

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по курсу «Высокопроизводительные вычисления»
на тему «Разработка параллельной программы для кластера облачного типа»



Оши

18.05 - 2021

Выполнила:

студентка группы 17ВВ1

Нешко Д.О.

Принял:

к. т. н., доцент Федюнин Р. Н.

Пенза 2021

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

профессор М.А. Митрохин



«___» _____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Высокопроизводительные вычисления

Студенту Нешко Д.О. Группа 17ВВ1

Тема проекта Разработка параллельной программы для кластера облачного типа

Исходные данные (технические требования) на проектирование

- 1) Клиент-серверное приложение для управления вычислительным процессом должно состоять из двух частей: клиента и сервера, которые должны образовывать единую распределенную вычислительную систему облачного типа.
- 2) Серверное приложение должно обеспечивать одновременное подключение нескольких клиентов.
- 3) Клиентское приложение должно обеспечивать подключение к серверу (порт и IP сервера вводятся в окне клиента) и иметь возможность осуществить отключение от сервера в любой момент.
- 4) В окне серверного приложения должен отображаться прогресс решения вычислительной задачи и примерное время, оставшееся до ее решения с учетом текущей скорости вычислительного процесса, а также указываться список подключившихся клиентов и диапазон интегрирования.
- 5) Для каждого клиента должна быть предусмотрена возможность указания предельно допустимого уровня использования вычислительных ресурсов (процессорного времени) в процентах. Соответствующее значение вводится на серверной стороне.
- 6) Кластер должен решать задачу вычисления определенного интеграла $\int \sqrt{x} dx$, шаг = 0,0000005
- 7) Для интегрирования должен использоваться метод средних прямоугольников.
- 8) Приложения (клиент и сервер) должны обладать интуитивно понятным графическим интерфейсом.
- 9) Для обмена данными использовать протокол TCP/IP и библиотека WinSock.
- 10) Язык программирования C++, среда разработки Microsoft Visual Studio.

Объем работы по курсу

1. Расчетная часть

2. Графическая часть

Диаграммы языка UML:

- 1) диаграмма классов;
- 2) диаграмма деятельности;
- 3) диаграмма вариантов использования;
- 4) диаграмма развёртывания;
- 5) диаграмма последовательности.

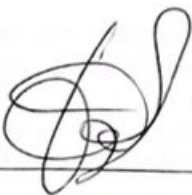
3. Экспериментальная часть

Тестирование распределенной вычислительной системы

Срок выполнения проекта по разделам

В соответствии с графиком

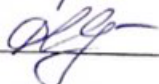
Руководитель _____



Дата выдачи задания "13" марта 2021 г.

Дата защиты проекта "___" _____

Задание получил "13" марта 2021 г.

Студент Никин Д.В. 

Содержание

Введение.....	4
1 Постановка задачи.....	6
2 Выбор решения.....	7
3 Описание разработки программы.....	9
4 Описание программы.....	12
5 Отладка и тестирование.....	14
6 Руководство пользователя.....	16
Заключение	21
Список используемых источников.....	22
Приложение А. Листинг программы.....	23
Приложение А.1 clientForm.cpp.....	23
Приложение А.2 Form1.h.....	23
Приложение А.3 Form1.h.....	31
Приложение А.4 serverForm.cpp	32
Приложение А.5 serverForm.h.....	32
Приложение А.6 Func.h	49

Введение

Высокопроизводительные вычисления приходят на помощь в тех случаях, когда нужно сократить время расчётов или получить доступ к большому объёму памяти. Например, ваша программа может проводить необходимые вычисления в течение недели, но вам нужно получить результаты завтра. Если разделить эту программу на части и выполнять каждую из них на отдельном узле, то теоретически можно ускорить расчёты пропорционально числу вовлечённых узлов. Но это только теоретически, а на практике этому всегда что-то мешает. Тут стоит упомянуть и другой случай, когда ваша программа требует большой объём оперативной памяти. Например, в вашем компьютере установлено только 4 Гб оперативной памяти, но для расчётов нужно хотя бы 64 Гб. В системах высокопроизводительных вычислений на каждом узле установлена память определённой ёмкости. Так если каждому узлу доступно 2 Гб памяти, то опять же можно разделить программу на 32 части, каждая из которых будет выполняться на отдельном узле, будет взаимодействовать с другими частями, обмениваться данными и, в конечном итоге, программа в целом будет иметь доступ к 64 Гб памяти.

