**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМ. Р. Е. АЛЕКСЕЕВА»**

**(НГТУ)**

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

**Методические указания**

**по выполнению лабораторных работ**

**по дисциплине «**Технологии распределённой обработки данных**»**

Направление подготовки

\_\_\_\_\_09.03.01 Информатика и вычислительная техника \_\_\_\_\_\_\_\_

код и наименование направления подготовки

Профиль подготовки

Вычислительные машины, комплексы, системы и сети

наименование профиля подготовки

Уровень высшего образования

Бакалавриат

Форма обучения

\_\_\_\_\_\_очная\_\_\_\_\_\_\_\_

Нижний Новгород

2016

Разработчик / составитель методических указаний по выполнению лабораторных работ по дисциплине «Технологии распределённой обработки данных»:

\_\_к.т.н., доцент, Гай В. Е. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ученое звание, степень, фамилия, инициалы

Кафедра «Вычислительные системы и технологии»

Дата, подпись \_\_3 марта 2016 г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Методические указания по выполнению лабораторных работ по дисциплине «Технологии распределённой обработки данных» рассмотрены на заседании кафедры

\_\_\_\_\_\_\_\_\_\_\_\_"Вычислительные системы и технологии"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

наименование кафедры

Протокол № 6 от «4»марта 2016 г.

Заведующий кафедрой член-корреспондент РАН Кондратьев В. В.

ученое звание, степень фамилия, имя, отчество

Дата, подпись \_\_4 марта 2016 г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Методические указания по выполнению лабораторных работ по дисциплине «Технологии распределённой обработки данных» утверждены учебно-методическим советом образовательно-научного института «Института радиоэлектроники и информационных технологий»

Протокол №7 от «10» марта 2016 г.

Методические рекомендации зарегистрированы в методическом отделе под учётным номером № \_\_\_\_.

Ведущий инженер Чуева Н. А. \_\_\_\_\_\_\_\_\_\_

Методические указания по выполнению лабораторных работ по дисциплине «Технологии распределённой обработки данных» предназначены для бакалавров третьего курса, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника», и содержат указания для проведения лабораторных занятий по курсу «Технологии распределённой обработки данных».

Цель методических указаний: помочь студентам при изучении учебной программы с использованием лекционных материалов и рекомендуемой учебно-методической литературы при формировании необходимых компетенций для разработки система распределённой обработки данных.

В процессе выполнения лабораторных работ по дисциплине «Технологии распределённой обработки данных» студент должен: строго выполнять весь объем самостоятельной подготовки, указанный в описаниях соответствующих лабораторных работ; знать, что выполнению каждой работы предшествует проверка готовности студента, которая проводится преподавателем; знать, что после выполнения работы студент должен представить отчет о проделанной работе с обсуждением полученных результатов и выводов.

В процессе выполнения лабораторных работ по данной дисциплине студент формирует и демонстрирует следующие общекультурные и профессиональные компетенции: ОПК-2. После проведения лабораторных работ бакалавр должен знать области применения технологий распределённой обработки данных, современные технологии распределённой обработки данных, принципы построения и методы работы в распределенных системах обработки информации (ОПК-2) и уметь применять технологии построения систем распределённой обработки данных (ОПК-2).

Окончательная оценка за выполнение лабораторной работы студенту ставится после выполнения лабораторной работы и представления отчета о проделанной работе с обсуждением полученных результатов и выводов. Порядок выполнения пропущенных работ по уважительным и неуважительным причинам: лабораторные работы взаимосвязаны, поэтому пока не будет выполнена первая работа, студент не допускается к выполнению второй работы и т.д.

В ходе выполнения лабораторной работы студент должен выполнить предложенное задание и подготовить отчет о проделанной работе. Форма сдачи лабораторной работы предполагает демонстрацию выполненного задания (программного продукта) и знаний теоретической части вопроса, рассмотренного в лабораторной работе.

Отчет по лабораторной работе должен содержать:

1. Тему и цель лабораторной работы;

2. Вариант задания на лабораторную работу;

3. Краткие теоретические сведения и описание алгоритма работы программы;

4. Листинг разработанной программы с подробными комментариями;

5. Результаты работы программы;

6. Выводы.

**Лабораторная работа № 1. Разработка сервиса с использованием технологии DSS**

**Цель**: создание простейшего DSS сервиса, реализующего параллельную обработку данных

**Краткие теоретические сведения**

**Обзор технологии DSS**

Сервисы могут взаимодействовать друг с другом как партнёры. Этот процесс называется координация (orchestration).

С помощью DSS Runtime выполняется запуск и остановка сервисов, организация потоков сообщений между сервисами через их порты. Фактически DSS состоит из нескольких сервисов, которые загружают конфигурацию сервиса, управляют безопасностью, доступом к локальным файлам и ресурсами, предоставляют список запущенных сервисов, интерфейс к сервису через Internet. Существует базовый класс DSS, от которого производны все сервисы.

Сервисы, которые входят в состав приложения, могут создаваться и уничтожаться динамически. В большинстве случаев используемые сервисы указываются в манифесте. Манифест – XML файл, в котором перечисляются сервисы, а также отношения между ними.

Сервис состоит из нескольких компонент:

1) соглашение – определяет сообщения, которые можно отправить сервису, а также глобально уникальную ссылку, называемую идентификатором соглашения, которая идентифицирует сервис и записывается в форме URI (Universal Resourse Identifier);

2) внутренне состояние – информация, которую хранит сервис; данная информация может использоваться операциями;

3) поведение – набор операций, которые сервис может выполнить, операции реализуются в виде обработчиков;

4) контекст исполнения – отношения сервиса с другими сервисами и начальное состояние сервиса.

Компоненты сервиса показаны в виде диаграммы на рис. 2.

|  |
| --- |
|  |
| Рис. 2. Компоненты сервиса |

**Общие соглашения**

Концепция общих соглашений по смыслу близка к абстрактным классам. Общие соглашения не имеют собственной реализации. Они состоят только из определений, необходимых для идентификатора соглашения, состояния и главного порта операций.

Для реализации общего сервиса необходимо разработать сервис, который основан на общем соглашении. Проще всего это можно сделать если при вызове команды DssNewService указать параметр /alt и реализуемое сервисом общее соглашение.

Реализация общего сервиса включает два порта операций: первый для нового сервиса, второй – для получения запросов, описанных в общем соглашении. В исходном коде для нового сервиса нужно реализовать обработчики для всех операций, определенных в сервисе общего соглашения. В новый сервис не обязательно добавлять обработчики (Lookup, Drop и Get добавляются по умолчанию). Простейшая реализация, состоит только из обработчиков для общих операций, в этом случае порт для собственных операций не требуется.

Общие соглашений представляют собой попытку создания аппаратно-независимого сервиса, подобного драйверу устройства, который отделяет детали аппаратного обеспечения от операционной системы.

