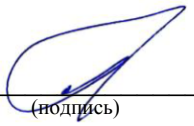
Министерство науки и высшего образования Российской Федерации

ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Шуйский филиал ИвГУ

Кафедра математики, информатики и методики обучения

ОДОБРЕНО:

Руководитель программы аспирантуры



_____ А.Н. Привалов
(подпись)

«18» марта 2026 г.

Рабочая программа дисциплины
СОВРЕМЕННЫЕ АРХИТЕКТУРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ

Уровень высшего образования:	Подготовка кадров высшей квалификации
Научная специальность:	2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей
Направленность образовательной программы:	
Срок освоения образовательной программы и форма обучения:	3 года

Шуя

1. Цели освоения дисциплины

Целью учебной дисциплины является формирование у аспирантов системы знаний в области архитектуры программного обеспечения вычислительных комплексов. В рамках данного курса предполагается получение общего представления об архитектуре программного обеспечения, процессе проектирования с учетом требований предметной области, анализе требований к предметной области, принципах построения распределенных систем обработки информации, многопоточном и параллельном программировании.

2. Место дисциплины в структуре ОП

Дисциплина «Современные архитектуры программного обеспечения вычислительных комплексов» является дисциплиной по выбору блока дисциплин «Дисциплины (модули) по выбору 2 (ДВ.2)».

Успешное освоение данной дисциплины будет способствовать готовности аспирантов к подготовке публикаций и (или) заявок на патенты на изобретения, полезные модели, свидетельства о государственной регистрации программ для электронных вычислительных машин, баз данных, топологий интегральных микросхем, подготовке к прохождению промежуточной аттестации по этапам выполнения научного исследования, итоговой аттестации.

Для освоения данной дисциплины обучающийся должен:

Знать:

– основные варианты использования компьютерных технологий в научных исследованиях.

Уметь:

– использовать информационные технологии при сборе и анализе данных;

– аккумулировать необходимую информацию для реализации определенных практических задач.

Владеть:

– современной терминологией в области проектирования информационных систем и программирования.

3. Планируемые результаты обучения по дисциплине

В результате освоения дисциплины обучающийся должен:

Знать:

– архитектуру электронно-вычислительных систем и программного обеспечения комплексов;

– назначение и принципы действия отдельных архитектурных конфигураций;

– общие понятия и принципы построения распределенных систем обработки информации;

– методологию и паттерны проектирования программного обеспечения вычислительных комплексов;

– методы организации связи между процессами;

– методы и средства проектирования структуры программного обеспечения;

– методологию и шаблоны многопоточного и параллельного программирования.

Уметь:

– выбирать рациональную конфигурацию вычислительных комплексов и оборудования;

– обеспечивать совместимость и синхронизацию аппаратных и программных средств вычислительных комплексов;

– выбирать средства реализации высокопроизводительных приложений, проектировать высокопроизводительные приложения с применением шаблонов параллельного программирования.

Иметь опыт:

- выбора структуры вычислительных комплексов и организации их функционирования;
- применения методов и средств распределения задач по вычислительным элементам;
- оценки проектируемой архитектуры с точки зрения надежности, производительности и масштабируемости.

4. Объем и содержание дисциплины

Объем дисциплины составляет 2 зачетных единицы (72 академических часа)

4.1. Содержание дисциплины по разделам (темам), соотнесенное с видами и трудоемкостью занятий лекционно-семинарского типа

№	Разделы (темы) дисциплины	Семестр	Виды занятий, их объем (в акад. часах, по очной форме обучения)			Формы текущего контроля успеваемости (по очной форме обучения) Формы промежуточной аттестации
			Занятия лекцион-ного типа	Занятия семинарского типа	Самостоятельная работа обучающихся	
1.	Введение в область организации вычислительных комплексов и систем	3	6	4 практические	8	Практические задания
2.	Архитектура программного обеспечения	3	6	4 практические	12	Практические задания
3.	Принципы построения распределенных систем обработки информации	3	4	8 практические	12	Практические задания
4.	Многопоточное и параллельное программирование	3	2	2 практические	4	Практические задания
5.	Заключительный. Подведение и анализ промежуточных результатов освоения дисциплины	3	6	4 практические	8	Практические задания
6.	Итого за семестр:		18	18	36	Зачет с оценкой

4.2. Развернутое описание содержания дисциплины по разделам (темам)

Тема 1 – Введение в область организации вычислительных комплексов и систем

Вычислительные системы и комплексы. Классификация вычислительных систем. Многопроцессорные системы. Архитектура вычислительных систем. Комплексирование в вычислительных системах. Уровни и средства комплексирования. Типовые структуры вычислительных комплексов и систем. Кластеры. Кластеризация. Понятие масштабируемости, надежности, суммарной производительности. Эффективное управление и контроль работы системы. Организация функционирования вычислительных систем. Операционные системы многомашинных ВС. Программное обеспечение многопроцессорных ВС.

Тема 2 – Архитектура программного обеспечения

Понятие архитектуры программного обеспечения. Архитектурные паттерны ПО. Проектирование архитектуры ПО. Укрупненное проектирование ПО. Формирование архитектуры программной системы. Оценка качества архитектур программной системы.

Анализ требований. Взаимодействия классов без учета состояний. Взаимодействие классов с учетом состояний. Диаграммы состояний (конечные автоматы).

Технологии структурного анализа требований: SADT, IDEF0. Технологии структурного проектирования: IDEF1X, DFD. Объектно-ориентированное проектирование с использованием языка моделирования UML.

Понятие паттерна проектирования. Типы паттернов проектирования. Основные паттерны проектирования. Проектирование в конкретных классах и проектирование в интерфейсах. Принципы проектирования компонентов.

Тема 3 – Принципы построения распределенных систем обработки информации

Введение в распределенные системы. Технология «клиент-сервер». Серверы приложений и прикладные протоколы. Представление данных в распределенных системах. Принципы построения серверного программного обеспечения. Основные задачи, выполняемые серверными программами. Инструментальные программные средства (ИПС) создания программ, выполняемых на стороне сервера. Технологии обеспечения коммуникаций в распределенных ИС. Технологии распределенных БД в распределенных ИС.

Тема 4 – Многопоточное и параллельное программирование

Современное состояние проблемы. Создание/завершение потоков. Примитивы синхронизации. Алгоритмы синхронизации. Атомарные снимки регистров. Ошибки параллельного программирования. Шаблоны параллельного программирования. Кластерные вычисления. Консенсус. Сети Петри.

Управление памятью в многопоточных системах. Оптимизации в компиляторах. Транзакционная память. Асинхронный ввод/вывод. Wait-free MRMW снимок регистров. Средства поиска ошибок. Модель памяти. Lock-free схемы управления памятью. Модель акторов.

Тема 5 – Заключительный. Подведение и анализ промежуточных результатов освоения дисциплины

5. Образовательные технологии

В программе данного курса предусмотрено использование в учебном процессе активных форм проведения занятий и организации самостоятельной работы. В учебном процессе предусмотрено использование технологий проблемного обучения.

Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине: технологии смешанного обучения, мультимедиа технологии, облачные технологии, технологии визуализации (инфографика, дашборды).

6. Учебно-методическое обеспечение самостоятельной работы обучающихся

Самостоятельная работа обучающихся по дисциплине осуществляется в следующих формах:

- повторение материала с использованием конспектов лекций;
- изучение и анализ предложенных электронных образовательных ресурсов, посвященных архитектуре программного обеспечения вычислительных комплексов;
- реферирование и аннотирование учебной и научно-технической литературы, содержащей информацию об архитектуре программного обеспечения вычислительных комплексов;
- выполнение заданий, из которых складывается портфолио студента, включающее обязательные и дополнительные (определяемые студентами совместно с преподавателем) материалы.

Задания и результаты их выполнения составляют часть информационного ресурса кафедры математики, информатики и методики обучения и хранятся на сервере кафедры, доступ к ним обеспечен с компьютеров в компьютерном классе.

Полностью весь методический материал по обеспечению самостоятельной работы студентов приводится в приложении 1 к настоящей программе. Доступ к методическим материалам осуществляется при помощи электронной информационно-образовательной среды Шуйского филиала ИвГУ.

7. Характеристика оценочных средств для текущего контроля успеваемости и промежуточной аттестации обучающихся по дисциплине

Промежуточная аттестация по дисциплине (зачет с оценкой) проводится с использованием рейтинговой оценки.

Промежуточная аттестация аспирантов предполагает зачет с оценкой в форме выполнения тестового задания, примерный вариант которого представлен в Приложении 2.

Критерии оценки:

– оценка «отлично» выставляется аспиранту, если аспирант выполнил тестового задание на 85 или более процентов;

– оценка «хорошо» выставляется аспиранту, если аспирант выполнил тестового задание на 71 или более процентов;

– оценка «удовлетворительно» выставляется аспиранту, если аспирант выполнил тестового задание на 61 или более процентов;

– оценка «неудовлетворительно» выставляется аспиранту, если аспирант выполнил тестовое задание на менее чем 61 процент;

8. Учебно-методическое и информационное обеспечение дисциплины

Основная литература:

1. Архитектуры и топологии многопроцессорных вычислительных систем: курс лекций : учебное пособие / А. В. Богданов, В. В. Корхов, В. В. Мареев, Е. Н. Станкова. – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2004. – 176 с. : ил.,табл., схем. – (Основы информационных технологий). – ISBN 5-9556-0018-3; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=232995

2. Бабич, А. В. UML. Первое знакомство: Пособие для подготовки к сдаче теста UM0-100 (OMG Certified UML Professional Fundamental) : учебное пособие / А. В. Бабич ; Национальный Открытый Университет "ИНТУИТ". – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ) : Бином. Лаборатория знаний, 2008. – 176 с. : ил. – (Основы информационных технологий). – ISBN 978-5-94774-878-9; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=233305

3. Хританков, А. С. Проектирование на UML: сборник задач / А. С. Хританков, В. А. Полежаев, А. И. Андрианов. – 3-е изд. стер. – Москва ; Берлин : Директ-Медиа, 2018. – 240 с. : ил. – Режим доступа: по подписке. – ISBN 978-5-4475-9493-0. – DOI 10.23681/483549; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=483549

Дополнительная литература:

1. Иванов, И. П. Программные средства обработки результатов расчетов в инженерных пакетах Ansys CFX и Abaqus для высокопроизво- дительных вычислительных установок : учебное пособие / И. П. Иванов, А. М. Чеповский ; Московский государственный технический университет им. Н.Э. Баумана. – Москва : МГТУ им. Н.Э. Баумана, 2009. – 191 с. : ил., табл. – ISBN 978-5-7038-3321-6; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=257478

2. Крищенко, В. А. Поддержка разработки распределенных приложений в Microsoft .NET Framework : / В. А. Крищенко, С. В. Горин. – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2007. – 316 с.; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=234143

3. Крищенко, В. А. Технологии создания кросс-платформенных распределенных приложений : учебное пособие : / В. А. Крищенко ; Московский государственный технический университет им. Н.Э. Баумана. – Москва : МГТУ им. Н.Э. Баумана, 2009. – 40 с. – ISBN 978-5-7038-3316-2; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=256822

4. Миков, А. И. Распределенные системы и алгоритмы: курс : учебное пособие / А. И. Миков, Е. Б. Замятина. – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2008. – 377 с.; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=234167

5. Распределенные базы данных: лабораторный практикум : практикум / авт.-сост. Н. Ю. Братченко ; Северо-Кавказский федеральный университет. – Ставрополь : Северо-Кавказский Федеральный университет (СКФУ), 2014. – 180 с. : ил.; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=457596

6. Царев, Р. Ю. Программные и аппаратные средства информатики : учебник / Р. Ю. Царев, А. В. Прокопенко, А. Н. Князьков ; Сибирский федеральный университет. – Красноярск : Сибирский федеральный университет (СФУ), 2015. – 160 с. : табл., схем., ил. – ISBN 978-5-7638-3187-0; То же [Электронный ресурс]. – URL: https://biblioclub.ru/index.php?page=book&id=435670

7. Червенчук, И. В. Моделирование объектно ориентированных систем с помощью UML : учебное пособие : / И. В. Червенчук, А. С. Грицай ; Омский государственный технический университет. – Омск : Омский государственный технический университет (ОмГТУ), 2019. – 108 с. : ил., табл., схем. – ISBN 978-5-8149-3012-5; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=682248

Ресурсы информационно-телекоммуникационной сети «Интернет»:
Профессиональные базы данных и информационно-справочные системы:
ЭБС «Университетская библиотека онлайн» www.biblioclub.ru
Национальный цифровой ресурс Руконт - https://rucont.ru/default.aspx
Электронная библиотека ИвГУ http://lib.ivanovo.ac.ru
Электронный каталог НБ ИвГУ http://lib.ivanovo.ac.ru/index.php/ek

Программное обеспечение: операционная система Microsoft Windows, пакет офисных программ Microsoft Office и(или) LibreOffice, интернет-браузер Microsoft Edge и(или) Yandex Browser.

9. Материально-техническое обеспечение дисциплины

Учебные аудитории:

– для проведения занятий лекционного типа с комплектом специализированной учебной мебели и техническими средствами обучения, служащими для предоставления учебной информации большой аудитории;

– для проведения занятий семинарского типа, консультаций, текущего контроля и промежуточной аттестации с комплектом специализированной учебной мебели и техническими средствами обучения.

Помещение для самостоятельной работы, оснащенное комплектом специализированной учебной мебели, компьютерной техникой с возможностью подключения к сети «Интернет» и обеспечением доступа в ЭИОС.

Демонстрационное оборудование и учебно-наглядные пособия для занятий лекционного типа, обеспечивающие тематические иллюстрации: электронные пособия (презентации), аудио-визуальные пособия (видеоматериалы), печатные пособия (таблицы, схемы)).

Автор(ы) рабочей программы дисциплины: профессор кафедры математики, информатики и методики обучения, доктор технических наук, профессор Привалов А. Н.

Программа рассмотрена и утверждена на заседании кафедры математики, информатики и методики обучения «02» марта 2022 г., протокол № 8

Программа обновлена
протокол заседания кафедры № 8 от «22» марта 2023 г.
Согласовано:
Руководитель ОП _____ А.Н. Привалов
(подпись)

Программа обновлена
протокол заседания кафедры № 7 от «9» февраля 2024 г.
Согласовано:
Руководитель ОП _____ А.Н. Привалов
(подпись)

Программа обновлена
протокол заседания кафедры № 9 от «20» марта 2025 г.
Согласовано:
Руководитель ОП _____ А.Н. Привалов
(подпись)

Программа обновлена
протокол заседания кафедры № 9 от «18» марта 2026 г.
Согласовано:
Руководитель ОП _____ А.Н. Привалов
(подпись)

Министерство науки и высшего образования Российской Федерации

ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Шуйский филиал ИвГУ

Кафедра математики, информатики и методики обучения

Приложение 1
к рабочей программе дисциплины

Методические указания
для обучающихся по освоению дисциплины
СОВРЕМЕННЫЕ АРХИТЕКТУРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ

Уровень высшего образования:	Подготовка кадров высшей квалификации
Научная специальность:	2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей
Направленность образовательной программы:	
Срок освоения образовательной программы:	3 года

Шуя

1. Методические указания к проработке лекционного материала по дисциплине

Общий план лекций включает следующие составные элементы:

Название этапа	**Содержание**	**Время, мин.**
Вступительная часть	Приветствие. Контроль посещаемости. Постановка темы, цели, основных задач.	4-7
Актуальность и основные понятия	Формулирование проблемы; основные понятия и суть изучаемого вопроса.	8-10
Основная часть	Представление материала лекции в соответствии с программой	60-65
Закрепление полученной информации	Вопрос-ответ, фронтальный опрос, разбор ошибок, допущенных при ответах	5-8
Подведение итогов	Сопоставление результатов лекции с установленной целью и намеченными задачами.	3

В рамках проведения занятий по дисциплине предполагается чтение следующих лекций:
Лекция 1. Введение в область организации вычислительных комплексов и систем
Тезисы лекции:

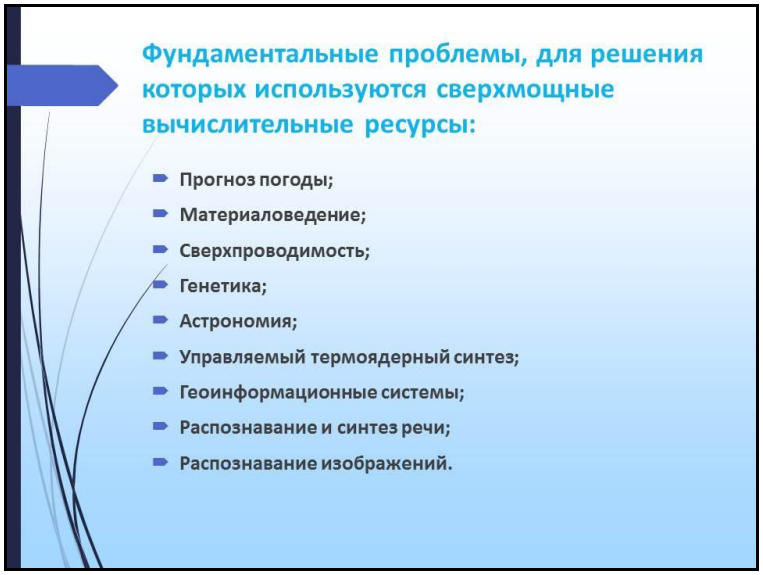
- Вычислительные системы и комплексы. Классификация вычислительных систем.
- Многопроцессорные системы. Архитектура вычислительных систем.
- Комплексирование в вычислительных системах. Уровни и средства комплексирования.
- Типовые структуры вычислительных комплексов и систем. Кластеры. Кластеризация.
- Понятие масштабируемости, надежности, суммарной производительности. Эффективное управление и контроль работы системы. Организация функционирования вычислительных систем.
- Операционные системы многомашинных ВС. Программное обеспечение многопроцессорных ВС.

Слайды презентации к лекции:








Фундаментальные проблемы, для решения которых используются сверхмощные вычислительные ресурсы:
Прогноз погоды;
Материаловедение;
Сверхпроводимость;
Генетика;
Астрономия;
Управляемый термоядерный синтез;
Геоинформационные системы;
Распознавание и синтез речи;
Распознавание изображений.


Многопроцессорные вычислительные системы (МВС) существуют в различных конфигурациях:
Системы высокой надежности (кластерные системы, катастрофоустойчивые системы, системы высокой готовности и т.д.);
Системы для высокопроизводительных вычислений (мейнфреймы, суперкомпьютеры);
Многопоточные системы (для обеспечения единого интерфейса к ряду ресурсов, например, группа WEB-серверов).


Главная отличительная особенность МВС – её производительность (кол-во операций, производимых системой за единицу времени):
Пиковая производительность (ПП): величина, равная произведению пиковой производительности одного процессора на число процессоров в машине.
ПП=ПП одного процессора x N процессоров
Реальная производительность: производительность, достигаемая на конкретном приложении. Она зависит от взаимодействия программной модели, в которой реализовано приложение, с архитектурными особенностями машины, на которой приложение запускается.

Существует 2 способа оценки пиковой производительности:

Производительность, выраженная в миллионах инструкций в секунду (MIPS, Million Instractions Per Second).

- Это показатель, поясняющий скорость выполнения компьютером своих же инструкций, однако, каждая программа обладает своей спецификой, то есть MIPS – дает общее представление о возможностях компьютера.

Производительность, выраженная в числе операций с плавающей точкой, производимых компьютером за 1 секунду Flops (Floating point operations per second).

- Указанная единица является более приемлемой для пользователя и представляет собой нижнюю оценку времени выполнения операции.

