:их расположение, права доступа и т.д…

Таблица дискрипторов бывает трех видов: Глобальная ТБ(GDT) Локальная ТД(LDT) и таблица дескрипторов прерываний(IDT) Таблицы эти расположены в ОП. GDT существует в единственном экземпляре и обычно с ОС. LDT может быть много, это характерно для многозадачного режима, в котором каждой задаче соотвествует своя LDT. LDT – это сегмент(сегментирование памяти?), имеющий свой дескриптор, который расположен в GDT.

Виртуальное адресное пространство делится на 2 равных части и при переключении задачи, GDT остается постоянной, а текущая LDT заменяется на LDT новой задачи. Рассмотрим формат дескриптора:



1. **Принципы магнитной и оптико-магнитной записи информации. Системы ввода-вывода.**

Одной из наиболее правильной оценок производительности системы является время ответа (время между момента ввода пользователем задания и получения им результата), который учитывает все накладные расходы, связанные с выполнением задания в системе, включая ввод-вывод.

**Система ввода-вывода** – комплекс средств обмена информации с внешними устройствами, который является важнейшей частью архитектуры процессора и ЭВМ в целом. К ней относятся:

* способы подключения к системной шине различного оборудования
* процедуры взаимодействия процессора с этим оборудованием
* команды процессора, предназначенные для обмена данными с внешними устройствами
* непосредственно устройства ввода-вывода.

Логическую схему современного компьютера можно представить в виде системной шины (магистрали) к которой подключен сам компьютер и все устройства компьютера.



*A – адресат, D – данные, M/IQ – один из сигналов управления (3 левые стрелки аналогичны всем остальным тройкам стрелок, ведущим к системной шине).*

Микропроцессор – арифметико-логическое устройство, кэш, управляющее устройство.

Системная шина представляет собой набор линий (проводов), к которым единообразно подключаются все устройства компьютера. В более широком плане понятие системной шины включает в себя электрические и логические характеристики сигналов, их назначение, а также правила взаимодействия этих сигналов при выполнении тех или иных операциях (протоколы обмена информации).

Процессор, желая записать данные по некоторому адресу в памяти, выставляет в линии адресов требуемый адрес, а по линии данных - данные. Устройство управления памятью расшифровывает поступивший адрес, и принимает с линии данных поступившие данные и заносит их в соответствующую ячейку.

*Системные и локальные шины.*

Одним из простейших механизмов, позволяющих организовать взаимодействие простых подсистем, является единственная центральная шина. Имеет 2 основных преимущества: низкая стоимость, универсальность. Основной недостаток: ограничение максимальной пропускной способности.

Причины трудностей:

* физические факторы: длина шины и количество подсоединяемых устройств.
* ограничение фактической пропускной способности.

Традиционно шины делятся на шины, обеспечивающие связь процессора с памятью и шины ввода/вывода. С целью снижения стоимости некоторые компьютеры имеют единственную шину и она называется системной. Необходимость сохранения баланса производительности по мере роста производительности микропроцессоров, привела к двухуровневой организации шины ПК на основе локальной шины.

**Локальной шиной** называется шина, электрически выходящая непосредственно на контакты микропроцессора. Она обычно объединяет процессор, память, схемы буферизации для системной шины, и её контроллер, а также некоторые вспомогательные шины.

В настоящее время используется 2 типа шин, отличающихся способом коммутации: 1) шины с коммутацией цепей 2) шины с коммутацией пакетов. Второй тип шин обеспечивает значительно большую пропускную способность по сравнению с первой за счет разделения транзакций на две логические части: запроса шины и ответа.

Накопитель на магнитных дисках (**НМД**) представляет собой набор пластин, магнитных головок, кареток, линейных двигателей плюс воздухонепроницаемый корпус. Дисковым устройством называется НМД с относящимися к нему электронными схемами.

Принцип магнитной записи электрических сигналов на движущийся магнитный носитель основан на явлении остаточного намагничивания магнитных материалов. Запись и хранение информации на магнитном носителе производится путем преобразования электрических сигналов в соответствующие им изменения магнитного поля, воздействия его на магнитный носитель и сохранения следов этих воздействий в магнитном материале длительное время, благодаря явлению остаточного магнетизма. Воспроизведение электрических сигналов производится путем обратного преобразования  
При цифровой магнитной записи в магнитную головку поступает ток, при котором поле записи через определенные промежутки времени изменяет свое направление на противоположное. В результате под действием поля рассеяния магнитной головки происходят намагничивание или перемагничивание отдельных участков движущегося магнитного носителя.

Другим направлением развития систем хранения информации являются **магнитооптические диски**. Запись на них выполняется при взаимодействии лазера и магнитной головки. Луч лазера разогревает до точки Кюри (температуры потери материалом магнитных свойств) микроскопическую область записывающего слоя, которая при выходе из зоны действия лазера остывает, фиксируя магнитное поле, наведенное магнитной головкой. В результате данные, записанные на диск, не боятся сильных магнитных полей и колебаний температуры. МО-диски уступают обычным жестким магнитным дискам лишь по времени доступа к данным.

Для достижения повышенного уровня отказоустойчивости, приходится жертвовать пропускной способностью ввода\вывода или емкостью памяти. В этой ситуации используются дополнительные диски, которые содержат избыточную информацию, позволяющую остановить исходные данные при отказе диска (например, вместо 25 дорожек используется 20, но наличие дополнительных дорожек позволяет восстановить сигнал на других дорожках).

Существует несколько способов объединения дисков **RAID**.

**RAID 1** – зеркальные диски. В этом случае все диски дублируются. Один из самых дорогостоящих.

**RAID 2** – матрица с поразрядным расслоением. Вводятся избыточные контрольные разряды для исправления одиночных и обнаружения двойных ошибок. Один диск контроля четности позволяет обнаружить одиночную ошибку, но для ее исправления требуется больше дисков.

**RAID 3** – аппаратное обнаружение ошибок и четность. Большинство контроллеров в состоянии определить, когда диск отказал. По существу, если контроллер может определить положение ошибочного разряда, то для восстановления данных требуется лишь один бит четности. Уменьшение числа контрольных дисков до одного на группу снижает избыточность емкости до вполне разумных размеров.

**RAID 4** – внутригрупповой параллелизм. RAID уровня 4 повышает производительность передачи небольших объемов данных за счет параллелизма, давая возможность выполнять более одного обращения по вводу/выводу к группе в единицу времени. В системах уровня 4 для записи небольших массивов данных используются два диска, которые выполняют четыре выборки (чтение данных плюс четности, запись данных плюс четности).

**RAID 5** – четность вращения для распараллеливания записей Система уровня 5 улучшает возможности системы уровня 4 посредством распределения контрольной информации между всеми дисками группы.

**RAID 6** – двумерная четность для обеспечения большой надежности.

1. **Обработка прерываний.**

**Прерывание** - это прекращение выполнения текущей команды или текущей последовательности команд для обработки некоторого события специальной программой - обработчиком прерывания, с последующим возвратом к выполнению прерванной программы. Событие может быть вызвано особой ситуацией, сложившейся при выполнении программы, или сигналом от внешнего устройства.

2 общих класса прер-ий:

- внутренние (инициируются сост-ем центр пр-ра или команды): деление на 0, переполнение

- внешние (иниции-ся сигналом подаваемым от др компонентов системы): запрос на обслуживание со стороны какого либо устр-ва

Переход к проц-ре прер-я осущ-ся из любой пр-мы, а после вып-я процедуры пр-я обязательно происходит процесс возврата в прерванную пр-му. Перед обращением к проц-ре пр-я должно быть сохранено сос-е всех регистров и флагов, используемых проц-й прер-я, а после окончания пр-я эти регистры должны быть восстановлены.

Пр-е вынуждает проц-р прекратить вып-е одной послед-ти команд и начать вып-е другой. При этом адрес очередной команды, кот должна была выполняться запомниться. Адрес команды, кот должна была вып-ся после возникновения пр-я выбирается из т-цы, хранящейся в начальной области памяти. Эта т-ца наз-ся т-ца векторов прер-я. В ней записано 256 адресов.

Когда устр-во вызывает пр-е пр-ра, оно сообщает ему какой адрес из т-цы следует вып-ть для перехода к новой посл-ти команд.

**Аппаратные**прерывания используются для организации взаимодействия с внешними устройствами. Запросы аппаратных прерываний поступают на специальные входы микропроцессора. Они бывают:

1. маскируемые, которые могут быть замаскированы программными средствами компьютера;
2. немаскируемые, запрос от которых таким образом замаскирован быть не может.

**Программные**прерывания вызываются следующими ситуациями:

1. особый случай, возникший при выполнении команды и препятствующий нормальному продолжению программы (переполнение, нарушение защиты памяти, отсутствие нужной страницы в оперативной памяти и т.п.);
2. наличие в программе специальной команды прерывания INT n, используемой обычно программистом при обращениях к специальным функциям операционной системы для ввода-вывода информации.

Каждому запросу прерывания в компьютере присваивается свой номер (**тип прерывания**), используемый для определения адреса обработчика прерывания.

При поступлении запроса прерывания компьютер выполняет следующую последовательность действий:

1. определение наиболее приоритетного незамаскированного запроса на прерывание (если одновременно поступило несколько запросов);
2. определение типа выбранного запроса;
3. сохранение текущего состояния счетчика команд и регистра флагов;
4. определение адреса обработчика прерывания по типу прерывания и передача управления первой команде этого обработчика;
5. выполнение программы - обработчика прерывания;
6. восстановление сохраненных значений счетчика команд и регистра флагов прерванной программы;
7. продолжение выполнения прерванной программы.

Этапы 1-4 выполняются аппаратными средствами ЭВМ автоматически при появлении запроса прерывания. Этап 6 также выполняется аппаратно по команде возврата из обработчика прерывания.

1. **Сравнительная характеристика микропроцессоров типа CISC и RISC.**

Архитектура набора команд служит границей между аппаратурой и программным обеспечением и представляет ту часть системы, которая видна программисту или разработчику компиляторов.

Двумя основными архитектурами набора команд, используемыми компьютерной промышленностью на современном этапе развития вычислительной техники являются архитектуры CISC и RISC. Основоположником CISC-архитектуры можно считать компанию IBM с ее базовой архитектурой /360, ядро которой используется с1964 года и дошло до наших дней, например, в таких современных мейнфреймах как IBM ES/9000.

Лидером в разработке микропроцессоров c полным набором команд (CISC - Complete Instruction Set Computer) считается компания Intel со своей серией x86 и Pentium. Эта архитектура является практическим стандартом для рынка микрокомпьютеров. Для CISC-процессоров характерно: сравнительно небольшое число регистров общего назначения; большое количество машинных команд, некоторые из которых нагружены семантически аналогично операторам высокоуровневых языков программирования и выполняются за много тактов; большое количество методов адресации; большое количество форматов команд различной разрядности; преобладание двухадресного формата команд; наличие команд обработки типа регистр-память.

Основой архитектуры современных рабочих станций и серверов является архитектура компьютера с сокращенным набором команд (RISC - Reduced Instruction Set Computer). Зачатки этой архитектуры уходят своими корнями к компьютерам CDC6600, разработчики которых (Торнтон, Крэй и др.) осознали важность упрощения набора команд для построения быстрых вычислительных машин. Эту традицию упрощения архитектуры С. Крэй с успехом применил при создании широко известной серии суперкомпьютеров компании Cray Research. Все они придерживались архитектуры, отделяющей команды обработки от команд работы с памятью, и делали упор на эффективную конвейерную обработку. Система команд разрабатывалась таким образом, чтобы выполнение любой команды занимало небольшое количество машинных тактов (предпочтительно один машинный такт). Сама логика выполнения команд с целью повышения производительности ориентировалась на аппаратную, а не на микропрограммную реализацию. Чтобы упростить логику декодирования команд использовались команды фиксированной длины и фиксированного формата.

Среди других особенностей RISC-архитектур следует отметить наличие достаточно большого регистрового файла (в типовых RISC-процессорах реализуются 32 или большее число регистров по сравнению с 8 - 16 регистрами в CISC-архитектурах), что позволяет большему объему данных храниться в регистрах на процессорном кристалле большее время и упрощает работу компилятора по распределению регистров под переменные. Для обработки, как правило, используются трехадресные команды, что помимо упрощения дешифрации дает возможность сохранять большее число переменных в регистрах без их последующей перезагрузки.

Развитие архитектуры RISC в значительной степени определялось прогрессом в области создания оптимизирующих компиляторов. Именно современная техника компиляции позволяет эффективно использовать преимущества большего регистрового файла, конвейерной организации и большей скорости выполнения команд. Современные компиляторы используют также преимущества другой оптимизационной техники для повышения производительности, обычно применяемой в процессорах RISC: реализацию задержанных переходов и суперскалярной обработки, позволяющей в один и тот же момент времени выдавать на выполнение несколько команд.

1. **Особенности RISC архитектуры.**

RISC - ReducedInstructionSetComputer – архитектура компьютера с сокращенным набором команд.В ходе анализа частоты выполнения тех или иных команд выяснено, что 40% команд используются крайне редко. RISC-архитектура предполагает реализацию в ЭВМ сокра­щенного набора простейших, но часто употребляемых команд, что позволяет упростить аппаратурные средства процессора и благодаря этому получить возможность повысить его быстро­действие.

Современные процессоры типа RISC характеризуются следующими особенностями:

1. упрощенный набор команд, имеющих одинаковую длину. Например, выполнение типичной команды можно разделить на этапы:
   * выборка команды
   * декодирование команды
   * выполнение операции
   * обращение к памяти
   * запоминание результата

Все команды в RISC имеют одну структуру, количество команд – обычно не более 50-100.

1. Большинство команд выполняется за 4-5 тактов процессора.
2. Отсутствуют макрокоманды, усложняющие структуру процессора и уменьшающие скорость его работы.
3. Взаимодействие с оперативной памятью ограничивается операциями пересылки данных.
4. Уменьшено число способов адресации (не используется косвенная, обычно 2-3 простых способов)
5. Используется конвейер команд.
6. Применяется высокоскоростная память.

В ЭВМ с RISC машинным циклом называют время, в течение которого производится выборка двух операндов из регистров, выполнение операции в АЛУ и запоминание результата в регист­ре. Большинство команд в RISC являются быстрыми командами типа «регистр - регистр» и выполняются без обращений к ОП. Обращения к ОП сохраняются лишь в командах загрузки регистров из памяти и запоминания в ОП. Чтобы это было возможным, про­цессор должен содержать достаточно большое число общих регистров.

Новый подход к архитектуре команд процессора значительно сократил площадь, требуемой для него на кристалле интегральной микросхемы. Это позволило резко увеличить число регистров (более 100 по лекциям, а вообще в типовых RISC-процессорах реализуются 32 или большее число регистров по сравнению с 8 - 16 регистрами в CISC-архитектурах). В результате процессор стал на 20-30% реже обращаться к оперативной памяти. Упростилась топология процессора, сократились сроки ее разработки, она стала дешевле.

Особенностью RISCархитектуры является механизм перекрывающихся окон, предназначенный для уменьшения числа обращений к оперативной памяти и межрегистровых передач, что способствует увеличению производительности ЭВМ. Процедурам динамически выделяются небольшие группы регистров фиксированной длины (регистровые окна). Окна последовательно выполняемых процедур перекрываются, благодаря чему возможна передача параметров из одной процедуры в другую. При этом не возникает необходимость передачи содержимого регистра в память.

Окно состоит из трех подгрупп регистров:

1. первая подгруппа содержит параметры, переданные данной процедуре от процедуры, вызвавшей её, и результаты для вызывающей процедуры при возврате в неё.
2. вторая подгруппа содержит локальные переменные процедуры.
3. третья является буфером для двухстороннего обмена.

В 1989 фирме Intel удалось на основе RISC-архитектуры создать однокри­стальный микропроцессор 80860, который практически представ­ляет собой кремниевый эквивалент суперЭВМCray-1.

В настоящее время многие архитектуры процессоров являются RISC-подобными, к примеру, ARM, DEC Alpha, SPARC, AVR, MIPS, POWER и PowerPC. Наиболее широко используемые в настольных компьютерах процессоры архитектуры x86 ранее являлись CISC-процессорами, однако новые процессоры, начиная с Intel 486DX, являются CISC-процессорами с RISC-ядром. Они непосредственно перед исполнением преобразуют CISC-инструкции x86-процессоров в более простой набор внутренних инструкций RISC.

1. **Архитектура суперскалярных процессоров. Предварительная выборка и предсказание переходов.**

Суперскалярная архитектура – Способность выполнения нескольких машинных инструкций за один такт процессора. Появление этой технологии привело к существенному увеличению производительности. Основная идея, определяющая развитие суперскалярных микропроцес­соров, состоит в построении возможно большего количества парал­лельных структур при сохранении традиционных последовательных про­грамм. Это означает, что компиляторы и аппаратура микропроцессора сами, без вмешательства программиста, обеспечивают загрузку параллель­но работающих функциональных устройств микропроцессора. В соответствии с моделью последовательного программирования, про­граммы пишутся в предположении, что команды будут выполнены в том же порядке, в каком они представлены в программе. Однако с целью достиже­ния большей эффективности современные процессоры пытаются выполнять несколько команд одновременно и в некоторых случаях в порядке, отличном от их исходной последовательности в программе. Это переупорядочение мо­жет быть выполнено в трансляторе и (или) в аппаратных средствах во время выполнения. Суперскалярные и VLIW-процессоры принадлежат классу ар­хитектур, которые используют параллельность уровня команды (ILP). ILP-процессоры и компиляторы обычно преобразуют полностью упо­рядоченное множество команд исходной программы в частично упорядо­ченное множество, структурированное зависимостями по данным и управ­лению. Зависимости по управлению (которые проявляются как переходы по условию) представляют главное препятствие высокопараллельному выполнению потому, что эти зависимости должны быть установлены пре­жде, чем будут выполнены все последующие команды. Текст последовательной программы, представленной на языке высо­кого уровня, компилируется в машинный код, отражающий статическую структуру программы, т. е. упорядоченное множество команд (инструк­ций) в памяти компьютера. Процесс выполнения программы с конкрет­ными наборами входных данных может быть представлен динамической структурой программы, т. е. множеством последовательностей инструк­ций в порядке их исполнения. Повысить степень параллелизма программы можно изменяя соответст­вующим образом ее статическую или динамическую структуру. Поскольку статическая структура программы однозначно соответствует ее исходному тексту (в предположении неизменности компилятора), то изменение статиче­ской структуры сводится к изменению исходного кода, что, в общем случае, не всегда возможно. Динамическая же структура программы может быть из­менена при неизменной статической структуре. И главной целью такого из­менения должно быть повышение степени параллельного исполнения команд. Допустимые границы преобразования динамической структуры про­граммы задают существующие на множестве инструкций отношения: за­висимость по управлению и зависимость по данным. При описании архи­тектур ССП часто используется модель окна ис­полнения. При исполнении программы микропроцессор как бы продви­гает по статической структуре программы окно исполнения. Команды в окне могут исполняться параллельно, если между ними нет зависимости. Для устранения зависимостей, вызванных командами переходов, исполь­зуется метод предсказания, позволяющий извлекать и условно исполнять команды предсказанного перехода. Если позднее обнаруживается, что пред­сказание было сделано, верно, то результаты условно исполненных команд принимаются. Если предсказание было ошибочным, состояние процессора восстанавливается на момент принятия решения о выполнении перехода. Все виды зависимостей по данным могут быть классифицированы по типу ассоциаций: RAR - "чтение после чтения", WAR - "запись после чте­ния" и WAW - "запись после записи", RAW - "чтение после записи". Действительной зави­симостью является только "чтение после записи" (RAW), так как необходи­мо прочитать предварительно записанные новые данные, а не старые.