**Lev2. DSS узел**

Программа DssHost.exe отвечает за реализацию среды выполнения для MRDS. DssHost запускает веб-сервер, что позволяет опрашивать DSS-узел и запущенные сервисы, используя веб-браузер. Это означает, что можно просматривать и управлять состоянием сервисов, запущенных на DSS узле через сеть без установки каких-либо специальных клиентов. После запуска DSS узла, несколько сервисов запускается автоматически:

1) Console Output: захватывает и фильтрует информационные сообщения и сообщения об ошибках;

2) Constructor: создает экземпляры сервисов;

3) Control Panel: поддерживает интерфейс для запуска и остановки сервисов вручную;

4) Manifest Loader: выполняет в режиме интерпретации манифесты для определения сервисов, входящих в приложение;

5) Mount Service: поддерживает доступ к локальной файловой системе (только в пределах каталога MRDS);

6) Embedded Resource Manager: позволяет работать с ярлыками, битовыми картами, XSLT файлами и другими ресурсами;

7) Security Manager: управляет безопасностью на DSS узле;

8) Service Directory: поддерживает список сервисов на DSS узле;

9) Partner Manager: выполняет чтение конфигурационного файла при запуске сервиса.

Расположение локального узла DSS можно определить с помощью статического класса LayoutPaths:

string logdir = LayoutPaths.RootDir+LayoutPaths.Logdir;

После выполнения команды в переменной logdir будет находиться полный путь к каталогу Log.

Запустите DSS Command Promt, выбрав Пуск→Программы→Microsoft Robotics Studio→DSS Command Prompt. Введите следующую команду, чтобы посмотреть все возможные параметры командной строки:

DssHost /?

Запустите DssHost, используя следующую команду:

DssHost /p:50000 /t:50001

Результат выполнения команды должен выглядеть, как показано на рис. 3. Из рисунка видно, что были запущены сервисы Directory, Constructor и Console Output, но не было загружено манифестов.

|  |
| --- |
|  |
| Рис. 3. Результат запуска узла |

Для использования команд MRDS нужно запустить DSS Command Prompt. При выполнении данной команды выполняется установка корневого каталога, переменных окружения и запуск дополнительных программ.

**Структура страниц веб-сервера**

После запуска DssHost, откройте веб-браузер и введите в адресную строку http://localhost:50000. На экране должно отобразиться что-то похожее на рис. 4. Заметьте, что в левой части окна размещено меню.

|  |
| --- |
|  |
| Рис. 4. Запущенный веб-сервер |

**Панель управления (Control Panel)**

После выбора пункт Control Panel отображаются сервисы, которые можно запустить на данном DSS узле. Прежде чем будет отображена информация может пройти немного времени, так как выполняется обновление кэша каталога сервисов. Сервисы из списка можно запустить вручную. Можно выполнить поиск сервиса, заполнив строку поиска. На рис. 5 показаны результаты поиска сервиса по строке «sim».

|  |
| --- |
|  |
| Рис. 5. Поиск сервиса |

В выпадающем списке рядом с каждым сервисом указан список манифестов, относящихся к сервису, которые были найдены DSS узлом. Для запуска экземпляра сервиса можно выбрать один из этих манифестов или оставить выпадающий список установленным в <<Start without manifest>> и нажать на кнопку Create. При наличии веб-камеры, убедитесь, что она подключена к компьютеру и нажмите на кнопку Create рядом с сервисом Webcam (не выбирайте манифест).

**Service Directory**

После запуска сервиса Webcam, этим сервисом можно управлять, выбрав пункт меню Service Directory (рис. 6).

|  |
| --- |
|  |
| Рис. 6. Каталог сервисов |

Каждый DSS узел предоставляет информацию о запущенных на нём сервисах. На рис. 6 в списке находятся два сервиса Webcam (информация о втором сервисе на рис. 6 уместилась не полностью), так как сервис реализует общее соглашение Webcam и своё собственное.

Выберите сервис Webcam в списке Service Instance Directory для просмотра информации о работе сервиса. В адресной строке будет отображён URL http://localhost:50000/webcam/c7354581-c28f-4b4c-b419-d5b7ea06aab9. Эти числа меняются при каждом запуске сервиса – они делают URI сервиса уникальными и позволяют уникально идентифицировать каждый экземпляр сервиса. Доступ к сервису Webcam можно получить лишь путем добавления его имени в конец URL адреса узла DSS.

Вывод сервиса WebCam представляет собой Web форму. С её помощью выполняется форматирование вывода состояния сервиса (для получения состояния сервиса и отображения его в веб-браузере используется запрос HttpGet). В форме можно устанавливать параметры камеры, включая интервал обновления (Refresh Interval), формат вывода (Display Format), разрешение изображения (Capture Format), указать режим обновления данных, получаемых с камеры (ручной или автоматический).

Рекомендуется запускать DSS узел из под учётной записи «Администратор». Это позволит избежать проблем с запуском DssHost из-за настроек брэндмауэра или настроек безопасности на компьютере.

На странице Security Manager отображается содержимое файла store\SecuritySettings.xml, который допускает редактирование. Если файл не найден, на DSS узле будут использоваться настройки безопасности по умолчанию. Имя файла, в котором записаны настройки безопасности, устанавливается в конфигурационном файле DssHost: bin\dsshost.exe.config. Ключ Security определяет расположение файла с настройками безопасности:

<!--comment the line below to disable security-->

<add key=\*security\* value="..\store\securitysettings.xml"/>

**Просмотр состояния сервиса**

Запустите пример из каталога Microsoft Robotics Developer Studio→Visual Simulation Environment→Lego NXT Tribot Simulation (меню Пуск). Откройте DSS узел в веб-браузере и выберите сервис Generic Contact Sensors для просмотра его состояния. Вывод на экран должен быть похож на рис. 7. Состояние выводится в виде XML кода. Значение параметра Pressed бампера установлено в false, т. е. в настоящий момент времени робот не взаимодействует ни с одним объектом симулятора.

|  |
| --- |
|  |
| Рис. 7. Состояние сервиса |

**Создание нового сервиса**

**Использование Visual Studio**

Создайте новый сервис (см. рис. 8), выполнив следующие шаги:

1) откройте Visual Studio и выберите File→New→Project;

2) выберите тип проекта Robotics и шаблон DSS Service (2.2);

3) убедитесь, что сервис расположен в папке Projects;

4) укажите для сервиса имя ServiceA.

|  |
| --- |
|  |
| Рис. 8. Создание сервиса |

После создания проекта в панели Solution Explorer появятся следующие исходные файлы:

1) AssemblyInfo.cs: содержит информацию о результирующей сборке (DLL библиотека);

2) ServiceA.cs: главный исходный файл сервиса (ядро сервиса);

3) ServiceA.manifest.xml: манифест, который используется DSS для загрузки сервиса (манифест – XML файл, описывающий сервисы которые, должны быть запущены при загрузке манифеста на DSS узел);

4) ServiceATypes.cs: содержит набор классов (типов), которые использются сервисом, идентификатор соглашения сервиса, типы сообщений, обрабатываемые главным портом сервиса, состояние сервиса.

Рассмотрим свойства проекта (страница Project Properties):

1) закладка Application: тип приложения – библиотека классов (DLL). Все сервисы MRDS являются динамически подключаемыми библиотеками. Имя сборки записывается в формате ServiceA.Yyyyy.Mmm, где yyyy – текущий год, а mm – текущий месяц. Дата в имени используется для создания различных версий сервисов с одинаковыми именами. Идентификаторы соглашений также содержат год и месяц. Имя сборки и идентификатор соглашения должны соответствовать. Пространством имён по умолчанию является Robotics.ServiceA;

|  |
| --- |
|  |
| Рис. 9. Закладка Application |

2) закладка Build: все сервисы MRDS находятся в папке bin, поэтому параметр Output Path установлен в ...\...\bin. Значение Output Path – относительный путь, это необходимо учитывать при переносе файлов проекта в другую папку, в этом случае указанный относительный путь не будет корректным;

|  |
| --- |
|  |
| Рис. 10. Закладка Build |

3) закладка Build Events:

"<MRDS path>\bin\dssproxy.exe"

/dll:"S(TargetPath)"/proxyprojectpath:"$(ProjectDir)Proxy

/keyfile:"$(AssemblyOriginatorKeyFile)"$(ProxyDelaySign)

S(CompactFrameworkProxyGen) /binpath:".