Особенности оценки производительности МВС:

- Пиковая производительность имеет место только в **идеальных условиях**, т.е. при отсутствии конфликтов при обращении к памяти.
- В реальных условиях на выполнение конкретной программы влияют такие **особенности компьютера** как: структура процессора, система команд, состав функциональных устройств и др.
- Одним из **определяющих факторов**, влияющих на производительность является **время взаимодействия с памятью**, которое определяется строением, объемом и архитектурой подсистем доступа к памяти.
- В современных компьютерах для повышения эффективности доступа используется **многоуровневая иерархическая память**, включающая: регистры и регистровую память, основную оперативную память, cash-память, виртуальные и жесткие диски.

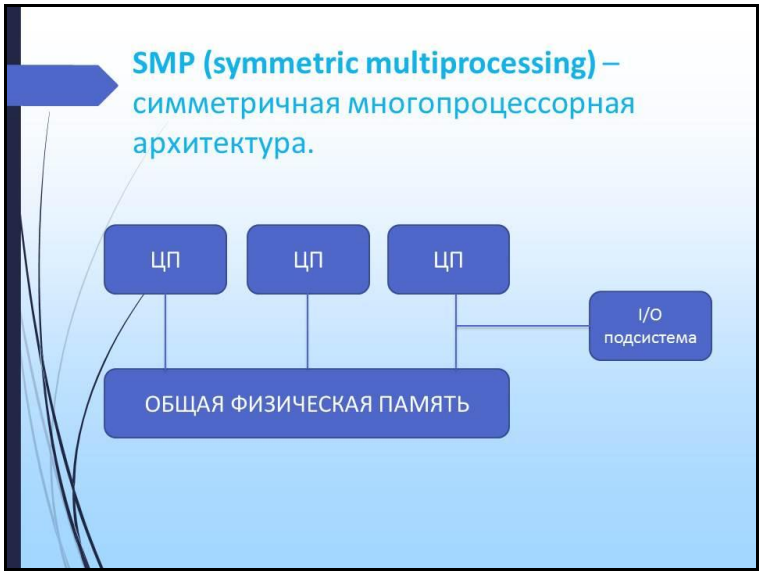
В 1966 году М. Флинном был предложен удобный подход к классификации архитектур вычислительных систем:

SISD (single instruction stream/single data stream) – одиночный поток команд и одиночный поток данных.

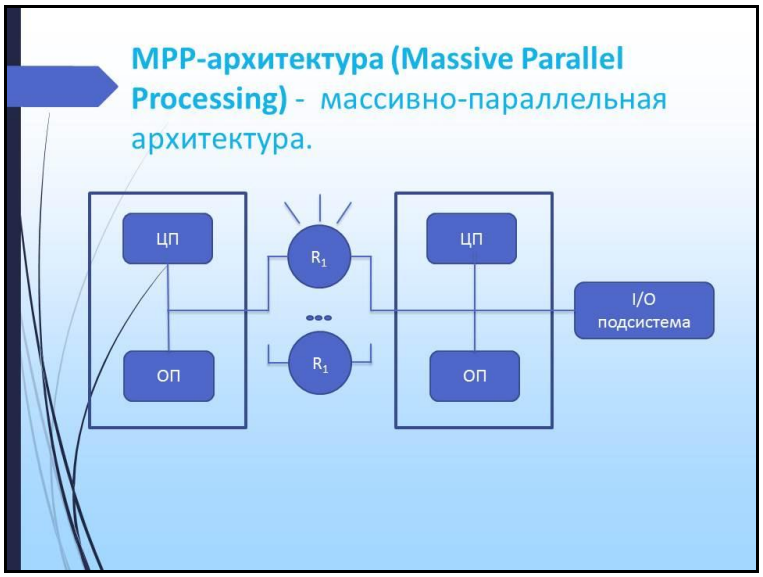
MISD (multiple instruction stream/single data stream) – множественный поток команд и одиночный поток данных.

SIMD (single instruction stream/multiple data stream) – одиночный поток команд и множественный поток данных.

MIMD (multiple instruction stream/multiple data stream) – множественный поток команд и множественный поток данных.


SMP (symmetric multiprocessing) – симметричная многопроцессорная архитектура.
ЦП
ЦП
ЦП
I/O подсистема
ОБЩАЯ ФИЗИЧЕСКАЯ ПАМЯТЬ


Симметричность архитектуры объясняется тем, что устройства имеют равные права и одну и туже адресацию для всех ячеек памяти.
Достоинства SMP –архитектуры:
- простота и универсальность для программирования;
- простота эксплуатации;
- использование общей памяти увеличивает скорость обмена;
- наличие средств эффективного распараллеливания решения задач.
Недостатки SMP –архитектуры:
- Системы с общей память плохо масштабируются.

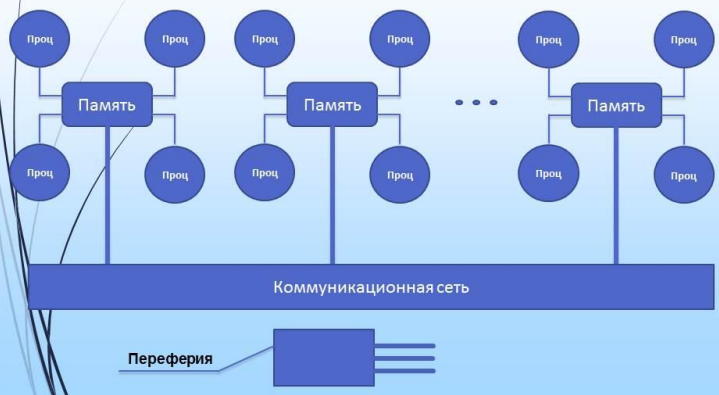

MPP-архитектура (Massive Parallel Processing) - массивно-параллельная архитектура.
ЦП
ОП
R1
R1
ЦП
ОП
I/O подсистема

Особенности архитектуры MPP:

- Система строится из отдельных модулей (каждый модуль представляет собой полнофункциональный компьютер);
- Память физически разделена;
- Модули соединяются специальными коммуникационными каналами;
- Высокая масштабируемость.

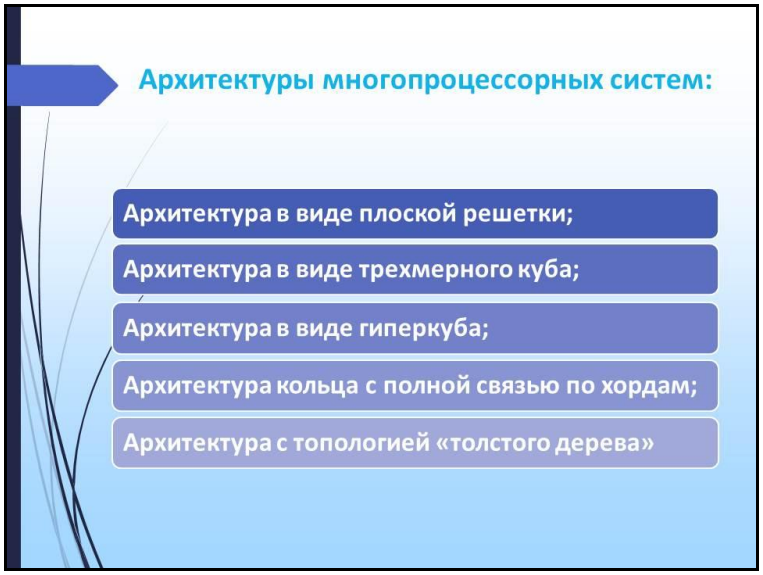
Недостатки:
- отсутствие общей памяти снижает скорость межпроцессорного обмена;
- Каждый процессор может использовать ограниченный объем памяти.

Гибридная архитектура NUMA (nonuniform memory access) – с неоднородным доступом к памяти.

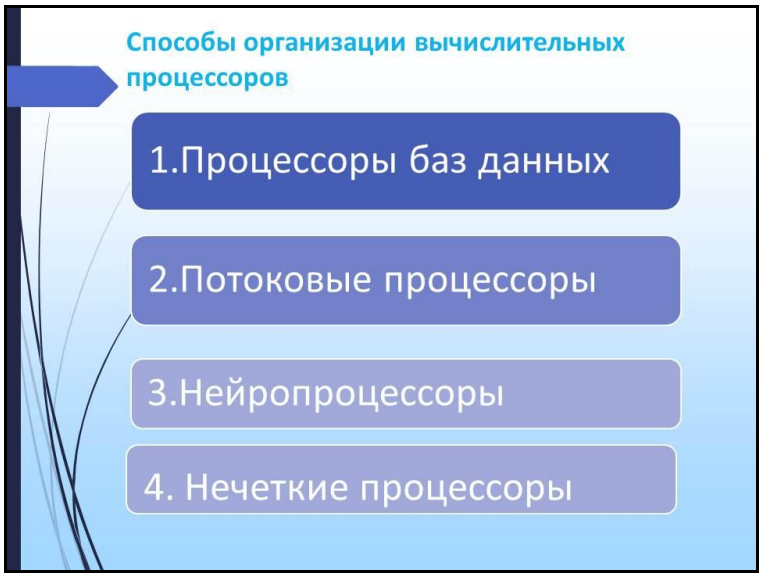


PVP (Parallel Vector Process) – параллельная архитектура с векторным процессором.

- 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.
- Парадигма программирования на PVP системах предусматривает **векторизацию циклов** (для достижения необходимой производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).


Архитектуры многопроцессорных систем:
Архитектура в виде плоской решетки;
Архитектура в виде трехмерного куба;
Архитектура в виде гиперкуба;
Архитектура кольца с полной связью по хордам;
Архитектура с топологией «толстого дерева»


1. Назначение , область применения и способы оценки производительности МВС.
2. SMP, MPP, NUMA, PVP - архитектуры.
3. Способы организации высокопроизводительных процессоров.


Способы организации вычислительных процессоров
1.Процессоры баз данных
2.Потоковые процессоры
3.Нейропроцессоры
4. Нечеткие процессоры

1. Процессоры баз данных

- **Процессорами(машинами) баз данных** называют программно-аппаратные комплексы, предназначенные для выполнения всех или некоторых функций систем управления базами данных (СУБД).
- Процессоры баз данных **выполняют функции**:
  - управления;
  - обеспечения дистанционного доступа к информации через шлюзы,
  - репликации обновленных данных с помощью различных механизмов тиражирования (копирования данных из одного источника на другой или на множество других и наоборот).

Процессоры баз данных обеспечивают **построение клиент-серверных архитектур (двухуровневых).**

2. Потоковые процессоры

- Потоковыми называют процессоры, в основе которых лежит принцип **обработки многих данных с помощью одной команды (SIMD).**
- Потоковые процессоры подразделяются на:
  - **отдельные потоковые процессоры** (SSP – single-streaming processor);
  - **многопотоковые процессоры** (MSP – Multi-Streaming Processor).
- Пример потокового процессора – семейство процессоров INTEL, начиная с Pentium 3. В основе функционирования процессоров Intel лежит технология SSE (streaming SIMD extensions – **потоковая обработка по принципу «одна команда-много данных»).** Используется для обработки речи, трехмерной графики изображений.
- **Векторные процессоры** – представитель SIMD класса потоковых процессоров. Векторная обработка **повышает производительность процессора за счет обработки набора данных (вектора) одной командой.**

3. Нейропроцессоры. Области применения нейросетей:

- **Прогнозирование;**
- **Распознавание образов;**
- **Классификация;**
- **Кластеризация и др.**

Отличия нейросетей от традиционных вычислительных систем:

- **Высокая скорость обработки данных;**
- **Высокий уровень отказоустойчивости;**
- **Возможность обучения.**

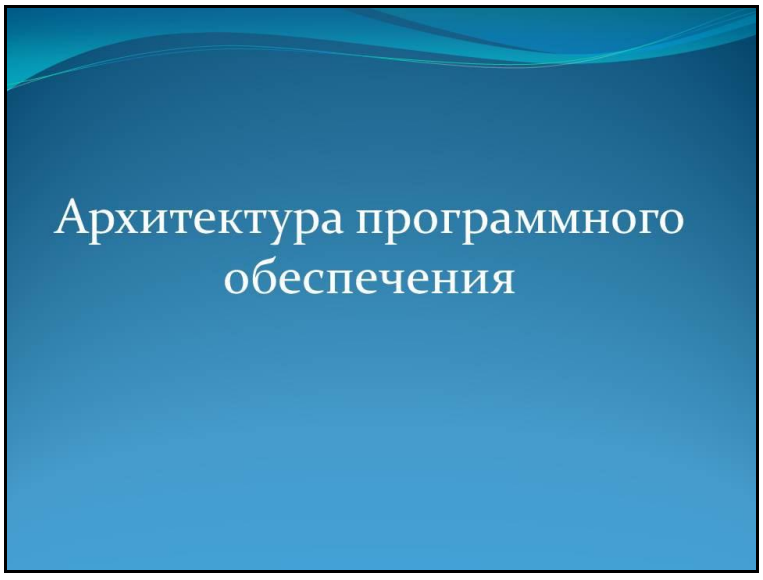
Литература: Бабич, А. В. UML. Первое знакомство: Пособие для подготовки к сдаче теста UM0-100 (OMG Certified UML Professional Fundamental) : учебное пособие / А. В. Бабич ; Национальный Открытый Университет "ИНТУИТ". – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ) : Бином. Лаборатория знаний, 2008. – 176 с. : ил. – (Основы информационных технологий). – ISBN 978-5-94774-878-9; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=233305

Лекция 2. Архитектура программного обеспечения

Тезисы лекции:

- Понятие архитектуры программного обеспечения. Архитектурные паттерны ПО. Проектирование архитектуры ПО. Укрупненное проектирование ПО. Формирование архитектуры программной системы. Оценка качества архитектур программной системы.
- Анализ требований. Взаимодействия классов без учета состояний. Взаимодействие классов с учетом состояний. Диаграммы состояний (конечные автоматы).
- Технологии структурного анализа требований: SADT, IDEF0. Технологии структурного проектирования: IDEF1X, DFD. Объектно-ориентированное проектирование с использованием языка моделирования UML.
- Понятие паттерна проектирования. Типы паттернов проектирования. Основные паттерны проектирования. Проектирование в конкретных классах и проектирование в интерфейсах. Принципы проектирования компонентов.

Слайды презентации к лекции:



Что такое паттерны (шаблоны) проектирования?

- Эффективные способы решения характерных задач проектирования
- Обобщенное описание решения задачи, которое можно использовать в различных ситуациях
- ОО паттерны проектирования часто показывают отношения и взаимодействия между классами и объектами
  - Алгоритмы не являются паттернами, т.к. решают задачу вычисления, а не программирования

Польза

- Каждый паттерн описывает решение целого класса проблем
- Каждый паттерн имеет известное имя
  - облегчается взаимодействие между разработчиками
  - Правильно сформулированный паттерн проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова
- Шаблоны проектирования не зависят от языка программирования, в отличие от идиом

Порождающие паттерны проектирования

- Абстрагируют процесс инстанцирования объектов
- Список паттернов
  - Абстрактная фабрика (Abstract Factory)
  - Строитель (Builder)
  - Фабричный метод (Factory method)
  - Прототип (Prototype)
  - Одиночка (Singleton)

Паттерн «Абстрактная фабрика»

- Шаблон проектирования, позволяющий изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы
- Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов

Реализация паттерна

- Паттерн «Абстрактная фабрика» реализуется созданием абстрактного класса **Factory**, который представляет собой интерфейс для создания абстрактных объектов-продуктов
- На основе данного класса создается один или несколько классов конкретных фабрик, создающих конкретные объекты-продукты

Применение паттерна

- Используйте паттерн, когда
  - Система не должна зависеть от того, как создаются, компонуются и представляются входящие в нее объекты
  - Входящие в семейство взаимосвязанные объекты должны использоваться вместе и вам необходимо обеспечить выполнение этого ограничения
  - Система должна конфигурироваться одним из семейств составляющих ее объектов
  - Требуется предоставить библиотеку объектов, раскрывая только их интерфейсы, но не реализацию.

Структура



Пример использования

- При разработке приложений с графическим интерфейсом пользователя необходимо создавать различные элементы управления
  - Кнопки, текстовые поля, радио-кнопки, выпадающие списки и т.п.
- Их создание и работа с ними в различных ОС осуществляется по-разному
  - Чтобы приложение можно было легче перенести в другую ОС в нем не должно быть жесткой привязки к типам конкретных классов элементов управления
  - Паттерн «Абстрактная фабрика» облегчает решение данной задачи

Преимущества использования паттерна «Абстрактная фабрика»

- Изоляция конкретных классов продуктов
  - Фабрика изолирует клиента от деталей реализации классов продуктов
    - Имена изготавливаемых классов известны только конкретной фабрике, в коде клиента они не упоминаются
    - Клиенты манипулируют экземплярами продуктов только через их абстрактные интерфейсы
- Упрощение замены семейств продуктов
  - Приложение может изменить семейство продуктов просто подставив новую конкретную фабрику
- Гарантия сочетаемости продуктов
  - Если продукты спроектированы для совместного использования, важно чтобы в каждый момент времени приложение работало с продуктами единственного семейства

Недостатки паттерна «Абстрактная фабрика»

- Трудность поддержания новых типов продуктов
  - Интерфейс абстрактной фабрики фиксирует набор продуктов, которые можно создать
  - Для поддержки новых продуктов необходимо расширить интерфейс фабрики, внеся изменения в класс AbstractFactory, а также во все его подклассы
- Обход этого ограничения – передача идентификатора типа создаваемого продукта в методы фабрики, создающие продукты
  - Ограничение: создаваемые продукты должны поддерживать общий абстрактный интерфейс
    - Клиент не может различать типы продуктов и делать какие-либо предположения о них

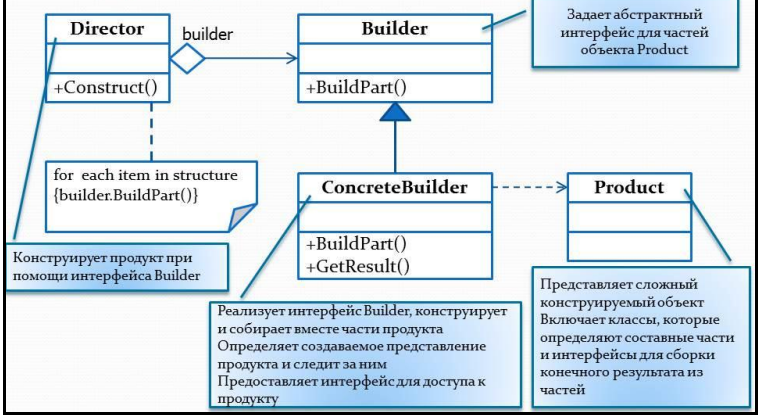
Назначение паттерна «Строитель»

- Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления

Область применения паттерна «Строитель»

- Алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой
- Процесс конструирования должен обеспечивать различные представления конструируемого объекта

Структура



Отношения между участниками паттерна

- Клиент создает новый объект «Распорядитель» (Director) и конфигурирует его новым объектом-строителем Builder
- Распорядитель уведомляет строителя о необходимости построения очередной части продукта
- Строитель обрабатывает запросы распорядителя и добавляет новые части к продукту
- Клиент забирает продукт у строителя

Достоинства паттерна «Строитель»

- Позволяет изменять внутреннее представление продукта
  - Распорядителю предоставляется абстрактный интерфейс Строителя, скрывающего структуру продукта и процесс сборки
    - Для изменения внутреннего представления достаточно определить новую реализацию Строителя
- Изолирует код, реализующий конструирование и представление
  - Клиенту не нужно знать о классах, задающих внутреннюю структуру продукта (в интерфейсе строителя они отсутствуют)
- Дает более тонкий контроль над процессом конструирования
  - Процесс построения продукта происходит не сразу, как в других порождающих паттернах, а шаг за шагом

Назначение паттерна «Фабричный метод»

- Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать
  - Фабричный метод позволяет классу делегировать инстанцирование подклассам
- Альтернативное название паттерна – «Виртуальный конструктор»

Структура



Отношения между участниками паттерна

- **Создатель** «полагается» на свои **подклассы** в определении фабричного метода, который будет возвращать экземпляр подходящего конкретного продукта

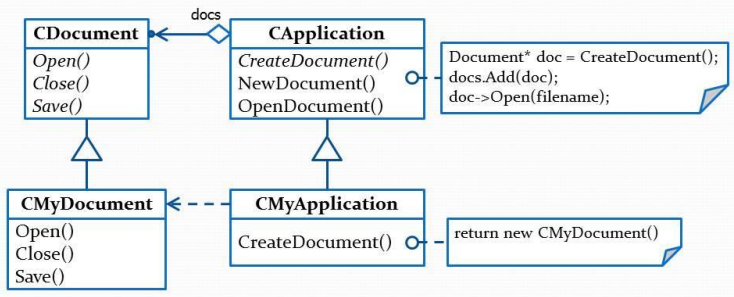
Применимость

- Классу заранее неизвестно, объекты каких классов ему нужно создавать
- Класс спроектирован так, чтобы объекты, которые он создает, специфицировались подклассами
- Класс делегирует свои обязанности одному из нескольких вспомогательных подклассов
  - тип конкретного вспомогательного подкласса выбирается в ходе выполнения программы

Пример использования

- Каркас приложения, позволяющего пользователю работать с различными типами документов
  - Основные абстракции в таком каркасе – классы CApplication и CDocument
    - CApplication – управляет созданием и открытием документов
    - CDocument – базовый абстрактный класс CDocument, основа для создания конкретных документов
- **Проблема** – класс CApplication знает, в какой момент документ должен быть создан, но ему известно лишь об абстрактном классе CDocument, который инстанцировать нельзя
  - Паттерн «Фабричный метод» предлагает решение данной проблемы

Иерархия классов



Методы структурного анализа и проектирования

Структурный анализ — один из формализованных методов анализа требований и проектирования ПО (автор Том Де Марко).