1. **Архитектура ЭВМ с длинным командным словом.**

Арх-ра процессоров с очень дл.ком.словом VLIW позволяет сократить обем оборуд-я, требуемого для паралл. выдачи неско-х команд.

потенциально чем больше кол-во команд выдается паралл-но, тем больше эта экономия. Базируется на множестве независ-х функц-х уст-в.Вместо того, чтобы пытаться паралл-но выдавать независ-е команды, в таких машинах неск-ко операций упаковываются в одну оч.длинную команду. (например: vliw-команда может включать в себя 2 целочисленные операции, 2 оп-ции с плавающ. точкой, 2 оп-ии обращения к памяти, и оп.перехода). При этом ответственность за паралл-но выполненные оп-ии полностью ложится на компилятор, а аппар-е ср-ва, необх-е для реал-ции суперскалярной обр-ки отсутствуют.

Для машин с VLIW арх-рой был разр-н новый метод планирования выдачи команд – трассировочное план-е. При исп-ии этого м-да из последовательности исх. программы генер-ся длинные команды путем просмотра пр-мы за пределами базовых блоков.

Рес-сы VLIW проц-ра не резервируются. Аппарат.рес-сы, предоставленные компилятору планируются статически. К ним относ-ся конвейерные функц. Устр-ва, шины и банки д-х.

Для высокой пропускной спос-ти м-ду функц. Устр-вами необх-мо использовать нес-ко наборов регистров.

В машинах типа VLIW принцип замены упр-я во время вып-я пр-мы план-я распространен и на систему памяти. Для поддержания занятости конвей. Функц. Уср-в должна быть обеспечена высокая пропускная спос-ть памяти. Одним из совр. Подходов к увел-ю памяти явл. расслоение памяти.

1. **Основные классы современных параллельных компьютеров. MPP, SMP, NUMA, PVP.**

Основным параметром классификации параллельных компьютеров является наличие общей (SMP) или распределенной памяти (MPP). Нечто среднее между SMP и MPP представляют собой NUMA-архитектуры, где память физически распределена, но логически общедоступна. Кластерные системы являются более дешевым вариантом MPP. При поддержке команд обработки векторных данных говорят о векторно-конвейерных процессорах, которые, в свою очередь могут объединяться в PVP-системы с использованием общей или распределенной памяти. Все большую популярность приобретают идеи комбинирования различных архитектур в одной системе и построения неоднородных систем.

При организациях распределенных вычислений в глобальных сетях (Интернет) говорят о мета-компьютерах, которые, строго говоря, не представляют из себя параллельных архитектур.

**Массивно-параллельные системы (MPP)**

|  |  |
| --- | --- |
| **Архитектура** | Система состоит из однородных *вычислительных узлов*, включающих:   1. один или несколько центральных процессоров (обычно RISC), 2. локальную память (прямой доступ к памяти других узлов невозможен), 3. коммуникационный процессор или сетевой адаптер 4. иногда - жесткие диски (как в SP) и/или другие устройства В/В   К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.) |
| **Примеры** | IBM RS/6000 [SP2](computers.html#sp2), Intel PARAGON/ASCI Red, SGI/CRAY [T3E](computers.html#crayt3e), Hitachi [SR8000](/parallel/news/hitachi_sr8000f1.html), транспьютерные системы [Parsytec](vendors.html#parsytec). |
| **Масштабируемость** | Общее число процессоров в реальных системах достигает нескольких тысяч (ASCI Red, Blue Mountain). |
| **Операционная система** | Существуют два основных варианта:   1. Полноценная ОС работает только на управляющей машине (front-end), на каждом узле работает сильно урезанный вариант ОС, обеспечивающие только работу расположенной в нем ветви параллельного приложения. Пример: Cray T3E. 2. На каждом узле работает полноценная UNIX-подобная ОС (вариант, близкий к [кластерному](#cluster) подходу). Пример: IBM RS/6000 SP + ОС AIX, устанавливаемая отдельно на каждом узле. |
| **Модель программирования** | Программирование в рамках модели **передачи сообщений** ( [MPI](/parallel/tech/tech_dev/mpi.html), [PVM](/parallel/tech/tech_dev/ifaces.html#pvm), [BSPlib](/parallel/tech/tech_dev/ifaces.html#bsplib)) |

**Симметричные мультипроцессорные системы (SMP)**

|  |  |
| --- | --- |
| Архитектура | Система состоит из нескольких однородных процессоров и массива общей памяти (обычно из нескольких независимых блоков). Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины (базовые 2-4 процессорные SMP-сервера), либо с помощью crossbar-коммутатора (HP 9000). Аппаратно поддерживается когерентность кэшей. |
| Примеры | [HP 9000 V-class](computers.html#exemplar), N-class; SMP-cервера и рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu и др.). |
| Масштабируемость | Наличие общей памяти сильно упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число - не более 32 в реальных системах. Для построения масштабируемых систем на базе SMP используются [кластерные](#clusters) или [NUMA](#numa)-архитектуры. |
| Операционная система | Вся система работает под управлением единой ОС (обычно UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы/нити по процессорам (scheduling), но иногда возможна и явная привязка. |
| Модель программирования | Программирование в модели общей памяти. (POSIX threads, [OpenMP](/parallel/tech/tech_dev/openmp.html)). Для SMP-систем существуют сравнительно эффективные средства [автоматического распараллеливания](/parallel/tech/tech_dev/auto_par.html). |

**Системы с неоднородным доступом к памяти (NUMA)**

|  |  |
| --- | --- |
| **Архитектура** | Система состоит из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. При этом доступ к локальной памяти в несколько раз быстрее, чем к удаленной.  В случае, если аппаратно поддерживается когерентность кэшей во всей системе (обычно это так), говорят об архитектуре **cc-NUMA** (cache-coherent NUMA) |
| **Примеры** | HP [HP 9000 V-class](computers.html#exemplar) в SCA-конфигурациях, SGI [Origin2000](computers.html#origin2000), Sun [HPC 10000](computers.html#starfire), IBM/Sequent [NUMA-Q 2000](computers.html#numa-q), SNI [RM600](computers.html#rm600). |
| **Масштабируемость** | Масштабируемость NUMA-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддежки когерентности кэшей и возможностями операционной системы по управлению большим числом процессоров. На настоящий момент, максимальное число процессоров в NUMA-системах составляет 256 (Origin2000). |
| **Операционная система** | Обычно вся система работает под управлением единой ОС, как в [SMP](#smp). Но возможны также варианты динамического "подразделения" системы, когда отдельные "разделы" системы работают под управлением разных ОС (например, Windows NT и UNIX в NUMA-Q 2000). |
| **Модель программирования** | Аналогично [SMP](#smp). |

**Параллельные векторные системы (PVP)**

|  |  |
| --- | --- |
| **Архитектура** | Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах.  Как правило, несколько таких процессоров (1-16) работают одновременно над общей памятью (аналогично [SMP](#smp)) в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора (аналогично [MPP](#mpp)). |
| **Примеры** | NEC SX-4/[SX-5](computers.html#sx-5), линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/[T90](computers.html#crayt90), [CRAY SV1](computers.html#sv1), серия Fujitsu [VPP](computers.html#vpp700). |
| **Модель программирования** | Эффективное программирование подразумевает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением). |

**Моделирование систем**

1. **Понятие имитационного моделирования. Формальная модель объекта. Типовые математические схемы моделирования.**

**Модель** – любое мысленное формальное физическое представ-е объекта окружающего мира обеспечивающее изучение определённых его св-в. **Мод-е** - процесс созд-я модели. **Система** – группа объектов, объед-ых к-л формой взаимод-я с целью вып-я опред задач. **Динамич с-мы** – сост-е с теч времени мен-ся непрерывно.

Методы мод-я: **1) аналитические** – позвол получить хар-ки с-мы как некот ф-ции параметров ее функ-вания. Аналит модель – с-ма дифф ур-ий. **2) имитационные** - представ-е динамич поведения с-мы посредством продвиж-я ее от одного сост-я к др в соотв-вии с опред-ми имитац-ми правилами. Шеннон: **Критерии прим-я** им-го мод-я: 1) не сущ-ет законченной матем постановки задачи 2) аналитич методы имеются, но настолько сложны и трудоемки, что рациональнее исп-ать им мод-е 3) кроме оценки опред-х парам-ов необх-мо наблюдение за ходом процесса.

**Проблемы применения им мод**: 1) нахождение компромисса м\д сложностью и упрощением модели 2) искусственном воспроизводство случ воздействий окр среды 3) оценка адекватности модели.

**Математические модели систем**

На базе мат модели происх анализ хар-ик с-мы, при комп-ом мод-ии на основе мат модели созд-ся алгоритм программ д\получ-я инф-ии о поведении с-мы, **формальное описание объекта.**

В общем сл мат модель любой динамической с-мы м\представить в виде: 

**x**- совок-сть входных воздействий, **h**- совок внутр-х параметров с-мы, **y**- совок выходных хар-ик, **t**– время, **F** - закон ф-ния. Процесс функц-я с-мы м\рассм-ать как послед-ную смену сост-ий

:

,

где z- совок начальных состояний.

Т.о, **общую мат модель** с-мы м\представить: .

При построении мат м\выделить осн подходы: **1)** непрерывно-детерминиров – исп-ет в кач-ве мат моделей с-мы дифф ур-ий (сост-е с-мы изм-ся с теч вр по непрерыв-му строго опред закону: колебания маятника) **2)** дискретно-детерм – с-ма мен сост-я по опред закону, h – конечное число сост-ий. Реализ-ся с пом мат аппарата теории автоматов (д\упр-я быт тех) **3)** дискретно-стохастический – вероятностный автомат (д\анализа надежности ИС) **4)** непрерывно-стох – с-ма мен сост-я с теч вр по случ закону (СМО) **5)** сетевой (сети Петри) 6) агрегативный подход(универсальный).

1. **Непрерывно-стохастические модели (Q-схемы).**

С-ма меняет свои сост-я с теч вр по случ закону (СМО).

Типовой мат схемой мод-я таких с-м явл Q-схемы. В обслуживании м\выделить 2 составляющие: ожидание обсл-я и обсл-е, а в любой СМО м\выделить элементарный прибор. В нем выделяют накопитель, некоторой емкостью; канал; потоки событий: поток заявок на обслуж-е wi, хар-щийся моментами поступления и атрибутами заявок, и поток обслуж-я ui, хар-щийся моментами начала и окончания обслуж-я заявок. Рис.

Различают: **поток однородных** событий (хар-ся только моментами поступления этих событий и задается послед-стью {tn }={0<= t1<= t2 ... <= tn <=...}, где tn - момент наступления n-го события. **Потоком неоднородных** событий (последовательность {tn,fn }, где fn - набор признаков события).

**Допущения**

**Ординарный поток** P>1(t, ) << P1(t, ). Для любого интервала  верно следующее:

P0(t, )+ P1(t, )+ P>1(t, )=1

**Стационарный поток** - поток, для кот вер-ть появления того или иного числа событий зав от длины интервала и не зав от того, где на оси времени взят этот интервал.

Среднее число событий, наступающих на интервале врем :

0·P0(t, )+1·P1(t, ) = P1(t, )

Тогда ср число событий орд-го потока в ед времени (**интенсивность потока**):



Для стационар потока a(t)=a= const/

Процесс ф-ния прибора м\представить как процесс изм-я сост-ий его элем-ов во времени z(t). Вектор сост-ий д\прибора имеет вид , где zн – сост-е накопителя, zк – сост-е канала.

Неоднородность заявок, отражающая процесс в реальной с-ме, учитывается с пом классов приоритетов. Различа статические (назначаются заранее) и динамические (возникают при мод-ии). **Относительный приоритет** - заявка с более выс приоритетом, поступившая в Н, ожидает окончания обслуж-я предшествующей заявки и только после этого занимает канал. **Абсолютный приорит**ет - заявка с более выс приоритетом, поступившая в Н, прерывает обслуж-е заявки с более низким приоритетом и занимает канал.

1. **Сущность метода статистического моделирования. Примеры использования.**

**Основа** - метод статистических испытаний Монте-Карло, кот базир-ся на исп-ии случ чисел, т.е возможных знач-ий некот случ вел-ны с заданным распред-ем вер-тей. **Сущность** метода: построение для процесса функц-я исследуемой с-мы некот-го моделирующего алг-ма, имитирующего поведение и взаимод-е эл-ов с-мы с учетом случ-х входных воздействий и воздействий внеш среды и реализация этого алг-ма на ЭВМ.

**Обл прим-я**: 1) изучение стохаст-х с-м; 2) решение детерминированных задач.

Рез-тат статист-го мод-я - серия частных знач-ий искомых величин или ф-ий, их статистическая обраб-ка. Если кол-во реализаций N → ∞, рез-ты устойчивы и достаточно точны.

**Пример1** (детерм). S фигуры.





1. **Генерация случайных чисел. Генерация последовательностей псевдослучайных чисел. Требования к генератору псевдослучайных чисел. Улучшение качества последовательностей.**

3 способа генерации случайных чисел:

**1)Аппаратный** - в осн лежит к-л физич эффект. «-» не гарант-ет кач-во послед-сти сл чисел во время мод-я; нельзя получать одинак послед-сти.

**2)Табличный** - случ числа оформлены в виде табл. «-» запас чисел ограничен, выч ресурсы исп-ся неэффект.

**3)Программный** (алгоритмический) – случ числа формир-ся с пом спец прог. Генерир-е некот базовых процессов и их последующее преобраз-е. Чаще всего в кач-ве базовой послед-ти исп-ют независимые случ величины, равномерно распред-ные на (0,1).**Рис**

Для получения случ чисел на ЭВМ исп-ся алг-мы, поэт такие последоввт-сти, являющиеся по сути детерминированными, назыв **псевдослучайными**.

ЭВМ оперирует n-разрядными числами, поэтому вместо непрерывной совок-ти равномерных случ чисел интервала (0,1) исп-ют дискретную послед-ть 2n случ чисел того же интервала - закон распред-я такой дискретной послед-ти назыв квазиравномерным распред-ем.

**Требования к идеальному генератору:**

1)Послед-ть д\состоять из квазиравномерно распред-х чисел 2)Числа д\б независимы 3)Послед-ти случ чисел д\б воспроизводимыми 4)Послед-ти д\иметь неповторяющиеся числа 5)Минимал затраты вычислительных ресурсов.

Наиб прим-е на ЭВМ д\генерации послед-тей псевдослучайных чисел находят алг-мы вида: xi+1=Ф(xi), представ собой реккурентные соотношения 1го порядка.

«-» наличие коррелляции м\д числами послед-ти, иногда случ-сть отсутствует.

Прим-ся конгруэнтные процедуры генерации псевдослуч-х послед-тей.

2 целых числа конгруэнтны (сравнимы). Осн на формуле: 

где  - неотрицат целые числа.

**Методы улучшения качества:**

1)Исп-е рекуррентных формул большего порядка r:



2)Метод возмущающих ф-ций:



1. **Моделирование случайных воздействий (моделирование случайных событий). Моделирование дискретной случайной величины.**

**1. Необх.реализовать случ.событие А, наступающее с заданной вероятностью p**: Определим А как событие, состоящ.в том, что выбран.знач-е xi равномерно распределенной на интервале (0,1) случ.величины удовлетворяет неравенству: xi ≤ p. Тогда вер-сть события А будет P(A) = ∫[0..p](dx) = p; Противоположн.событие состоит в том, что xi >p, его вер-сть равна 1-р.

**2. Рассм. группу событий**: Пусть А1, А2 ,..., Аs - полная группа событий, наступающ.с вер-стями p1, p2 ,..., ps. Определим событие Аm как событие, состоящ. в том, что выбран. Знач-е xi случ. величины удовлетв. Нер-ву: lm-1 < xi ≤ lm; где lr = ∑[i=1..r](pi);

Процедура моделир-я испытаний в этом случае состоит в последоват. сравнении случ.чисел xi со значениями lr. Если условие выполн-ся, исходом испытания оказыв-ся событие Аm .

**3. Рассм. независимые события А и В с вер-стями наступления рА и рВ:** Возможными исходами совмест.испытаний в этом случае будут события АВ, ]AB, A]B,]A]B с вер-стями рА∙рВ, (1-рА)∙рВ, рА∙(1-рВ), (1-рА)∙(1-рВ). Для моделир-я совместн. испытаний можно исп-ть два вар-та процедуры: \* Последоват.е выполнение процедуры.\* Определ-е одного из исходов АВ, ]AB, A]B,]A]B по жребию с соответствующими вер-стями.

**4. События А и В явл-ся зависимыми и наступают с вер-стями pА и pВ :** Обозначим через pА(В) условную вер-сть наступления события В при условии, что событие А произошло. Алгоритм модели может быть:

[Пуск] → [Генерация] → <[xi < pA]> → (1+ or 2-) →

(1+) → [KA = KA+1] → [Генерация] → (1a+ or 1b-) →

(1a+) → [KAB=KAB+1] → [выход]

(1b-) → [KANB=KANB+1] → [выход]

(2-) → [KNA=KNA+1] → [генерация] → <[xi < pNA(B)]> → (2a+ or 2b-) →

(2a+) → [KNAB=KNAB+1] → [выход]

(2b-) → [KNANB=KNANB+1] → [выход];

**Моделир-е дискретных случ.величин:**

Дискрет. Случ.вел-на Y принимает знач-я y1 ≤ y2 ≤ … ≤ yj ≤ … с вер-стями p1, p2, …, pj,… составляющими дифференц. Распредел-е вер-стей.

Для получения дискрет. Случ. величин можно воспольз-ся методом обратных функций: если X – равномерно распред. на интервале (0,1) случ. величина, то искомую случ. величину получают при выполнении действий:

Если x1 < p1 , то Y= y1 , иначе,

Если x1 < p1 + p2, то Y= y2 , иначе, ……..

Если x1 < ∑[j=1..m]Pj; то Y= ym;

1. **Моделирование непрерывных случайных величин (метод обратных преобразований, показательный закон, треугольный закон распределения).**

Для получения НСВ (непрерывных случайных величин) с заданным законом распределения можно использовать **метод обратной ф-ии**. Если СВ (случайная величина) Y имеет плотность распределения f(y), то распределение СВ: F(y)=∫[0..y]f(y)dy; явл-ся равномерным на интервале (0,1). Чтобы получить число, принадлежащее последоват-ти СЧ (случайных чисел) {yi}, имеющих ф-ю плотности f(y), надо разрешить относительно **yi** ур-е: xi=∫[0..y]f(y)dy;

где xi - число, принадлежащее последовательности СЧ равномерно распределенных на интервале от (0,1).