/referencepath:"<MRDS path>\bin\" /referencepath: "<MRDS path>\bin\"

Данная команда предназначена для создания proxy DLL сервиса.

|  |
| --- |
|  |
| Рис. 11. Закладка Build Events |

4) закладка Debug: после запуска отладчика, автоматически запускается программа Dsshost.exe. Dsshost.exe запускает DSS узел, который является средой выполнения сервиса. Рабочим каталогом для DSSHost является корневой каталог MRDS. Команде DSSHost передаются следующие параметры:

/p:50000

/t:50001

/manifest:"samples\config\LEGO.NXT.Tribot.Simulation.manifest.xml"

После компиляции проекта сервиса генерируются три dll библиотеки:

**1) сборка реализации сервиса, формируемая на основе файлов исходного кода, включенных в проект сервиса (Service.Y2012.M07.dll);**

**2) сборка DSS Proxy предоставляющая соглашение сервиса для его использования другими сервисами. Сборка proxy включает общедоступные операции и типы состояния, предоставляемые сервисом. Организация взаимодействия сервисов реализуется посредством Proxy dll (Service.Y2012.M07.proxy.dll).**

**3) сборка DSS Transform содержит описание соответствия между типами, определенными в реализации сервиса и реализации proxy. Данная сборка загружается автоматически с помощью среды выполнения DSS и взаимодействует только со сборкой реализации сервиса (Service.Y2012.M07.transform.dll).**

**Все три сборки копируются в каталог MRDS\bin.**

**Команда DssNewService**

Запустите DSS Command Prompt из меню MRDS и выполните следующие шаги:

1) перейдите в папку Projects;

2) запустите команду DssNewService для создания сервиса:

DssNewService /Service:ServiceB

3) откройте созданный сервис.

Команда DssNewService имеет несколько параметров. Параметры отображаются после запуска DssNewService со следующим параметром:

DssNewService /?

Параметры командной строки команды DssNewService:

1) URI префикс, который используется в идентификаторе соглашения, указывается с помощью. По умолчанию – shemas.tempuri.org;

2) параметры /month и /year определяют идентификатор соглашения и имя сборки (по умолчанию – текущий месяц и год);

3) /namespace – пространство имен сервиса;

4) /clone – используется для копирования существующего сервиса. В результате, создаётся образ сервиса, копирование исходных файлов не выполняется. После построения solution с помощью команды DssNewService проект будет содержать методы без реализации;

5) /alt указывает альтернативный сервис, то есть общее соглашение, которое должен реализовать разрабатываемый сервис в дополнение к собственному набору операций (при использовании /alt нужно добавить параметр /i).

**Исходные файлы сервиса**

В проекте ServiceA откройте References в панели Solution Explorer. При создании сервиса автоматически вставляются ссылки для Ccr.Core, DssBase и DssRuntime. Почти во всех сервисах также требуется ссылка на Robotics.Common.Proxy, которая используется для работы с реальными роботами.

Исходный код сервиса разделен на два файла: основной код (ServiceA.cs) и описание типов (ServiceATypes.cs).

**Файл описания типов**

В начале файла ServiceATypes.cs находится соглашение для ServiceA:

namespace Robotics.ServiceA

{

/// <summary>

/// класс описания соглашения ServiceA

/// </summary>

public sealed class Contract

{

/// <summary>

/// соглашение сервиса

/// </summary>

public const String Identifier =

"http://www.promrds.com/contracts/2008/01/servicea.html";

}

}

Класс Contract – обязательная часть соглашения. Данный класс должен содержать строку Identifier. Строка Identifier представляет собой URI в следующем формате:

http://somehost.domain/path/year/month/servicename.html

Идентификатор соглашения используется для идентификации сервисов, в связи с этим идентификатор должен быть уникальным.

Идентификатор выглядит как URL. Данный URL не обязательно должен существовать в Интернете. По умолчанию (для примеров данной книги) именем хоста в идентификаторе является www.promrds.com, но большинство сервисов Майкрософт используют путь schemas.microsoft.com/robotics. Включение в идентификатор года и месяца создания сервиса помогает сохранить URI уникальным. Ограничения на имена узлов и пути не накладываются. Все URI должны быть записаны в нижнем регистре во избежание проблем с регистром символов.

Далее в файле ServiceATypes.cs описывается состояние сервиса. Cостояние сервиса – это класс, описывающий свойства, которые нужны для реализации операций сервиса. Некоторые свойства не могут изменяться во время выполнения сервиса. Состояние может включать различные параметры сервиса. Внешне доступное состояние сервиса позволяет реконфигурировать сервис без его рекомпиляции.

Каждый сервис включает класс описания состояния. Добавление полей в состояние сервиса является одним из шагов при создании нового сервиса:

/// <summary>

/// состояние serviceA

/// </summary>

[DataContract]

public class serviceAState { }

Перед описанием состояния сервиса указан атрибут [DataContract]. Он необходим для того, чтобы класс был скопирован в Proxy DLL при компиляции кода. Следующая часть кода – описание главного порта операций, PortSet:

/// <summary>

/// порт главных операций serviceA

/// </summary>

[ServicePort]

public class ServiceAOperations : Portset<DsspDefaultLookup, DsspDefaulDrop, Get>

{

}

Главный порт операций сервиса должен иметь атрибут [ServicePort], который отмечает его как порт операций. Подобные атрибуты используются в MRDS для декларативного создания инфраструктуры сервиса.

При создании нового сервиса порт операций (PortSet) по умолчанию содержит только операции DssDefaultLoolup, DssDefaultDrop и Get.

Операция Lookup позволяет получать информацию о данном сервисе другим сервисам, Drop используется для завершения работы сервиса, Get – для получения копии состояния сервиса. Этот набор типов сообщений создается автоматически при создании нового сервиса.

ServiceA является управляющим сервисом, поэтому в описание операций данного сервиса не требуется добавлять операции, помимо операций, добавленных по умолчанию. ServiceB вызывается ServiceA, поэтому при описании ServiceB нужно определить дополнительные операции.

Последняя секция файла содержит определение операций для запроса состояния сервиса. В ServiceA описывается только запрос типа Get:

/// <summary>

/// операция Get

/// </summary>

public class Get: Get<GetRequestType, Portset<serviceAState, Fault>>

{

/// <summary>

/// операция Get

/// </summary>

public Get()

{

}

/// <summary>

/// операция Get

/// </summary>

public Get(Microsoft.Dss.serviceModel.Dssp.GetRequestType body): base(body)

{

}

/// <summary>

/// операция Get

/// </summary>

public Get(Microsoft.Dss.serviceModel.Dssp.GetRequestType body,

Microsoft.ccr.core.Portset<serviceAState, w3c.soap.Fault> responsePort):

base(body, responsePort)

{

}

}

В коде выше не определены операции DssDefaultLookup и DssDefaultDrop, так как они неявно обрабатываются классом DssServiceBase. Однако эти операции можно переопределить.