Из этого примера понятно, что высокопроизводительные вычисления – это вычисления, проводимые на компьютерных системах со спецификациями, которые значительно превышают обычные компьютеры. Это понятие довольно условное. Существуют параллельные, распределённые высокопроизводительные вычисления, а также их комбинации.

Параллельные вычисления предусматривают разработку программ, которые во время их выполнения представляют собой несколько параллельных и взаимодействующих между собой процессов. Например, моделирование характеристик ячейки солнечной батареи предусматривает взаимодействие трёх моделей описывающих: перенос носителей заряда, распространение падающего света внутри ячейки, температурные эффекты, растяжение-сжатие. Так перенос носителей, растяжение-сжатие и показатель преломления материала, который

используется в оптической модели падающего света, зависят от температуры и модели, описывающие эти эффекты, должны взаимодействовать друг с другом в процессе расчёта. Чтобы ускорить расчёты можно код модели описывающей транспорт носителей выполнять на одном узле, код отвечающий за распространение света – на другом, температурную модель – на третьем, и так далее. То есть, узлы будут выполнять взаимодействующие расчёты параллельно.

Распределённые вычисления предусматривают использование нескольких не взаимодействующих друг с другом узлов и процессов. Очень часто в таком случае выполняется один и тот же код на разных узлах. Например, нам нужно оценить растяжение и сжатие той же ячейки солнечной батареи в зависимости от температуры. В таком случае, температура – входной параметр модели и один и тот же программный код этой модели можно выполнить на разных узлах для разных значений температуры.

Выбор между распределёнными и параллельными расчётами зависит от организации программного кода, используемого для расчётов, самой физической модели, доступности систем высокопроизводительных вычислений для конечного пользователя.

1 Постановка задачи

Необходимо разработать клиент-серверное приложение для управления вычислительным процессом должно состоять из двух частей: клиента и сервера, которые должны образовывать единую распределенную вычислительную систему облачного типа. Обе части (клиент и сервер) должны иметь интуитивно понятный интерфейс. Серверное приложение должно обеспечивать одновременное подключение нескольких клиентов и отображать прогресс решения вычислительной задачи, примерное время, оставшиеся до окончания решения с учетом текущей скорости решения. В свою очередь клиентское приложение должно обеспечивать подключение и отключение от сервера, а также должна быть предусмотрена возможность указания предельно допустимого уровня использования вычислительных ресурсов (процессорного времени) в процентах. Соответствующее значение вводится на серверной стороне. Для этого нужно определить зоны ввода необходимых для работы программы данных и область вывода информации о соединении, о ходе выполнения вычислений и других требуемых данных.

Программа должна выполнять вычисление определенного интеграла $\int \sqrt{x} dx$ методом средних прямоугольников. Для выполнения данного требования необходимо реализовать алгоритм вычислений по соответствующей формуле.

2 Выбор решения

Для написания данной программы будет использован язык программирования Си++. Язык Си++ является одним из наиболее важных и популярных языков программирования. Его использование все более расширяется, поскольку часто программисты предпочитают язык Си++ всем другим языкам после первого знакомства с ним. Он включает в себя те управляющие конструкции, которые рекомендуются теоретическим и практическим программированием. Его структура побуждает программиста использовать в своей работе нисходящее проектирование, структурное программирование и пошаговую разработку модулей. Результатом такого подхода является надежная и читаемая программа. Структура языка Си++ позволяет наилучшим образом использовать возможности современных ЭВМ. На языке Си++ программы обычно отличаются компактностью и быстротой исполнения. Программа, написанная на Си для одной вычислительной системы, может быть перенесена с небольшими изменениями (или вообще без них) на другую.