Основная задача – описание:

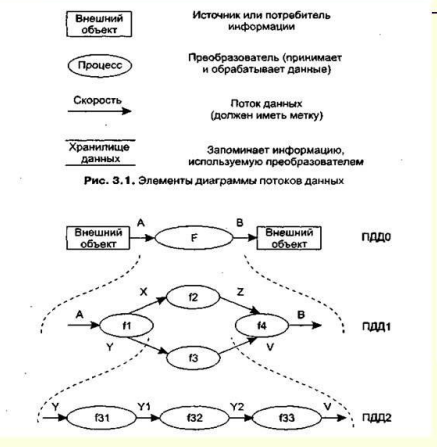
а) функциональной структуры системы;
б) последовательности выполняемых действий;
в) передачи информации между функциональными процессами;
г) отношений между данными.

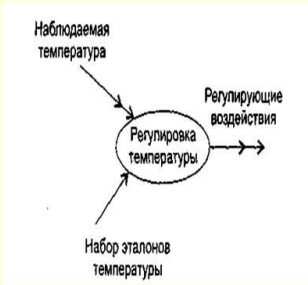
Модели структурного анализа и проектирования:

- Функциональная модель SADT (Structured Analysis and Design Technique);
- Модель IDEF3;
- DFD (Data Flow Diagrams) – диаграммы потоков данных

- **Метод SADT** (IDEF0) – совокупность правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области (производимые действия и связи между ними).
- **Модель IDEF3** – предназначена для моделирования последовательности выполняемых действий и взаимозависимости между ними, основа модели – сценарий процесса

- DFD (Data Flow Diagrams) – иерархия функциональных процессов, связанных потоками данных.





Методы объектно-ориентированного анализа и проектирования

Концептуальная основа ООАП – объектная модель, ее основные принципы (абстрагирование, инкапсуляция, модульность, иерархия) и понятия (объект, класс, атрибут, операция, интерфейс и др.).

UML (Unified Modeling Language) – язык для определения, представления, проектирования и документирования систем различной природы.

Основные задачи UML:

- предоставить пользователям готовый к использованию выразительный язык визуального моделирования, позволяющий им разрабатывать осмысленные модели и обмениваться ими;
- предусмотреть механизмы расширяемости и специализации для расширения базовых концепций;
- обеспечить независимость от конкретных языков программирования и процессов разработки;
- обеспечить формальную основу для понимания этого языка моделирования (язык должен быть одновременно точным и доступным для понимания, без лишнего формализма);
- стимулировать рост рынка объектно-ориентированных инструментальных средств;
- интегрировать лучший практический опыт.

Структурные (structural) модели:

- •диаграммы классов (class diagrams)– для моделирования статической структуры классов системы и связей между ними;
- •диаграммы компонентов (component diagrams) – для моделирования иерархии компонентов (подсистем) системы;
- •диаграммы размещения (deployment diagrams) – для моделирования физической архитектуры системы.

Модели поведения (behavioral):

- •диаграммы вариантов использования (use case diagrams) – для моделирования функциональных требований к системе (в виде сценариев взаимодействия пользователей с системой);

•диаграммы взаимодействия (interaction diagrams):

- •диаграммы последовательности (sequence diagrams)и кооперативные диаграммы (collaboration diagrams) – для моделирования процесса обмена сообщениями между объектами;
- •диаграммы состояний (statechart diagrams) – для моделирования поведения объектов системы при переходе из одного состояния в другое;
- •диаграммы деятельности (activity diagrams) – для моделирования поведения системы в рамках различных вариантов использования, или потоков управления.

Анализ требований

Что должно делать будущее ПО?

1. Система должна предоставлять пользователю доступ к балансу его банковского счета.
2. Балансы счетов клиентов будут храниться в таблице под названием «балансы» в базе данных Access.

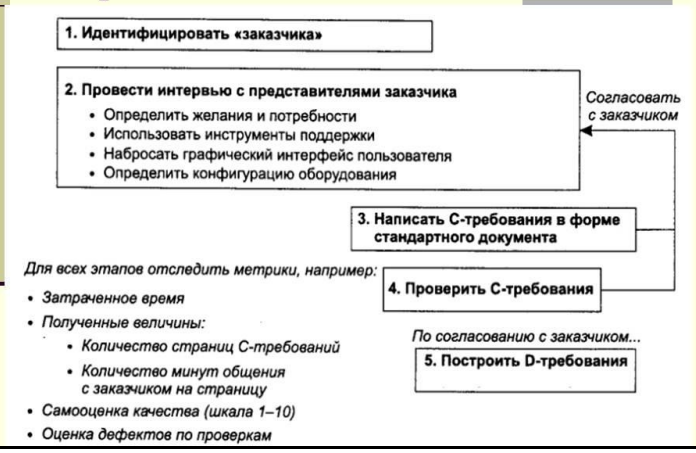
Результат анализа требований – спецификация требований к ПО (SRS – Software Requirements Specification)

Требования заказчика и разработчика

- C-требования – требования заказчика
- D-требования – требования разработчика

Каждое требование должно быть:
четко выражено;
легко доступно;
пронумеровано;
сопровождаться подтверждающими требованиями;
предусматриваться проектом;
учтено кодом;
протестировано отдельно;
протестировано совместно с остальными требованиями;
подтверждено тестированием после сборки приложения.

Типичная схема процесса анализа C-требований



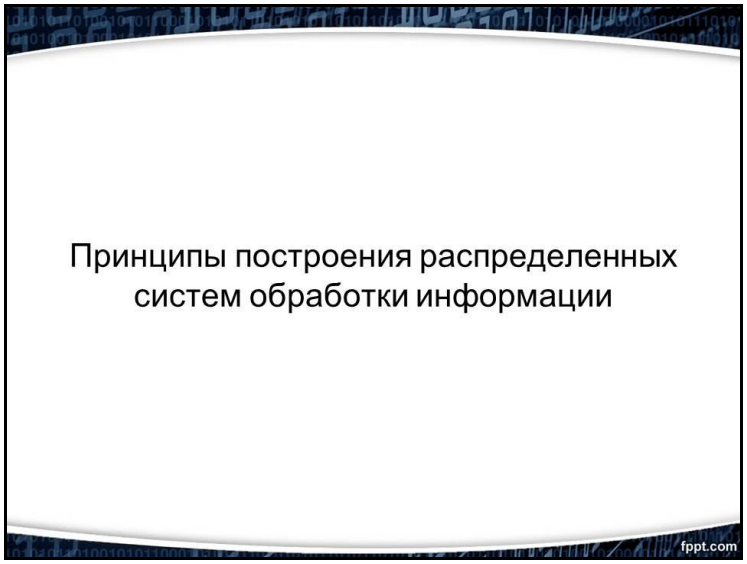


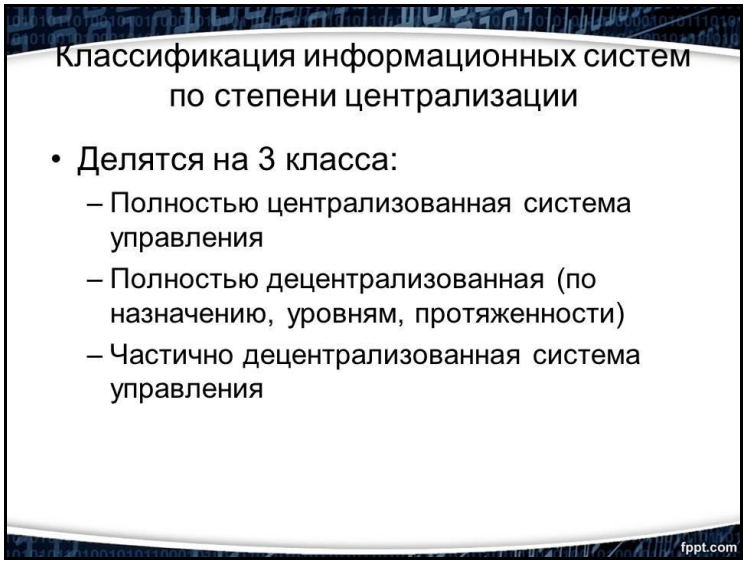
Литература: Хританков, А. С. Проектирование на UML: сборник задач / А. С. Хританков, В. А. Полежаев, А. И. Андрианов. – 3-е изд. стер. – Москва ; Берлин : Директ-Медиа, 2018. – 240 с. : ил. – Режим доступа: по подписке. – ISBN 978-5-4475-9493-0. – DOI 10.23681/483549; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=483549

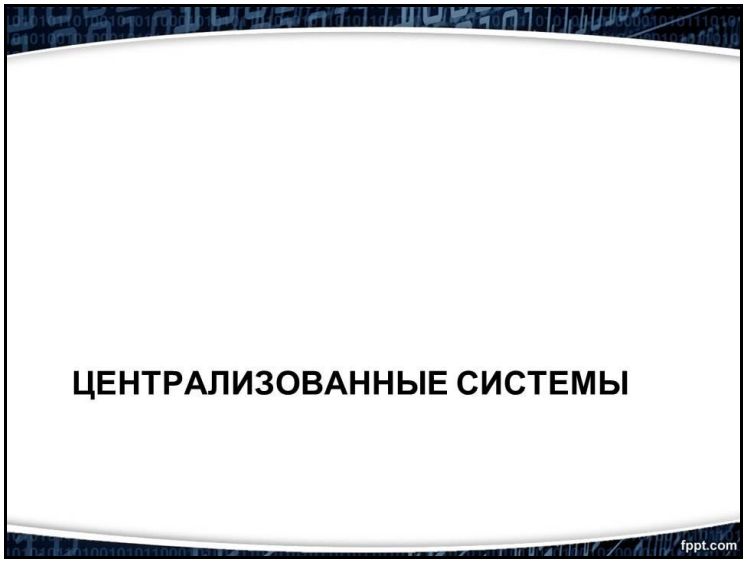
Лекция 3. Принципы построения распределенных систем обработки информации
Тезисы лекции:

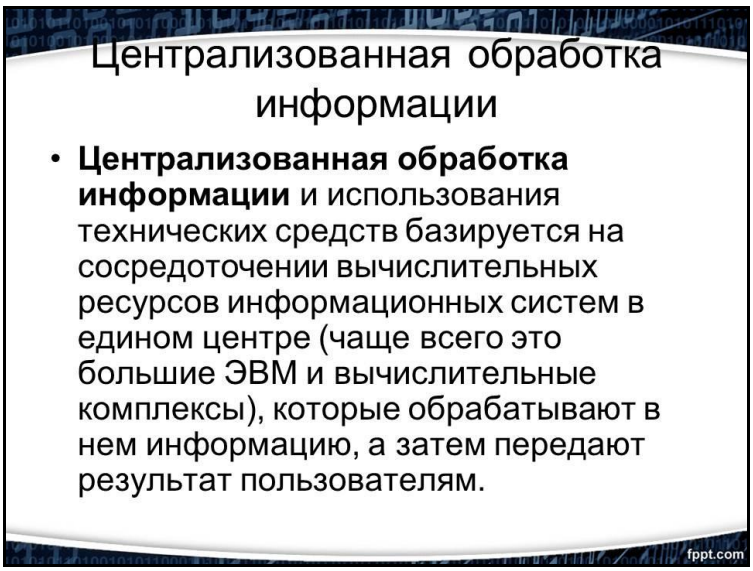
- Введение в распределенные системы. Технология «клиент-сервер». Серверы приложений и прикладные протоколы.
- Представление данных в распределенных системах. Принципы построения серверного программного обеспечения.
- Основные задачи, выполняемые серверными программами. Инструментальные программные средства (ИПС) создания программ, выполняемых на стороне сервера. Технологии обеспечения коммуникаций в распределенных ИС.
- Технологии распределенных БД в распределенных ИС.

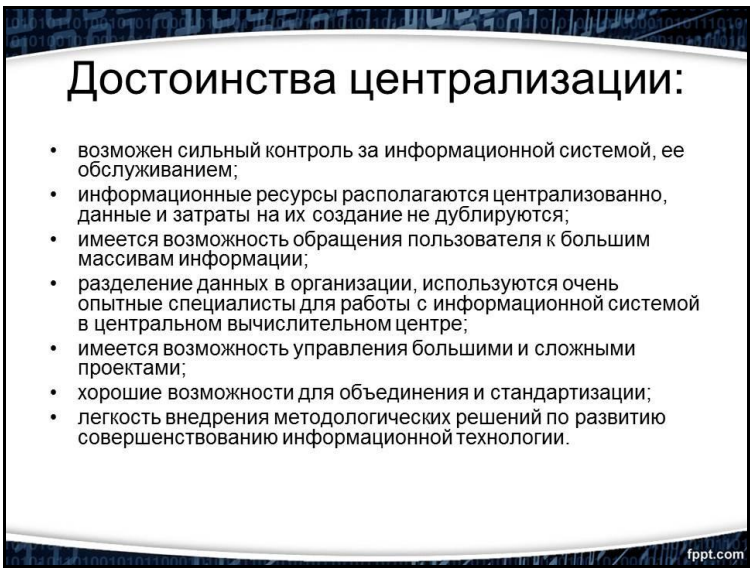
Слайды презентации к лекции:

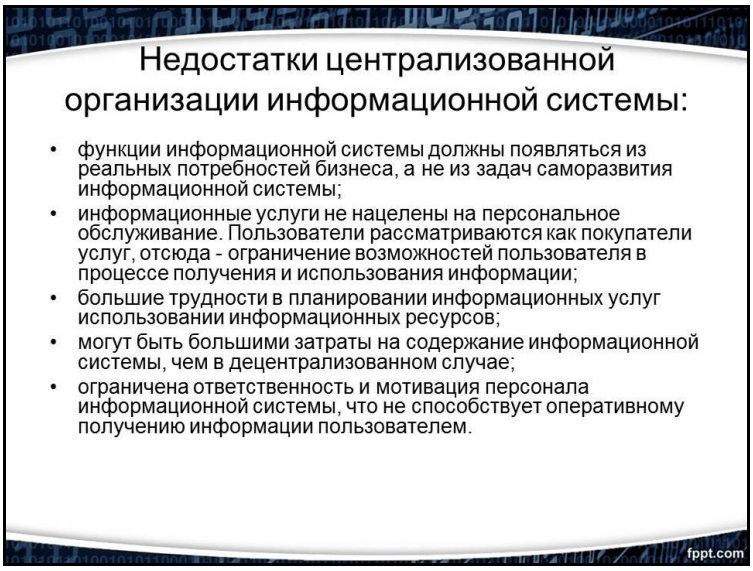


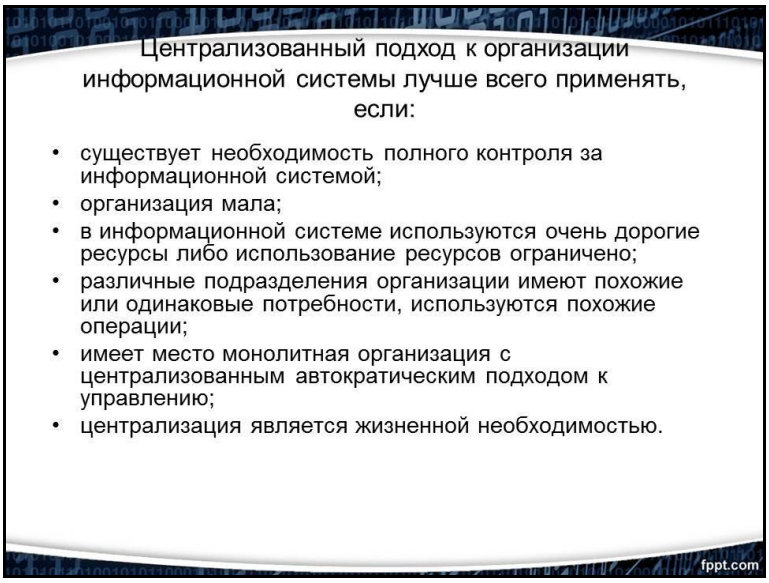


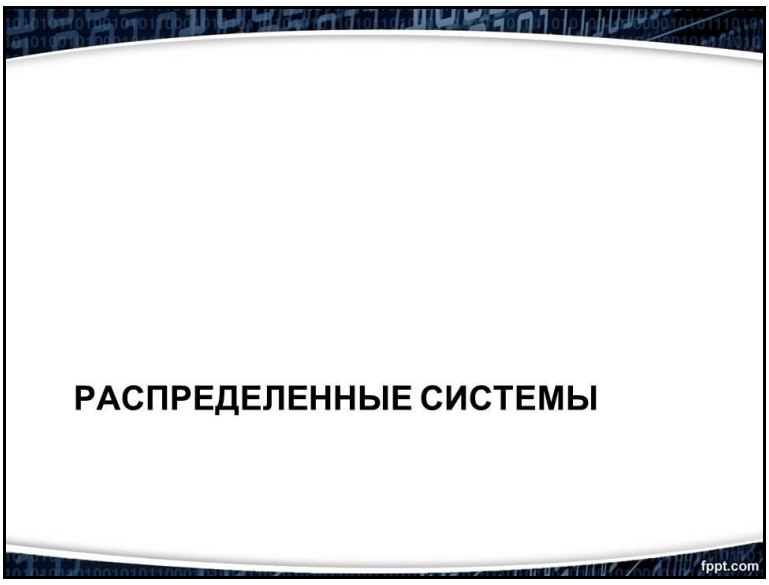


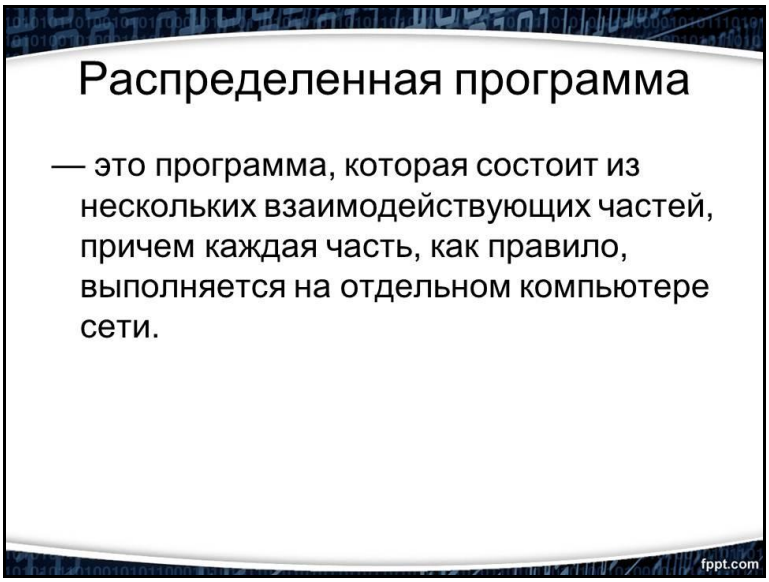

Централизованная обработка информации
• Централизованная обработка информации и использования технических средств базируется на сосредоточении вычислительных ресурсов информационных систем в едином центре (чаще всего это большие ЭВМ и вычислительные комплексы), которые обрабатывают в нем информацию, а затем передают результат пользователям.
fppt.com