**Файл реализации сервиса**

Файл ServiceA.cs включает код, выполняющий инициализацию сервиса, а также код, содержащий определение операций сервиса:

namespace ServiceA

{

/// <summary>

/// класс, реализующий ServiceA

/// </summary>

[DisplayName("ServiceA")]

[Description("The ServiceA service)"]

[contract(contract.identifier)]

public class serviceAService : DsspServiceBase

{

Атрибуты [DisplayName] и [Description] используются для оформления документации. Соглашение сервиса задается в поле Contract.Identifier, описание которого находится в файле ServiceATypes.cs. Класс разрабатываемого сервиса наследуется от класса DsspServiceBase, что даёт доступ к множеству вспомогательных функций.

Каждый сервис включает определение экземпляра своего состояния (даже если состояние пустое) и определение главного порта операций:

/// <summary>

/// состояние сервиса

/// </summary>

private ServiceAState \_state = new ServiceAState();

/// <summary>

/// порт операций

/// </summary>

[servicePort("/servicea", AllowMultipleinstances=false)]

private ServiceAOperations \_mainPort = new ServiceAOperations();

Состояние сервиса в данном примере хранится в переменной \_state. Порт операций \_mainPort создан для отправки сообщений ServiceA. Конструктор класса всегда пустой:

/// <summary>

/// конструктор сервиса по умолчанию

/// </summary>

public ServiceAService(DsspServiceCreationPort creationPort):base(creationPort)

{

}

Метод Start вызывается во время создания сервиса для выполнения операций по инициализации:

/// <summary>

/// метода Start сервиса

/// </summary>

protected override void Start()

{

base.Start();

}

Вызов base.Start вставляется автоматически. Обработчик для операции Get:

/// <summary>

/// обработчик Get

/// </summary>

[ServiceHandler(ServiceHandlerBehavior.Concurrent)]

public virtual IEnumerator <ITask> GetHandler(Get get)

{

get.ResponsePort.Post(\_state);

yield break;

}

Данный обработчик возвращает копию состояния сервиса. Атрибут [ServiceHandler] указывает на принадлежность обработчика группе Concurrent.

Соглашение (contract.identifier) включается в Proxy DLL. Эта библиотека автоматически создается при компиляции сервиса с помощью DSSProxy. Напрямую DLL сервиса не вызывается, вместо этого методы вызываются с помощью соответствующих сигнатур в ProxyDLL. При создании связи с другими сервисами в References проекта нужно делать ссылку на Proxy DLL, а не на DLL.

**Манифест сервиса**

Файл ServiceA.manifest.xml – манифест сервиса ServiceA:

<?xml version="1.0"?>

<Manifest

xmlns="http://schemas.microsoft.com/xw/2004/10/manifest.html"

xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">

<CreateServiceList>

<ServiceRecordType>

<dssp:Contract>http://schemas.tempuri.org/2008/01/servicea.html</dssp:Contract>

</ServiceRecordType>

</CreateServiceList>

</Manifest>

После запуска отладчика, этот манифест отправляется в DssHost. В манифесте указываются запускаемые сервисы и партнерские отношения между ними. Приведённый манифест включает одну запись ServiceRecord с идентификатором соглашения ServiceA.

Перед компиляцией сервиса ServiceA нужно добавить ссылки на ServiceB и выполнить реализацию операций сервиса. После этого необходимо выполнить компиляцию обоих сервисов.

Для компиляции сервиса выберите Build Solution в меню Build. В метод Start в ServiceA.cs добавьте строку кода, информирующую о запуске сервиса:

protected override void Start()

{

base.Start();

Console.WriteLine("ServiceA starting");

Скомпилируйте ServiceA. Откройте ServiceB в Visual Studio и добавьте команду Console.WriteLine в метод Start (файл ServiceB.cs).

Рассмотрим задачу умножения матрицы A (m × n элементов) на вектор B (1 × n элементов). Алгоритм вычисления вектора C (m × 1 элементов) легко распараллеливается, так как значение i-го элемента вектора не зависит от значений других его элементов.

Библиотека Concurrent and Coordination Runtime (CCR) предназначена для организации обработки данных с помощью параллельно и асинхронно выполняющихся методов. Взаимодействие между такими методами организуется на основе сообщений. Рассылка сообщений основана на использовании портов.

Основные понятия CCR:

1) сообщение – экземпляр любого типа данных;

2) порт – очередь сообщений типа FIFO (First-In-First-Out), сообщение остаётся в порте пока не будут извлечено из очереди порта получателем.

Определение порта:

Port<int> p = new Port<int>();

Отправка сообщения в порт:

p.Post(1);

3) получатель – структура, которая выполняет обработку сообщений.

Данная структура объединяет:

а) один или несколько портов, в которые отправляются сообщения;

б) метод (или методы), которые используются для обработки сообщений (такой метод называется задачей);

в) логическое условие, определяющее ситуации, в которых активизируется тот или иной получатель.

Описание получателя:

Arbiter.Receive(<тип\_получателя>, <порт>, <выполняемый\_делегат>);

Пример:

Activate(Arbiter.Receive(false, intPort, delegate(int n)

{

Console.WriteLine("Получено сообщение: " + n.ToString());

}));

Делегат, входящий в получатель, выполнится, когда в порт intPort придёт сообщение.

Получатели сообщений бывают двух типов: временные и постоянные (в примере получатель – временный). Временный получатель, обработав сообщение (или несколько сообщений), удаляется из списка получателей сообщений данного порта.

4) процессом запуска задач управляет диспетчер. После выполнения условий активации задачи (одним из условий активации может быть получение портом сообщения) диспетчер назначает задаче поток из пула потоков, в котором она будет выполняться.

Описание диспетчера с двумя потоками в пуле:

Dispatcher d = new Dispatcher(2, "MyPool");

Описание очереди диспетчера, в которую задачи ставятся на выполнение:

DispatcherQueue dq = new DispatcherQueue("MyQueue", d);

Выполните следующие действия:

1) установите библиотеку CCR (CCR входит в состав Microsoft Robotics Developer Studio);

2) создайте проект консольного приложения и добавьте к проекту библиотеку Microsoft.Ccr.Core.dll.

Теперь можно использовать все возможности CCR, часть из которых будет описана ниже.

Время выполнения вычислений будем определять с помощью класса

Stopwatch:

Stopwatch sWatch = new Stopwatch();

sWatch.Start();

<выполняемый код>

sWatch.Stop();

Console.WriteLine(sWatch.ElapsedMilliseconds.ToString());

Матрицу A, векторы B и C, переменные для хранения их размеров определим глобально:

int[,] A;

int[] B;

int[] C;

int m;

int n;

Метод TestFunction() запускает вычисления. Сначала в методе выполняется умножение матрицы на вектор с помощью последовательного алгоритма, затем та же задача решается с помощью параллельных вычислений. Рассмотрим этот метод.