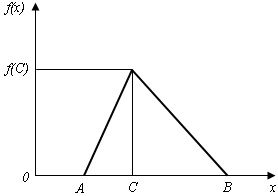
**Показательный закон распределения**: Необходимо получить случайные числа с показательным законом распределения (напр-р, интервалов времени м\ду поступлениями заявок на обслуживание): f(t)=λ∙e-λt; xi=∫[0..t](λ∙e-λt)dt; t=-(1/λ)∙ln(xi);

Этот способ получения случ-ых чисел с заданным законом распределения имеет ограниченную сферу применения, т.к. для многих законов распределения, встречающихся в практических задачах моделирования, интеграл не берется, => приходится прибегать к численным методам реш-я, что увеличивает затраты вычислительных ресурсов на получение каждого числа; Поэтому на практике пользуются приближенными способами преобраз-я СЧ, кот-е делят на:

а) универсальные способы, с пом-ю кот-х можно получать СЧ с законом распределения любого вида; б) неуниверсальные способы, пригодные д\получения случ-ых чисел с конкретным законом распределения.

**Треугольное распределение:** применяется когда о случайной величине ничего неизвестно, кроме наиболее вероятного значения и диапазона возможных значений этой случайной величины

Для получения последовательностей СЧ, подчиненных треугол-му распределению исп-ся метод обратных функций.



1. **Моделирование непрерывных случайных величин (универсальный метод (кусочная аппроксимация), нормальный закон распределения).**

**Показат.з-н распредел-я**: Необх.получить случ. числа с показат. З-ном распредел-я (например, интервалов времени между поступлениями заявок на обслуживание): f(t)=λ∙e-λt; xi=∫[0..t](λ∙e-λt)dt; t=-(1/λ)∙ln(xi);

Этот способ получения случ. чисел с заданным з-ном распредел.имеет огранич.сферу применения, т.к. для многих з-нов распредел-я, встречающ. в задачах моделир-я, интеграл не берется, т.е. приходится прибегать к числен. методам решения, что увелич. затраты вычислит. ресурсов на получение каждого числа; на практике пользуются приближен.способами преобразов-я случ. чисел, кот.можно классифицировать:

а) универс. способы, с пом. которых можно получать случ. числа с з-ном распредел-я любого вида; б) неуниверсальн. способы, пригодные для получения случ. чисел с конкретным з-ном распредел-я.

Рассм.приближ.универс.способ получения случ. чисел, основанный на кусочной аппроксимации ф-ии плотности.

Пусть треб-ся получить послед-сть случ.чисел {yj} с ф-ей плотности fη(y) , знач-я кот. лежат в интервале (a,b). Разобьем интерв. (a,b) на m интервалов, и будем считать fη(y) на каждом интерв.постоянной. Разбивать необх-мо так, чтобы вер-сть попадания случ.вел-ны в любой интервал (ak, ak+1) была постоянной, т.е.: ∫[ak…ak+1]fn(y)dy;

алгоритм этого способа получения случ. чисел сводится к выполн-ю след. действий:

1) Генерир-ся случ.равномерно распредел. число xi из интервала (0,1);

2) с пом. этого числа выбир-ся интерв. (ak, ak+1);

3) генерир-ся число xi+1 ;

4) вычисл-ся случ. число yj =ak+ xi+1(ak+1-ak) с требуемым з-ном распредел-я.

Вид норм.распред-я

В современ. системах имитац.моделир-я обычно исп-ся не менее двух програм. датчи­ков случ.величин, распред.по норм. Закону.



1. **Управление модельным временем (принцип Δt и принцип δz).**

При создании модели важным явл-ся реализация 2-х функций: 1) корректировка временной координаты состояния сист-ы ("продвижение" времени, организация "часов"); 2) обеспечение согласованности различных блоков и событий в сист-е (синхронизация во времени, координация с др. блоками). Т.о., функционирование модели должно проте­кать в искусственном (не в реальном и не в машинном) времени, обеспечивая появление событий в требуемом логикой работы ис­следуемой системы порядке и с надлежащими временными интерва­лами м\ду ними.

1) **принцип Δt (квантования времени).** Состояние с-мы хар-ся вектором сост-ий Z(t). Организуем счетчик систем-го времени, кот в начальный момент показ время t0. Прибавим интервал врем Δt, тогда счетчик будет показывать t1= t0 +Δt. Вычислим знач-я Z(t0 +Δt), затем перейдем к мом врем t2= t1 +Δt и т.д. Если шаг Δt достаточно мал, то таким путем можно получать приближенные знач-я Z.

**2) принцип δz.** При рассм процессов функц-ния некот сист-м можно обнаружить, что для них характерны 2 типа состоя­ний: 1) особые, присущие процессу ф-ия с-мы то­лько в некоторые мом-ы времени 2) неособые, в кот-ых процесс находится всё остальное время. Особые состояния хар-ны еще и тем, что функции сост-ий Z(t) в эти мом врем изм-ся скач­ком, а м\ду особыми сост-ми изменение координат Z(t) происходит плавно и непрерывно или не происх вообще. Для описанного типа сист-м м\б построены моделирующие алг-мы по «принципу особых состо­яний». Обозначим скачкообразное изм-ие сост-я z как δz. Хар-ки процесса функц-я таких сист-м оцениваются по инф-ии об особых сост-ях, а неособые состояния при мод-ии не рассм-ся.

1. **Механизм протяжки модельного времени.**

1 Время реальной системы – время в которой живет и функционирует реальная система.

2 Модельное время – искусственное время в кот-ом живет модель, или время которое явл-ся имитационным временем реальной сист-ы.

3 Реальное время – время в кот-ом живет исследователь и компьютер; время необходимое для моделирования.

В моделирующих сист-ах выделяют 2 списка событий: cписок текущих событий (СТС) и список будущих событий (СБС). Каждое событие ассоциируется с динамическим объектом. В СТС входят все события, запланированные на текущий момент модельного времени. Прог-а управления моделированием в 1-ую очередь смотрит этот список и старается переместить по модели те динамич-е объекты, д\кот-ых выполнены условия. Если в списке таких объектов нет, процесс управления моделированием обращается к СБС. Он переносит все события, кот-е запланированы на ближайший мом-т модельного времени, из этого списка в СТС и повторяет его просмотр. При моделировании модель­ное время может меняться быстрее или медленнее, чем в реальной сис­т-е. Это зависит от степени детализации модели и сложности описа­ния изучаемого процесса. Модельное время изменяется при выполнении некот-ых событий, а события в сист-е моделирова­ния возникают в результате перемещения динамических объектов. Причиной изме­нения модельного времени может послужить явная задержка динамического объекта на некот-ый отрезок модельного времени.

Упрощенная схема протяжки модельного времени.

1- Начало моделирования;

2- СТС пуст?

3- Продвижение активного объекта в модели;

4- Движение активного объекта прекращено;

5- Извлечение динамического объекта из СБС с ближайшим временем активизации;

6- Изменение текущего модельного времени;

7- Поместить все объекты со временем выхода, равным тек. модельному времени, из СБС в СТС;

8- Конец моделирования?

9- Нормальное завершение моделирования.

2

3

1

9

завершение моделирования

6

7

4

5

8

моделиро-вания?

1. **Объектно-ориентированная моделирующая система.**

В большинстве случаев с помощью имитационных моделей исследуются характеристики и поведение системы на определенном отрезке времени, поэтому для моделирующей системы необходимым является реализация **двух функций**:

1. предоставление средств для формализованного описания системных компонентов, дисциплин выполнения различных работ, для задания структуры модели, привязки объектов модели к временной и пространственной координате;
2. осуществление координации событий, определение путей прохождения динамических объектов модели, изменение состояний узлов и передачу управления между блоками модели.

Существует несколько основных понятий, являющихся общими для большинства современных моделирующих систем:

1. **Граф модели.** Все процессы независимо от количества уровней структурного анализа, объединяются в виде направленного графа (многослойный иерархический).
2. **Динамические объекты.** Это объекты, моделирующие формальный запрос на какое-либо обслуживание.

Динамический объект может выполнять следующие действия:

* Порождать группы (семейства) других динамических объектов (GPSS, Split);
* Поглощать другие динамические объекты конкретного семейства;
* Захватывать ресурсы и использовать их некоторое время, а затем освобождать;
* Определять времена обслуживания, накапливать информацию о пройденном пути и иметь информацию о своем дальнейшем пути и о путях других динамических объектов.

Основные параметры динамических объектов:

* Уникальный идентификатор объекта;
* Идентификатор (номер) семейства, к которому принадлежит объект;
* Наборы различных ресурсов, которые динамический объект может захватывать и использовать какое-то время;
* Время жизни динамического объекта;
* Приоритет – неотрицательное число;
* Параметры обслуживания в каком-либо обслуживающем устройстве.

1. **Узлы графа модели** представляют собой центры обслуживания динамических объектов. В узлах динамические объекты могут задерживаться, обслуживаться, порождать семейства новых динамических объектов, уничтожать другие динамические объекты. Динамический объект всегда принадлежит одному из узлов графа модели.
2. **Событие** – факт входа или выходы из узла одного динамического объекта. Функция управления событиями отдана специально управляющей программе – координатору, автоматически внедряемому в состав модели.
3. **Ресурс** независимо от его природы в процессе моделирования может характеризоваться тремя общими параметрами: мощностью, остатком и дефицитом. Мощность – это макс. число ресурсных единиц. Остаток ресурса – число незанятых на данный момент единиц. Дефицит ресурса – количество единиц ресурса в суммарном запросе динамических объектов, стоящих в очереди к данному ресурсу.
4. **Пространство** – географическое, декартова плоскость. Узлы, динамические объекты и ресурсы могут быть привязаны к точкам пространства и мигрировать в нем.

Программная реализация моделирующих систем использует объектно-ориентированный способ представления процессов. Динамические объекты (транзакты), узлы, события и ресурсы являются основными объектами имитационной модели.

1. **Событийный и процессно-ориентированный подход к построению моделей.**

Событие – послед-сть логически связанных действий, происходящих в некот фиксированный момент мод-го времени. Появление события связ с изм-ем сост-я модельной среды.

**Событийный** подход осн на формировании потока событий. Такой поток образует сгруппиров. Последоват-сти действий:



Группы событий последоват-но, происх-щих в один момент вр, условно заключены в скобки.

Языки, реализующие **процессно-ориентир.подход**, имеют в своем составе блоки или операторы, позволяющие описать процесс продвижения компонентов ч\з с-му. В моделях, исп-щих подобную схему, описыв-ся не события и условия их возник-я, а процесс, развивающ-ся в ней.

1-создавать транзакты 20-25 ед мод времени; → 2-занести в 1-ый параметр тр-та знач-я; → 3-отправить на обраб-ку; → (А)

4-создавать транзакты 25-30 ед мод времени; → 5=2; → 6=3; → (А)

(А) = занять очередь → 7-занять прибор обраб-ки; → 8-задержаться на время обраб-ки; → 9-освободить прибор; → 10-уничтожить активный транзакт;

Оба подхода имеют как достоинства, так и недостатки. К достоинствам процессно-ориентир.представления моделей следует отнести компактность и наглядность.

Событийн. модели обладают большей гибкостью, но они уступают процессно-ориентиров.системам в простоте и наглядности составления моделей.

**Агентное моделирование-**метод , исследующ. поведение децентрализованных [агентов](http://ru.wikipedia.org/wiki/%D0%90%D0%B3%D0%B5%D0%BD%D1%82) и то, как такое поведение определяет поведение всей системы в целом. В отличие от [системной динамики](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B4%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D0%BA%D0%B0) аналитик определяет поведение агентов на индивид. уровне, а глобальное поведение возникает как результат деят-сти множ-ва агентов (моделир-е «снизу вверх»).

Агентное моделир-е вкл. в себя элементы теории игр, сложных систем, мультиагентных систем и эволюционного программир-я, методы Монте-Карло, использует случ. числа

1. **Инструментальные средства моделирования.**

Универс. Инструм.ср-вом создания моделей яв-ся языки программир. общего пользования (Pascal, C/C++ и др.). Кроме того, сущ-ет множ-во специализир.средств моделир-я, позволяющих быстрее и с меньшими затратами создавать и исследовать модели. В развитии специализир. Ср-в можно выделить **два направления**: **1.** Ср-ва моделир-я для анализа достаточно широкого класса систем: языки имитац.моделир-я (GPSS, SIMSCRIPT и др.), пакеты приклад. программ, использующих для моделир-я аналитич. методы, такие как MathCad, MathLab, SAS и др. Основной недостаток: их применение требует от исследователя спец.подготовки.**2.** Програм. комплексы, специализирующиеся на моделир-нии узкого круга систем одной конкретной предметной области. Недостаток: ограниченность применения таких программ одной предметной областью.Преимущ-ва: легкость их освоения специалистами в данной предмет.области, и эффективность применения вследствие узкой специализации.

В кажд. цикле создания программ. модели можно выделить **этапы**:**1. Формулирование проблемы**: описание исслед. проблемы, установление границ и ограничений моделируемой системы, определение целей исследования.**2. Разработка модели**: переход от реальной системы к некоторой логической схеме (абстрагирование).**3. Подготовка данных**: отбор данных, необх. для построения модели, и представление их в соответств. форме.**4. Трансляция модели**: описание модели на языке имитац. Моделир-я.**5. Оценка адекватности**: повышение до приемлемого уровня степени уверенности, с кот. можно судить относительно корректности выводов о реальной системе, получ. на основании обращения к модели.**6. Планирование**: определение условий проведения машинного эксперимента с имитац.моделью.**7. Экспериментирование**: многократн. прогон имит. модели на компьютере для получения требуемой инф-ии.**8. Анализ результатов**: изучение рез-тов для подготовки выводов и рекомендаций по решению проблемы.**9. Реализация и документирование**: реализация рекомендаций, получ.на основе имитации, и составление документации по модели и ее использованию.

1. **Система имитационного моделирования AnyLogic.**

AnyLogic– уникальный инструмент имит.моделир-я, поддерживающ. на единой платформе абсолютно все существующ. подходы дискретно-событийного и непрерывного моделир-я. AnyLogicимеет развитый базовый язык дискретного и смешан. дискретно-непрерывного моделир-я.Библиотека AnyLogicEnterprise Library предоставляет высокоуровнев. интерфейс для быстрого создания дискретно-событийных моделей с пом. блок-схем. Графич. представление систем с пом. блок-схем широко исп-ся во многих важных сферах деятельности: произв-ве, логистике, системах обслуживания, бизнес-процессах, моделир-ии компьют. и телекоммуникацион. сетей.

Класс Entity яв-ся базовым классом для всех сообщений, кот. посылаются между активными объектами библиотеки Enterprise Library.

Под заявкой в библиотеке Enterprise Library может пониматься:

• заявка в ее обычном понимании (продукт, потребитель, пакет данных, документ),

• ресурс (оператор, машина, критическая секция),

• транспортер (поезд, автобус, корабль, автопогрузчик).

Заявки в их традиц. понимании генерир-ся объектами Source, проходят через смоделир. систему, где они обрабатыв-ся, обслужив-ся, транспортир-ся, конкурируют за право обладания ресурсами и эту систему покидают.

Ресурсы, созданные объектами Resource, м б.заняты заявками для выполнения каких-то задач, после чего они освобожд-ся и возвращ-ся в объект Resource.

Транспортеры, так же как и обычные заявки, создаются объектами Source, затем передаются в объекты Node и исп-ся для транспортировки других заявок между узлами и вдоль сегментов сети.

Объект класса Entity может исп-ся в любой из этих ролей, и при необходимости даже менять свою роль во время работы модели.

**Source** - Источник заявок. Обычно исп-ся в кач.начальной точки потока заявок, или как генератор ресурсов, транспортеров.

**Sink -** Уничтожает поступивш. заявки. Обычно исп-ся в кач.конечной точки потока заявок.

**Enter -** Пересылает заявки, переданные этому объекту либо “явно” через входной порт inputExternal, либо с помощью функции объекта take(), дальше по блок-схеме.

**SelectOutput -** Принимает заявку, и затем, в зависимости от заданного условия, передает ее на один из двух выходных портов.

**Queue -**  моделирует очередь, он хранит поступающ. заявки в опред.порядке: FIFO (заявки помещ-ся в очередь в порядке поступления), LIFO (заявки помещ-ся в порядке, обратном поступлению), RANDOM (заявки помещ-ся в произвольные места очереди) или PRIORITY (заявки помещ-ся в очередь в соответствии со значением своих полей priority).

1. **Общие понятия сетей Петри.**

Сетевые модели (сети Петри) исп-ся для анализа причинно-следственных связей в сложных системах. Аппарат теории сетей Петри позволяет описывать стр-ру и взаимод-е параллельных систем и процессов. Сеть Петри (N-схема) задается 4мя элементами:N = <B,D,I,O>,где B - конечное множество позиций; D - конечное множество переходов; I - входная функция (прямая функция инцидентности), I:BXD→{0,1}; O - выходная функция (обратная функция инцидентности), O:DXB→{0,1}.

Iотображает переход dj в множество входных позицийbiϵI(dj), а выходная функция О отображает переход dj в множество выходных позиций biϵD(dj). Для каждого djϵDможно определить множество входных позиций перехода I(dj) и выходных позиций перехода O(dj) какI(dj)={biϵB|I(bi,dj)=1}, O(dj)={biϵB|I(dj,bi)=1}.

Аналогично, для каждого biϵB можно определить множество входных переходов позиции I(bi) и выходных переходов позицииO(bi):I(bi)={djϵD|I(dj,bi)=1}, O(bi)={djϵD|O(bi,dj)=1}.

Графически сеть Петри изображается в виде двудольного ориентированного мультиграфа, представляющего собой совокупность позиций и переходов.

Для представления динамич. Св-в объекта вводится ф-я маркировки (разметки) M:B🡪{0,1,2,…}. Маркировка - присвоение абстрактных объектов, называемых метками (фишками), позициям:NM = <B,D,I,O,M>.

Функционир-е сети Петри отражается путем перехода от разметки к разметке. M0 - начальная разметка. Смена разметок - срабатывание одного из переходов сети. Необходимое условие срабатывания перехода dj: biϵI(dj) {M(bi)≥1}, где M(bi) – разметка позиции bi.

Срабатывание перехода dj изменяет разметку сетиM(b) на M’(b) по следующему правилу:M’(b)=M(b)-I(dj)+O(dj), т.е. переход djизымает по одной метке из каждой своей входной позиции и добавляет по одной метке в каждую из выходных позиций.

Сети Петри предст. удобный математич. аппарат для моделир-я параллел.технологич. процессов с разделяемыми ресурсами. Преимущ-вом сетей Петри явл-ся легкость построения иерархич. конструкций, что позволяет сначала исследовать отдельные подсистемы, а затем всю систему в целом. Модели, построенные на основе сетей П., предназначены для анализа с пом. имитации на компьютере. Такие модели довольно легко реализуются программно даже с помощью универс. языков программир-я.