Достоинства централизации:
• возможен сильный контроль за информационной системой, ее обслуживанием;
• информационные ресурсы располагаются централизованно, данные и затраты на их создание не дублируются;
• имеется возможность обращения пользователя к большим массивам информации;
• разделение данных в организации, используются очень опытные специалисты для работы с информационной системой в центральном вычислительном центре;
• имеется возможность управления большими и сложными проектами;
• хорошие возможности для объединения и стандартизации;
• легкость внедрения методологических решений по развитию совершенствованию информационной технологии.
fppt.com


Недостатки централизованной организации информационной системы:
• функции информационной системы должны появляться из реальных потребностей бизнеса, а не из задач саморазвития информационной системы;
• информационные услуги не нацелены на персональное обслуживание. Пользователи рассматриваются как покупатели услуг, отсюда - ограничение возможностей пользователя в процессе получения и использования информации;
• большие трудности в планировании информационных услуг использовании информационных ресурсов;
• могут быть большими затраты на содержание информационной системы, чем в децентрализованном случае;
• ограничена ответственность и мотивация персонала информационной системы, что не способствует оперативному получению информации пользователем.
fppt.com

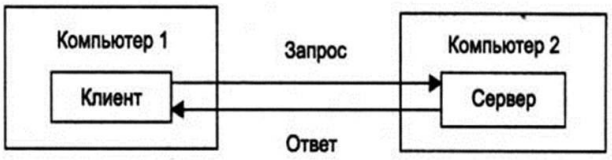

Централизованный подход к организации информационной системы лучше всего применять, если:
• существует необходимость полного контроля за информационной системой;
• организация мала;
• в информационной системе используются очень дорогие ресурсы либо использование ресурсов ограничено;
• различные подразделения организации имеют похожие или одинаковые потребности, используются похожие операции;
• имеет место монолитная организация с централизованным автократическим подходом к управлению;
• централизация является жизненной необходимостью.
fppt.com


РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ
fppt.com


Распределенная программа
— это программа, которая состоит из нескольких взаимодействующих частей, причем каждая часть, как правило, выполняется на отдельном компьютере сети.
fppt.com

Распределенные системы

- Взаимодействие частей распределенного приложения



fppt.com

Недостатки распределенных систем

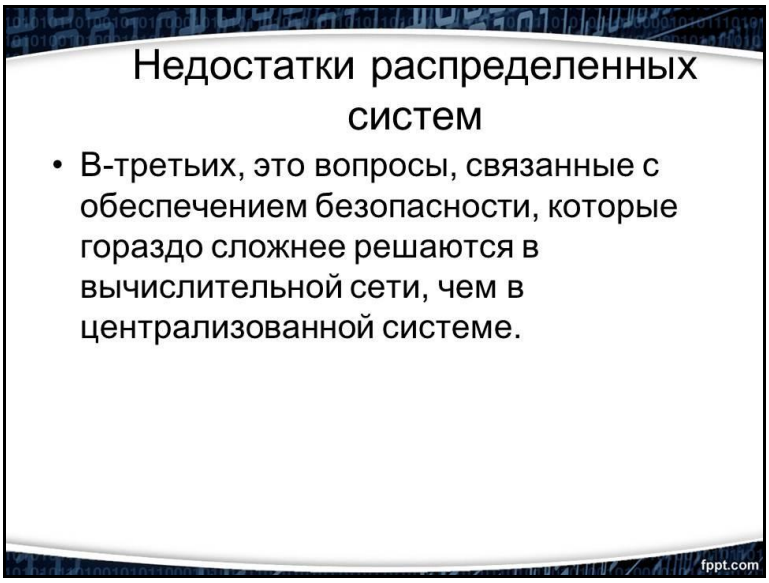
- Во-первых, это сложности, связанные с программным обеспечением — операционными системами и приложениями.

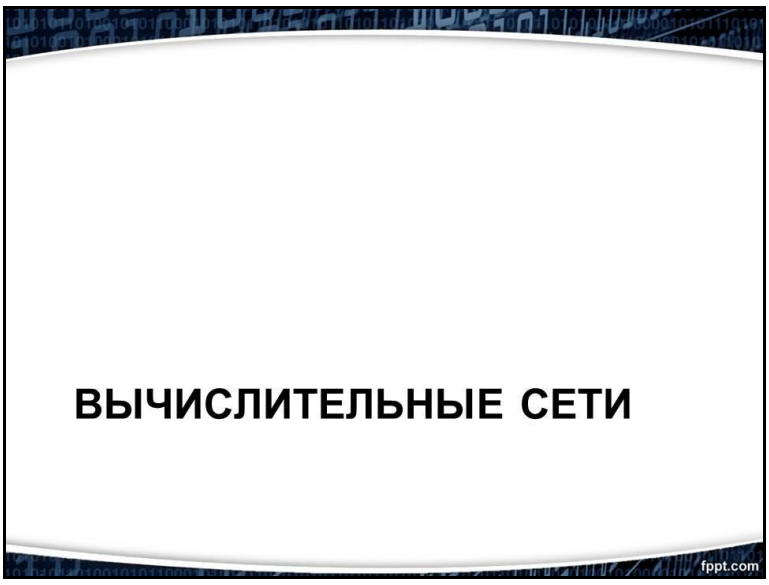
fppt.com

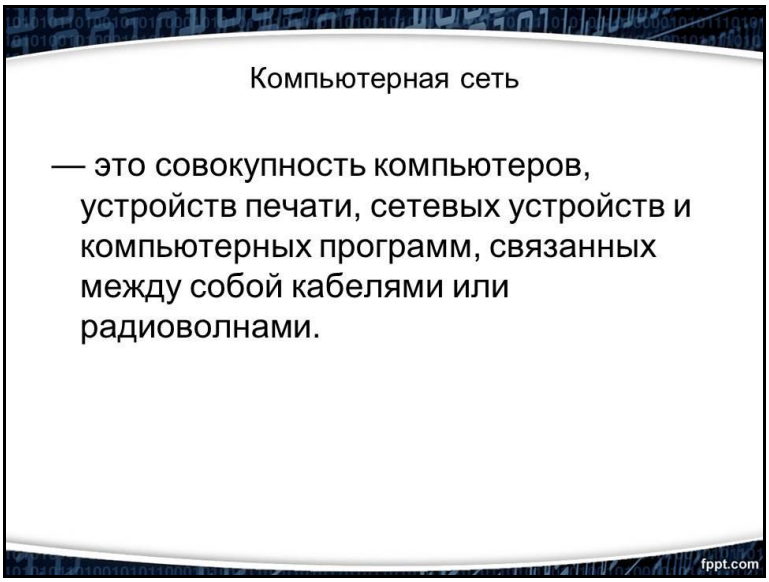
Недостатки распределенных систем

- Во-вторых, много проблем связано с транспортировкой сообщений по каналам связи между компьютерами.

fppt.com


Недостатки распределенных систем
• В-третьих, это вопросы, связанные с обеспечением безопасности, которые гораздо сложнее решаются в вычислительной сети, чем в централизованной системе.
fppt.com


ВЫЧИСЛИТЕЛЬНЫЕ СЕТИ
fppt.com


Компьютерная сеть
— это совокупность компьютеров, устройств печати, сетевых устройств и компьютерных программ, связанных между собой кабелями или радиоволнами.
fppt.com

Сетевые коммуникации с использованием кабеля, оптоволокна и радиоволн

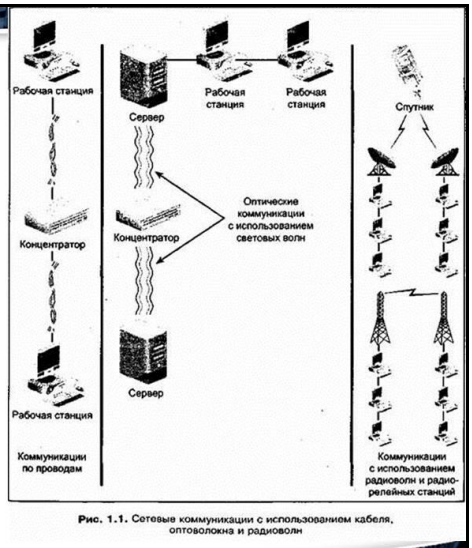
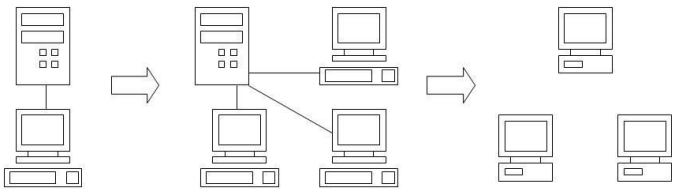


Рис. 1.1. Сетевые коммуникации с использованием кабеля, оптоволокна и радиоволн

fppt.com

Применение компьютерных сетей

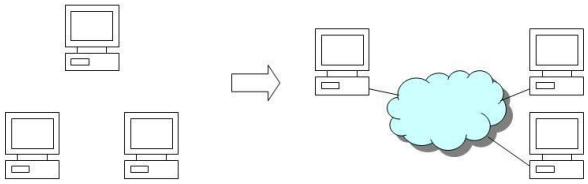


- При использовании "больших" ЭВМ с одним или несколькими терминалами все пользователи разделяли доступ к одним и тем же ресурсам (аппаратным, программным, информационным)
- При переходе к использованию нескольких ЭВМ ("больших" или персональных) возникла проблема организации доступа к ресурсам

Компьютерные сети МИОЭС

fppt.com

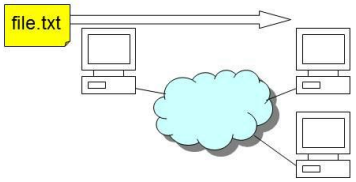
Применение компьютерных сетей



- При работе в автономной среде (без подключения к сети) ЭВМ может использовать только локальные ресурсы
- Работа в сетевой среде обеспечивает передачу данных между ЭВМ

fppt.com

Применение компьютерных сетей
Разделение файлов



- Сеть позволяет передавать данные (в виде файлов) между устройствами без использования дополнительных носителей (CD/DVD-ROM, flash-диски и т.д.)

fppt.com

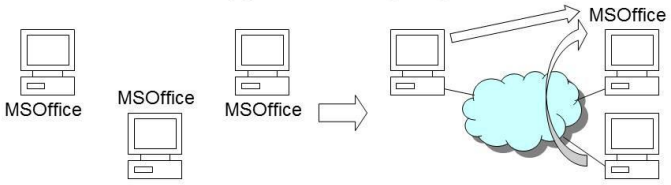
Применение компьютерных сетей
Разделение ресурсов



- В автономной среде пользователь может использовать только те устройства, которые непосредственно подключены к его компьютеру
- В сетевой среде можно организовать доступ к устройствам другого компьютера
- Разделяемые ресурсы
  - файловая система
  - дисковое пространство
  - принтеры
  - подключение к Интернет
  - другие ресурсы

fppt.com

Применение компьютерных сетей
Разделение программ



- В сетевой среде можно установить некоторые приложения на единственный компьютер в сети, а остальные настроить таким образом, чтобы они использовали эту установку
  - (+) Централизованное управление программным обеспечением
  - (-) Повышенные требования к пропускной способности сети и характеристикам компьютера, на котором установлено ПО
  - (-) Сложность управления распределенными настройками

fppt.com

Применение компьютерных сетей
Клиент-серверные приложения…

- Взаимодействие клиента и сервера обычно состоит из следующих этапов

Клиент		Сервер
1. Формирование запроса		
2. Отправка запроса серверу	⇨	1. Прием запроса от клиента
		2. Обработка запроса и формирование результата
3. Прием результата от сервера	⇦	3. Отправка результата клиенту
4. Интерпретация результата		

fppt.com

Применение компьютерных сетей
Клиент-серверные приложения…

- Тип запроса, способ обработки и результат зависят от приложения
  - Клиент может запросить текущее время, в этом случае сервер выясняет свое локальное время и отправляет его клиенту
  - Клиент может запросить отчет из базы данных, в этом случае сервер должен организовать выборку данных из БД и предъявить результат клиенту (например, в виде таблицы)
- Клиентская и серверная части могут выполняться как на одном компьютере, так и на разных
- Фактически, любое программное взаимодействие имеет клиент-серверную архитектуру

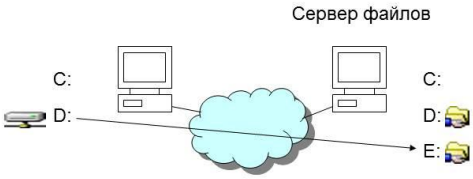
fppt.com

Применение компьютерных сетей
Клиент-серверные приложения…

- В **одноранговой** сети все компьютеры равноправны, на каждом выполняются как клиентские, так и серверные части приложений
- Часто в сети присутствует **выделенный сервер**, на котором выполняются серверные части одного или нескольких приложений
- Использование выделенных серверов
  - позволяет эффективнее выполнять соответствующие задачи за счет концентрации необходимых ресурсов
  - обеспечивает централизацию управления

fppt.com

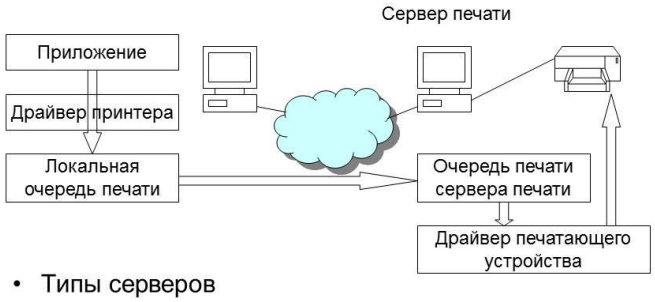
Применение компьютерных сетей
Клиент-серверные приложения…



- Существует большое количество клиент-серверных приложений и соответствующих типов серверов
  - Сервер файлов предоставляет в совместное использование свои файловые системы и предназначен для хранения данных и управления доступом к ним пользователей

fppt.com

Применение компьютерных сетей
Клиент-серверные приложения…



- Типы серверов
  - Сервер печати поддерживает очередь печати и обеспечивает доступ приложений к печатающему устройству

fppt.com

Применение компьютерных сетей
Клиент-серверные приложения

- Типы серверов
  - Сервер служб безопасности содержит информацию об устройствах и пользователях сети, обеспечивая функционирование системы защиты ресурсов
  - Сервер приложений выполняет серверные части клиент-серверных приложений
  - Почтовый сервер отвечает за функционирование системы электронной почты; в зависимости от имеющихся в его распоряжении каналов связи, он может обеспечивать передачу сообщений внутри сети либо в пределах Интернет или другой глобальной сети
  - и т.д.

fppt.com

Применение компьютерных сетей
Удаленное управление…

- Администратор обслуживает десятки или сотни компьютеров, расположенных в различных кабинетах или зданиях
- Для эффективной работы ему нужны средства, позволяющие
  - Удаленно узнать ситуацию на интересующем его компьютере
  - Удаленно изменить настройки в случае необходимости

fppt.com

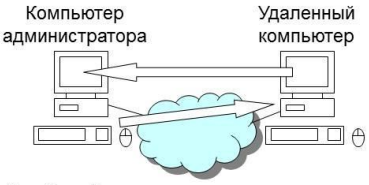
Применение компьютерных сетей
Удаленное управление…

- Средства удаленной настройки приложений
  - В наиболее простом варианте конфигурационные файлы приложения предоставлены в совместное использование
  - В Windows встроены средства, позволяющие получить доступ к конфигурации удаленного узла (Remote Registry, MMC,…)
  - Web-интерфейс доступа к конфигурации (swat)



fppt.com

Применение компьютерных сетей
Удаленное управление…



- Удаленный рабочий стол – администратор может видеть содержимое дисплея удаленного компьютера, а его нажатия на клавиатуре или действия мышью передаются на удаленный компьютер и обрабатываются им как локальные
  - Встроенные средства операционной системы (например, удаленный помощник Windows XP/Vista)
- Средства сторонних разработчиков (Remote Desktop, PC Anywhere, RAdmin и т.д.)

fppt.com

Применение компьютерных сетей
Удаленное управление



- Удаленная сессия – пользовательский интерфейс приложений выполняется на одном компьютере, а сами приложения – на другом
  - telnet/ssh – удаленные терминалы (UNIX)
  - X Window – архитектура, разделяющая выполнение приложения и функционирование его пользовательского интерфейса (ПИ), позволяет организовывать выполнение приложения и его ПИ как на одном компьютере, так и на разных
  - Начиная с Windows NT 4.0 Terminal Server серверные версии ОС Windows поддерживают выполнение удаленных сессий

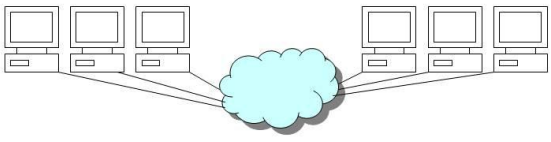
fppt.com

Применение компьютерных сетей
Распределенные вычисления…

- Существует большое количество задач, для решения которых недостаточно ресурсов одной ЭВМ
  - Расчет деформации неоднородного объекта
  - Прогноз погоды
  - Расчет движения воды в океане
  - Расчет обтекаемости объекта (например, самолетного крыла или автомобиля)
  - и т.д.
- Распределенные вычисления – параллельная обработка данных, при которой используется несколько обрабатывающих устройств, достаточно удаленных друг от друга
  - например, при организации вычислений в многомашинных вычислительных комплексах, образуемых объединением нескольких отдельных ЭВМ с помощью каналов связи локальных или глобальных сетей

fppt.com

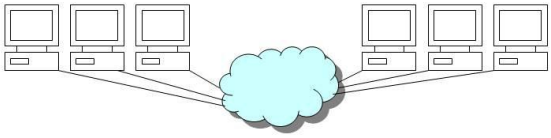
Применение компьютерных сетей
Распределенные вычисления…



- Для распределенных вычислений используются многомашинные системы, состоящие из компьютеров самых различных архитектур, между которыми некоторым образом организованы сетевые соединение
- С точки зрения аппаратной составляющей, каждый такой комплекс в некотором смысле уникален

fppt.com

Применение компьютерных сетей
Распределенные вычисления…



- С программной точки зрения (1)
  - Можно самостоятельно разработать логику сетевого взаимодействия и реализовать ее, используя программный интерфейс передачи данных через сеть (например, программный интерфейс сокетов)
  - Можно использовать популярные парадигмы параллельных вычислений и соответствующие библиотеки (например, реализации MPI)

fppt.com

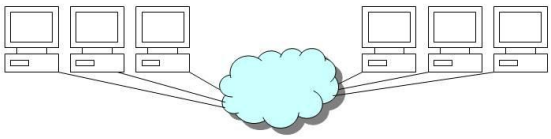
Применение компьютерных сетей
Распределенные вычисления…



- С программной точки зрения (2)
  - Можно использовать архитектуру с выделенным главным узлом, который планирует выполнение вычислений и раздает задания другим узлам
    - Такой подход может быть использован, если задания решаются независимо
    - Популярна при использовании удаленных систем с существенно различными характеристиками производительности