Выполняется инициализация структур данных:

nc = 2; // количество ядер

m = 11000; // количество строк матрицы

n = 11000; // количество столбцов матрицы

A = new int[m, n];

B = new int[n];

C = new int[m];

Генерируется матрица A и вектор B:

Random r = new Random();

for (int i = 0; i < m; i++)

{

for (int j = 0; j < n; j++)

A[i, j] = r.Next(100);

}

for (int j = 0; j < n; j++)

B[j] = r.Next(100);

Рассмотрим метод SequentialMul:

Stopwatch sWatch = new Stopwatch();

sWatch.Start();

for (int i = 0; i < m; i++) {

C[i] = 0;

for (int j = 0; j < n; j++)

C[i] += A[i, j] \* B[i]; }

sWatch.Stop();

Console.WriteLine("Последовательный алгоритм = {0} мс.",

sWatch.ElapsedMilliseconds.ToString());

Параллельная обработка выполняется с помощью запуска нескольких копий вычислительного метода. Каждая копия метода выполняет обработку определённой части исходных данных. Для описания задания для каждого метода используется класс InputData:

public class InputData

{

public int start; // начало диапазона

public int stop; // начало диапазона

}

Поля start / stop класса хранят номер начальной / конечной строки вектора C. Поля рассчитываются с помощью экземпляра вычислительного метода. Рассмотрим метод ParallelMul:

// создание массива объектов для хранения параметров

InputData[] ClArr = new InputData[nc];

for (int i = 0; i < nc; i++)

ClArr[i] = new InputData();

Далее, задаются исходные данные для каждого экземпляра вычислительного метода:

// делим количество строк в матрице на nc частей

int step = (Int32)(m / nc);

// заполняем массив параметров

int c = -1;

for (int i = 0; i < nc; i++)

{

ClArr[i].start = c + 1;

ClArr[i].stop = c + step;

c = c + step;

}

Создаётся диспетчер с пулом из двух потоков:

Dispatcher d = new Dispatcher(nc, "Test Pool");

DispatcherQueue dq = new DispatcherQueue("Test Queue", d);

Описывается порт, в который каждый экземпляр метода Mul() отправляет сообщение после завершения вычислений:

Port<int> p = new Port<int>();

Метод Arbiter.Activate помещает в очередь диспетчера две задачи (два экземпляра метода Mul):

for (int i = 0; i < nc; i++)

Arbiter.Activate(dq, new Task<InputData, Port<int>>(ClArr[i], p, Mul));

Первый параметр метода Arbiter.Activate – очередь диспетчера, который будет управлять выполнением задачи, второй параметр – запускаемая задача.

С помощью метода Arbiter.MultipleItemReceive запускается задача (приёмник), которая обрабатывает получение двух сообщений портом p:

Arbiter.Activate(Environment.TaskQueue,

Arbiter.MultipleItemReceive(true, p, nc, delegate(int[] array)

{

Console.WriteLine("Вычисления завершены");

}));

Приёмник используется для определения момента окончания вычислений. Он сработает только после того, как в порт p придёт два сообщения. В делегат, описанный в приёмнике, можно включить действия, которые должны быть выполнены после завершения процесса умножения.Метод Mul() выполняет перемножение части матрицы на вектор:

void Mul(InputData data, Port<int> resp)

{

Stopwatch sWatch = new Stopwatch();

sWatch.Start();

for (int i = data.start; i < data.stop; i++)

{

C[i] = 0;

for (int j = 0; j < n; j++)

C[i] += A[i, j] \* B[j];

}

sWatch.Stop();

Console.WriteLine("Поток № {0}: Паралл. алгоритм = {1} мс.",

Thread.CurrentThread.ManagedThreadId,

sWatch.ElapsedMilliseconds.ToString());

resp.Post(1);

}

Метод Mul() имеет два параметра:

1) индекс, хранящий значение элемента массива, который определяет параметры, передаваемые на вход метода;

2) порт завершения, в который отправляется целое число после завершения вычислений.

После завершения вычислений метод Mul отправляет в порт p целое значение (отправляемое значение может быть любым).

**Задание на лабораторную работу**

1. Изучить методические рекомендации по выполнению лабораторной работы;

2. Выполнить вариант задания, выданный преподавателем;

3. Подготовить отчёт по лабораторной работе;

4. Подготовить ответы на контрольные вопросы.

**Варианты заданий**

Создать сервис, предназначенный для решения одной из задач, приведённой в лабораторной работе № 1.

**Контрольные вопросы**

1. Опишите структуру DSS сервиса;

2. Опишите назначение DSS сервиса;

3. Опишите порядок создания DSS сервиса;

4. Опишите принцип организации вычислений с помощью DSS сервиса.

**Лабораторная работа № 2. Создание простейшего распределённого приложения на основе технологии .NET Remoting.**

**Цель:** изучение возможностей технологии.NET Remoting.

**Краткие теоретические сведения**

.NET Remoting – компонент, созданный компанией Microsoft. API для межпроцессного взаимодействия. Реализация от Microsoft протокола SOAP (веб-сервисы). Выпущен в 2002 году вместе с версией 1.0 пакета .NET Framework. Это одна из серии технологий от Microsoft, начатой в 1990 году первой версией OLE для 16-разрядной Windows. Промежуточными шагами в разработке подобных технологий были COM, выпущенная в 1993 году и доработанная в 1995 году под названием COM-95, DCOM, выпущенная в 1997 году (и переименованная в ActiveX), и COM+ с её Microsoft Transaction Server (MTS), выпущенная в 2000 году. В данный момент на смену .NET Remoting пришёл WCF, являющийся частью .NET Framework 3.0.

.NET Remoting позволяет приложению создать объект (именуемый remotable object) доступный в рамках remoting boundaries и расположенный в домене приложения внутри одного процесса, другом процессе, исполняющемся на этом компьютере, или даже на другом компьютере, соединённом сетью. Процесс .NET Remoting содержит приёмник запросов к объекту в домене серверного приложения. На стороне клиента любые запросы к удалённому объекту направляются средой выполнения .NET Remoting через объекты Channel, являющиеся обёрткой для средств транспортного уровня, таких как потоки TCP, потоки HTTP и именованные каналы. В результате, запросы к удаленным объектам для клиентского кода ничем не отличаются от локальных вызовов, а созданием экземпляра нужного Channel-объекта, приложение .NET Remoting можно без перекомпиляции перевести на другой коммуникационный протокол. Среда выполнения сама по себе выполняет этапы сериализации и маршалинга объектов в среде между клиентским и серверным доменами приложения.

.NET Remoting делает ссылку на удалённый (remotable) объект доступной клиентскому приложению, которое затем направляет запросы к экземпляру удалённого объекта так, как если бы это был локальный объект. Однако, фактическое исполнение кода происходит на серверной стороне. Удалённый объект имеет идентификаторы в форме URL активации. В зависимости от конфигурации серверного процесса, экземпляр удаленного объекта создаётся при подключении по данным URL, в момент создания ссылки на удалённый объект или в момент первого обращения к удалённому объекту. Прослушивающий приёмник (listener) для объекта создаётся исполняющей средой .NET Remoting в момент, когда сервер регистрирует канал, который будет использоваться для подключения к удалённому объекту. На клиентской стороне инфраструктура .NET Remoting создаёт объект-заместитель (proxy), который является псевдоэкземпляром замещаемого объекта. Он не реализует функциональность удалённого объекта, но предоставляет похожий интерфейс, перенаправляя все запросы серверному объекту и возвращая результаты от него клиенту. Как следствие, инфраструктуре .NET Remoting для создания заместителя нужно обладать метаданными, описывающими публичный интерфейс удалённого объекта. Это можно обеспечить статическим связыванием со сборкой, содержащей метаданные, во время компиляции или динамическим получением метаданных в виде WSDL или проприетарного формата, разработанного фирмой Microsoft.