**Теория информационных процессов и систем**

1. **Основные положения общей теории систем.**

Основоположником этого направления считается биолог Л. фон Берталанфи.

К числу задач, решаемых теорией систем, относятся:

-определение общей структуры системы;

-организация взаимодействия между подсистемами и элементами;

-учет влияния внешней среды;

-выбор оптимальной структуры системы;

-выбор оптимальных алгоритмов функционирования системы.

Теория систем как наука развивается в двух направлениях:

*-феноменологический подход* (иногда называемый причинно-следственным или терминальным).

*-разработка теории сложных целенаправленных систем*.

**Система** – целостный комплекс взаимосвязанных элементов.

В зависимости от количества учитываемых факторов и степени абстрактности определение понятия "система" можно представить в следующей символьной форме:

-Система есть нечто целое:

*S=А(1,0)*

-Система есть организованное множество:

*S=(орг, М),*

где орг - оператор организации; М - множество.

-Система есть множество вещей, свойств и отношений:

*S=({т},{n},{r}),*

где m - вещи, n - свойства, r - отношения.

-Система есть множество входов, множество выходов, множество состояний, характеризуемых оператором переходов и оператором выходов:

*S=(Х, Y, Z, H, G),*

где *Х* - входы, *Y* - выходы, *Z* - состояния, *Н* - оператор переходов, *G* - оператор выходов.

-Для организационных систем удобно в определении системы учитывать следующее:

*S=(РL, RO, RJ, EX, PR, DT, SV, RD, EF),*

где *РL* - цели и планы, *RO* - внешние ресурсы, *RJ* - внутренние ресурсы, *ЕХ* - исполнители, *PR* - процесс, *DТ* - помехи, *SV* - контроль, *RD* - управление, *ЕF* - эффект.

**Элемент** – часть системы, обладающая некоторой самостоятельностью и имеющей связи с другими частями.

**Связь** – обязательное свойство системы. Она рассматривается как способ воздействия, взаимодействия или отношения элементов между собой, обуславливающий структуру системы и ее размещение в пространстве и во времени.

Классификация связей:

-первого порядка – связи, функционально, необходимый друг другу,

-второго порядка – дополнительные связи.

-третьего порядка – избыточные и противоречивые связи.

Типы связей:

-механические

-энергетические

-информационные

-связи управления и др.

Могут быть направленными и ненаправленными, сильными и слабыми, могут иметь свой характер, быть прямыми и обратными.

**Структура** – это совокупность элементов и связей между ними. Она является внутренней основой системы, которая обеспечивает ее функционирование, отражает уровень организации системного объекта, выражает его целостность и обладает относительной самостоятельностью.

**Свойство** – сторона системы, обуславливающая различия и сходства с другими системами или элементами и проявляющаяся при взаимодействии с ними. Свойство – понятие относительное.

**Характеристика** – это то, что выражает некоторые свойства объекта, но не оценивается числом.

**Параметр** – характеристика, представленная величиной.

**Состояние системы** – существенные свойства системы в их конкретных проявлениях, которые выражаются через множества значений характеристик в данный момент времени. Переход из одного состояния в другое определяется внутренним состоянием системы и состоянием внешней среды.

**Среда** – множество объектов вне системы, которые оказывают влияние на систему, либо находятся под воздействием системы, либо и то, и другое.

**Поведение** – последовательность состояний системы, принимаемое во времени.

**Равновесие** – способность системы в отсутствие внешних возмущающих воздействий (или при постоянных воздействиях) сохранить свое состояние сколь угодно долго.

**Устойчивость** – способность системы возвращаться в состояние равновесия после того, как она была из этого состояния выведена под влиянием внешних возмущающих воздействий. Состояние равновесия, в которое система способна возвращаться называют **устойчивым состоянием равновесия**.

**Объект** – это то, что существует вне нас, независимо от нашего сознания и выступает предметом познания и воздействия.

**Цель** – область состояния среды или системы, которую необходимо достичь при функционировании системы.

**Управление** – совокупность информационных воздействий для достижения поставленной цели.

1. **Закономерности систем. Закон Эшби.**
2. *Закономерности взаимодействия части и целого* помогают глубже понять диалектику части и целого в системе и формировать более адекватные модели принятия решений.

*Закономерность целостности* проявляется в системе в возникновении новых интегративных качеств, не свойственных образующим ее компонентам. Данная закономерность учитывает следующие особенности:

-свойства системы QS не являются простой суммой свойств, составляющих ее элементов qi, т.е. QS≠∑[i=1..n]qi;

-свойства системы зависят от свойств составляющих ее элементов, т.е. QS = f(qi).

Свойство целостности связано с целью, для выполнения которой предназначена система.

Введены два коэффициента для оценки целостности:

-α – оценивает степень целостности,

-β – коэффициент использования элементов.

*Интегративность.* Эту закономерность выделяют как самостоятельную, стремясь подчеркнуть интерес не к внешним факторам проявления целостности, а к более глубоким причинам формирования этого свойства и, главное,— к его сохранению.

1. *Закономерности иерархической упорядоченности систем* связаны с закономерностью целостности и расчленением целого на части, однако характеризует и взаимодействие системы с ее окружением, со средой, с надсистемой, с подчиненными системами.

*Коммуникативность*. Система не изолирована от других систем, она связана множеством коммуникаций со средой, представляющих в свою очередь сложное и неоднородное образование, которое содержит надсистему (т.е. систему более высокого порядка, задающую требования и ограничения исследуемой системе), подсистемы (т.е. подведомственные системы) и системы одного уровня с рассматриваемой.

*Иерархичность (иерархическая упорядоченность).* Сущность этой закономерности в том, что закономерность целостности проявляется на каждом уровне иерархии.

1. *Закономерности функционирования и развития систем*

*Историчность.* Время является непременной характеристикой системы, поэтому каждая система исторична.

*Самоорганизация.* Ранее рассмотренные закономерности охарактеризовали способности систем противостоять энтропийным тенденциям, способности адаптироваться к изменяющимся условиям, преобразуя при этом свою структуру…

1. *Закономерности осуществимости систем*

*Эквифинальность*характеризует предельные возможности систем определенного класса сложности. **Эквифинальность** – способность (в отличие от состояний равновесия в закрытых системах, полностью детерминированных начальными условиями систем) достигать не зависящего от времени состояния (которое не зависит от ее исходных условий и определяется исключительно параметрами системы).

* 1. *Закон необходимого разнообразия* (У. Р. Эшби).

**Закон Эшби**: чтобы создать систему, способную справиться с решением проблемы, обладающей определенным, известным разнообразием, нужно, чтобы сама система имела еще большее разнообразие, чем разнообразие решаемой проблемы, или была способна создать в себе это разнообразие. Этот закон достаточно широко применяется на практике. Он позволяет, например, получить рекомендации по совершенствованию системы управления предприятием, объединением, отраслью.

* 1. *Закономерность потенциальной эффективности систем*.

1. *Закономерность целеобразования.* Исследования процесса целеобразования в сложных системах философами, психологами и кибернетиками позволили сформулировать некоторые общие закономерности процессов обоснования и структуризации целей в конкретных условиях совершенствования сложных систем

*Закономерности возникновения и формулирования целей*

*-Зависимость представления о цели и формулировки цели от стадии познания объекта или процесса и от времени*.

*-Зависимость цели от внутренних и внешних факторов*.

*-Возможность сведения задачи формирования общей (главной, глобальной) цели к задаче структуризации цели*.

* 1. *Закономерности формирования структур цели.*

*-Зависимость способа представления структуры целей от стадии познания объекта или процесса.*

*-Проявление в структуре целей закономерности целостности.*

*-Закономерности формирования иерархических структур цели.* Рекомендации по формированию древовидных иерархических структур.

-Существует два подхода формирования:

--Формирование структур сверху

---Методы структуризации

---Методы декомпозиции

---Целевой/целенаправленный подход

--Формирование структур снизу

---Морфологический

---Лингвистически

---Тезаурусный

---Терминальный

-Цели нижележащего уровня иерархии можно рассматривать как средства для достижения цели вышестоящего уровня.

-Расчленение на каждом уровне должно быть соразмерным, а выделенные части логически независимыми. Число уровней иерархии и число компонентов на каждом уровне должно быть 7±2.

1. **Методы формализованного представления систем (МФПС, количественные).**

Необходимость в методах моделирования возникает в разрешении конкретных задач. Часто задача усложняется на столько, что для её постановки и решения не может быть сразу определён подходящий аппарат формализации, когда решение задачи требует участие специалистов различных областей знаний. В этом случае постановка задачи становится проблемой, для решения которой нужно разрабатывать специализированные подходы и методы. Возникает необходимость определить область проблемы принятия решения, выявить факторы, влияющие на её решение, подобрать приёмы и методы которые позволят сформулировать или поставить задачу таким образом, чтобы решение было принято.

Для принятия решения необходимо получить выражение связывающее цель со средствами, связывающими её достижение. Данное выражение называют критерием функционирования, показателем эффективности целевой функции. Если удаётся получить такое выражение, то задача практически решена.

Для того чтобы в сжатые сроки поставить задачу, проанализировать цели, определить возможные средства, отобрать требуемую информацию, характеризующую условия принятия решения и влияющих на выбор критериев и ограничений, а в идеале получить связывающую цель со средствами, применяют системное представление, приёмы и методы системного анализа.

Постановка задачи заключается в том, чтобы перевести её словесное вербальное описание в формальное, если полученная формальная модель опирается на фундаментальный закон или подтверждается экспериментом, то этим доказывается её адекватность в ситуации и модель рекомендуется для решения задач соответствующего класса. Методы МФПС - методы формализованного представления систем, количественные методы.

МФПС: *-Аналитические* – методы, в которых ряд свойств многомерной и многосвязной системы или какой-либо ее части отображаются в n-мерном пространстве одной единственной n-мерной точки, совершающей какое-либо движение в n-мерном пространстве. Это отображение осуществляется либо с помощью функции f(Sx), либо посредством функционала Ф(Sx).

Данные методы применяются только в тех случаях, когда свойство системы можно отобразить с помощью детерминированных величин или зависимостей. *Статистические*. В тех случаях, когда не удается представить систему с помощью детерминированных категорий, можно применить отображение ее с помощью случайных стохастических событий или процессов, которые описываются соответствующими вероятностными (статистическими) характеристиками и статистическими закономерностями.

Статистическое отображение системы в общем случае можно представить как бы в виде размытой точки или области в n-мерном пространстве, в которую переводит систему оператор. **Размытую точку** следует понимать как некоторую область, характеризующее движение системы, при этом границы области заданы с некоторой вероятностью, и движение точки определяется некоторой случайной функцией.

*-Теоретико-множественное представление*. Базируется на понятиях: множество, элементы множества и отношения на множествах.

*-Логические методы.* Логические представления переводят реальную систему и отношения в ней на язык одной из алгебр логики. Основан на применении алгебраических методов для выражения законов формальной логики.

*-Лингвистические и символьные представления.* Лингвистические представления базируются на понятиях **тезауруса** (множество смысловыражающих элементов языка с заданными смысловыми отношениями), **грамматики** (правила образования смысловыражающих элементов разных уровней тезауруса), **семантики** (смысловое содержание формируемых фраз, предложений и других смысловыражающих элементов) и **прагматики** (смысл для заданной задачи).

Лингвистические и символьные представления являются удобным аппаратом для первого этапа постепенной формализации задач принятия решений плохо формализуемых ситуаций.

*-Графические представления* – отнесены любые графики и возникшие на основе графических отображений теории.

Продолжение 4

абсолютно верным.

Процедура метода:

-выявить в исследуемом объекте его признаки, характеристики, свойства или иные атрибуты, существенные для выполнения его главной полезной функции,

-отрицают один из выявленных признаков и заменяют его принципиально иным признаком, не обязательно противоположным отброшенному, конструируют новый объект с этой замененной характеристикой,

1. **Методы описания систем, направленные на активизацию использования опыта и интуиции специалиста (МАИС, качественные).**

Необходимость в методах моделирования возникает в разрешении конкретных задач. Часто задача усложняется на столько, что для её постановки и решения не может быть сразу определён подходящий аппарат формализации, когда решение задачи требует участие специалистов различных областей знаний. В этом случае постановка задачи становится проблемой, для решения которой нужно разрабатывать специализированные подходы и методы. Возникает необходимость определить область проблемы принятия решения, выявить факторы, влияющие на её решение, подобрать приёмы и методы которые позволят сформулировать или поставить задачу таким образом, чтобы решение было принято.

Для принятия решения необходимо получить выражение связывающее цель со средствами, связывающими её достижение. Данное выражение называют критерием функционирования, показателем эффективности целевой функции. Если удаётся получить такое выражение, то задача практически решена.

Для того чтобы в сжатые сроки поставить задачу, проанализировать цели, определить возможные средства, отобрать требуемую информацию, характеризующую условия принятия решения и влияющих на выбор критериев и ограничений, а в идеале получить связывающую цель со средствами, применяют системное представление, приёмы и методы системного анализа.

Постановка задачи заключается в том, чтобы перевести её словесное вербальное описание в формальное, если полученная формальная модель опирается на фундаментальный закон или подтверждается экспериментом, то этим доказывается её адекватность в ситуации и модель рекомендуется для решения задач соответствующего класса. Методы МАИС - методы направленные на активизацию опыта и интуицию специалиста.

В качественных методах основное внимание уделяется организации постановки задачи, новому этапу ее формализации, формированию вариантов, выбору подхода к оценке вариантов, использованию опыта человека, его предпочтений, которые не всегда могут быть выражены в количественных оценках.

Качественные методы системного анализа применяются, когда отсутствуют описания закономерностей систем в виде аналитических зависимостей.

*1.Методы типа “мозговой атаки”* (“мозговой штурм”, “конференция идей”, “коллективная генерация идей” (КГИ)). Обычно при проведении мозговой атаки или сессий КГИ стараются выполнять определенные правила, суть которых:

-рекомендуется сформулировать проблему в основных терминах, выделив центральный пункт обсуждения;

-необходимо обеспечить как можно большую свободу мышления участников КГИ и высказывания ими новых идей;

-приветствуются любые идеи, даже если вначале они кажутся сомнительными или абсурдными (обсуждение и оценка идей производится позднее);

-не допускается критика, не объявляется ложной и не прекращается обсуждение ни одной идеи;

-необходимо стараться создавать как бы цепные реакции идей;

-желательно высказывать как можно больше идей, особенно нетривиальных.

2.*Методы типа “сценариев”*. Методы подготовки и согласования представлений о проблеме или анализируемом объекте, изложенные в письменном виде, называются **сценарием**.

Сценарий позволяет создать предварительное представление о проблеме или системе, которое не удается сразу отобразить в формальной модели. Его следует рассматривать как основу для разработки более формального представления о будущей системе или решаемой проблеме.

На практике по типу сценариев разрабатывались прогнозы в некоторых отраслях промышленности.

3.*Методы типа дерева целей*. Метод подразумевает использование иерархической структуры, получаемой путей разделения общей цели на подцели, а их, в свою очередь, на более детальные составляющие, называемые подцелями нижележащих уровней, направлениями, проблемами, а начиная с некоторого уровня функциями.

Древовидные иерархические структуры используются и при исследовании и совершенствовании организационных структур.

4.*Методы типа “«Дельфи”*. Метод предполагает полный отказ от коллективных обсуждений. Это делается для того, чтобы уменьшить влияние таких психологических факторов, как присоединение к мнению наиболее авторитетного специалиста, нежелание отказаться от публично выраженного мнения, следование за мнением большинства.

Процедура Дельфи-метода:

-в упрощенном виде организуется последовательность циклов мозговой атаки;

-в более сложном виде разрабатывается программа последовательных индивидуальных опросов обычно с помощью вопросников, исключая контакты между экспертами, но предусматривающая ознакомление их с мнениями друг друга между турами; вопросники от тура к туру могут уточняться;

-в наиболее развитых методиках экспертам присваиваются весовые коэффициенты значимости их мнений, вычисляемые на основе предшествующих опросов, уточняемые от тура к туру и учитываемые при получении обобщенных результатов оценок.

Исследуемые проблемы: научные открытия, рост народонаселения, автоматизация производства, освоение космоса, предотвращение войны, военная техника.

*5.Методы экспертных оценок*. При использовании экспертных оценок обычно предполагается, что мнение группы экспертов надежнее, чем мнение отдельного эксперта.

Все множество проблем, решаемых методами экспертных оценок, делится на два класса. К первому относятся такие, в отношении которых имеется достаточное обеспечение информацией. Ко второму классу относятся проблемы, в отношении которых знаний для уверенности в справедливости указанных гипотез недостаточно.

*6.Морфологические методы*. Основная идея морфологических методов – систематически находить все “мыслимые” варианты решения проблемы или реализации системы путем комбинирования выделенных элементов или их признаков.

Цвикки предложил три метода морфологического исследования.

*-Метод систематического покрытия поля.*

*-Метод отрицания и конструирования* – любое утверждение, сформулированное в конечных и полностью определенных терминах, не может быть

1. **Структура информационной системы (каноническое представление).**

**Информационная система** – взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели.

Структуру информационной системы составляет совокупность отдельных ее частей, назы­ваемых подсистемами.

**Подсистема –** это часть системы, выделенная по какому-либо признаку.

Общую структуру информационной системы можно рассматривать как совокупность подсистем независимо от сферы применения. В этом случае говорят о структурном признаке классификации, а подсистемы называют обеспечивающими. Среди обеспечивающих подсистем обычно выделяют информационное, техническое, математическое, программное, организационное и правовое обеспечение.

**Информационное обеспечение** – совокупность единой системы классификации и кодирования информации, унифицированных систем документации, схем потоков, циркулирующих в организации, а также методология построения баз данных. Назначение подсистемы информационного обеспечения состоит в своевременном формиро­вании и выдаче достоверной информации для принятия управленческих решений.

*Унифицированные системы документации* создаются на государственном республиканском, отраслевом и региональном уровнях. Главная цель - это обеспе­чение сопоставимости показателей различных сфер общественного производства.

*Схемы информационных потоков* отражают маршруты движения инфор­мации и ее объемы, места возникновения первичной информации и использования резуль­татной информации.

*Методология построение баз данных* базируется на теоретических основах их проектирования. Для понимания концепции методологии приведем основные идеи в виде двух последовательно реализуемых на практике этапов:

*1-й этап* – обследование всех функциональных подразделений фирмы.

*2-й этап* – построение концептуальной информационно-логической модели данных для обследованной на 1-м этапе сферы деятельности.

**Техническое обеспечение** – комплекс технических средств, предназна­ченных для работы информационной системы, а также соответствующая документация на эти средства и технологические процессы

*Комплекс технических средств*составляют:

* компьютеры любых моделей;
* устройства сбора, накопления, обработки, передачи и вывода информации;
* устройства передачи данных и линий связи;
* оргтехника и устройства автоматического съема информации;
* эксплуатационные материалы и др.