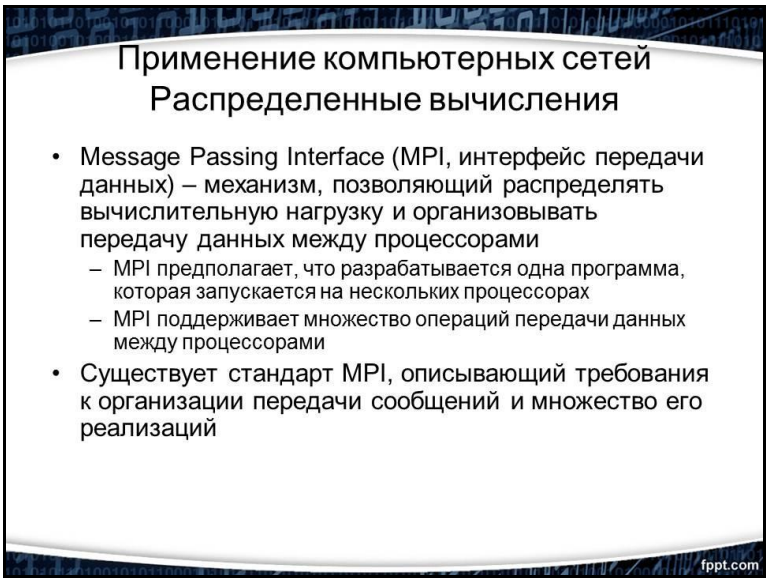
fppt.com

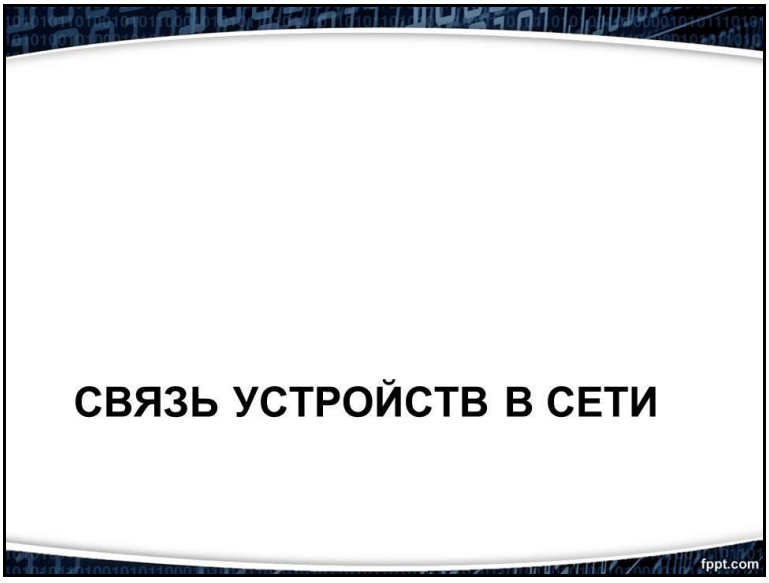
Применение компьютерных сетей
Распределенные вычисления…

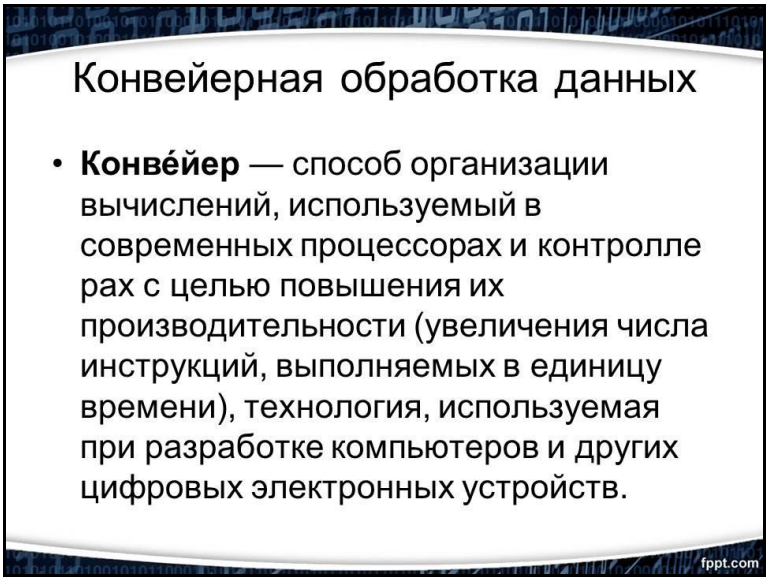


- С программной точки зрения (2)
  - Можно использовать архитектуру параллельного выполнения задачи на нескольких равноправных узлах
    - Используется, если параллельный алгоритм требует передачи больших объемов данных
    - Максимальная эффективность достигается при использовании одинаковых узлов, соединенных высокоскоростной сетью

fppt.com


Применение компьютерных сетей
Распределенные вычисления
• Message Passing Interface (MPI, интерфейс передачи данных) – механизм, позволяющий распределять вычислительную нагрузку и организовывать передачу данных между процессорами
– MPI предполагает, что разрабатывается одна программа, которая запускается на нескольких процессорах
– MPI поддерживает множество операций передачи данных между процессорами
• Существует стандарт MPI, описывающий требования к организации передачи сообщений и множество его реализаций
fppt.com


СВЯЗЬ УСТРОЙСТВ В СЕТИ
fppt.com


Конвейерная обработка данных
• Конве́йер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени), технология, используемая при разработке компьютеров и других цифровых электронных устройств.
fppt.com

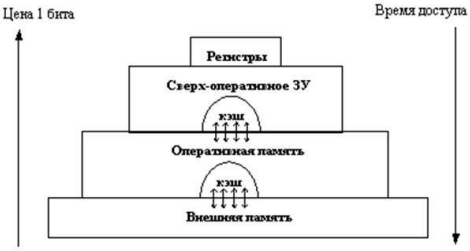
Конвейерная обработка данных

- Идея заключается в разделении обработки компьютерной инструкции на последовательность независимых стадий с сохранением результатов в конце каждой стадии.
- Это позволяет управляющим цепям процессора получать инструкции со скоростью самой медленной стадии обработки, однако при этом намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.

fppt.com

Иерархия запоминающих устройств

- Память вычислительной машины представляет собой иерархию запоминающих устройств (внутренние регистры процессора, различные типы сверхоперативной и оперативной памяти, диски, ленты), отличающихся средним временем доступа и стоимостью хранения данных в расчете на один бит



fppt.com

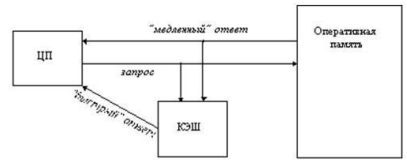
Принцип кэширования данных

- *Кэш-память* - это способ организации совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который позволяет уменьшить среднее время доступа к данным за счет динамического копирования в "быстрое" ЗУ наиболее часто используемой информации из "медленного" ЗУ.
- Кэш-памятью часто называют не только способ организации работы двух типов запоминающих устройств, но и одно из устройств - "быстрое" ЗУ. Оно стоит дороже и, как правило, имеет сравнительно небольшой объем. Важно, что механизм кэш-памяти является прозрачным для пользователя, который не должен сообщать никакой информации об интенсивности использования данных и не должен никак участвовать в перемещении данных из ЗУ одного типа в ЗУ другого типа, все это делается автоматически системными средствами.

fppt.com

Частный случай использования кэш-памяти

- Используется для уменьшения среднего времени доступа к данным, хранящимся в оперативной памяти.
- Для этого между процессором и оперативной памятью помещается быстрое ЗУ, называемое просто кэш-памятью.



Структура кэш-памяти

Адрес данных в ОП	Данные	Управл. информация	
		Бит модиф.	Бит обращ.

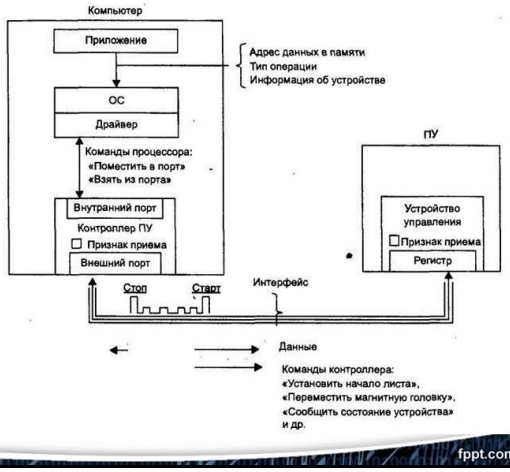
fppt.com

В системах, оснащенных кэш-памятью, каждый запрос к оперативной памяти выполняется в соответствии со следующим алгоритмом:

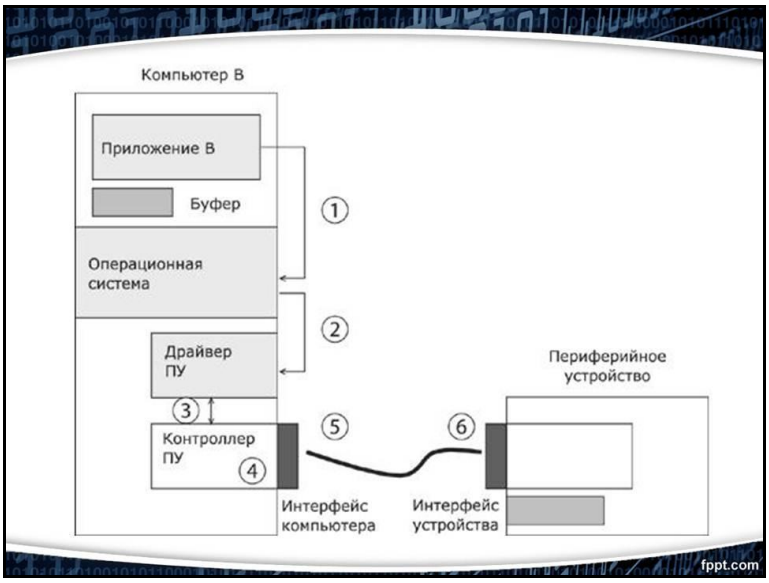
- Просматривается содержимое кэш-памяти с целью определения, не находятся ли нужные данные в кэш-памяти; кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому - значению поля "адрес в оперативной памяти", взятому из запроса.
- Если данные обнаруживаются в кэш-памяти, то они считываются из нее, и результат передается в процессор.
- Если нужных данных нет, то они вместе со своим адресом копируются из оперативной памяти в кэш-память, и результат выполнения запроса передается в процессор.

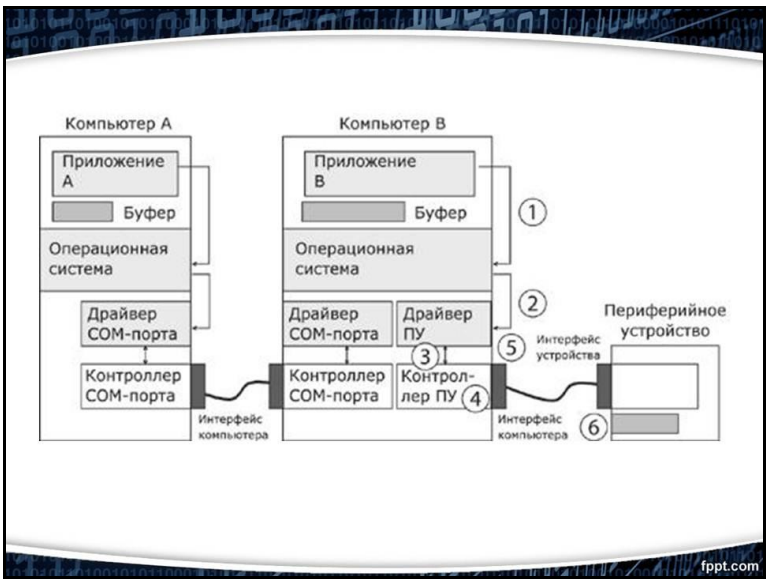
fppt.com

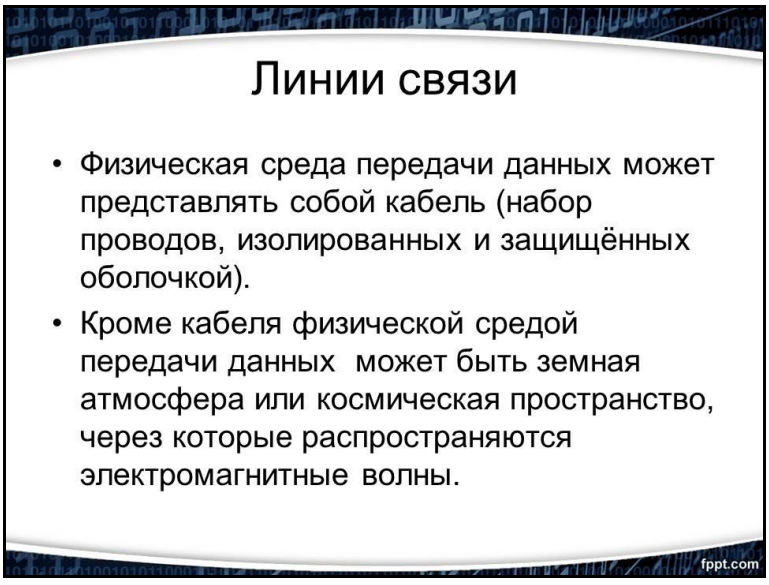
Связь компьютера с периферийными устройствами



fppt.com


Компьютер В
Приложение В
Буфер
Операционная система
Драйвер ПУ
Контроллер ПУ
Периферийное устройство
Интерфейс компьютера
Интерфейс устройства
1
2
3
4
5
6
fppt.com


Компьютер А
Компьютер В
Приложение А
Приложение В
Буфер
Операционная система
Драйвер COM-порта
Драйвер ПУ
Контроллер COM-порта
Контрол-лер ПУ
Периферийное устройство
Интерфейс устройства
Интерфейс компьютера
1
2
3
4
5
6
fppt.com


Линии связи
• Физическая среда передачи данных может представлять собой кабель (набор проводов, изолированных и защищённых оболочкой).
• Кроме кабеля физической средой передачи данных может быть земная атмосфера или космическая пространство, через которые распространяются электромагнитные волны.
fppt.com

Линии связи

- В зависимости от среды передачи данных линии связи можно разделить на три группы:
- 1) Проводные (воздушные) линии связи — это провода без изолирующих и экранирующих оплёток, проложенные между столбами и висящие в воздухе. Они проводят телефонные и телеграфные сигналы;
- 2) Радиоканалы земной и спутниковой связи — образуются с помощью передатчика и приёмника радиоволн. Может использоваться для организации сетей в пределах больших помещений типа ангаров или павильонов, там, где использование обычных линий связи затруднено или нецелесообразно.
- 3) Кабельные — состоят из проводников, заключённых в несколько слоёв изоляции.

fppt.com

Проблемы физической передачи данных по линиям связи

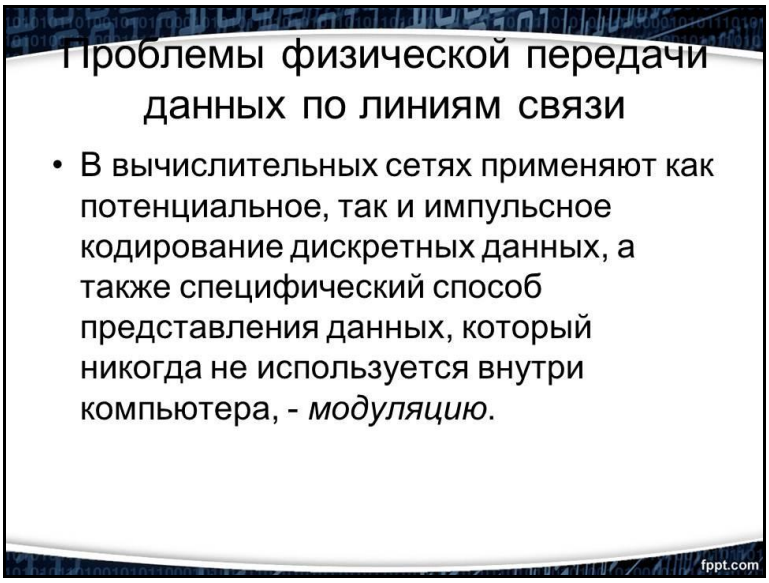
- В вычислительной технике для представления данных используется двоичный код.
- Внутри компьютера единицам и нулям данных соответствуют дискретные электрические сигналы.
- Представление данных в виде электрических или оптических сигналов называется *кодированием*.
- Существуют различные способы кодирования двоичных цифр 1 и 0, например, потенциальный способ, при котором единице соответствует один уровень напряжения, а нулю - другой, или импульсный способ, когда для представления цифр используются импульсы различной или одной полярности.

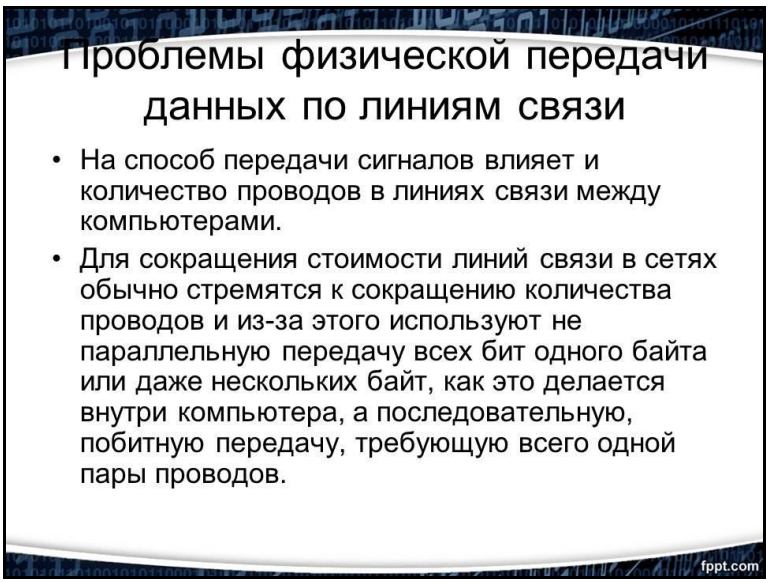
fppt.com

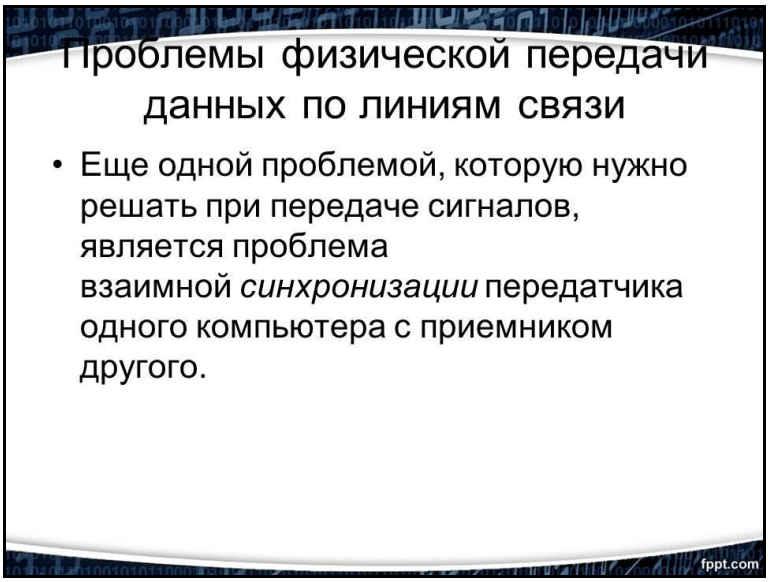
Проблемы физической передачи данных по линиям связи