В процессе выполнения запросов любые вызовы методов, направленные объекту, включая идентификатор метода и любые передаваемые параметры, сериализуются в байтовый поток и передаются посредством канала связи, реализованного для конкретного протокола, принимающему прокси-объекту на серверной стороне («маршализируются»). Передача происходит путём записи данных в транспортный ввод канала. На серверной стороне прокси читает поток данных из вывода канала и выполняет вызов удалённого компонента от лица клиента. Результаты сериализуются и передаются через канал клиенту, где прокси читает результат и передаёт его вызывающему приложению.

Если удалённому объекту нужно обеспечить обратный вызов (callback) клиентскому объекту, клиентский объект обратного вызова должен быть помечен как remotable, а инфраструктура .NET Remoting должна быть сконфигурирована на создание прослушивателя для него. Сервер может подключиться к нему по другому каналу, или по уже существующему, если соединение, на котором он основан, поддерживает двунаправленный обмен данными.[5] Канал может быть составлен из нескольких канальных объектов, возможно, даже с разными транспортными механизмами. Таким образом, система, основанная на .NET Remoting, может состоять из нескольких подсистем, связанных подключёнными друг к другу гетерогенными сетями, включая Интернет.

Удаленные объекты требуются для реализации распределенных вычислений. Объект, который планируется вызывать удаленно из другой системы, должен наследоваться от System.MarshalByRefObject. Объекты MarshalByRefObject ограничены доменом прило- жений, в котором они созданы. Это значит, что они никогда не выходят за пределы домена приложений; для доступа к удаленному объекту из другого домена приложений используется прокси-объект. При этом другой домен приложений может существовать внутри того же процесса, в другом процессе или на другой системе. Удаленный объект имеет распределенную идентичность. По этой причине ссылка на него может передаваться другим клиентам, и они будут обращаться к тому же объекту. Прокси знает об идентичности удаленного объекта. Вдобавок к методам, унаследованным от класса Object, класс MarshalByRefObject имеет методы для инициализации и получения служб времени жизни. Службы времени жизни (lifetime services) определяют, насколько долго существует удаленный объект. Службы времени жизни и средства аренды рассматриваются далее в главе. Чтобы увидеть .NET Remoting в действии, создайте библиотеку классов для удаленного объекта. Класс Hello унаследован от System.MarshalByRefObject. В конструкторе на консоль выводится сообщение, в котором указывается информация о времени жизни объекта. Добавьте также метод Greeting(), который будет вызываться со стороны клиента. Чтобы не путать сборку с классом в последующих разделах, назначьте им разные имена в аргументах вызовов метода. Именем сборки будет RemoteHello, а именем класса – Hello.

using System;

{

public class Hello : System.MarshalByRefObject

{

public Hello()

{

Console.WriteLine("Вызван конструктор");

}

public string Greeting(string name)

{

Console.WriteLine("Вызван метод Greeting"); return "Hello, " + name;

}

}

}

Для сервера создайте новое консольное приложение C# по имени HelloServer. Чтобы пользоваться классом TcpServerChannel, необходимо сослаться на сборку System. Runtime.Remoting. Также потребуется ссылка на сборку RemoteHello, которая была соз- дана ранее. В методе Main() создается объект типа System.Runtime.Remoting.Channels. Tcp.TcpServerChannel с портом номер 8086. Этот канал зарегистрирован с классом System.Runtime.Remoting.Channels.ChannelServices для обеспечения его доступ- ности удаленным объектам. Тип удаленного объекта зарегистрирован вызовом метода RemotingConfiguration.RegisterWellKnownServiceType(). В примере клиент использует тип URI класса удаленного объекта с указанием режима. Режим WellKnownObjectMode.SingleCall означает, что новый экземпляр создается для каждого вызова метода; в примере приложения для удаленного объекта никакого состояния не удерживается.

using System;

using System.Collections.Generic;

using System.Runtime.Remoting;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

namespace Wrox.ProCSharp.Remoting

{

class Program

{

static void Main()

{

var channel = new TcpServerChannel(8086);

ChannelServices.RegisterChannel(channel,true);

RemotingConfiguration.RegisterWellKnownServiceType( typeof(Hello), "Hi",

WellKnownObjectMode.SingleCall);

Console.WriteLine("Для завершения нажмите Enter"); Console.ReadLine();

}

}

}

Клиентом будет также консольное приложение под названием HelloClient. В этом проекте понадобится сослаться на сборку System.Runtime.Remoting, чтобы использовать класс TcpClientChannel. Вдобавок нужно сослаться на сборку RemoteHello. Хотя объект будет создан на удаленном сервере, эта сборка необходима клиенту, чтобы прокси мог прочитать информацию о типе во время выполнения. В клиентской программе создайте объект TcpClientChannel, зарегистрированный в ChannelServices. Для TcpChannel можно использовать конструктор по умолчанию, выбирающий свободный порт. Затем с помощью класса Activator возвращается прокси удаленного объекта. Прокси имеет тип System.Runtime.Remoting.Proxies.TransparentProxy. Этот объект выглядит как реальный объект, потому что предоставляет те же методы. Прозрачный прокси использует реальный прокси для отправки сообщений в канал

using System;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

namespace Remoting

{

class Program

{

static void Main()

{

Console.WriteLine("Нажмите Enter после запуска сервера");

Console.ReadLine();

ChannelServices.RegisterChannel(new TcpClientChannel(), true);

Hello obj = (Hello)Activator.GetObject( typeof(Hello), "tcp://localhost:8086/Hi");

if (obj == null)

{

Console.WriteLine("не удается найти сервер");

return;

}

for (int i=0; i< 5; i++)

{ Console.WriteLine(obj.Greeting("Stephanie"));

}

}

}

}

**Задание на лабораторную работу**

1. Изучить методические рекомендации по выполнению лабораторной работы;

2. Выполнить вариант задания, выданный преподавателем;

3. Подготовить отчёт по лабораторной работе;

4. Подготовить ответы на контрольные вопросы.

**Варианты заданий**

Разработайте систему распределённой обработки данных в соответствии с вариантом задания. Клиент отправляет данные для обработки на сервер. Сервер - обрабатывает их и возвращает результат.

1. Разработайте распределённую систему сложения двух чисел. В обмене данными участвуют один клиент и один сервер.

2. Разработайте распределённую систему деления двух чисел. В обмене данными участвуют один клиент и один сервер.

3. Разработайте распределённую систему вычитания двух чисел. В обмене данными участвуют один клиент и один сервер.

4. Разработайте распределённую систему вычисления косинуса угла. В обмене данными участвуют один клиент и один сервер.

5. Разработайте распределённую систему вычисления синуса угла. В обмене данными участвуют один клиент и один сервер.

6. Разработайте распределённую систему решения системы уравнений с двумя неизвестными. В обмене данными участвуют один клиент и один сервер.

7. Разработайте распределённую систему решения квадратного уравнения. В обмене данными участвуют один клиент и один сервер.

8. Разработайте распределённую систему решения кубического уравнения. В обмене данными участвуют один клиент и один сервер.

**Контрольные вопросы**

1. Опишите структуру распределённого приложения, построенного по технологии .NET Remoting;

2. Опишите функции серверного приложения;

3. Опишите функции клиентского приложения;

4. Опишите функции удалённой библиотеки.