*Документацией* оформляются предварительный выбор технических средств, ор­ганизация их эксплуатации, технологический процесс обработки данных, технологическое оснащение. Документацию можно условно разделить на три группы:

* общесистемную, включающую государственные и отраслевые стандарты по техничес­кому обеспечению;
* специализированную, содержащую комплекс методик по всем этапам разработки тех­нического обеспечения;
* нормативно-справочную, используемую при выполнении расчетов по техническому обеспечению.

**Математическое и программное обеспечение** – совокупность математических методов, моделей, алгоритмов и программ для реализации целей и задач информационной системы, а также нормального функционирования комплекса технических средств

К средствам *математического обеспечения* относятся:

* средства моделирования процессов управления;
* типовые задачи управления;
* методы математического программирования, математической статистики, теории массового обслуживания и др.

В состав *программного обеспечения* входят общесистемные и специальные программные продукты, а также техническая документация.

К *общесистемному* программному обеспечению относятся комплексы про­грамм, ориентированных на пользователей и предназначенных для решения типовых задач обработки информации.

*Специальное* программное обеспечение представляет собой совокупность про­грамм, разработанных при создании конкретной информационной системы.

*Техническая документация* на разработку программных средств должна содержать описание задач, задание на алгоритмизацию, экономико-математическую модель задачи, контрольные примеры.

**Организационное обеспечение –** совокупность методов и средств, регла­ментирующих взаимодействие работников с техническими средствами и между собой в процессе разработки и эксплуатации информационной системы

Организационное обеспечение реализует следующие функции:

* анализ существующей системы управления организацией, где будет использоваться ИС, и выявление задач, подлежащих автоматизации;
* подготовку задач к решению на компьютере, включая техническое задание на проектирование ИС и технико-экономическое обоснование ее эффективности;
* разработку управленческих решений по составу и структуре организации, методологии решения задач, направленных на повышение эффективности системы управления.

**Правовое обеспечение** – совокупность правовых норм, определяющих создание, юридический статус и функционирование информационных сис­тем, регламентирующих порядок получения, преобразования и использования информации

Главной целью правового обеспечения является укрепление законности.

В состав правового обеспечения входят законы, указы, постановления государствен­ных органов власти, приказы, инструкции.

Правовое обеспечение этапов разработки информационной системы включает норма­тивные акты, связанные с договорными отношениями разработчика и заказчика и правовым регулированием отклонений от договора.

1. **Динамическое описание информационных систем.**

Функционирование сложной системы можно представить как совокупность двух функций времени: x(t) – внутреннее состояние системы; y(t) – выходной процесс системы. Обе функции зависят от u(t) – входного воздействия и от f(t) – возмущения.

Для каждого t∈T существует множество z∈Z.

Z = Z1 ×Z2 ...×Zn – множество n мерного пространства. Состояние системы z(t) – точка или вектор пространства Z с обобщенными координатами z1, z2, z3, z4, ....., zn.

U = T×Z – фазовое пространство системы.

* + 1. *Детерминированная система без последствий* – система, состояние которой z(t) зависит только от z(t0) и не зависит от z(0) ... z(t0), т.е. z(t) зависит от z(t0) и не зависит от того каким способом система попала в состояние z(t0).

Для систем без последствия ее состояние можно описать как:

, где

– множество всевозможных отрывков входящих сообщений, соответствующих интервалу (t0, t],

H – оператор переходов системы.

t∈T, t0∈T, z(t0) ∈Z, ∈.

Формальная запись отображения:

T×T×→Z.

Начальные условия .

Если =, то =

Если t0<t1<t2 и t0, t1, t2∈T, то =, так как есть сочленение отрезков и .

Оператор выходов системы G реализует отношение:

{(t, t0)} ×Z× (t, XL)T} →Y,

.

(x, y) ∈X×Y - расширенное состояние системы.

* + 1. *Детерминированные системы без последствия с входными сигналами двух классов*

Расширение понятие системы идет по трем путям:

* учет специфики воздействий;
* учет последствий;
* учет случайных факторов.

Учет специфики воздействий. Вводится понятие управляющих сигналов u∈U; u=M(t), или если сигнал u∈U описывается набором характеристик. U = U1×U2×UL.

Отличие от предыдущего случая в том, что множество моментов времени tu и tx могут не совпадать.

Вводится расширенное множество X\*= X×U, таким образом, состояние системы описывается вектором x = (x, u) = (x1, x2, .... , xn, u1, u2, .... , uL).

С учетом этого предыдущие формулы приобретают вид. Оператор переходов:

 или , что соответствует отображению

T×T××→Z.

* + 1. *Детерминированные системы с последствием.* Большой класс систем характеризуется тем, что для представления их состояния необходимо знать состояние системы на некотором множестве моментов времени.

,

, где

 – семейство всевозможных состояний системы.

* + 1. *Стохастические системы.* Системы, функционирующие под воздействием случайных факторов, называются **стохастическими**. Для их описания вводится случайный оператор: ω∈Ω - пространство элементарных событий с вероятностной мерой P(A).

Случайный оператор H1, переводящий множество X в множество Z: z = H1(x, ω), реализующий отображение множества Ω в множество {X→Z }

Оператор переходов будет представлен соответственно:

,

y(t) = G1(t, z(t), ω’’ ), где

ω0, ω’, ω’’ – выбираются из множества Ω в соответствии с вероятностями P0(A), Px(A), Py(A).

При фиксированныхω’, ω’’ - система со случайными начальными состояниями.

При фиксированныхω0, ω’’ - система со случайными переходами.

При фиксированныхω0, ω’ - система со случайными выходами.

1. **Агрегатное описание информационных систем.**

Агрегат - унифицированная схема, получаемая наложением дополнительных ограничений на множества состояний, сигналов и сообщений и на операторы перехода а так же выходов. t ∈ T - моменты времени; x ∈ X - входные сигналы; u ∈ U - управляющие сигналы; y ∈ Y - выходные сигналы; z ∈ Z - состояния, x(t), u(t), y(t), z(t) - функции времени.

Агрегат - объект определенный множествами T, X, U, Y, Z и операторами H и G реализующими функции z(t) и y(t). Структура операторов H и G является определяющей для понятия агрегата.

T – это фиксированное подмножество действительных чисел, множество рассматриваемых моментов времени.

U – это множество управляющих сигналов

X – множество входных сигналов

Z – множество состояний

Y – множество выходных сигналов

x(t), y(t), z(t), u(t) операторы времени

операторы H,G реализуют функцию z(t) и y(t).

Вводится пространство параметров агрегата b=(b1, b2, ...,bn) ∈ B.

Оператор выходов G реализуется как совокупность операторов G` и G``. Оператор G` выбирает очередные моменты выдачи выходных сигналов, а оператор G`` - содержание сигналов.у=G``{t, z(t),u(t),b}.

В общем случае оператор G`` является случайным оператором, т.е. t, z(t), u(t) и b ставится в соответствие множество y с функцией распределения G``. Оператор G` определяет момент выдачи следующего выходного сигнала.

Операторы переходов агрегата. Рассмотрим состояние агрегата z(t) и z(t+0).

Оператор V реализуется в моменты времени tn , поступления в агрегат сигналов xn(t). Оператор V1 описывает изменение состояний агрегата между моментами поступления сигналов.

z(t’n + 0) = V{ t’n, z(t’n), x(t’n), b}.

z(t) = V1(t, tn, z(t+0),b}.

Особенность описания некоторых реальных систем приводит к так называемым агрегатам с обрывающимся процессом функционирования. Для этих агрегатов характерно наличие переменной соответствующий времени оставшемуся до прекращения функционирования агрегата.

Все процессы функционирования реальных сложных систем по существу носят случайный характер, по этому в моменты поступления входных сигналов происходит регенерация случайного процесса. То есть развитие процессов в таких системах после поступления входных сигналов не зависит от предыстории.

Автономный агрегат - агрегат который не может воспринимать входных и управляющих сигналов.

Неавтономный агрегат - общий случай.

Частные случаи агрегата:

-Кусочно-марковский агрегат - агрегат процессы в котором являются обрывающими марковскими процессами. Любой агрегат можно свести к марковскому.

-Кусочно-непрерывный агрегат - в промежутках между подачей сигналов функционирует как автономный агрегат.

-Кусочно-линейный агрегат. dzv(t)/dt = F(v)(zv).

Представление реальных систем в виде агрегатов неоднозначно, вследствие неоднозначности выбора фазовых переменных.

Иерархические системы

Иерархический принцип построения модели как одно из определений структурной сложности. Иерархический и составной характер построения системы.

Вертикальная соподчиняемость.

Право вмешательства. Обязательность действий вышестоящих подсистем.

Страты - уровни описания или обстрагирования. Система представляется комплексом моделей - технологические, информационные и т.п. со своими наборами переменных.

Слои - уровни сложности принемаемого решения:

1. срочное решение;

2. неопределенность или неоднозначность выбора.

Разбитие сложной проблемы на более простые: слой выбора способа действия, слой адаптации, слой самоорганизации.

Многоэшелонные системы. Состоит из четко выраженных подсистем, некоторые из них являются принимающими решения иерархия подсистем и принятия решений.

Декомпозиция на подсистемы - функционально-целевой принцип, декомпозиция по принципу сильных связей.

1. **Кибернетический подход к описанию систем.**

Кибернетический подход к описанию систем состоит в том, что всякое целенаправленное поведение рассматривается как управление. **Управление** – в широком, кибернетическом смысле – это обобщение приемов и методов, накопленных разными науками об управлении искусственными объектами и живыми организмами. **Язык управления** – это использование понятий “объект”, “среда”, “обратная связь”, “алгоритм”…

Под управлением будем понимать процесс организации такого целенаправленного воздействия на некоторую часть среды, называемую **объектом управления**, в результате которого удовлетворяются потребности субъекта, взаимодействующего с этим объектом.

Анализ управления заставляет выделить тройку – среду, объект и субъект, внутри которой разыгрывается процесс управления (рис). В данном случае субъект ощущает на себе воздействие среды Х и объекта Y. Если состояние среды Х он изменить не может, то состоянием объекта Y он может управлять с помощью специально организованного воздействия U, называемого управлением.

Пусть Ux\*– оптимальное поведение субъекта, минимизирующее его потребности А. Способ решения задачи, позволяющий определить Ux\*, называется **алгоритмом управления**: Ux\* = ϕ(At,X), где

ϕ – алгоритм, позволяющий синтезировать управление по состоянию среды Х и потребностей Аt. Потребности субъекта изменяются не только под влиянием среды или объекта, но и самостоятельно, отражая жизнедеятельность субъекта, что отмечается индексом t.

Процесс управления как организация целенаправленного воздействия на объект может реализовываться как на интуитивном, так и на осознанном уровне. Первый используют животные, второй – человек. Осознанное удовлетворение потребностей заставляет декомпозировать алгоритм управления и вводить промежуточную стадию – формулировку цели управления, т.е. действовать по двухэтапной схеме:

At –>Z\*–>U\*

1 2

На первом этапе определяется цель управления Z\*, причем задача решается на интуитивном уровне: Z\* = ϕ1(X,At),

где ϕ1 – алгоритм синтеза цели Z\* по потребностям Аt и состоянию среды X.

На втором этапе определяется управление Ux\*, реализация которого обеспечивает достижение цели, сформированной на первой стадии, что и приводит к удовлетворению потребностей субъекта. Именно на этой стадии может быть использована вся мощь формального аппарата, с помощью которого по цели Z\* синтезируется управление: Ux\* = ϕ2(Z\*,X), где ϕ2 – алгоритм управления. Этот алгоритм и есть предмет изучения кибернетики как науки.

Объект

YY()

Субъект



Среда

x

x

x

y

Cy

U

Среда

объект

субъект

X

X

Y

U

Таким образом, разделение процесса управления на два этапа отражает известные стороны науки – неформальный (интуитивный, экспертный) и формальный (алгоритмизуемый). Если первая пока полностью принадлежит человеку, то вторая является объектом приложения формальных подходов. Естественно, что эти различные функции выполняются разными структурными элементами. Первую функцию ϕ1, выполняет субъект, а вторую ϕ2 – управляющее устройство (УУ). На рисунке 2 показано взаимодействие этих элементов. Штриховой линией выделена система управления (СУ), выполняющая функцию реализации целей управления, формируемых субъектом.

**Управление** – целенаправленная организация того или иного процесса, протекающего в системе. В общем случае процесс управления состоит из следующих четырех элементов:

-получение информации о задачах управления (Z\*),

-получение информации о результатах управления (т.е. о поведении объекта управления) Y’;

-анализ полученной информации и выработка решения (J),

-исполнение решения (т.е. осуществление управляющих воздействий) U.

**Процесс управления** – это информационный процесс, заключающийся в сборе информации о ходе процесса, передаче ее в пункты накопления и переработки; анализе поступающей, накопленной и справочной информации; принятии решения на основе выполненного анализа; выработке соответствующего управляющего воздействия и доведении его до объекта управления. Каждая фаза процесса управления протекает во взаимодействии с окружающей средой при воздействии различного рода помех. Цели, принципы и границы управления зависят от сущности решаемой задачи.

**Система управления** – совокупность взаимодействующих между собой объекта управления и органа управления, деятельность которых направлена заданной цели управления.

В системах управления решаются четыре основные задачи управления: стабилизация, выполнение программы, слежение, оптимизация.

Прежде чем принимать решение о создании СУ, необходимо рассмотреть все его этапы, независимо от того, с помощью каких технических средств они будут реализованы. Такой алгоритмический анализ управления является основой для принятия решения о создании СУ и степени ее автоматизации. При этом анализе следует обязательно учитывать фактор сложности объекта управления:

-отсутствие математического описания системы;

-стохастичность поведения;

-негативность к управлению;

-не стационарность, дрейф характеристик;

-невоспроизводимость экспериментов (развивающаяся система все время как бы перестает быть сама собой, что предъявляет специальные требования к синтезу и коррекции модели объекта управления).

Особенности сложной системы часто приводят к тому, что цель управления таким объектом в полной мере никогда не достигается, как бы совершенно ни было управление.

1. **Виды систем (классификация).**

1.по признаку структурированности задач. При создании или при классификации информационных систем неизбежно возникают про­блемы, связанные с формальным – математическим и алгоритмическим описанием решае­мых задач. От степени формализации во многом зависят эффективность работы всей системы, а также уровень автоматизации, определяемый степенью участия человека при принятии решения на основе получаемой информации. Различают три типа задач, для которых создаются информационные системы:

-структурированные (формализуемые) – задача, где известны все ее элементы и взаимосвязи между ними. В структурированнойзадаче удается выразить ее содержание в форме математической модели, имеющей точный алгоритм решения.

-неструктурированные (неформализуемые) – задача, в которой невозможно выделить элементы и установить между ними связи. Решение неструктурированныхзадач из-за невозможности создания матема­тического описания и разработки алгоритма связано с большими трудностями.

-частич­но структурированные – задача, где известны лишь часть ее элементы и связей между ними. Получаемая информация анализируется че­ловеком, который будет играть определяющую роль.

Типы ИС для частично структурированных и неструктурированных задач:

-системы, создающие управленческие отчеты,

-системы, разрабатывающие альтернативные решения:

--модельные ИС предоставляют пользователю математические, статистические, финансовые и другие модели, использование которых облегчает выработку и оценку альтернатив решения,

--экспертные ИС обеспечивают оценку инициатив пользователя за счет создания экспертных систем, связанных с обработкой данных.

2.по степени автоматизации

-ручные;

-автоматизированные:

3.по сфере применения

-интегрированные системы,

-организационного управления,

-системы управления технологическими процессами,

-САПР;

4.по характеру информации:

-информационно-поисковые – производят ввод, систематизацию, хранение, выдачу информации по запросу пользователя без сложных преобразований данных;

-информационно-решающие – осуществляют все операции переработки информации по определенному алгоритму:

--управляющие – вырабатывают информацию, на основе которой человек принимает решение. Для этих систем характерны: тип задач расчетного характера и обработка больших объемов данных,

-советующие – вырабатывают информацию, которая принимается человеком к сведению и не превращается немедленно в серию конкретных действий. Эти системы обладают более высокой степенью интеллекта, т.к. для них характерна обработка знаний, а не данных;

-автоматические;

1. **Системный подход. Постулаты и принципы системного подхода. Этапы реализации.**

Системный подход как метод исходит из того положения, что любая организация, процесс рассматривается как сложное целое, как совокупность взаимосвязанных частей – функционирующих элементов, составляющих определенную систему. Основными свойствами любой системы являются измеримость и эффективность. При этом измеримость – способность системы измерять свои характеристики, а эффективность - возможность решить проблему с помощью данной системы.

**Системный подход** – это совокупность некоторых общих принципов, предопределяющих научную и практическую деятельность при анализе и синтезе сложных систем, которые вытекают из особенностей представления сложных объектов. При исследовании объекта создается его абстрактное представление в виде системы, цель которого - служить инструментом для описания, понимания и изменения рассматриваемого объекта.

Системный подход базируется на двух постулатах:

Постулат 1. Любая система может быть описана в терминах системных объектов, свойств и связей.

Постулат 2. Структура, функции системы и решение проблемы являются стандартными для любых систем и любых проблем.

К числу принципов системного подхода относятся принципы:

-цели – ориентирует на то, что, прежде всего, необходимо выявить цель (предназначение) системы

-целостности – предполагает, что исследуемый объект рассматривается или выделяется из совокупности объектов как нечто целое по отношению к окружающей среде, имеющее свои специфические функции и развивающееся по своим законам

-сложности – указывает на необходимость рассматривать объект как сложную совокупность различных элементов, находящихся в разнообразных связях между собой и со средой.

-историзма – требует, чтобы каждый объект рассматривался исторически с точки зрения того, как он возник и какие этапы прошел до момента исследования

-двойственности – предполагает, что систему необходимо рассматривать как самостоятельную систему, так и как подсистему более высокого уровня иерархии

-всесторонности – указывает на то, объект необходимо изучать со всех сторон

-множественности – утверждает, что при исследовании объекта необходимо использовать множество моделей

-динамизма – требует, чтобы все свойства объекта рассматривались как изменяющиеся

-сходства – предполагает использование ранее полученных результатов при изучении других сходных объектов

С прикладной точки зрения системный подход состоит в определенной направленности и последовательности исследования объектов, которое обычно реализуется в шесть этапов.

1 этап. Четкое определение цели исследования объекта.

2 этап. Точное и полное определение цели функционирования объекта с позиций системы более высокого уровня. Необходимо определить общие и частные цели, осуществимость, требуемые ресурсы для осуществления цели, причем все это необходимо делать взаимосвязано.

3 этап. Выделение и изучение структуры системы и среды. Выделение системы осуществляется разделением (точным) на две части. Основой отнесения элементов к системе и среде является характер связей между элементами. Так как внутренние связи значительно сильнее внешних, то это дает право относить элементы с внутренними связями к системе. Процесс выделения элементов и связей называется **структуризацией**.

4 этап. Последовательное раскрытие механизма функционирования системы.

5 этап. Система рассматривается на всех этапах жизненного цикла: происхождение, развитие, разрушение (гибель).