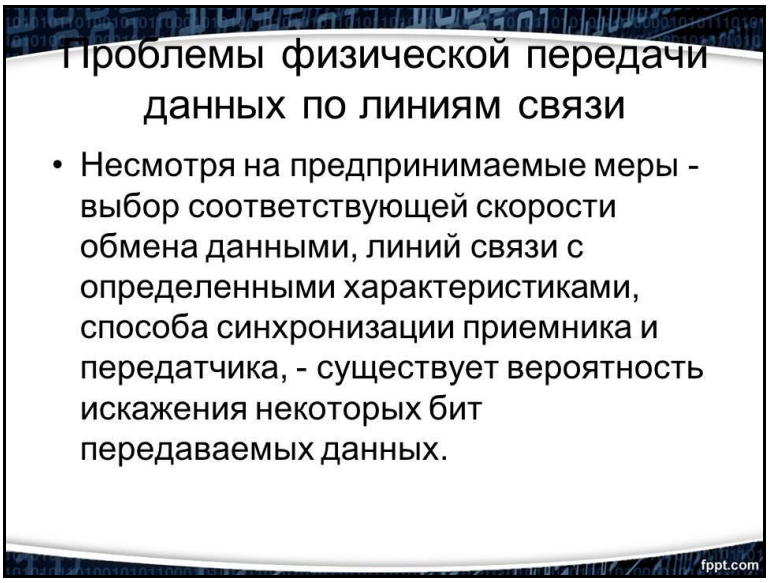
- Аналогичные подходы могут быть использованы для кодирования данных и при передаче их между двумя компьютерами по линиям связи.
- **Однако эти линии связи отличаются по своим электрическим характеристикам от тех, которые существуют внутри компьютера.**

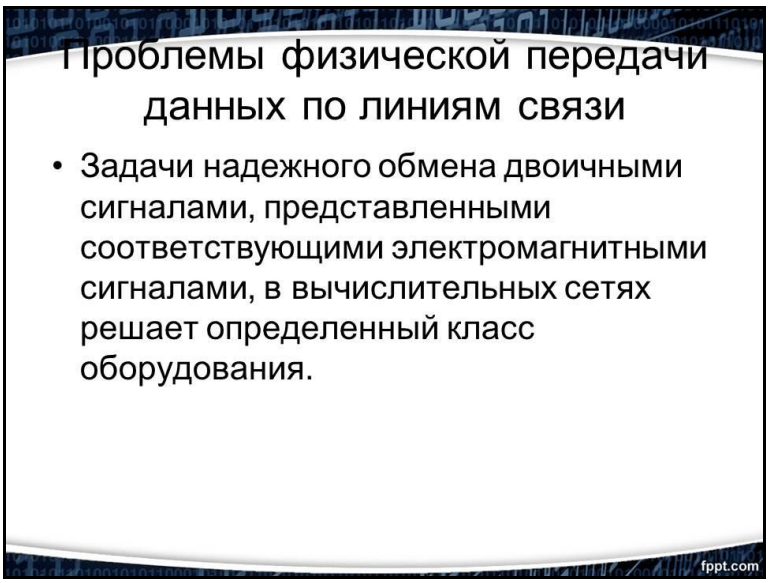
fppt.com

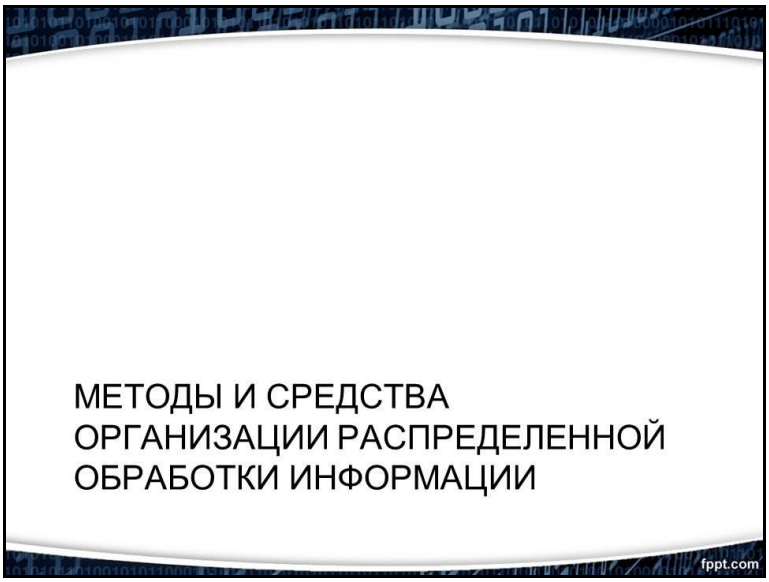

Проблемы физической передачи данных по линиям связи
• В вычислительных сетях применяют как потенциальное, так и импульсное кодирование дискретных данных, а также специфический способ представления данных, который никогда не используется внутри компьютера, - *модуляцию*.
fppt.com

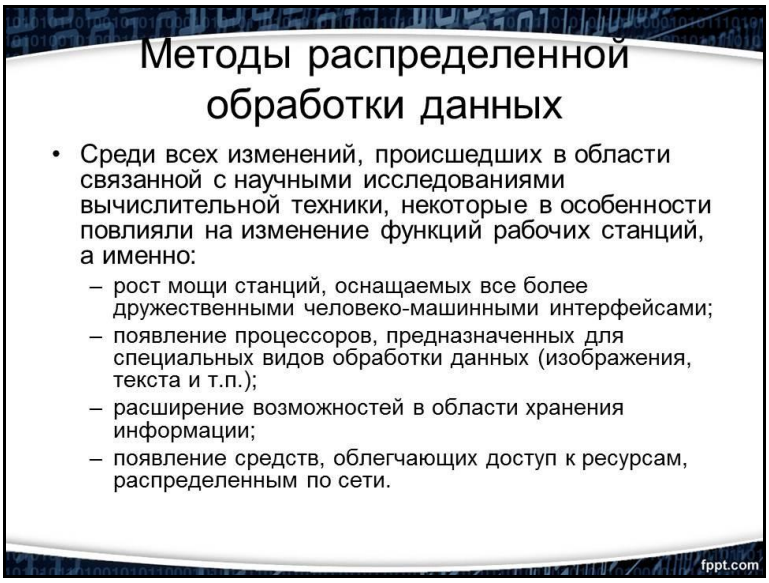

Проблемы физической передачи данных по линиям связи
• На способ передачи сигналов влияет и количество проводов в линиях связи между компьютерами.
• Для сокращения стоимости линий связи в сетях обычно стремятся к сокращению количества проводов и из-за этого используют не параллельную передачу всех бит одного байта или даже нескольких байт, как это делается внутри компьютера, а последовательную, побитную передачу, требующую всего одной пары проводов.
fppt.com

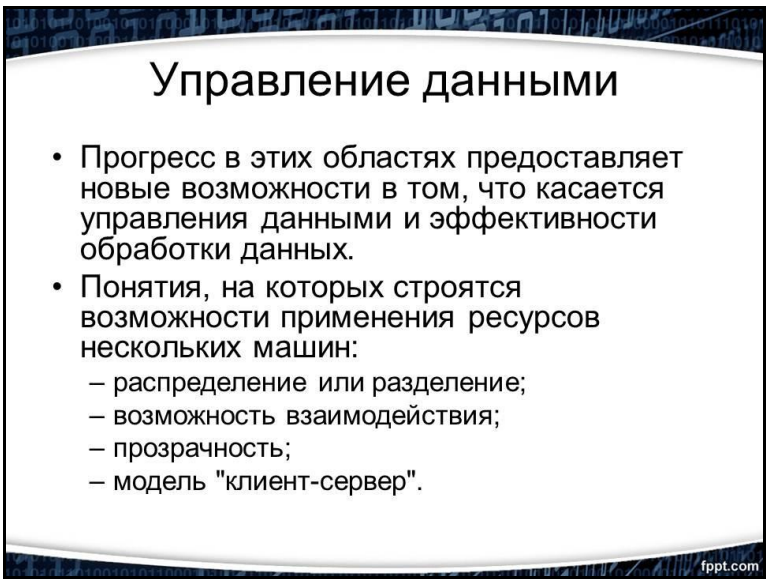

Проблемы физической передачи данных по линиям связи
• Еще одной проблемой, которую нужно решать при передаче сигналов, является проблема взаимной *синхронизации* передатчика одного компьютера с приемником другого.
fppt.com

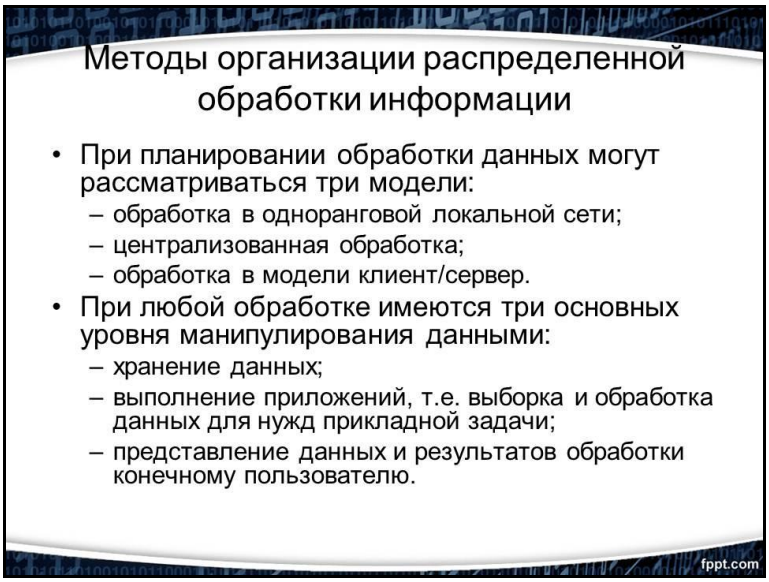

Проблемы физической передачи данных по линиям связи
• Несмотря на предпринимаемые меры - выбор соответствующей скорости обмена данными, линий связи с определенными характеристиками, способа синхронизации приемника и передатчика, - существует вероятность искажения некоторых бит передаваемых данных.
fppt.com


Проблемы физической передачи данных по линиям связи
• Задачи надежного обмена двоичными сигналами, представленными соответствующими электромагнитными сигналами, в вычислительных сетях решает определенный класс оборудования.
fppt.com


МЕТОДЫ И СРЕДСТВА ОРГАНИЗАЦИИ РАСПРЕДЕЛЕННОЙ ОБРАБОТКИ ИНФОРМАЦИИ
fppt.com


Методы распределенной обработки данных
• Среди всех изменений, происшедших в области связанной с научными исследованиями вычислительной техники, некоторые в особенности повлияли на изменение функций рабочих станций, а именно:
– рост мощи станций, оснащаемых все более дружественными человеко-машинными интерфейсами;
– появление процессоров, предназначенных для специальных видов обработки данных (изображения, текста и т.п.);
– расширение возможностей в области хранения информации;
– появление средств, облегчающих доступ к ресурсам, распределенным по сети.
fppt.com

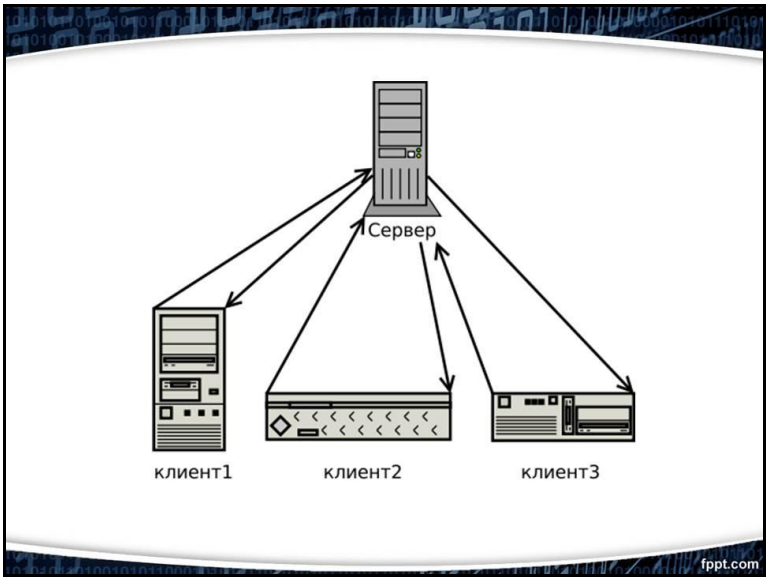

Управление данными
• Прогресс в этих областях предоставляет новые возможности в том, что касается управления данными и эффективности обработки данных.
• Понятия, на которых строятся возможности применения ресурсов нескольких машин:
– распределение или разделение;
– возможность взаимодействия;
– прозрачность;
– модель "клиент-сервер".
fppt.com


Методы организации распределенной обработки информации
• При планировании обработки данных могут рассматриваться три модели:
– обработка в одноранговой локальной сети;
– централизованная обработка;
– обработка в модели клиент/сервер.
• При любой обработке имеются три основных уровня манипулирования данными:
– хранение данных;
– выполнение приложений, т.е. выборка и обработка данных для нужд прикладной задачи;
– представление данных и результатов обработки конечному пользователю.
fppt.com

Модель «клиент-сервер»

- **Клиент-сервер** (англ. *Client-server*) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами.
- Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением.

fppt.com



fppt.com

ТИПЫ РАСПРЕДЕЛИТЕЛЬНЫХ СИСТЕМ

fppt.com

Мультипроцессорные компьютеры

Имеется несколько процессоров, каждый из которых может относительно независимо от остальных выполнять свою программу.

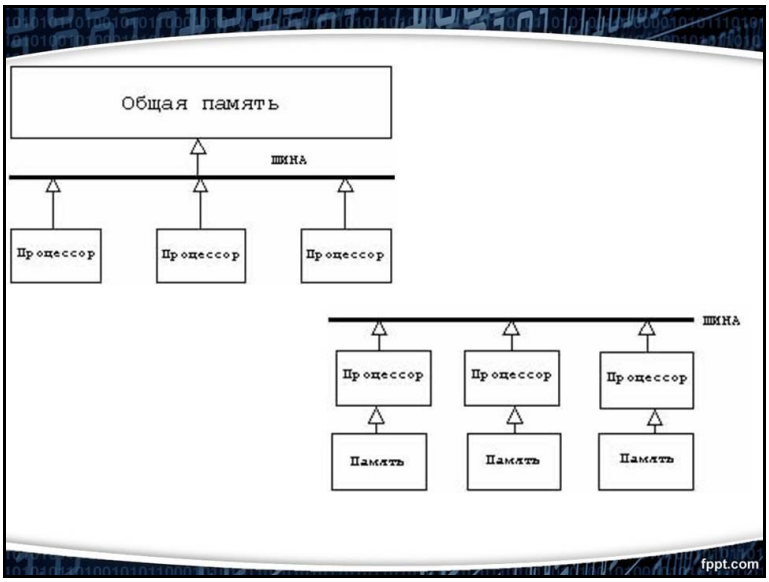
- Существует общая для всех процессоров операционная система, которая оперативно распределяет вычислительную нагрузку между процессорами.
- Взаимодействие между отдельными процессорами организуется наиболее простым способом — через общую оперативную память.

fppt.com

Мультипроцессоры

- Мультипроцессорные машины подразделяются на два семейства:
- жестко связанные или жестко соединенные мультипроцессоры (tightly coupled), в которых процессоры связаны через общую память;
- слабо связанные или слабо соединенные мультипроцессоры (loosely coupled), в которых процессоры связаны через средство связи (как правило, шину), отличное от общей памяти.

fppt.com



fppt.com

Многомашинные системы

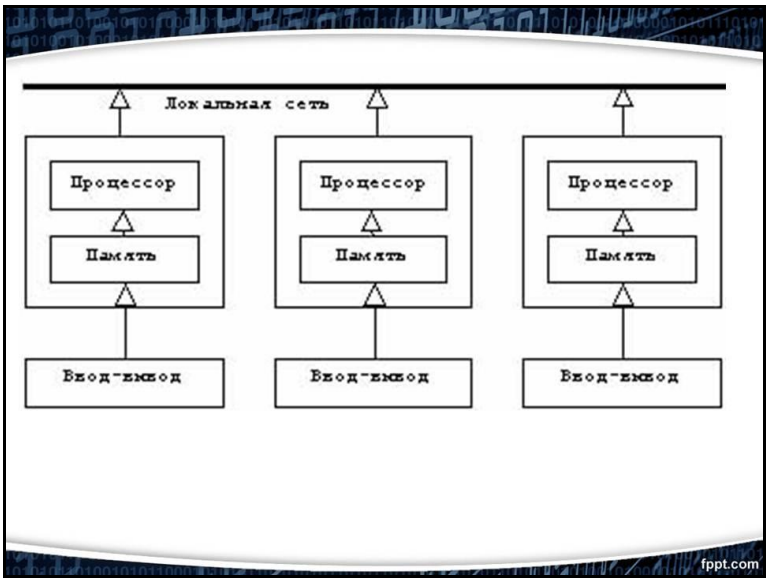
— это вычислительный комплекс, включающий в себя несколько компьютеров (каждый из которых работает под управлением собственной операционной системы), а также программные и аппаратные средства связи компьютеров, которые обеспечивают работу всех компьютеров комплекса как единого целого.

fppt.com

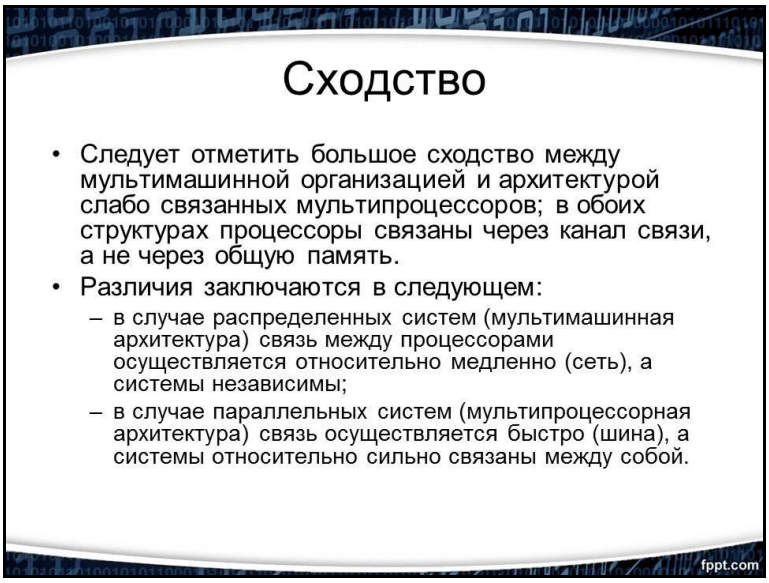
Мультимашинная организация

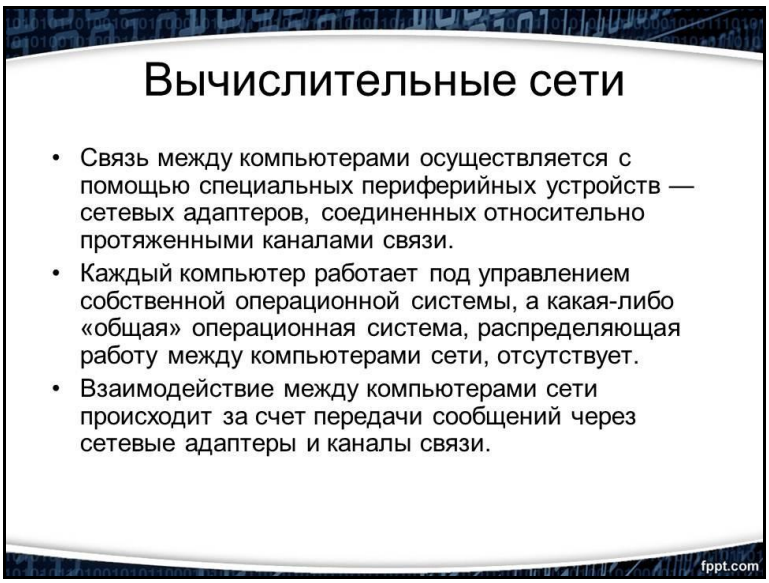
- Появление сетей, предназначенных для взаимной связи различных компьютеров, привело к разработке средств, а затем и операционных систем, позволяющих осуществлять управление, так называемой, мультимашинной архитектурой, то есть совокупности полносоставных компьютеров (процессоры, память, вводы-выводы...), связанных в сеть.
- В этом случае речь идет о распределенных вычислительных системах.