**Лабораторная работа № 3. Создание простейшего распределённого приложения для математических расчётов на основе технологии .NET Remoting**

**Цель**: использование технологии .NET Remoting при разработке распределённых приложений обработки данных

**Краткие теоретические сведения**

Реальная распределённая система, в отличие от приложения из предыдущей лабораторной работы, состоит из нескольких более сложных компонент, ведущих обмен данными друг с другом.

Разработанный программный комплекс должен состоять из Сервера и Клиента.

Функции сервера: хранение удалённого объекта, предоставляющего доступ к заданиям для обработки и результату обработки.

Предусмотреть на сервере возможность одновременного доступа к критической

секции кода нескольких клиентов (с помощью оператора lock). Критическая секция кода - та, к которой гипотетически одновременно могут обратиться

несколько клиентов.

Функции клиента (на сервере хранится список клиентов - эта функция уже предусмотрена исходным кодом библиотеки RemoteBase):

1) управляющие функции (выполняет только один клиент из всего множества клиентов, выполнение данной функции должно выполняться через вызов методов удалённого объекта (удалённый объект хранится на сервере)):

1.1) формирование и ведение списка заданий (под ведением понимается удаление уже обработанных и предоставление клиенту задания по запросу);

1.2) получение, объединение и вывод результатов вычислений (результаты вычислений должны выводиться в каждом клиенте, для этого необходимо проверять окончание обработки всех данных по таймеру; объединение

результатов вычисление также можно реализовать с использованием таймера);

1.3) устанавливает флаг того, что управляющий клиент назначен, на сервере сохраняется идентификатор клиента;

2) вычислительные функции

2.1) запрос задания с сервера (клиент должен запросить задание только после того, как эти задания были сформированы);

2.2) обработка данных;

2.3) отправка результатов обработки на сервер.

Требования к системе:

1) предусмотреть возможность отключения одного из клиентов, получившего задание на обработку.

**Задание на лабораторную работу**

1. Изучить методические рекомендации по выполнению лабораторной работы;

2. Выполнить вариант задания, выданный преподавателем;

3. Подготовить отчёт по лабораторной работе;

4. Подготовить ответы на контрольные вопросы.

**Варианты заданий**

Разработать распределённую систему, выполняющую решение задачи в соответствии с вариантом.

|  |  |
| --- | --- |
| № | Пояснение задания |
| 1 | Разработать алгоритм вычисления скалярного произведения векторов a и b: |
| 2 | Разработать алгоритм поиска среднего значения массива a: |
| 3 | Разработать алгоритм поиска максимального и минимального значения массива a: , |
| 4 | Разработать алгоритм вычисления массива значений синуса: |
| Используемые обозначения: *a*, *b* – нижний и верхний пределы интегрирования, 2⋅*N* – число равных частей на которые делится отрезок [*a*; *b*], *h* – шаг | |
| 5 | Разработать алгоритм вычисления значения определённого интеграла с использованием метода трапеций: |
| 6 | Разработать алгоритм вычисления значения определённого интеграла с использованием метода парабол (метод Симпсона). Разбив интервал интегрирования на 2⋅*N* равных частей, приближённое значение интеграла можно вычислить следующим образом: |
| 7 | Разработать алгоритм вычисления значения определённого интеграла с использованием метода прямоугольников: |
| 8 | Разработать алгоритм вычисления значения числа π на основе метода прямоугольников: |
| 9 | Разработать алгоритм умножения матрицы a (m × n элементов) на вектор b (n элементов) по следующей формуле: . При разработке параллельного алгоритма необходимо учитывать, что значение i-го элемента вектора c не зависит от значений других элементов вектора |
| 10 | Разработать алгоритм умножения матрицы a (m × n элементов) на матрицу b (n × l элементов) по следующей формуле: |
| 11 | Разработать алгоритм сортировки массива чисел слиянием |
| 12 | Разработать алгоритм сортировки массива чисел методом Шелла |
| 13 | Разработать алгоритм сортировки массива чисел методом пузырька |
| 14 | Разработать алгоритм быстрой сортировки массива чисел |
| 15 | Разработка алгоритма вычисления суммы двух матриц |
| 16 | Разработка алгоритма вычисления суммы элементов массива |
| 17 | Разработка алгоритма вычисления произведения матрицы и вектора |
| 18 | Разработать алгоритм вычисления значения определённого интеграла с использованием метода Монте-Карло |
| 19 | Разработать алгоритм вычисления значения определённого интеграла с использованием метода геометрического метода Монте-Карло |

Интегралы для вычислений.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 |  | 2 |  | 3 |  |
| 4 |  | 5 |  | 6 |  |
| 7 |  | 8 |  | 9 |  |
| 10 |  | 11 |  | 12 |  |
| 13 |  | 14 |  | 15 |  |
| 16 |  | 17 |  | 18 |  |
| 19 |  | 20 |  | 21 |  |
| 22 |  | 23 |  | 24 |  |
| 25 |  | 26 |  | 27 |  |

**Контрольные вопросы**

1. Опишите назначение функции ChannelServices.RegisterChannel;

2. Опишите назначение функции Activator.GetObject;

3. Опишите структуру класса TcpClientChannel;

4. Опишите назначение функции RemotingConfiguration.RegisterWellKnownServiceType;

5. Опишите структуру WellKnownObjectMode.

**Лабораторная работа № 4.** Изучение методов реализации логических часов

**Цель**: практическое закрепление знаний, связанных с различными методами реализации логических часов.

Задание по данной лабораторной работе состоит в изучении работы способов реализации логического времени. В таблице ниже приведены соответствие между вариантом задания, методом реализации логического времени и схемой, предназначенной для исследования. В каждом варианте задания указаны два метода логического времени для изучения. Порядок выполнения задания для каждого из вариантов логического времени приведён ниже.

**1. Cкалярное время Лэмпорта**

Задания:

1.0. При выполнении заданий используйте концепцию скалярного времени Лэмпорта;

1.1. По заданной пространственно-временной диаграмме для каждого события определите отметки времени, подпишите отметки времени для передаваемых сообщений;

1.2. Приведите примеры пар событий, для которых выполняется причинно-следственный порядок, и пар событий которые являются параллельными, обоснуйте свой выбор;

1.3. Опишите в текстовом виде выполнение распределённой системы в соответствии с порядком наступления событий;

1.4. Привести пример пар событий, для которых выполняются следующие условия:

а)  ';

б) .

1.5. Поясните свойство скалярного времени Лэмпорта, связанное с отсутствием строгой непротиворечивости на примере диаграммы;

1.6. Оцените объём информации, передаваемой между процессами в ходе выполнения распределённой системы, при оценке используйте допущение о том, что каждый элемент сообщения имеет тип int64.

**2. Векторное время**

Задания:

2.0. При выполнении заданий используйте концепцию векторного времени;

2.1. По заданной пространственно-временной диаграмме для каждого события определите отметки времени;

2.2. Опишите в текстовом виде выполнение распределённой системы в соответствии с порядком наступления событий;

2.3. Приведите пример пар несравнимых событий, обоснуйте свой выбор;

2.4. Поясните понятие "частичный порядок на множестве";

2.5. Приведите примеры пар событий, для которых выполняется причинно-следственный порядок, и пар событий которые являются параллельными, обоснуйте свой выбор;

2.6. Оцените объём информации, передаваемой между процессами в ходе выполнения распределённой системы, при оценке используйте допущение о том, что каждый элемент сообщения имеет тип int64.