6 этап. Осуществляется сравнение системы с другими, в какой то степени близкими ей системами для обнаружения сходства. В случае обнаружения сходства, полученные ранее свойства переносятся или могут быть перенесены на систему и (или) наоборот.

Таким образом, системный подход к исследованию сложных объектов предполагает проведение исследования в трех взаимосвязанных аспектах: исторический анализ (генетический и прогностический); структурный анализ (анализ связей и состава); функциональный анализ (анализ внутреннего и внешнего функционирования).

1. **Методика системного анализа.**

Системный анализ это методология решения крупных проблем основанная на концепции систем.

Методика системного анализа разрабатывается и применяется в тех случаях когда у лиц принимающих решение на начальном этапе нет достаточных сведений о проблемной ситуации позволяющие выбрать метод её формализованного представления, сформировать математическую модель или применить один из подходов сочетающих качественные и количественные приёмы моделирования, в таких случаях может помочь представление объекта в виде системы, организация коллективного принятия решений с привлечением специалистов различных областей знаний с использованием разных методов, как формализованного представления систем МФПС так и МАИС со сменой методов со знанием их представления.

Приёмы и методы системного анализа направлены на выдвижение альтернатив вариантов решения проблемы, выявления масштабов неопределённости по каждому варианту и сопоставление вариантов по их эффективности. Целью системного анализа является упорядочивание последовательности действий при решении крупных проблем, основываясь на системном подходе.

Системный анализ как методологи решения проблем претендует на то, чтобы выполнять роль каркаса, объединяющего все необходимые знания методы и действия для решения проблем.

Проблемой называется разница между желаемым выходом и существующим выходом. Выход является желаемым если его отсутствие создаёт угрозу существованию или развитию системы.

Существующий выход обеспечивается существующей системой, желаемый выход желаемой системой. Решение проблемы есть то, что заполняет промежуток между существующей и желаемой системой.

Проблема характеризуется существующей в ней неопределённостью и условием.

Неизвестные могут быть выражены только в терминах известного такого как объекты свойства и связи, а известное определяется как количество значение которого установлено. Необходимо определить цели функционирования системы так как система должна быть ориентирована на достижение цели, определение целей может быть дано, только в терминах требований к системе, а требование к системе это средства фиксации однозначных утверждений определяющих цель.

Требования в терминах системных объектов, свойств связи.

Цели: сохранение состояния определяется как способность удерживать выход системы в предписанных пределах улучшения состояния системы, определяется как способность определить выход выше или другой в отличие от существующего состояния. Если известен хотя бы один способ решения проблемы, проблема превращается в задачу.

Обнаружение проблемы есть результат идентификации симптомов, после выявления проблемы, прогнозируется её развитие и оценка актуальности её решения.

Процесс нахождения решения проблемы, концентрируется вокруг итеративно выполняемых операций, идентификации условия, цели и её решения.

Необходимо так же произвести редуцирование, т.е. уменьшение числа переменных на основе анализа чувствительности проблемы к изменению отдельных переменных, агрегировать в сводные факторы и выбор подходящих форма критериев.

Укрупнённые этапы системного анализа

1. Выявление проблемы
2. Постановка цели
3. Разработка вариантов и модели принятия решений
4. Оценка альтернатив и поиск решения
5. Реализация решения
6. Оценка эффективности и в последствие их реализации
7. Проектирование организации для достижения цели

Методика системного анализа по Оптнеру

1. Идентификация симптомов
2. Определение актуальности проблемы
3. Определение цели
4. Вскрытие структуры системы и выделение её дефектных элементов
5. Определение структуры возможностей
6. Нахождение альтернатив
7. Оценка альтернатив
8. Выбор альтернативы
9. Составление решения
10. Признание решения коллективом и руководителем
11. Запуск процесса реализации процесса
12. Управление процессом реализации решения
13. Оценка реализации и его последствий

Методика по Квейду

1. Постановка задачи (определение существа проблемы, выявление целей и определение границ задач)
2. Поиск (сбор необходимой информации, определение альтернативных средств достижения цели)
3. Толкование (построение модели и её использование)
4. Рекомендация (определение предпочтительной альтернативы или курса действий)
5. Подтверждение (экспериментальная проверка решения)

Методика Янга

1. Определение цели системы
2. Выявление проблем организации
3. Исследование проблем и постановка диагноза
4. Поиск решения проблемы
5. Оценка всех альтернатив и выбор наилучшей из них
6. Согласование решения в организации
7. Утверждение решения
8. Подготовка к вводу
9. Управление применением решения
10. Проверка эффективности решения

Методика Голубкова

1. Постановка задачи
2. Исследование
3. Анализ
4. Предварительное суждение
5. Подтверждение
6. Окончательное суждение
7. Реализация принятого решения

Методика Черняка

1. Анализ проблемы
2. Определение системы
3. Анализ структуры системы
4. Формирование общей цели и критерия
5. Декомпозиция цели, выявление потребности в ресурсах и процессах
6. Выявление ресурсов и процессов, композиция цели
7. Прогноз и анализ будущих условий
8. Оценка целей и средств
9. Отбор вариантов
10. Диагноз существующей системы
11. Построение комплексной программы развития
12. Проектирование организации для достижения цели
13. **Виды и формы представления структур (сетевые, многоуровневые иерархические и др.).**
14. Сетевая структура

Представляет собой декомпозицию системы во времени, могут отображать порядок действия технической системы, этапы деятельности человека и т.д.

Сетевым структурам можно привести в соответствие матричным структурам

1. Иерархические структуры

Представляют собой декомпозицию системы в пространстве. Различают структуры со сильными и со слабыми связями. В случае слабых связей одна вершина подчинена двум или более вершинам вышестоящего уровня.

Иерархическим структурам можно привести в соответствие матричным структурам

1. Многоуровневая иерархическая система

Месарович и Такахара. Вводиться три понятия уровней для описания сложной системы:

1. Уровень описания или абстрогирования
2. Уровень сложности принимаемого решения
3. Организационный уровень

4 Матричные структуры

В форме матричного представления могут быть представлены взаимоотношения между уровнями иерархической структуры, сетевые структуры, иерархии с сильными и слабыми связями.

Матричные структуры могут быть многомерными.

5 Смешанные иерархические структуры с вертикальными и горизонтальными связями.

В системах организационного управления могут быть использованы одновременно несколько видов иерархических структур, от древовидных до много эшелонных, такие структуры называют смешанными.

6 Структуры с произвольными связями.

1. **Сети. Программирование на сетях.**

Исходные данные в СП: перечни проводимых работ, уставленные для каждой из них сроки, описание существующих между ними отношений предшествования.

**Сеть** – конечный граф без циклов и петель, ориентированный в одном, общем направлении от вершины I, называемой **истоком**, к вершине S, называемой **стоком**.

Сетевое план-е (СП) – модель системы, представленная во времени.

Для наглядности будем представлять, что по ребрам (i,j), где i<j, перемещается какой-либо груз, вещество… Максимальное количество веществ rij, которое может пропустить ребро (i,j) за единицу времени называется его **пропускной способностью**. Количество вещества, проходящее через ребро (i,j) в единицу времени xij, называется **потоком по ребру**.

Свойства потоков:

-если из вершины i в j направляется поток xij, то величина потока из j в i: xji = –xij. Кроме того xii = 0,

-величина потока не может превышать его пропускной способности: xij ≤ rij,

-количество вещества, притекающего в вершину, равно количеству вещества, вытекающего из него: ∑[j=1..n]xij = 0, где i ≠ I, S.

На практике важной является задача упорядочения элементов в графе. Под **упорядочением вершин связанного ориентированного графа** понимают такое разбиение его вершин на группы, при котором:

-вершины первой группы не имеют предшествующих, а последней – последующих,

-вершины любой группы не имеют предшествующих в следующей группе,

-вершины одной и той же группы дугами не соединены.

В результате получают граф, изоморфный данному.

Различают графический и табличный способ упорядочения вершин.

Графический способ упорядочения вершин. **Алгоритм Фалкерсона упорядочения вершин и дуг**:

-найти дуги, не имеющие непосредственно предшествующих. Они образуют первую группу,

-вычеркнуть найденные дуги. После этого появится, по крайней мере, одна дуга, не имеющая непосредственно предшествующих. Это дуги второй группы. Этот шаг повторяют до тех пор, пока все дуги не будут разбиты на группы.

Табличный способ упорядочения вершин

-составить матрицу смежности вершин,

-построить изоморфный граф.

Для решения задачи о максимальном потоке на сети, в качестве исходных данных выступает матрица смежности вершин, имеющая две разновидности.

Предположим, что задана некоторая сеть. Разобьем множество вершин этой сети на 2 пересекающихся подмножества A и B так, чтобы исток I попал в подмножество A и сток S попал в подмножество B. В этом случае говорят, что на сети произведен разрез, отделяющий I от S (A/B обозначение).

R(A/B) = ∑[i∈A]∑ [j∈B]rij X(A/B) = ∑[i∈A]∑ [j∈B]xij

**Теорема Форда-Фалкерсона.**

На любой сети максимальное количество вершин потока из истока I в сток S равно минимальной пропускной способности разреза, отделяющего I от S.

1. **Синтез сложных систем.**

Увеличение размерности систем приводит к качественно новым свойствам у них, любая сложная система может быть реализована на раличных элементах и с различными взаимосвяязями между ними. В связи с этим возникает задача синтеза при заданных ресурсах такой структуры которые максимизируют критерии качества функционирования системы.

При этом различают формальную структуру и материальную структуру.

Основными задачами при синтезе стуктурыявляються

1. Определение состава систем
2. Описание и классификация допустимых структур
3. Определение классов преобразований структур и иные варианты по отношению к заданой цели для поисков порэто оптимальных решений.
4. Анализ соотношений между формальными и материальными структурами
5. Иследование возможностей пстроение системы с переменной структурой

В практических приложениях синтез структуры разделяют на три составляющих

1. Синтез структуры системы управления (нахождение оптимального состава элементов их взаимосвязи, факторизация множества элементов по типам характеристик связей)
2. Синтез структуры управляющей системы (выбор числа уровней и подсистем, установление принципов организации управления с учётом координации цели подсистем различных уровней с глобальной целью системы, выбор организационной иерархии)
3. Синтез структуры системы передачи и обработки информации (определение связей между объектами и передаваемыми массивами информации, размещение центро обработки информации)

В зависимости от исходных данных и представлений о функционировании системы различают три класса зада синтеза

1. Синтез структуры системы при заданых алгоритмах её синтезирования
2. Синтез оптимального поведения при известной структуре
3. Синтез структуры и алгоритма функционирования распределение функций по элементам и определение их оптимального состава

Определение оптимальной структуры системы управления

Исходными данными являються

1. Выполняеме системой функции могут быть формализованны в виде множества решаемых задач, каждая из задач состоит из qэтапов
2. Связи между задачами и их этапами. Соответствует направлению информационных потоков
3. Множество возможных узлов АСУ и связи между ними.
4. Виды и характеристики комплекса технических средств применение которых возможно в системе
5. Внешние для системы источники и потребители информации

Задача определения оптимальной структуры состоит в нахождениичисла узлов М и связей между ними, возлогаемых на комплекс технических средств задач и вариантов их решений при которых максимизируеться эффект решения задач.

Предпологаеться что каждый этап задачи решается в одном узле.

Решение задачи синтеза оптимальной структуры весьма трудно, поэтому часто определяют рациональную структуру, которая обычно имеет иерархическую структуру, узлы одного уровня модно разбить на группы которые идентичны.

**Управление данными**

1. **СУБД. Функции СУБД. Типовая организация СУБД.**

СУБД – комплекс программ, предназначенная для управления БД (организации информации, помещения ее в таблицы, манипулирования ею). Основные функции СУБД:

• непосредственное управление во внешней памяти, обеспечение необходимых структур внешней памяти как для хранения данных, вводящихся в БД, так и для служебных целей (индексы). В развитых СУБД пользователь не знает особенностей организации внешней памяти.

• управление буферами оперативной памяти

• управление транзакциями (Транзакция – последовательность операций, рассматриваемых СУБД как единое целое, переводящее БД из одного целостного состояния в другое.

• журнализация СУБД должна быть в состоянии восстановить последнее состояние БД после аппаратного или программного сбоя. Для восстановления БД необходимо иметь информацию. Существует журнал – особая часть БД, недоступная пользователю СУБД, в который поступают записи обо всех изменениях основной части БД

• поддержка языков БД. В ранних – языки манипулирования, организация схемы СУБД, в современных – SQL

Типовая структура современной СУБД.

Организация соответствует набору функций СУБД. Логически в современной реляционной СУБД можно выделить наиболее внутреннюю часть - ядро СУБД (часто его называют Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения, набор утилит. В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала. Функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и проверенным протоколам. Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL (или в подсистеме поддержки выполнения таких программ) и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. При использовании архитектуры "клиент-сервер" ядро является основной составляющей серверной части системы.

Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу. Основной проблемой реляционных СУБД является то, что языки этих систем (а это, как правило, SQL) являются непроцедурными, т.е. в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не является процедурой, а лишь описывает в некоторой форме условия совершения желаемого действия (вспомните примеры из первой лекции). Поэтому компилятор должен решить, каким образом выполнять оператор языка прежде, чем произвести программу. Результатом компиляции является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением подсистемы поддержки времени выполнения, представляющей собой, по сути дела, интерпретатор этого внутреннего языка.

Наконец, в отдельные утилиты БД обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

1. **Этапы разработки базы данных.**

БД - это совокупность сведений о каких-либо объектах реального мира, предназначенных для совместного использования.

На стадии проектирования информационной системы проектировщик должен сделать:

1. обследовать предметную область

2. определить объекты и их атрибуты

3. установить все структуры и запросы, установить структурные и иерархические связи между объектами, запросные связи, начертить схему проекта

4. выработать технологию обслуживания информационной системы

5. выбрать технические и инструментальные средства для реализации проекта, реализовать проект

6. протестировать проект

На этапе 1 необходимо внимательно выслушать заказчика, необходимо прояснить следующие вопросы:

• Каковы границы предметной области?

• Будет ли предметная область изменяться?

• Каков перечень фрагментов предметной области?

• Какая информация, с какой степенью детальности нужна пользователям каждого фрагмента?

• Определить перечень пользователей и их информационные потребности?

• Какие процессы передачи, обработки данных происходят в каждом фрагменте, с какой интенсивностью?

• Какие существуют технологии накопления и обработки информации?

• Какие технические средства для реализации системы будут использоваться?

• Определить требования технологического функционирования системы, ознакомиться со всеми входными и выходными документами.

Основные этапы разработки структуры БД.

1. Инфологическая (концептуальная модель). Это описание предметной области выполненное без ориентации на используемые в дальнейшем программы и технические средства. Транслируется в даталогическую модель

Требования к этой модели:

i. адекватность отражения предметной области

ii. непротиворечивость

iii. недопустимость неосознанной трактовки модели

iv. легкая расширяемость модели (изменение)

Компоненты модели:

i. описание объектов предметной области и связей между ними (знаковая система или ER-модель)

ii. алгоритмические связи показателей

iii. описание информационных потребностей пользователей

iv. ограничения целостности

2. Даталогическая модель – трансляция концептуальной модели в логическую модель данных на языке описания данных.

3. Даталогическая модель (логическая)- описание логической связи между элементами данных без отношения к среде их содержания. Строиться датмодель в терминах и информационных единиц допустимых конкретной СУБД. Описание логической структуры данных называют схемой. Датмодель отображается затем в физической памяти.

Критерии качества логической модели данных.

- адекватность БД предметной области

- легкость разработки и сопровождения БД

-скорость выполнения операции обновления данных

-скорость выполнения операции выборки данных

**Физическая модель**. Определяет используемые запоминающие устройства. Способы физической организации данных, выбор метода доступа к данным, управление свободной памятью, описание физической структуры данных называется **схемой хранения**.

1. **ER - модель. Основные понятия.**

“Сущность – связь” (Entity - Relationship). Отображает инфологическую модель.

1 Сущность – класс однотипных объектов, информация о которых должна быть учтена в модели.

Каждая сущность в модели показывается:

2 Экземпляр сущности – конкретный представитель данной сущности. Экземпляры сущности различны, они должны иметь различные свойства. Пример: студент Иванов, 2 курс, 2 группа

3 Атрибут – поименованная характеристика, являющаяся некоторым свойством сущности или информационное отображение свойств объекта. выражается существительным в ед ч. Пример: ФИО.

4 Ключ сущности – неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности. Неизбыточный значит, что удаление любого атрибута нарушит уникальность ключа.

5 Связь – некоторая ассоциация между 2 сущностями. Позволяет по одной сущности находить другую сущность, связанную с ней.

3 типа связей:

• 1:1 (один к одному). Означает, что один экземпляр первой сущности связан с одним экземпляром второй сущности.

• 1:M (один ко многим). Один экземпляр первой сущности связан с несколькими экземплярами второй сущности. Сущность со стороны 1 – родитель, со стороны M – потомок.

• M:M (многие ко многим). Каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности.

6 Модальность - Обязательность или необязательность связи. Пример: при поступлении в вуз абитуриент обязан знать иностранный язык, но никто не обязан знать более 4х языков.

7 Класс объектов – совокупность объектов, обладающих одинаковым набором свойств.

1. **Общая характеристика реляционной модели данных. Типы данных. Простые типы данных. Структурированные типы данных. Ссылочные типы данных. Типы данных, используемые в реляционной модели. Домены. Отношения. Атрибуты. Кортежи.**

Кодд предложил реляционную модель в 1970. Дейт дает наиболее распространенную трактовку этой модели. Реляционная модель (РМ) данных состоит из 3 частей:

1. структурная. Описывает, как объекты рассматриваются реляционной моделью. Единственной структурной единицей являются формализованные отношения.
2. целостная. Описывает ограничения, которые должны выполняться для отношений
3. манипуляционная. Описывает способы манипулирования реляционными данными (реляционная алгебра и реляционные исчисления)

Типы данных:

* + простые (логические, строковые, численные)
  + структурированные (массивы, записи)
  + ссылочные (указатели)

Для реляционной модели тип данных не важен, важно, чтобы тип был простым

**Домен** – набор значений элементов данных одного типа, отвечающих поставленным условиям. D={n∈N: n>=18 and n<=60}Свойства домена:

* уникальное имя
* определен на простом типе данных или на другом домене
* может нести логическое условие для описания подмножества данных, допустимых для данных домена.
* Домен имеет определенную смысловую нагрузку. Этим он отличается от понятия подмножества.

**Атрибут отношения** – пара вида <Имя атрибута : Имя домена>. Имя атрибута должно быть уникальным.

**Отношение** *R* определенное на множестве доменов D1, D2, …Dn содержит 2 части: заголовок отношения - фиксированное количество атрибутов отношения <A1:D1><A2:D2>…<An:Dn>. Тело отношения содержит множество кортежей отношения.

Отношение “Сотрудники”: Сотрудники (Номер сотр, Фамилия, Зарплата, Номер отдела).