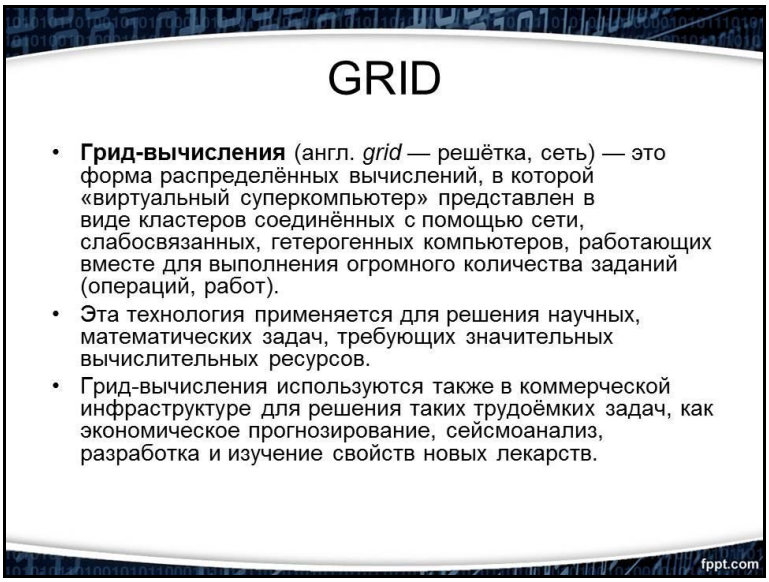
fppt.com



fppt.com


Сходство
• Следует отметить большое сходство между мультимашинной организацией и архитектурой слабо связанных мультипроцессоров; в обоих структурах процессоры связаны через канал связи, а не через общую память.
• Различия заключаются в следующем:
– в случае распределенных систем (мультимашинная архитектура) связь между процессорами осуществляется относительно медленно (сеть), а системы независимы;
– в случае параллельных систем (мультипроцессорная архитектура) связь осуществляется быстро (шина), а системы относительно сильно связаны между собой.
fppt.com


Вычислительные сети
• Связь между компьютерами осуществляется с помощью специальных периферийных устройств — сетевых адаптеров, соединенных относительно протяженными каналами связи.
• Каждый компьютер работает под управлением собственной операционной системы, а какая-либо «общая» операционная система, распределяющая работу между компьютерами сети, отсутствует.
• Взаимодействие между компьютерами сети происходит за счет передачи сообщений через сетевые адаптеры и каналы связи.
fppt.com


GRID
• **Грид-вычисления** (англ. *grid* — решётка, сеть) — это форма распределённых вычислений, в которой «виртуальный суперкомпьютер» представлен в виде кластеров соединённых с помощью сети, слабосвязанных, гетерогенных компьютеров, работающих вместе для выполнения огромного количества заданий (операций, работ).
• Эта технология применяется для решения научных, математических задач, требующих значительных вычислительных ресурсов.
• Грид-вычисления используются также в коммерческой инфраструктуре для решения таких трудоёмких задач, как экономическое прогнозирование, сейсмоанализ, разработка и изучение свойств новых лекарств.
fppt.com



Литература: Архитектуры и топологии многопроцессорных вычислительных систем: курс лекций : учебное пособие / А. В. Богданов, В. В. Корхов, В. В. Мареев, Е. Н. Станкова. – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2004. – 176 с. : ил.,табл., схем. – (Основы информационных технологий). – ISBN 5-9556-0018-3; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=232995

Лекция 4. Многопоточное и параллельное программирование

Тезисы лекции:

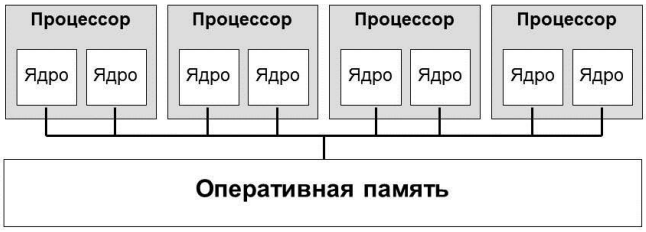
- Современное состояние проблемы. Создание/завершение потоков. Примитивы синхронизации. Алгоритмы синхронизации. Атомарные снимки регистров.
- Ошибки параллельного программирования. Шаблоны параллельного программирования.
- Кластерные вычисления. Консенсус. Сети Петри.
- Управление памятью в многопоточных системах. Оптимизации в компиляторах. Транзакционная память. Асинхронный ввод/вывод. Wait-free MRMW снимок регистров. Средства поиска ошибок. Модель памяти. Lock-free схемы управления памятью. Модель акторов.

Слайды презентации к лекции:

Многопоточное и параллельное программирование

Многопоточное программирование

- Используется для создания параллельных программ для систем с общей памятью



- И для других целей…

Модель памяти OpenMP

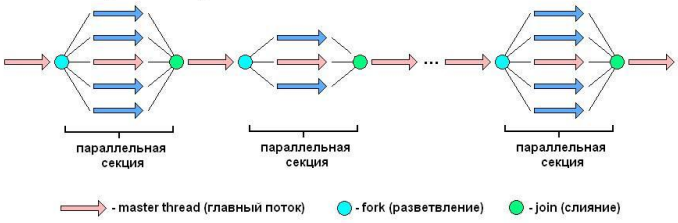
- Модель разделяемой памяти
  - Нити взаимодействуют через разделяемые переменные
  - Разделение определяется синтаксически
  - Любая переменная, видимая более чем одной нити, является разделяемой
  - Любая переменная, видимая только одной нити, является private
- Возможны Race conditions (Гонки)
  - Требуется синхронизация для предотвращения конфликтов
  - Возможно управление доступом (shared, private) для минимизации накладных расходов на синхронизацию

OpenMP – это…

- Стандарт интерфейса для многопоточного программирования над общей памятью
- Набор средств для языков C/C++ и Fortran:
  - Директивы компилятора
    #pragma omp …
  - Библиотечные подпрограммы
    get_num_threads()
  - Переменные окружения
    OMP_NUM_THREADS

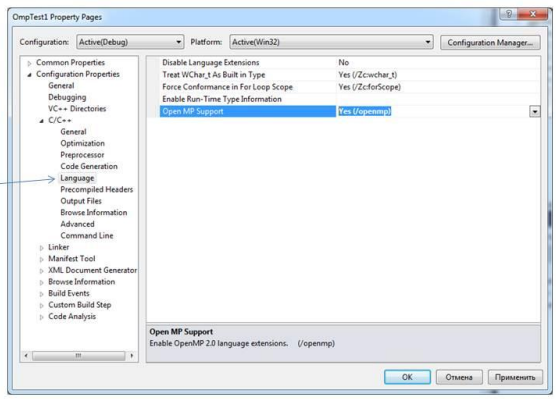
Модель программирования

- Fork-join параллелизм



- Явное указание параллельных секций
- Поддержка вложенного параллелизма
- Поддержка динамических потоков

Подключение в Visual Studio



Формат записи директив и клауз OpenMP

- Формат записи

```
#pragma omp имя_директивы [clause,…]
```

- Пример

```
#pragma omp parallel default(shared)private(beta,pi)
```

Директива

Клаузы (clause)

Директива – описывает, что делать

Клауза – это что-то вроде настроек директив

А ещё бываю функции – это просто команды делающие что-то полезное, типа получения номера нити, но это не директивы.

Пример: Объявление параллельной секции

```
#include <omp.h>
int main()
{
  // последовательный код
  #pragma omp parallel
  {
    // параллельный код
  }
  // последовательный код

  return 0;
}
```

Некоторые функции OpenMP

omp_get_thread_num(); - возвращает номер нити типом int. Вне параллельной секции всегда вернёт 0

omp_get_num_threads(); - возращает общее количество нитей типом int. Вне параллельной секции всегда вернёт 1.

omp_get_wtime(); - возращает время в секундах типа double с момента некого «момента в прошлом». «Момент в прошлом» является произвольным, но он гарантировано не меняется с момента зпуска программы. Т.е. для подсчёта времени нужно вычесть одно время из другого.

Пример:

Hello, World!

```
#include <stdio.h>
#include <omp.h>
int main()
{
  printf("Hello, World!\n");
  #pragma omp parallel
  { int i,n;
    i = omp_get_thread_num();
    n = omp_get_num_threads();
    printf("I'm thread %d of %d\n",i,n);
  }
  return 0;
}
```

Задание числа потоков

- Переменная среды OMP_NUM_THREADS
- Функция omp_set_num_threads(int)

```
omp_set_num_threads(4);
#pragma omp parallel
{ . . .
}
```

- Параметр(клауза, условие) num_threads

```
#pragma omp parallel num_threads(4)
{ . . .
}
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - **private**
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copyin

Своя локальная переменная в каждом потоке. Делает переменные приватными.

```
int num;
#pragma omp parallel private(num)
{
  num=omp_get_thread_num()
  printf("%d\n",num);
}
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - **firstprivate**
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copyin

Локальная переменная с инициализацией. Делает переменные локальными и помещает в локальные переменные значение глобальной

```
int num=5;
#pragma omp parallel \
            firstprivate(num)
{
  printf("%d\n",num);
}
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - **lastprivate**
  - shared
  - default
  - reduction
  - threadprivate
  - copyin

Локальная переменная с сохранением последнего значения (в последовательном исполнении). В глобальную переменную помещает последнее значение из приватной.

```
int i,j;
#pragma omp parallel for \
            lastprivate(j)
  for (i=0;i<100;i++) j=i;

printf("Последний j = %d\n",j);
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - **shared**
  - default
  - reduction
  - threadprivate
  - copyin

Разделяемая (общая) переменная. Делает переменные разделяемыми (глобальными)

```
int i,j;

#pragma omp parallel for \
            shared(j)
for (i=0;i<100;i++) j=i;

printf("j = %d\n",j);
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - **default**
  - reduction
  - threadprivate
  - copyin

Задание области видимости не указанных явно переменных

```
int i,k,n=2;
#pragma omp parallel shared(n) \
            default(private)
{
  i = omp_get_thread_num() / n;
  k = omp_get_thread_num() % n;
  printf("%d %d %d\n",i,k,n);
}
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - **reduction**
  - threadprivate
  - copyin

Переменная для выполнения редукционной операции

```
int i,s=0;
#pragma omp parallel for \
            reduction(+:s)
  for (i=0;i<100;i++)
    s += i;

printf("Sum: %d\n",s);
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - **threadprivate**
  - copyin

Объявление глобальных переменных локальными для всех потоков, кроме нулевого. Для нулевого x остаётся глобальным.

```
int x;
#pragma omp threadprivate(x)

int main()
{
  . . .
}
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - **copyin**

Объявление глобальных переменных локальными для всех потоков, кроме нулевого с инициализацией

```
int x;
#pragma omp threadprivate(x)
#pragma omp copyin(x)
int main()
{
  . . .
}
```

#pragma omp **parallel**

«Единственная причина для существования времени — чтобы все не случилось одновременно»
Альберт Эйнштейн

Директива определяет параллельную область. Область программы, которая выполняется несколькими потоками одновременно. Это фундаментальная конструкция, которая начинает параллельное выполнение.

```
#pragma omp parallel [клауза [ клауза]...]
// структурный блок кода, который будет выполнен параллельно
```

#pragma omp **parallel**

При входе в параллельную область порождаются новые OMP_NUM_THREADS-1 нитей, каждая нить получает свой уникальный номер, причём порождающая нить получает номер 0 и становится основной нитью группы (“мастером”). Остальные нити получают в качестве номера целые числа с 1 до OMP_NUM_THREADS-1. Количество нитей, выполняющих данную параллельную область, остаётся неизменным до момента выхода из области. При выходе из параллельной области производится неявная синхронизация и уничтожаются все нити, кроме породившей.

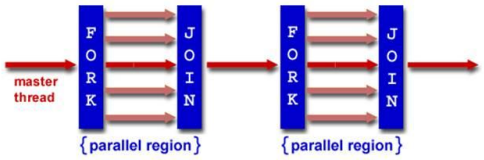
Когда поток встречает параллельную конструкцию, то создаётся группа потоков в одном из следующих случаев:

- Не присутствует клауза **if**
- Значение скалярного выражения в клаузе **if** не равно нулю

Клауза **if** может отключить параллельную секцию. Это может быть удобно для программного управления, типа «применять параллелизм или нет»

#pragma omp **parallel**

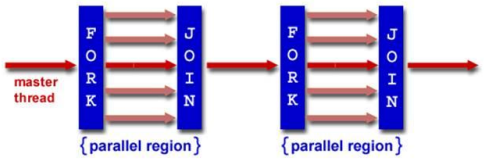
Главный поток группы (master thread), получает номер 0, и все потоки, включая основной, выполняют параллельно. Число потоков в группе управляется переменной окружения или библиотечной функцией. После создания параллельной области число, потоков на время выполнение этой области остается неизменным. После завершения этой области число потоков может быть изменено. Если выражение в клаузе **if** равно 0 (false), то область выполняется последовательно.



#pragma omp **parallel**

Директива **parallel** сообщает компилятору, что структурированный блок кода должен быть выполнен параллельно, в нескольких потоках.

Каждый поток будет выполнять один и тот же поток команд, но не один и тот же набор команд – все зависит от операторов, управляющих логикой программы, таких как if-else.



#pragma omp **parallel**
Пример

```
#include <stdio.h>
#include <omp.h>

void test(int val)
{
   #pragma omp parallel if (val)
   if (omp_in_parallel())
   {
      #pragma omp single
      printf_s("val = %d, parallelized with %d threads\n",
            val, omp_get_num_threads());
   }
   else
   {
      printf_s("val = %d, serialized\n", val);
   }
}

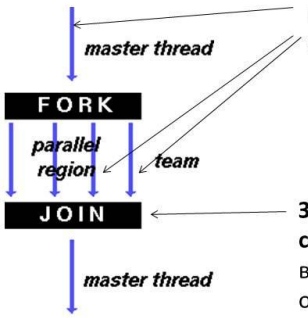
int main( )
{
   omp_set_num_threads(2);
   test(0);
   test(2);
}
```

Результат работы:

val = 0, serialized val = 2, parallelized with 2 threads

#pragma omp **parallel**

В конце параллельной области осуществляется, не указанная явно, барьерная синхронизация.



Нити (они потоки, они же threads)

Здесь по умолчанию происходит синхронизация. Главная нить не выйдет, пока не сработают остальные. Но это можно отключить клаузой **nowait**

#pragma omp **parallel**

Клауза одна из следующих:

- **if (скалярное выражение)**
- **num_threads (целое число)**
- **private (список)**
- **shared (список)**
- **default (shared | none)**
- **firstprivate (список)**
- **reduction (оператор : список)**
- **copyin (список)**
- **proc_bind (master | close | spread)**

#pragma omp **parallel**
Клаузы

- if (условие) – выполнение параллельной области по условию. Вхождение в параллельную область осуществляется только при выполнении некоторого условия. Если условие не выполнено, то директива не срабатывает и продолжается обработка программы в прежнем режиме;
- num_threads (целочисленное выражение) – явное задание количества нитей, которые будут выполнять параллельную область; по умолчанию выбирается последнее значение, установленное с помощью функции omp_set_num_threads(), или значение переменной OMP_NUM_THREADS;
- default(shared|none) – всем переменным в параллельной области, которым явно не назначен класс, будет назначен класс shared; none означает, что всем переменным в параллельной области класс должен быть назначен явно;
- private (список) – задаёт список переменных, для которых порождается локальная копия в каждой нити; начальное значение локальных копий переменных из списка не определено;
- firstprivate (список) – задаёт список переменных, для которых порождается локальная копия в каждой нити; локальные копии переменных инициализируются значениями этих переменных в нити-мастере;

#pragma omp **parallel**
Клаузы

- shared (список) – задаёт список переменных, общих для всех нитей;
- copyin (список) – задаёт список переменных, объявленных как threadprivate, которые при входе в параллельную область инициализируются значениями соответствующих переменных в нити-мастере;
- reduction (оператор:список) – задаёт оператор и список общих переменных; для каждой переменной создаются локальные копии в каждой нити; локальные копии инициализируются соответственно типу оператора (для аддитивных операций – 0 или его аналоги, для мультипликативных операций – 1 или её аналоги); над локальными копиями переменных после выполнения всех операторов параллельной области выполняется заданный оператор; оператор это: +, *, -, &, |, ^, &&, ||; порядок выполнения операторов не определён, поэтому результат может отличаться от запуска к запуску.

#pragma omp **parallel**

Если поток в группе, выполняя параллельную область кода, встретит другую параллельную конструкцию, она создает новую группу и становится основным потоком в новой группе. Вложенные параллельные области выполняются по умолчанию последовательно. И как следствие, по умолчанию вложенная параллельная область выполняется группой, состоящей из одного потока.

Поведение по умолчанию может быть изменено путем применения библиотечной функции времени выполнения **omp_set_nested** или переменной среды **OMP_NESTED**. Если вызывается библиотечные функции то нужно внести заголовочный файл **omp.h** .