**3. Векторное время. Дифференциальная пересылка векторного времени**

Задания:

3.0. При выполнении заданий используйте концепцию векторного времени и, в частности, реализацию векторного времени на основе дифференциальной пересылки;

3.1. По заданной пространственно-временной диаграмме для каждого события определите отметки времени;

3.2. Опишите в текстовом виде выполнение распределённой системы в соответствии с порядком наступления событий, также приведите изменения значений массивов LS и LU для каждого из процессов, приведите сообщения, передаваемые между процессами;

3.3. Оцените объём информации, передаваемой между процессами в ходе выполнения распределённой системы, при оценке используйте допущение о том, что каждый элемент сообщения имеет тип int64.

**4. Векторное время. Часы, фиксирующие прямую зависимость**

Задания:

4.0. При выполнении заданий используйте концепцию векторного времени и, в частности, реализацию векторного времени на основе часов, фиксирующих прямую зависимость;

4.1. По заданной пространственно-временной диаграмме для каждого события определите отметки времени;

4.2. Опишите в текстовом виде выполнение распределённой системы в соответствии с порядком наступления событий, также приведите изменения значений массива D для каждого из процессов, приведите сообщения, передаваемые между процессами;

4.3. Оцените объём информации, передаваемой между процессами в ходе выполнения распределённой системы, при оценке используйте допущение о том, что каждый элемент сообщения имеет тип int64.

**5. Векторное время. Адаптивный метод Жарда-Жордана**

Задания:

5.0. При выполнении заданий используйте концепцию векторного времени и, в частности, реализацию векторного времени на основе адаптивного метода Жарда-Жордана;

5.1. По заданной пространственно-временной диаграмме для каждого события определите отметки времени;

5.2. Опишите в текстовом виде выполнение распределённой системы в соответствии с порядком наступления событий, также приведите изменения значений частично-векторных часов PV для каждого из процессов, приведите сообщения, передаваемые между процессами;

5.3. Приведите пример событий, связанных псевдо-прямой зависимостью.

5.4. Оцените объём информации, передаваемой между процессами в ходе выполнения распределённой системы, при оценке используйте допущение о том, что каждый элемент сообщения имеет тип int64.

5.5. Самостоятельно выберите моменты времени для регистрации событий.

**6. Матричное время**

6.0. При выполнении заданий используйте концепцию матричного времени;

6.1. По заданной пространственно-временной диаграмме для каждого события определите отметки времени;

6.2. Опишите в текстовом виде выполнение распределённой системы в соответствии с порядком наступления событий;

6.3. Приведите пример пар несравнимых событий, обоснуйте свой выбор;

6.4. Поясните понятие "частичный порядок на множестве";

6.5. Приведите примеры пар событий, для которых выполняется причинно-следственный порядок, и пар событий которые являются параллельными, обоснуйте свой выбор;

6.6. Оцените объём информации, передаваемой между процессами в ходе выполнения распределённой системы, при оценке используйте допущение о том, что каждый элемент сообщения имеет тип int64.

Варианты для выполнения лабораторной работы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № варианта | Реализация логических  часов | Схема | № варианта | Реализация логических  часов | Схема |
| 1 | 1, 2 | 1 | 31 | 1, 2 | 1 |
| 2 | 1, 3 | 2 | 32 | 1, 3 | 2 |
| 3 | 1, 4 | 3 | 33 | 1, 4 | 3 |
| 4 | 1, 5 | 4 | 34 | 1, 5 | 4 |
| 5 | 1, 6 | 5 | 35 | 1, 6 | 5 |
| 6 | 2, 3 | 6 | 36 | 2, 3 | 6 |
| 7 | 2, 4 | 7 | 37 | 2, 4 | 7 |
| 8 | 2, 5 | 8 | 37 | 2, 5 | 8 |
| 9 | 2, 6 | 5 | 39 | 2, 6 | 5 |
| 10 | 3, 4 | 5 | 40 | 3, 4 | 5 |
| 11 | 3, 5 | 3 | 41 | 3, 5 | 3 |
| 12 | 3, 6 | 4 | 42 | 3, 6 | 4 |
| 13 | 4, 5 | 5 | 43 | 4, 5 | 5 |
| 14 | 4, 6 | 3 | 44 | 4, 6 | 3 |
| 15 | 5, 6 | 7 | 45 | 5, 6 | 7 |
| 16 | 1, 2 | 8 | 46 | 1, 2 | 8 |
| 17 | 1, 3 | 1 | 47 | 1, 3 | 1 |
| 18 | 1, 4 | 2 | 48 | 1, 4 | 2 |
| 19 | 1, 5 | 3 | 49 | 1, 5 | 3 |
| 20 | 1, 6 | 4 | 50 | 1, 6 | 4 |
| 21 | 2, 3 | 5 | 51 | 2, 3 | 5 |
| 22 | 2, 4 | 6 | 52 | 2, 4 | 6 |
| 23 | 2, 5 | 7 | 53 | 2, 5 | 7 |
| 24 | 2, 6 | 8 | 54 | 2, 6 | 8 |
| 25 | 3, 4 | 1 | 55 | 3, 4 | 1 |
| 26 | 3, 5 | 2 | 56 | 3, 5 | 2 |
| 27 | 3, 6 | 3 | 57 | 3, 6 | 3 |
| 28 | 4, 5 | 5 | 58 | 4, 5 | 5 |
| 29 | 4, 6 | 3 | 59 | 4, 6 | 3 |
| 30 | 5, 6 | 6 | 60 | 5, 6 | 6 |

Схемы приведены на рисунках ниже. Точками показаны внутренние события процессов. Нумерация процессов сверху вниз, начиная с единицы.

|  |  |
| --- | --- |
|  |  |
| Схема № 1 | Схема № 2 |
|  |  |
| Схема № 3 | Схема № 4 |
|  |  |
| Схема № 5 | Схема № 6 |
|  |  |
| Схема № 7 | Схема № 8 |

**Задание на лабораторную работу**

1. Изучить методические рекомендации по выполнению лабораторной работы;

2. Выполнить вариант задания, выданный преподавателем;

3. Подготовить отчёт по лабораторной работе;

4. Подготовить ответы на контрольные вопросы.

Контрольные вопросы:

1) Опишите концепцию скалярного времени Лэмпорта, приведите достоинства и недостатки данного способа реализации логического времени;

2) Опишите концепцию векторного времени, приведите достоинства и недостатки данного способа реализации логического времени;

3) Опишите концепцию матричного времени, приведите достоинства и недостатки данного способа реализации логического времени;

**Литература**

1. Д. Бэкон Операционные системы. Параллельные и распределенные системы : Пер.с англ. / Д. Бэкон, Т. Харрис. // СПб.: Питер; БХВ, 2004.

2. Э. Таненбаум Распределенные системы. Принципы и парадигмы / Э. Таненбаум, Стеен ван М. // Питер, 2003.

3. М. К. Гофф Сетевые распределенные вычисления: достижения и проблемы : Пер.с англ. / М. К. Гофф. // М. : КУДИЦ-ОБРАЗ, 2005.

4. С. А. Лупин Технологии параллельного программирования / С. А. Лупин, М. А. Посыпкин // М. : ФОРУМ; ИНФРА-М, 2008.

5. А. В. Старченко Практикум по методам параллельных вычислений / А. В. Старченко [и др.] // М. : Изд-во МГУ, 2010.