Каждый**кортеж отношения**представляет собой множество пар вида: <имя атрибута: значение атрибута> Отношение записывается в виде R(<A1:D1><A2:D2>…<An:Dn>); R(A1, A2,…,An) Пример:

(1, Иванов, 1000)

(2, Петров, 2000)

Количество атрибутов в отношении называется *степенью отношения* (n – арное отношение). Мощность множества кортежей называется *мощностью отношения*.

Отношения представляются в виде таблицы, но не являются таблицами. Схема реляционной БД является набор заголовков отношений.

1. **Свойства отношений. Потенциальные ключи. Целостность сущностей. Внешние ключи. Целостность внешних ключей. Стратегия поддержания ссылочной целостности.**

**Свойства отношений:**

1. в отношении нет одинаковых кортежей
2. кортежи не упорядочены
3. атрибуты не упорядочены
4. все значения атрибутов атомарны

Реляционная БД – набор взаимосвязанных отношений.

**Потенциальные ключи**– пусть дано отношение R, подмножество атрибутов K отношения R называют потенциальным ключом. Если K обладает свойствами:

* свойство уникальности
* свойство неизбыточности - никакое подмножество из K не обладает свойством уникальности.

Потенциальный ключ может быть простым и составным (сложным). Отношение может иметь несколько потенциальных ключей, один из которых первичный, все остальные альтернативные. Не понимая смысла данных, невозможно судить о том, какая совокупность атрибутов будет являться потенциальным ключом. Потенциальный ключ является средством идентификации объектов предметной области. П. кл. является единственным средством адресации на уровне кортежей, поэтому значения этих идентификаторов не могут содержать неизвестные значения.

|  |  |
| --- | --- |
| ID\_o | Tel |
| 1  2 | 222  333 |

Правило целостности сущностей:

* Атрибут, входящий в состав п. кл. не может принимать NULL значения

Пусть дано отношение R. Подмножество атрибутов FK отношения R называют **внешним ключом**, если существует отношение S с п. кл. K, и каждое значение FK в отношении R всегда совпадает со значением K для некоторого кортежа из S, либо является NULL значением.Отношение S- родительское, отношение R- дочернее. Внешние ключи не обладают свойством уникальности.

|  |  |  |
| --- | --- | --- |
| ID | Name | ID\_otd |
| 1  2  3 | Ваня  Петя  Вася | 1  2  1 |

ID\_otd – внешний ключ по отношению к ID\_o.

Правила целостности внешних ключей:

* Внешние ключи должны быть согласованными, для каждого значения внешнего ключа должно существовать значение потенциального ключа в родительском отношении.

**Нарушение ссылочной целостности:**

* Обновление кортежа в родительском отношении
* Удаление кортежа в родительском отношении
* Вставка кортежа в дочернее отношение
* Обновление кортежа в дочернем отношении

**Стратегии поддержания ссылочной целостности:**

* 1. Основные
     + RESTRICT – запрет выполнения операций, приводящих к нарушению целостности
     + CASCADE – разрешать выполнение операций, внося поправки в другое отношение так, чтобы не нарушить ссылочную целостность и сохранить все имеющиеся связи
  2. дополнительные
     + SET NULL – разрешить выполнение операции, некоторые атрибуты изменить на NULL
     + SET DEFAULT – разрешает выполнение требуемой операции но все возникающие некорректные значения внешних ключей заменяются значениями по умолчанию
     + IGNORE – не соблюдать ограничения ссылочной целостности

1. **Реляционные операторы: выборка, проекция, соединение, деление. Реализация на SQL.**

**Реляционная алгебра** – набор операторов, использующих отношения в качестве переменных и возвращающих отношение в качестве результата. R=f(R1,R2,…,Rn)

Выборка (A Where C):

Выборка на отношении A с условием C называется отношение с тем же заголовком, что и у A и телом, состоящим из кортежей, значения атрибутов которых при подстановке в условие C дают значение истина. С представляет собой логическое выражение в которое могут входить атрибуты А и скалярные выражения

Проекция (A[x,y,z]):

Проекция отношения A по атрибутам X,Y,Z, где каждый из этих атрибутов принадлежит A называется отношение с заголовком (X,Y,Z) и телом (x,y,z), состоящим из множества кортежей, таких, для которых в отношении A найдутся кортежи со значением атрибута X=x, Y=y, Z=z

Соединение ((A TIMES B) Where C):

1 - (A TIMES B) Where С - соединение отношений A и B по условию C

2 - (A TIMES B) Where X θ Y, θ - оператор сравнения, A[X θ Y]B

3 - A[X = Y]B – эквисоединие

4 - A JOIN B естественное соединение - В синтаксисе естественного соединения не указывается по каким атрибутам производится соединение. Оно производится по всем одинаковым атрибутам.

Деление (A DEVID BY B): A – делимое, B – делитель

Пусть данные отношения A(X1,X2,…Xn,Y1,Y2,…Ym) и B(Y1, Y2,…Ym) атрибуты Y являются общими для 2 отношений. Делением отношения A на B называют отношение с заголовком X1 X2 Xn и телом из кортежей (x1,x2,…,xn), таких, что для кортежей (y1,y2,…,ym)∈B в отношении A всегда найдется кортеж (x1,x2,…,xn,y1,y2,…ym)

1. **Реляционные операторы: объединение, пересечение, вычитание, декартово произведение множеств. Реализация на SQL.**

**Реляционная алгебра** – набор операторов, использующих отношения в качестве переменных и возвращающих отношение в качестве результата. R=f(R1,R2,…,Rn)

Объединение:

Объединением двух совместных по типу отношений A и B (A UNION B) называют отношение с тем же заголовком, что и отношение A и B и телом, состоящим из кортежей, принадлежащих или A или B или обоим отношениям. В результаты объединения входят все кортежы А и недостающие кортежы В.

Пересечение:

Пересечением двух совместных по типу отношений A и B (A INTERSECT B) называют отношение с тем же заголовком, что и у отношений A и B и телом, состоящим из кортежей, принадлежащих одновременно обоим отношениям.

Вычитание:

Вычитанием двух совместных по типу отношений A и B (A MINUS B) называется отношение с тем же заголовком, что и у отношений A и B и телом, состоящим из кортежей, принадлежащих отношению A и не принадлежащих отношению B

Декартово произведение (A TIMES B):

Декартовым произведением 2 отношений A(A1,A2,…,An) B(B1,B2,…,Bn) называют отношение, заголовок которого является сцеплением заголовков отношений A и B (A1,A2,…,An,B1,B2,…,Bn), а тело состоит из кортежей (a1,a2,…,an,b1,b2,…,bn) отношений A и B, таких что (a1,a2,…,an)∈A и (b1,b2,…,bn)∈B. Мощность произведения равна произведению мощностей. Если есть одинаковые атрибуты, то их необходимо переименовать.

1. **Концептуальная схема выполнения оператора SELECT. Синтаксис оператора выборки данных. Синтаксис соединенных таблиц. Синтаксис условных выражений.**

Концептуальная схема выполнения оператора SELECT.

1. вычисляется прямое декартово произведение всех таблиц, указанных в обязательном разделе FROM, получаем некоторую таблицу A.
2. если присутствует WHERE, то сканируется таблица A, для каждой строки таблицы A вычисляется условие из этого раздела; в результат включаются строки, для которых условие возвращает значение TRUE, результат: таблица B, если WHERE нет, то шаг 3
3. если присутствует GROUP BY, то строки таблицы B группируются в соответствии со списком группировки из раздела, получаем таблицу С. Если нет, то шаг 4.
4. если присутствует HAVING, то группы, не удовлетворяющие условию из этого раздела исключаются, в результате таблица D
5. каждая группа, полученная на шаге 4, генерирует одну строку результата следующим образом: вычисляются все скалярные выражения, указанные в разделе SELECT для каждой группы, вычисляются агрегатные функции, приведенные в разделе SELECT. Если GROUP BY нет, а функции есть, то считается, что имеется одна группа. Если нет агрегатных функций, ни раздела, то считается, что есть столько групп, сколько строк. В результате таблица E содержит столько колонок, сколько элементов приведено в разделе SELECT и столько строк, сколько отобрано групп. Если есть ORDER BY, то строки таблицы упорядочиваются в соответствии со списком упорядочивания.

Синтаксис оператора выборки данных:

Табличное выражение:

[ORDER BY {{имя столбца результата [ASC|DESC|}

|{положит целое [ASC|DESC}]

табл выражение := select выражение

[UNION | EXCEPT | INTERSECT | ALL }

{SELECT выражение | TABLE имя табл | конструктор значений таблицы}

SELECT выр-е:= SELECT [ALL|DISTINCT]

{{скалярное выражение | функция агрегирования |

SELECT выр-е} [AS имя столбца]}…}

|{имя табл | имя корреляции}…}

FROM {имя табл [AS] [имя корреляции]

[(имя столбца,…)]}

|SELECT выр-я [AS] имя корреляции

[(имя столбца,…)]}

| соединенная таблица}…

[WHERE условное выражение]

[GROUP BY [имя табл|имя корреляции}]

имя столбца]…}

[HAVING условное выражение];

Условное выражение вычисляется для каждой строки, являющейся кандидатом в результирующее множество строк. Можно использовать подзапросы. Раздел HAVING содержит условное выражение, вычисляющееся для каждой группы в соответствии со списком группировки в разделе GROUP BY. Если в разделе SELECT присутствует агрегатная функция и присутствует раздел GROUP BY, то агрегатная функция вычисляется для каждой группы отдельно. Если раздел отсутствует, то агрегатная функция вычисляется по всем строкам, удовлетворяющим условному выражению в разделе WHERE. Скалярное выражение – имя столбцов таблицы или функции.

Синтаксис соединенных таблиц:

Соединенная таблица :=

* перекрестное соединение - A CROSS JOIN B - декартово произведение таблицы
* естественное соединение - A NATURAL [тип соединения] JOIN B - произведение по всем столбцам таблиц, имеющим одинаковые имена.
* соединение посредством предиката - A [тип соединения] JOIN B ON предикат - соединение по общим столбцам - BETWEEN | IN | LIKE | IS NULL | EXIST | UNIQUE | MATCH | OVERLAPS
* соединение посредством имен столбцов - A [тип соед-я] JOIN B USING {имя столбца}
* соединение объединения - A UNION JOIN B – обратное к внутреннему соединению, обратно FULL

Тип соединения

INNER - соединяются те строки, для которых найдены совпадения

| LEFT [OUTER] – результат включает все строки из левой таблицы и те из правой таблицы, для которых найдены совпадения. Для строк из A, для которых не найдены совпадения в B, заносятся со значениями NULL.

| RIGHT [OUTER]

| FULL [OUTER] – все строки таблиц, для совпадающих – реальные значения, для несовпадающих – NULL.

Синтаксис условных выражений раздела WHERE:

Условное выражение:= [( ] [NOT]

{Предикат сравнения | BETWEEN | IN | LIKE | NULL | предикат количественного сравнения | EXIST | UNIQUE | MATСH | OVERLAPS | } [ { AND | OR } условное выражение] [ ) ] [IS [NOT] {TRUE | FALSE | UNKNOWN}

BETWEEN – диапазоны значения

IN – проверяет вхождение во множество

EXIST – проверяет существует ли предикат подзапроса

MATСH – проверяет будет ли значение в любой строке совпадать с результатом подзапроса.

1. **Определение функциональной зависимости. Вторая нормальная форма. Третья нормальная форма. Алгоритм приведения к 3НФ.**

Функциональная зависимость – пусть R является отношением множества атрибутов Y функционально зависит от множества атрибутов Х (Х->Y). Когда для любого состояния R во всех кортежах, имеющих одинаковое значение атрибутов Х, значение атрибутов Y так же совпадают.

r1,r2∈R

r1X=r2X следует, что r1Y=r2Y

Y- зависимая часть, Х- детерминант функциональной зависимости.

Пусть дано соотношение R с атрибутами Х и Y. Если каждому значению атрибута Х соответствует ровно одно значение Y, то Y функционально зависит от Х.

Н\_отделения -> телефон

Н\_сотрудника -> фамилия

Н\_проекта-> проект

{Н\_сотрудник,Н\_проект}->Н\_задание

Вторая нормальная форма.

Отношение находится во 2ой норм форме, если оно находиться в 1ой норм форме и отсутствует зависимость неключевых атрибутов от части сложного ключа. Чтобы устранить зависимость неключевых атрибутов от части сложного ключа, компоненты вносятся в отдельное отношение.

Декомпозиция: проекты (H\_пр, проект)

Задание (Н\_сотр, Н\_пр, Н\_задание)

Сотрудники\_отделы(Н\_сотр, фамилия, Н\_отд, телефон)

Третья нормальная форма

Отношение R находиться в 3ей нормальной форме если отношение находиться во 2ой норм форме и все неключевые атрибуты взаимно независимы.

Н\_\_отдел-> телефон

Чтобы устранить взаимозависимость неключевых атрибутов проводиться декомпозиция, при этом независимые атрибуты вносятся в отдельное отношение. Детерминант функциональной зависимости становиться ключом.

Сотрудники (Н\_сотр,фамилия, Н\_отд)

Отделы (Н\_отд, телефон)

Алгоритм привидения отношения к третьей нормальной форме

* + 1. Задается одно или несколько отношений, образующих понятие предметной области.
    2. Если в отношении обнаружена зависимость от части сложного ключа, то приводиться декомпозиция этих отношений на несколько, причем атрибуты которые зависят от части сложного ключа вносятся в отдельное отношение вместе с частью этого ключа. В исходном отношении остаются все ключевые атрибуты.

R(K1,K2, A1,A2,…,An,B1,B2,…,Bm)

{K1,K2}->{ A1,A2,…,An,B1,B2,…,Bm }

{K2}->{ B1,B2,…,Bm }

R1(K1,K2, A1,A2,…,An)

R2(K2, B1,B2,…,Bm)

* + 1. Если в некоторых отношения обнаружена зависимость некоторых неключевых атрибутов от др неключевых атрибутов, то проводиться декомпозиция этих отношений, т.е. не ключевые атрибуты, которые зависят от др не ключевых атрибутов, выносятся в отдельное отношение. В новом отношении ключом становиться детерминант функц зависимости.

R(K,A1,A2,…,An,B1,B2,…,Bm)

K->{ A1,A2,…,An,B1,B2,…,Bm }

{A1…An}->{B1…Bm}

R1(K,A1,A2,…,An)

R2(A1,A2,…,An,B1,B2,…,Bm)

1. **Нормальная форма Бойса-Кодда. Четвертая нормальная форма.**

Поставки

|  |  |  |  |
| --- | --- | --- | --- |
| PNUM | PNAME | DNUM | VOL |
| 1 | Фирма 1 | 1 | 100 |
| 1 | Фирма 1 | 2 | 200 |
| 1 | Фирма 1 | 3 | 300 |
| 2 | Фирма 2 | 1 | 150 |
| 2 | Фирма 2 | 2 | 250 |
| 3 | Фирма 3 | 1 | 1000 |

Считаем что наименование поставщиков является уникальным, каждый поставщик имеет свой уникальный номер.

2 потенциальных ключа {PNUM, DNUM}, {PNAME,DNUM}. Данные хранятся с избыточностью.

PNUM->PNAME

PNAME->PNUM

{PNUM, DNUM}->VOL

{PNUM, DNUM}->PNAME

{PNAME,DNUM}->VOL

{PNAME,DNUM}->PNUM

Отношение относиться в 3ей норм форме поскольку единственный неключевой атрибут зависит от всего ключа сразу. Очевиден его способ декомпозиции: P(PNUM,PNAME) – поставщики, PD(PNUM,DNUM,VOL)- поставки.

Отношение находиться в норм форме Бойса-Кодда, если детерминант всех функц зависимостей являются потенциальными ключами.

Правило: для того, чтобы устранить зависимость от детерминантов, не являющихся потенциальными ключами, проводиться декомпозиция, при этом детерминант и зависимые от них части вносятся в отдельное отношение.

Четвертая нормальная форма

4НФ Пусть необходимо хранить данные об абитуриентах, поступающих в ВУЗ. Каждый абитуриент имеет право сдавать экзамены на несколько факультетов одновременно. Каждый факультет имеет список сдаваемых предметов. Один и тот же предмет может сдаваться на нескольких факультетах. Абитуриент обязан сдавать предмет на факультете, независимо от сдачи этого предмета на другом факультете.

Отношения – Абитуриент.Факультет.Предмет(Абитуриент, Факультет, Предмет)

Декомпозируем отношение. В результате получим:

Абитуринты(Номер, Фамилия); Факультеты(Номер, Факультет); Предмет(Номер, Предмет); Абитуриент.Факультет.Предмет(Номер\_А, Номер\_Ф, Номер\_П)

Декомпозиция отношения А.Ф.П не может быть выполнена на основе функциональной зависимости. В отношении нет функциональных зависимостей.

Многозначная зависимость.

Пусть дано отношение R(X,Y,Z). Атрибуты Y и Z многозначно зависят от Х, если из того, что в отношении R соединяются кортежы r1=(x,y,z1) r2=(x,y1,z). Следует, что в отношении r содержится и кортеж r3=(x,y,z)

x->>y|z

Факультет(х)->>Абитуриент(y)|Предмет(z)

Для каждого факультета х каждый поступающий абитуриент у сдает один и тот же список предметов z или для каждого факультет каждый сдаваемый на факультете экзамен сдается одним и тем же списком абитуриентов.

Замечание: если в отношении R имеется не менее 3х атрибутов и х->у, то имеется многозначная зависимость x->>y|z

Многозначная зависимость атрибутов у и z от х называется нетривиальной многозначной зависимостью, если не существует х->у и х->z

Теорема Фейджина-Пусть x,y,z – непересекающееся множество атрибутов отношения R. Декомпозиция отношения R на проекции R1[x,y] и R2[x,z] является декомпозицией без потерь, когда имеется многозначная зависимость атрибутов y,z от x.

Отношение R находиться в 4ой норм форме тогда и только тогда, когда отношения находиться в нормальной форме Бойса-Кодда и не содержит нетривиальных многозначных зависимостей.

1. **Корректность процедуры нормализации - декомпозиция без потерь. Теорема Хеза.**

При декомпозиции из одного отношения получают одно или более отношений, каждое из которых содержит часть исходного отношения. В полученных новых отношениях удаляются дубликаты строк. Это означает, что декомпозиция отношений является проекциями сходного отношения, при чем эти проекции в совокупности содержат все атрибуты исходного отношения. При декомпозиции не должны теряться данные. Данные считаются не потерянными в случае, если возможна обратная операция – восстановления исходного отношения в точности в прежнем виде по декомпозированным отношениям. Операция, обратная операции проекции – это операция соединения (естественное соединение).

Проекция R [x] на множество атрибутов х является собственной, если множество атрибутов х является собственным подмножеством множества атрибутов отношения R.