#pragma omp **parallel**
Вложенность. Пример

```
#include <omp.h>
#include <stdio.h>
void report_num_threads(int level)
{
  #pragma omp single
  {
    printf("Level %d: number of threads in the team - %d\n",
           level, omp_get_num_threads());
  }
}
int main()
{
  omp_set_dynamic(0);
  omp_set_nested(...);//Меняем
  #pragma omp parallel num_threads(2)
  {
    report_num_threads(1);
    #pragma omp parallel num_threads(2)
    {
      report_num_threads(2);
      #pragma omp parallel num_threads(2)
      {
        report_num_threads(3);
      }
    }
  }
  return 0;
}
```

Результат работы:

```
// omp_set_nested(true);

Level 1: number of threads in the team - 2
Level 2: number of threads in the team - 2
Level 2: number of threads in the team - 2
Level 3: number of threads in the team - 2
Level 3: number of threads in the team - 2
Level 3: number of threads in the team - 2
Level 3: number of threads in the team - 2

// omp_set_nested(false);

Level 1: number of threads in the team - 2
Level 2: number of threads in the team - 1
Level 3: number of threads in the team - 1
Level 2: number of threads in the team - 1
Level 3: number of threads in the team - 1
```

Области видимости переменных

- Переменные, объявленные внутри параллельного блока, являются локальными для потока:

```
#pragma omp parallel
{
  int num;
  num = omp_get_thread_num()
  printf("Поток %d\n",num);
}
```

Области видимости переменных

- Переменные, объявленные вне параллельного блока, определяются параметрами директив OpenMP:
  - private
  - firstprivate
  - lastprivate
  - shared
  - default
  - reduction
  - threadprivate
  - copying

Синхронизация потоков

- Директивы синхронизации потоков:
  - master
  - critical
  - barrier
  - atomic
  - flush
  - ordered
- Блокировки
  - omp_lock_t

Синхронизация потоков

- Директивы синхронизации потоков:
  - **master**
  - critical
  - barrier
  - atomic
  - flush
  - ordered

Выполнение кода только главным потоком

```
#pragma omp parallel
{
  //code
  #pragma omp master
  {
    // critical code
  }
  // code
}
```

Синхронизация потоков

- Директивы синхронизации потоков:
  - master
  - **critical**
  - barrier
  - atomic
  - flush
  - ordered

Критическая секция

```
int x;
x = 0;

#pragma omp parallel
{
  #pragma omp critical
    x = x + 1;
}
```

Синхронизация потоков

- Директивы синхронизации потоков:
  - master
  - critical
  - **barrier**
  - atomic
  - flush
  - ordered

Барьер

```
int i;

#pragma omp parallel for
  for (i=0;i<1000;i++)
  {
    printf("%d ",i);
    #pragma omp barrier
  }
```

Синхронизация потоков

- Директивы синхронизации потоков:
  - master
  - critical
  - barrier
  - **atomic**
  - flush
  - ordered

Атомарная операция

```
int i,index[N],x[M];

#pragma omp parallel for \
            shared(index,x)
  for (i=0;i<N;i++)
  {
    #pragma omp atomic
      x[index[i]] += count(i);
  }
```

Синхронизация потоков

- Директивы синхронизации потоков:
  - master
  - critical
  - barrier
  - atomic
  - **flush**
  - ordered

Согласование значения переменных между потоками

```
int x = 0;
#pragma omp parallel sections \
                    shared(x)
{
  #pragma omp section
  { x=1;
    #pragma omp flush
  }
  #pragma omp section
    while (!x);
}
```

Синхронизация потоков

- Директивы синхронизации потоков:
  - master
  - critical
  - barrier
  - atomic
  - flush
  - **ordered**

Выделение упорядоченного блока в цикле

```
int i,j,k;
double x;
#pragma omp parallel for ordered
  for (i=0;i<N;i++)
  {   k = rand(); x = 1.0;
      for (j=0;j<k;j++) x=sin(x);
      printf("No order: %d\n",i);
    #pragma omp ordered
      printf("Order: %d\n",i);
  }
```

Синхронизация потоков

- Блокировки
  - omp_lock_t
    - void omp_init_lock(omp_lock_t *lock)
    - void omp_destroy_lock(omp_lock_t *lock)
    - void omp_set_lock(omp_lock_t *lock)
    - void omp_unset_lock(omp_lock_t *lock)
    - int omp_test_lock(omp_lock_t *lock)
  - omp_nest_lock_t
    - void omp_init_nest_lock(omp_nest_lock_t *lock)
    - void omp_destroy_nest__lock(omp_nest_lock_t *lock)
    - void omp_set_nest__lock(omp_nest_lock_t *lock)
    - void omp_unset_nest__lock(omp_nest_lock_t *lock)
    - int omp_test_nest__lock(omp_nest_lock_t *lock)

Функции OpenMP

- void omp_set_num_threads(int num_threads)
- int omp_get_num_threads(void)
- int omp_get_max_threads(void)
- int omp_get_thread_num(void)
- int omp_get_num_procs(void)
- int omp_in_parallel(void)
- void omp_set_dynamic(int dynamic_threads)
- int omp_get_dynamic(void)
- void omp_set_nested(int nested)
- int omp_get_nested (void)
- double omp_get_wtick(void)
- Функции работы с блокировками

Порядок создания параллельных программ

1. Написать и отладить последовательную программу
2. Дополнить программу директивами OpenMP
3. Скомпилировать программу компилятором с поддержкой OpenMP
4. Задать переменные окружения
5. Запустить программу

Литература: Архитектуры и топологии многопроцессорных вычислительных систем: курс лекций : учебное пособие / А. В. Богданов, В. В. Корхов, В. В. Мареев, Е. Н. Станкова. – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2004. – 176 с. : ил.,табл., схем. – (Основы информационных технологий). – ISBN 5-9556-0018-3; То же [Электронный ресурс]. - URL: https://biblioclub.ru/index.php?page=book&id=232995

Лекция 5. Итоговая лекция. Подведение и анализ промежуточных результатов освоения дисциплины

Итоговая лекция завершает изучение дисциплины, обобщает пройденное за весь период. При ее подготовке учитываются пробелы в знаниях аспирантов, выявленные на практических занятиях и при оценке самостоятельной работы. Аспиранты получают установку на их устранение и на дальнейшее совершенствование своей подготовки в данной области. На лекции подводятся итоги изучения дисциплины, показывается ее значение в профессиональной сфере и формировании научного мировоззрения, обсуждается вопрос о подведении итогов изучения дисциплины – сдаче зачета или экзамена.

2. Методические указания по подготовке к занятиям семинарского типа

2.1. Планы и (или) методические рекомендации к практическим занятиям

В рамках проведения занятий по дисциплине предполагается организация следующих практических занятий:

Практическое занятие 1. Введение в область организации вычислительных комплексов и систем

Тематика: Программное обеспечение многопроцессорных ВС.

Текст задания: Используя возможности поисковых систем, найти ответы на следующие вопросы:

1. Какой класс ЭВМ называется многопроцессорными?
2. Охарактеризуйте основные подклассы многопроцессорных систем.
3. Опишите способы построения коммуникационных сред для высокопроизводительных ВС?
4. В каких областях современной науки и техники высокопроизводительные вычислительные системы наиболее востребованы?
5. Охарактеризуйте существующие подходы к построению программных комплексов для высокопроизводительных вычислительных систем.
6. Опишите возможности, назначение и основные особенности программирования для систем с общей памятью.
7. Опишите возможности и основные особенности программирования для систем с передачей сообщений.
8. В каком случае, при использовании OpenMP или MPI, программисту потребуется больше времени для адаптации последовательной программы к возможностям параллельной системы?
9. Охарактеризуйте функции посылки и приема сообщения в библиотеке MPI.
10. Может ли MPI-программа быть выполнена на SMP–системе?

Шкала оценивания:

Оценка «отлично» за практическую работу выставляется, если на все вопросы дан полный, обоснованный и правильный ответ;

Оценка «хорошо» выставляется, если аспирант ошибся при ответе на один-два вопроса;

Оценка «удовлетворительно» выставляется, если аспирант ошибся при ответе на три-четыре вопроса.

Оценка «неудовлетворительно» выставляется, если аспирант допустил больше четырех ошибок.

Практическое занятие 2. Архитектура программного обеспечения

Тематика: Технологии структурного анализа требований: SADT, IDEF0.

Текст задания:

1. В любой специализированной программе в нотации IDEF0 построить модель описанной в задании предметной области. Модель должна содержать контекстную диаграмму, диаграмму первого уровня и одну диаграмму детализации любого из процессов диаграммы первого уровня. В отчете привести распечатки или изображения всех диаграмм, кратко описать построение и привести обоснование построения.

2. Построить дерево узлов для полученной модели. В отчете привести распечатку или изображение диаграммы дерева узлов, кратко описать построение.

Выбор предметной области согласовывается с преподавателем.

Торговая фирма. Создать функциональную модель деятельности торговой фирмы по реализации продовольственной продукции, учитывая работу фирмы с клиентами, поставщиками, доставку продукции от поставщиков и по торговым точкам клиентов.

Библиотека. Создать функциональную модель деятельности библиотеки, учитывая работу библиотеки с клиентами и поставщиками книг. Следует отметить, что кроме выдачи книг современные библиотеки оказывают своим клиентам дополнительные услуги: выдают клиентам CD, видео и аудио кассеты, проводят конференции, делают копирование, ламинирование, позволяют работать с электронными каталогами и выходить в Интернет.

Банк. Создать функциональную модель деятельности банка, учитывая, что современные банки оказывают своим клиентам широкий спектр услуг, начиная от обслуживания счетов, принятия вкладов, кредитования и заканчивая работой на рынке ценных бумаг, работой с инвестициями, валютными операциями, и другие возможные направления деятельности.

ВУЗ. Создать функциональную модель деятельности ВУЗа, учитывая работу ВУЗа как по основным направлениям деятельности: обеспечение учебного процесса, научной работы, так и по дополнительным процессам: международная деятельность, работа по договорам, социальная работа.

Компьютерная фирма. Создать функциональную модель деятельности компьютерной фирмы, учитывая, что фирма торгует компьютерами в собранном виде и комплектующими. Фирма работает как с производителями компьютерной техники, так и с клиентами. Фирма оказывает ряд дополнительных услуг: установка программного обеспечения, подключает к Интернету клиентов, гарантийное обслуживание и т.д.

Кафедра ВУЗа. Создать функциональную модель деятельности кафедры ВУЗа, учитывая следующие направления: работа по обеспечению учебного процесса, работа по хоз. договорам, научно- исследовательская работа сотрудников и студентов и т.д.

Аэропорт. Создать функциональную модель работы аэропорта, учитывая работу аэропорта с авиакомпаниями, клиентами, поставщиками и т.д. Учесть, всевозможные работы аэропорта по техническому обслуживанию самолетов, обслуживанию клиентов через кассы, работу диспетчерской службы аэропорта.

Строительная фирма. Создать функциональную модель работы строительной фирмы. Описать работу фирмы, как с поставщиками, так и с клиентами. Следует отметь, что в настоящее время строительные организации обеспечивают полный технологический процесс, начиная

проведения исследований рынка, создания проекта, закупки материалов, непосредственного строительства и заканчивая продажей квартир.

Шкала оценивания:

Оценка «отлично» за практическую работу выставляется, если работа выполнена полностью правильно и самостоятельно, а аспирант способен уверенно отвечать на вопросы преподавателя о проделанной работе;

Оценка «хорошо» выставляется, если работа выполнена в целом корректна, но имеются неточности в ответах на вопросе преподавателя или некоторые действия выполнены неоптимально;

Оценка «удовлетворительно» выставляется, когда работа не может быть завершена аспирантом самостоятельно, без помощи преподавателя.

Оценка «неудовлетворительно» выставляется, если работа не была выполнена.

Практическое занятие 3. Принципы построения распределенных систем обработки информации

Тематика: Введение в распределенные системы.

Текст задания: Подготовить доклад по любой из представленных тем.

1. Технология Grid: базовые понятия и определения, эволюция Grid систем, концепция Gird сети, архитектура Grid сети, инструментальные средства построения Grid систем, основные приложения, современное состояние и перспективы развития.

2. Технология Peer-to-Peer: базовые понятия и определения, эволюция пиринговых систем, концепция пиринговой сети, архитектура пиринговой сети, инструментальные средства построения пиринговых систем, основные приложения, современное состояние и перспективы развития.

3. Технология мультиагентных систем (МАС): базовые понятия и определения, эволюция МАС, концепция МАС, архитектуры МАС, инструментальные средства построения МАС, основные приложения, современное состояние и перспективы развития.

4. Обзор современных технологий построения распределенных приложений: программирование сокетов, RPC, COM , CORBA, Java RMI. Общая характеристика, достоинства и недостатки, принципиальные отличия, области применения.

5. Технология программирования сокетов: базовые понятия и определения, основы спецификации BSD Sockets, взаимодействие приложений на базе механизма сокетов. Реализация механизма сокетов в средах RAD.

6. Технология COM: общая характеристика и принципы организации, приемы построения распределенных приложений на базе COM в средах RAD.

7. Удаленный вызов процедур RPC: общая характеристика и принципы организации, расширение семантики локальных вызовов.

8. Технология CORBA: общая характеристика и принципы работы, приемы построения распределенных приложений на базе COM в средах RAD.

9. Распределенные вычисления. Общая характеристика и особенности языков параллельного программирования.

10. Технология распределенных баз данных в корпоративных ИС: базовые понятия и определения, структура сети, логика работы и основные принципы функционирования.

11. Технология тиражирования данных в корпоративных ИС: базовые понятия и определения, структура сети, логика работы и основные принципы функционирования

12. Язык SQL – общая характеристика, интерфейс с традиционными языками программирования.

13. Обзор языков параллельного программирования.

14. Интранет системы. Основные понятия и определения. Построения интранет систем. Технологии и средства создания web-приложений.

15. Технология Java RMI: общая характеристика и принципы работы, приемы построения распределенных приложений на базе Java RMI в средах RAD.

16. Финансово экономические информационные системы: информатизация банковской деятельности обзор существующих систем, принципы построения и логика функционирования, области применения.

17. Финансово экономические информационные системы: автоматизированные системы в торговле обзор существующих систем, принципы построения и логика функционирования, области применения.

18. Корпоративные Интернет порталы: основные понятия и определения, принципы построения и логика функционирования, области применения.

19. Корпоративные торговые площадки и электронная коммерция: основные понятия и определения, принципы построения и логика функционирования, области применения.

20. Системы дистанционного образования: основные понятия и определения, принципы построения и логика функционирования, области применения.

21. B2B системы (B2B–"business to business"): общая характеристика, принципы построения и функционирования, области применения.

22. Облачные технологии (Cloud Computing).

Требования к докладам:

1) Время выступления 5-7 мин (без учета вопросов преподавателя).

2) Презентация объемом не более 12-15 слайдов, содержащая основные положения доклада, а также иллюстративный материал к докладу (рисунки, схемы, графики, листинги программ и т.д.)

3) Текст доклада объемом не более 25 машинописных страниц, содержащий оглавление, введение, основной текст доклада, заключение, список литературы. Машинописный вариант доклада оформляется как контрольная работа/реферат в соответствие с правилами, описанными в руководстве по оформлению курсовых и дипломных работ для студентов Ивановского государственного университета. Сдается только в электронном виде вместе с презентацией.

Критерии оценки:

Оценки в баллах	Критерии
2	Самостоятельность в подготовке материала
3	Аргументированность и полнота материала
2	Использование современной и актуальной литературы
3	Качество презентации
10	**Итого**

Шкала оценивания:

Оценка «отлично» выставляется аспиранту, если сумма баллов по всем критериям равна 9-10 баллам;

Оценка «хорошо» выставляется аспиранту, если сумма баллов по всем критериям равна 7-8 баллам;

Оценка «удовлетворительно» выставляется аспиранту, если сумма баллов по всем критериям равна 5-6 баллам;

Оценка «неудовлетворительно» выставляется аспиранту, если сумма баллов по всем критериям равна 0-4 баллам.

Практическое занятие 4. Многопоточное и параллельное программирование
Тематика: Создание/завершение потоков.

Текст задания:

1. Разработайте многопоточное приложение, выполняющее вычисление произведения матриц A (m×n) и B (n×k). Элементы c_{ij} матрицы произведения C = A×B вычисляются параллельно p однотипными потоками. Если некоторый поток уже вычисляет элемент c_{ij} матрицы C, то следующий приступающий к вычислению поток выбирает для расчета элемент c_{ij+1}, если j<k, и c_{i+1k}, если j=k. Выполнив вычисление элемента матрицы-произведения, поток проверяет, нет ли элемента, который еще не рассчитывается. Если такой элемент есть, то приступает к его расчету. В противном случае отправляет (пользовательское) сообщение о завершении своей работы и приостанавливает своё выполнение. Главный поток, получив сообщения о завершении вычислений от всех потоков, выводит результат на экран и запускает поток, записывающий результат в конец файла-протокола. В каждом потоке должна быть задержка в выполнении вычислений (чтобы дать возможность поработать всем потокам). Синхронизацию потоков между собой организуйте через критическую секцию или мьютекс.

2. Пул потоков – это набор потоков, которые могут использоваться для решения различных задач. Задачи помещаются в общую очередь. Свободный поток (поток, который не решает в настоящий момент никакую задачу) обращается к очереди для получения очередной задачи, выполняет ее, после чего запрашивает в очереди следующую задачу. В результате исполнения некоторой задачи могут появляться новые задачи. Они также добавляются в очередь. Программа завершается, когда в очереди не остается задач и все потоки оказываются свободными. Необходимо реализовать пул потоков и на его основе решить задачу: генерации простых чисел с помощью алгоритма «Решето Эратосфена».

Шкала оценивания:

Оценка «отлично» за практическую работу выставляется, если работа выполнена полностью правильно и самостоятельно, а аспирант способен уверенно отвечать на вопросы преподавателя о проделанной работе;

Оценка «хорошо» выставляется, если работа выполнена в целом корректна, но имеются неточности в ответах на вопросе преподавателя или некоторые действия выполнены неоптимально;

Оценка «удовлетворительно» выставляется, когда работа не может быть завершена аспирантом самостоятельно, без помощи преподавателя.

Оценка «неудовлетворительно» выставляется, если работа не была выполнена.

Практическое занятие 5. Заключительное занятие. Подведение и анализ промежуточных результатов освоения дисциплины

При подготовке материалов для заключительного практического занятия, преподаватель учитывает недостаточную степень развития навыков аспирантов, определенную в ходе практических занятий и при оценке самостоятельной работы. На практическом занятии обсуждаются итоги проведения занятий по дисциплине, проверяется полнота и качество собранного аспирантом портфолио материалов и результатов выполнения заданий по дисциплине. Обсуждаются вопросы, вынесенные для проведения зачета с оценкой. Оцениваются индивидуальные достижения, полученные аспирантов в ходе внеурочной работы (сертификаты

олимпиад, конкурсов, грантов, написание тезисов/статей в сборники конференций различного уровня).

3. Методические указания по выполнению других видов самостоятельной работы по дисциплине

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ВОПРОСОВ ДЛЯ ЗАЧЕТА С ОЦЕНКОЙ

1. Понятие архитектуры ВС.
2. Классификация ВС.
3. Объектно-ориентированный анализ (ООА). Модели ООА.
4. Классы анализа. Их выявление и описание.
5. Построение статической модели анализа.
6. Динамической модели без учета состояния. Порядок построения и описания.
7. Конечные автоматы.
8. Динамической модели с учетом состояния. Порядок построения и описания.
9. Объектно-ориентированное проектирование (ООП).
10. Проектирование классов программного обеспечения.
11. Проектирование связей между классами программного обеспечения.
12. Уточнение отношений между классами, выявленными на этапе анализа.
13. Выделение подсистем на этапе проектирования.
14. Шаблоны архитектуры программного обеспечения.
15. Интерфейсы и компоненты.
16. Характеристика распределенной обработки данных.
17. Основные принципы построения распределенных информационных систем.
18. Различные способы представления данных в информационных системах, языки гипертекстовой разметки.
19. Принципы построения и основные задачи, выполняемые серверными программами
20. Основные технологии построения распределенных информационных систем.
21. Технологии и модели «Клиент-сервер».
22. Связь. Понятие прикладных протоколов.
23. Типовые задачи, решаемые при помощи программ, выполняемых на стороне клиента.
24. Средства создания программ, выполняемых на стороне сервера.
25. Клиенты. Пользовательские интерфейсы. Клиентское программное обеспечение, обеспечивающее прозрачность распределения

По представленным вопросам предполагается выполнение аспирантом конспектов или повторение материала с использованием конспектов лекций.

4. Примеры контрольных заданий (оценочных средств), используемых при текущем контроле успеваемости и промежуточной аттестации по дисциплине

Демоверсия итогового тестирования:

1. В рамках какой модели для всех параллельных процессов используется один и тот же код?

а) MIMD;

б) SPMD (правильный ответ);

в) MPMD;

2. Какие процессы обязательно участвуют в операциях типа точка-точка?

а) процесс-отправитель;

б) процесс-получатель;

в) как тот, так и другой (правильный ответ);

3. Процедуры с синхронизацией приостанавливают работу процесса...

а) до выполнения некоторого условия (правильный ответ);

б) во время проверки массивов данных;

в) после инициализации соответствующей коммуникационной операции;

4. Выполнение какой процедуры не зависит от соответствующего вызова процедуры приема сообщения?

а) MPI_BSEND (правильный ответ);

б) MPI_SSEND;

в) MPI_RSEND;

5. Локальная вычислительная система, состоящая из множества независимых компьютеров и сети, связывающей их, носит название:

а) мейнфрейм;

б) кластер (правильный ответ);

в) сервер;

6. К программному обеспечению управления ресурсами кластера следует относить средства:

а) инсталляции (правильный ответ);

б) администрирования (правильный ответ);

в) планирования потоков работ (правильный ответ);

7. Методы, которые могут быть перенесены для исполнения на другие машины, выделяются в C# с помощью ключевого слова:

а) type;

б) struct;

в) movable (правильный ответ);

г) conf;

8. Методы, которые должны быть исполнены асинхронно локально, выделяются в C# с помощью ключевого слова:

а) async (правильный ответ);

б) define;

в) conf;

г) struct.

Автор(ы)-составитель(и): *Привалов А.Н.*, профессор кафедры математики, информатики и методики обучения, доктор технических наук, профессор