Собственные проекции R1 и R2 называются декомпозицией без потерь если отношение R точно восстанавливается из них при помощи естественного соединения R1 JOIN R2 любого состояния отношения R.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| R Номер | Фамилия | | | Зарплата | | |
| 1 | Иванов | | | 1000 | | |
| 2 | Петров | | | 1000 | | |
| R1  Номер | | Зарплата | | |
| 1 | | 1000 | | |
| 2 | | 1000 | | |
| R2 Фамилия | | | Зарплата | | |
| Петров | | | 1000 | | |
| Иванов | | | 1000 | | |

|  |  |
| --- | --- |
| R2  Номер | Фамилия |
| 1 | Иванов |
| 2 | Петров |

|  |  |  |
| --- | --- | --- |
| R1 JOIN R2  Номер | Фамилия | Зарплата |
| 1 | Иванов | 1000 |
| 1 | Петров | 1000 |
| 2 | Иванов | 1000 |
| 2 | Петров | 1000 |

Теорема Хеза:

Пусть R(A,B,C) является отношением, а А,В,С- множество атрибутов этого отношения. Если имеется функциональная зависимость А->В, то проекции R1=R[A,B] и R2=R[A,C] образуют декомпозицию без потерь.

1. **Понятие транзакции. Свойства транзакций. Работа транзакций. Проблемы при работе транзакций.**

**Транзакция** – последовательность операторов манипулирования данными, выполняющаяся как единое целое и переводящая БД из одного целостного состояния в другое. Это логическая единица работы, единицы восстановления данных после сбоя.

4 свойства транзакций:

* атомарность –транзакция выполняется как атомарная, либо выполняется целиком, либо не выполняется
* согласованность – транзакция переводит БД из одного согласованного состояния в другое
* изоляция - транзакции разных пользователей не должны мешать друг другу
* долговечность - если транз выполнена, то результат ее работы должен сохраниться в БД, даже если в следующий момент произойдет сбой системы

Продолжается транзакция до наступления одного из 4 событий.

* Commit– завершить транзакцию
* Rollback – откатить
* Произошло отсоединение пользователя от СУБД
* Сбой системы

При последовательном запуске системы происходит анализ транзакций, которые можно восстановить.

Проблемы при параллельной работе транзакции.

1. Проблема потери результатов обновления. Две транзакции по очереди делают запись в одну и туже строку и фиксируют изменения. После окончания транзакции P имеет значение Р2. Транзакция А ничего не знает о транзакции В и следовательно транзакция А потеряла результаты своей работы.
2. Проблема незафиксированной зависимости (чтения грязных данных). Транзакция В изменяет данные в строке, затем транзакция А читает измененные данные и работает с ними. Транзакция В откатывается и восстанавливает измененные данные.
3. Проблема несовместимого анализа.
   1. Неповторяемое считывание. Транзакция А дважды читает одну и туже строку, между считываниями вклинивается транзакция В, изменяющая значения в строке. Транзакция А работает с данными, которые с точки зрения транзакций самопроизвольно изменяются
   2. Фантомы. Транзакция А выполняет выборку строк удовлетворяющих некоторому условию
   3. Несовместный анализ. Длинная транзакция выполняет анализ по таблице, подсчитывает сумму денег на счетах клиентов банка. Пусть на всех счетах сумма 100 денежных единиц. Короткая транзакция выполняет перевод 50 денежных единиц со счета на счет так, чтобы сумма на счетах не уменьшилась

Транзакции называются конкурирующими если они пересекаются во времени и обращаются к одним и тем же данным. В результате конкуренции транзакции возникают конфликты:

1. WRITE- WRITE. Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается изменить этот объект. В результате происходит конфликт обновления.
2. READ-WRITE. Первая транзакция прочитала объект и не закончилась. Вторая пронзакция пытается изменить объект.
3. WRITE-READ. Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается прочитать объект. В результате чтение грязных данных

Имеется 2 способа разрешить конкуренцию между поступающими в произвольные моменты времени транзакциями:

1. Обеспечить чтобы конкурирующие транзакции выполнялись в разное время. Реализуется с помощью временных меток.
2. Обеспечить чтобы конкурирующие транзакции работали с разными версиями данных. Реализуется с помощью использования журнала транзакции
3. **Ограничения целостности. Ограничения атрибута, кортежа, отношения, базы данных.**

Ограничение целостности – это некоторое утверждение которое может быть либо истинным, либо ложным в зависимости от БД. БД находиться в согласованном целостном состоянии, если выполнены все ограничения целостности, определенные для БД.

Реакции на попытку нарушения целостности:

1. отказ выполнить незаконную операцию

2. выполнение компенсирующих действий

Классификация ограничения целостности

1. По способам реализации:

a. декларативная поддержка ограничения целостности. Заключается в определении ограничений средствами языка DDL.

CREATE TABLE PERSON

(P\_ID integer primary key,

P\_NAME char(30) not null,

P\_id REFERENCES DEPART (P\_id)

ON DELETE CASCADE ON UPDATECASCADE);

b. Процедурная поддержка ограничения целостности заключается в использовании триггеров и хранимых процедур

2. По времени проверки:

a. немедленно проверяемые ограничения (пример- проверка уникальности ключа)

b. ограничение с отложенной проверкой (пример- в момент фиксации транзакции)

3. По области действия:

a. ограничение целостности домен

b. оц атрибута- представляет собой ограничение, накладываемое на значение атрибута. Совпадает с ограничениями домена

c. оц кортежа- представляет собой ограничение, накладываемое на значение конкретного кортежа и не является ограничением атрибута. Является немедленно проверяемым ограничением

d. оц отношения представляет собой ограничения накладываемые на отдельные значения отношения и не являются ограничением целостности кортежа

e. оц функциональной зависимости

f. ограничение ц БД представляет собой ограничения накладываемые на значения 2х или более связанных между собой отношений (пример- оц ссылок)

1. **Блокировки. Совместимость блокировок. Протокол доступа к данным. Решение проблем параллелизма при помощи блокировок.**

2 типа блокировок: S и Х блокировки. S – разделяемая блокировка (блокировка чтения), Х – монопольная блокировка(блокировка без взаимного доступа, блокировка записи).

|  |  |  |
| --- | --- | --- |
| Блокировки тр. А. | Блокировки тр. В. | |
| S | X |
| S | Да | Нет R-W |
| X | Нет W-R | Нет W-W |

Доступ к объектам БД осуществляется с помощью протокола БД:

1. прежде чем прочитать объект транзакция должна наложить на объект S блокировку. Прежде чем обвить объект транзакция должна наложить Х блокировку
2. если объект заблокирован S, то перед обновлением объекта S блокировка должна быть заменена на Х блокировку
3. если блокировка объекта транзакции отвергается от того, что объект заблокирован конкурирующей транзакцией, то транзакция, пытающаяся захватить объект, переходит в состояние ожидания и находиться в этом состоянии ожидания до тех пор, пока захватившая объект транзакция не освободит его
4. Х блокировки, наложенные транзакцией А, сохраняются до конца транзакции А

Проблема потери результатов обновления

|  |  |  |
| --- | --- | --- |
| S-бл. P успешно | t1 | - |
| Чтение P->P0 | t2 | - |
| - | t3 | S-бл. P успешно |
| - | t4 | Чтение P->P0 |
| x-бл отвергается | t5 | - |
| режим ожидания | t6 | x-бл отвергается, ожидание |

Обе транзакции накладывают S-бл и читают объект P. Тр.А. пытается наложить x-бл. Она отвергается, так как объект s-блокирован тр. В. Тр. А. переходит в режим ожидания до тех пор, пока тр. В. не освободит объект. В аналогичной ситуации находится тр. В. Обе транзакции ожидают друг друга и не могут продолжаться. Возникла ситуация ТУПИКА.

Проблема чтения грязных данных

|  |  |  |
| --- | --- | --- |
| - | t1 | S-блкировка строки Р, успешно |
| - | t2 | Чтение P->P0 |
| - | t3 | x-блокировка, успешно |
| - | t4 | Запись P1->P |
| S-блокировка отвергается | t5 | - |
| ожидание | t6 | Откат P0->P блокировки снимаются |
| S блокировка Р успешна | t7 | - |
| Чтение P=P0 | t8 | - |
| Фиксация | t9 | - |

Работа тр. А. притормаживается до окончания работы тр. В. После этого тр. А. продолжает работу с правильными данными. “-“ увеличение времени работы тр. А.

Проблема неповторяемого считывания

|  |  |  |
| --- | --- | --- |
| S-бл Р успешно | t1 | - |
| P=P0 | t2 | - |
| - | t3 | x-бл Р отвергается |
| - | t4 | ожидание |
| Чтение P=P0 | t5 |  |
| Фиксация, снимаются блокировки | t6 |  |
| - | t7 | x-бл успешно |
| - | t8 | Запись P1->P |
| - | t9 | фиксация |

Тр. В. притормаживается до окончания работы тр. А. В результате тр. А. дважды читает одни и те же данные правильно. После окончания тр. А. тр.В продолжает работу

Проблема фантомов

|  |  |  |
| --- | --- | --- |
| S-бл строк, удовл усл. Заблок N строк | t1 | - |
| Выборка по усл α, отобр N строк | t2 | - |
| - | t3 | Вставл строки удовл усл α |
| - | t4 | фиксация |
| Повторная выборка, соотв условию α (N+1) | t5 | - |
| Фиксация | t6 | - |

Блокировка на уровне строк не решила проблему фантомов.

Несовместимый анализ

|  |  |  |
| --- | --- | --- |
| S-бл счета P1 успешно | t1 | - |
| Чтение P1 = 100, sum=100 | t2 | - |
| - | t3 | x-бл счета Р3 успешно |
| - | t4 | Снимает деньги со счета Р3, Р3: 100->50 |
| - | t5 | x-бл счета Р1 отвергается |
| S-бл счета P2 успешно | t6 | ожидание |
| Чтение P2 =100, sum=200 | t7 | ожидание |
| S-бл P3 отвергается | t8 | ожидание |
| ожидание | t9 | ожидание |

При использовании протокола доступа к данным проблема незафиксированной зависимости разрешилась, неповторяемого чтения разрешилась, проблема фантомов не разрешилась, проблема потери результатов обновления и несовместного анализа привели к новой проблеме – ТУПИК.

2 подхода решения тупика:

1 подход: СУБД не следит за выбором транзакции жертвы

2 подход: за возникновение тупиковой ситуации следит СУБД. Строиться граф ожидания транзакции.

1. **Преднамеренные блокировки. Метод временных меток. Механизм выделения версий данных.**

При блокировании объектов разной величины возникает проблема обнаружения ранее наложенных блокировок. Протокол преднамеренных блокировок: перед тем как наложить блокировку на объект необходимо наложить специальную преднамеренную блокировку на объект, в состав которого входит блокируемый объект. Наличие блокировки таблицы будет свидетельствовать о наличии блокировки строк таблицы для другой транзакции, пытающейся блокировать целую таблицу.

Виды блокировок:

1) IS –преднамеренная блокировка с возм-ю взаимного доступа. Накл-ся на некий составной объект Т и означает намерение блок-ть входящий в Т- объект в режиме S блок-ки.

2) IХ – преднамеренная блокировка без взаимного доступа. Накл-ся на некий составной объект Т и означает намерение блокировать вход-й в него объект в режиме Х блок-ки.

3) SIХ – преднамеренная блокировка как с возможностью взаимного доступа, так и без него. Накладывается на некий составной объект Т и означает разделяемую блокировку всего этого объекта с намерением в последствии блокировать входящие в него объекты в режиме Х-блокировки

Матрица совместимости блокировок.

IS S SIX IX X

IS + + + + -

S + + - - -

SIX + - + - -

IX + - - - -

X - - - - -

Сила блокировок: Х самая сильная блокировка, а IS самая слабая блокировка.

Протокол пр блок для доступа к данным:

1) При задании Х-блок-к для сложных объектов неявно зад-ся Х-бл-ки для дочерних объектов этого об-та.

2) При задании S-блок-к или SIX для сложных объектов неявно зад-ся X-бл-ки для дочерних объектов этого об-та.

3) Перед налож-ем требуемой S или IS на об-т, зад-ся IS-бл-ка или более сильная хотя бы для одного родительского объекта.

4) Перед налож-ем требуемой X или IS, SIX на об-т, зад-ся IX-бл-ка или более сильная для всех родительских объектов.

5) Перед отменой бл-ки объекта для транз-и, д.б. отменены все бл-ки для доч-х об-в этого об-та.

Проблема фантомов решается, если транзакция А использует преднамеренную блокировку этой таблицы S или IX

Метод временных меток:

Если транзакция T1 началась раньше транзакции T2, то система обеспечивает такой режим выполнения, как если бы T1 была целиком выполнена до начала T2.

Для этого каждой транзакции T предписывается временная метка t, соответствующая времени начала транзакции. При выполнении операции над объектом r транзакция T помечает его своей временной меткой и типом операции (чтение или изменение).

Перед выполнением операции над объектом r транзакция T1 проверяет, не закончилась ли транзакция T, пометившая этот объект. 1 - Если T закончилась, T1 помечает объект r и выполняет свою операцию. 2 - Если транзакция T не завершилась, то T1 проверяет конфликтность операций. Если операции неконфликтны, при объекте r остается или проставляется временная метка с меньшим значением, и транзакция T1 выполняет свою операцию. 3 - Если операции T1 и T конфликтуют, то если t(T) > t(T1) (т.е. транзакция T является более "молодой", чем T), производится откат T и T1 продолжает работу. 4 - Если же t(T) < t(T1) (T "старше" T1), то T1 получает новую временную метку и начинается заново.

Механизм выделения версий данных:

1. Для кажд. т. зап-ся текущий системный номер (SCN); чем позже начата транзакция, тем он >.
2. При записи страниц данных фиксируется SCN транз-и, произв-ей запись. Этот номер ставится текущим в системе данных.
3. Если транзакция А читает страницу данных, то SCNа сравнивается с SCNp, читающей со страницы данных.
4. Если SCNp <= SCNa то транзакция читает эту страницу
5. Если SCNa<SCNp это означает, что некоторая транзакция В, начавшаяся позже транзакции А успела изменить и сейчас изменяет данные страницы. В этом случае транзакция А просматривает журнал транзакции назад в поиске первой записи об изменении нужной страницы данных с SCNa<SCNp. Найдя такую запись транзакция А использует старый вариант данных страницы.

|  |  |  |
| --- | --- | --- |
| A | t | B |
| Проверка SCNа>SCNp1. Чтение p1 = 100 sum = 100 | t1 | - |
|  | t2 | Х-бл. р3 успешно |
| - | t3 | Снятие денег со счета р3 : 100 -> 50 |
| - | t4 | Х-бл. р1 успешно |
| - | t5 | Помещение денег на счет р1 : 100 -> 150 |
| - | t6 | Фиксация В |
| Проверка SCN(A) > SCN(р2). Чтение р2 = 100 sum = 200 | t7 |  |
| Проверка SCN(A) < SCN(р3). Чтение старого варианта счета р3 = 100 sum = 300 | t8 | - |
| Фиксация А | t9 | - |

1. **Виды восстановления данных. Индивидуальный откат транзакции. Восстановление после жесткого сбоя. Восстановление после мягкого сбоя.**

Восстановление БД должно проводиться в следующих видах:

1) Индивидуальный откат: откат транзакции может быть инициирован самой транзанкцией путем подачи команды Rollback или системой. СУБД может инициировать откат транзакции в случае ошибки или при разрешении тупиковой ситуации

2) Мягкий сбой системы (аварийный отказ ПО). Характеризуется утратой оперативной памяти, при этом поражаются выполняющиеся на момент сбоя транзакции, теряется содержимое буферов БД, но данные, хранящиеся на диске, остаются неповрежденными. Мягкий сбой может произойти по факту отключения питания.

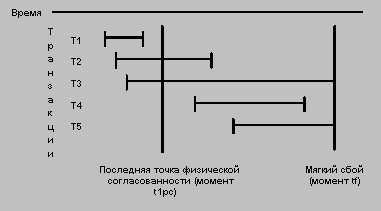
3) Жесткий сбой системы (аварийный отказ аппаратуры). Характеризуется повреждением внешних носителей памяти.

Индивидуальный откат транзакции:

Все записи в журнале от данной транзакции связываются в обратный список. Началом списка для не закончившихся транзакций является запись о последнем изменении БД, произведенном данной транзакцией. Для закончившихся транзакций (индивидуальные откаты которых уже невозможны) началом списка является запись о конце транзакции, которая обязательно вытолкнута во внешнюю память журнала. Концом списка всегда служит 1ая запись об изменении БД, произведенном данной транзакцией. Каждой записи проставляется уникальный ID транзакции, чтобы можно было восстановить прямой список записей об изменениях БД данной транзакцией.

1. Просматривается список записей, сделанных данной т. в журнале т.
2. Выбирается очередная запись из списка данной транзакции.
3. Выполняется противоположная по смыслу операция: вместо INSERT - DELETE.
4. Любая из операций журнализуются.
5. При успешном завершении отката в журнал заносится запись о конце транзакции.

Мягкий сбой : Существуют 5 вариантов состояния т. к посл. контр. точке.



T1 - закончилась до принятия контр точки. Все ее результаты отражены во внешней памяти базы данных. Никаких операций по восстановлению не требуется.

T2 – начата до контр. точки и успешно завершилась до момента мягкого сбоя. Записи журнала вытолкнуты во внешнюю память, страницы данных – частично. Нужно повторно выполнить оставшуюся часть операций.

T3 - начата до контр. точки и не завершилась к моменту сбоя. Часть страниц данных – во внеш. памяти (до контр. точки), после контр точки следов изменений нет. Записи журнала до контр точки – вытолкнуты, записи журнала после контр точки – нет. Транзакцию нужно откатить.

T4 – начата после контр точки и успела закончиться до сбоя. Записи журнала – вытолкнуты, страницы – отсутствуют. Нужно повторить транзакцию целиком.

T5 - начата после контр точки и не завершена в результате сбоя. Следов т. нет во внешней памяти, значит действий не требуется.

Восст-е с-мы после мягкого сбоя ос-ся как часть проц-ры перезагрузки системы.

1. Созд-ся 2 списка тр-й Undo и Redo. В Undo зан-ся тр-и из посл-й записи контр-й точки. Undo = {T2, T3}. Redo ост-ся пустым
2. Начиная с записи контр-й точки, просм-ся вперед журнал транз-и и если в журн обн-ся запись о начале транз-и, то она доб-ся в список Undo. Undo = {T2, T3, T4}
3. Если в ж-ле обн-ся запись Commit об ок-и т, то она доб-ся в список Redo. Redo = {T2, T4}.
4. При достижении конца журнала т списки анализ-ся. Из списка Undo удал-ся т, попавшие в список Redo.
5. Система просм-ет журнал транз-и назад с момента контр-й точки, отк-ет транз-и из списка Undo. Система просм-ет журнал транз-и вперед с контр. точки и повторно вып-ет все опер-и транз-и из списка Redo.

Жесткий сбой

БД на диске нар-ся физически. Основой восстановления являются журнал и архивная копия БД. Восстановление начинается с обратного копирования базы данных из архивной копии. Затем просм-ся журнал для вып-ия в прямом направлении всех т, которые успешно зак-ся до сбоя. Нет необходимости отката т, прерванных в результате сбоя, т к их нет в резервной копии.