В качестве среды программирования был выбран программный продукт Microsoft Visual Studio 2017.

Разработанная программа состоит из двух отдельных приложений: клиентского приложения и серверного приложения.

В качестве языка программирования был выбран язык C++.

В качестве сетевой библиотеки была выбрана библиотека WinSock от Microsoft. Библиотека поддерживает построение сетевого взаимодействия по широкому набору сетевых протоколов (TCP, UDP и др).

На этапе проектирования программы было принято решения реализовать программу в объектно-ориентированном стиле: деление ПО на совокупность классов помогает инкапсулировать логику программы, программные параметры библиотек, что значительно упрощает разработку подобного приложения по сравнению с функциональным стилем.

Пользователю будет предоставлена возможность управления вычислениями на серверной стороне, задавать параметры задач (диапазон) и подключать/отключать клиентское приложение посредством элементов ввода «кнопка» и «текстовое поле». Для отладки будет введён лог сообщений, собирающий информацию от программы при посылке соответствующего события.

Каждому объекту будет выделена отдельная нить для ускорения работы программы и обеспечения асинхронной реакции на различные события, возникающие в процессе функционирования программы. Спроектированная диаграмма классов для серверного (Рисунок 1) и клиентского (Рисунок 2) приложения представлена ниже.

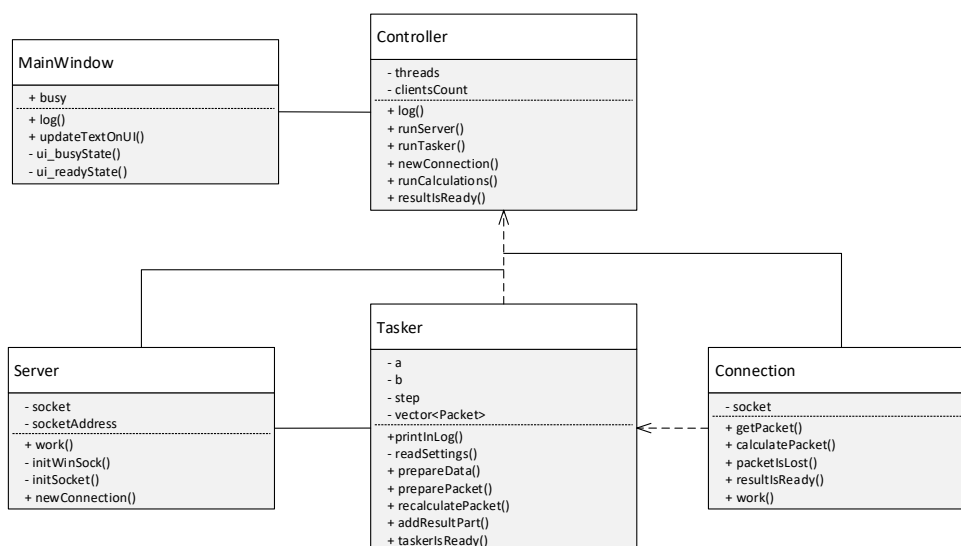


Рисунок 1. Диаграмма классов серверного приложения

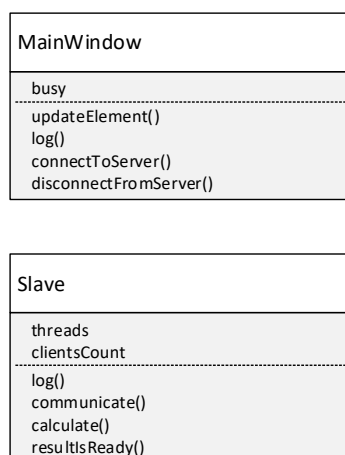


Рисунок 2. Диаграмма классов клиентского приложения

3 Описание разработки программы

Разработка программы началась с разработки рабочей логики приложения, после чего был добавлен графический интерфейс. Это позволило к моменту написания интерфейса приложения управлять состояниями программы без дополнительных средств отладки, параметров командной строки и др. Диаграмма активностей для серверного (Рисунок 3) и клиентского (Рисунок 4) приложения представлена ниже.

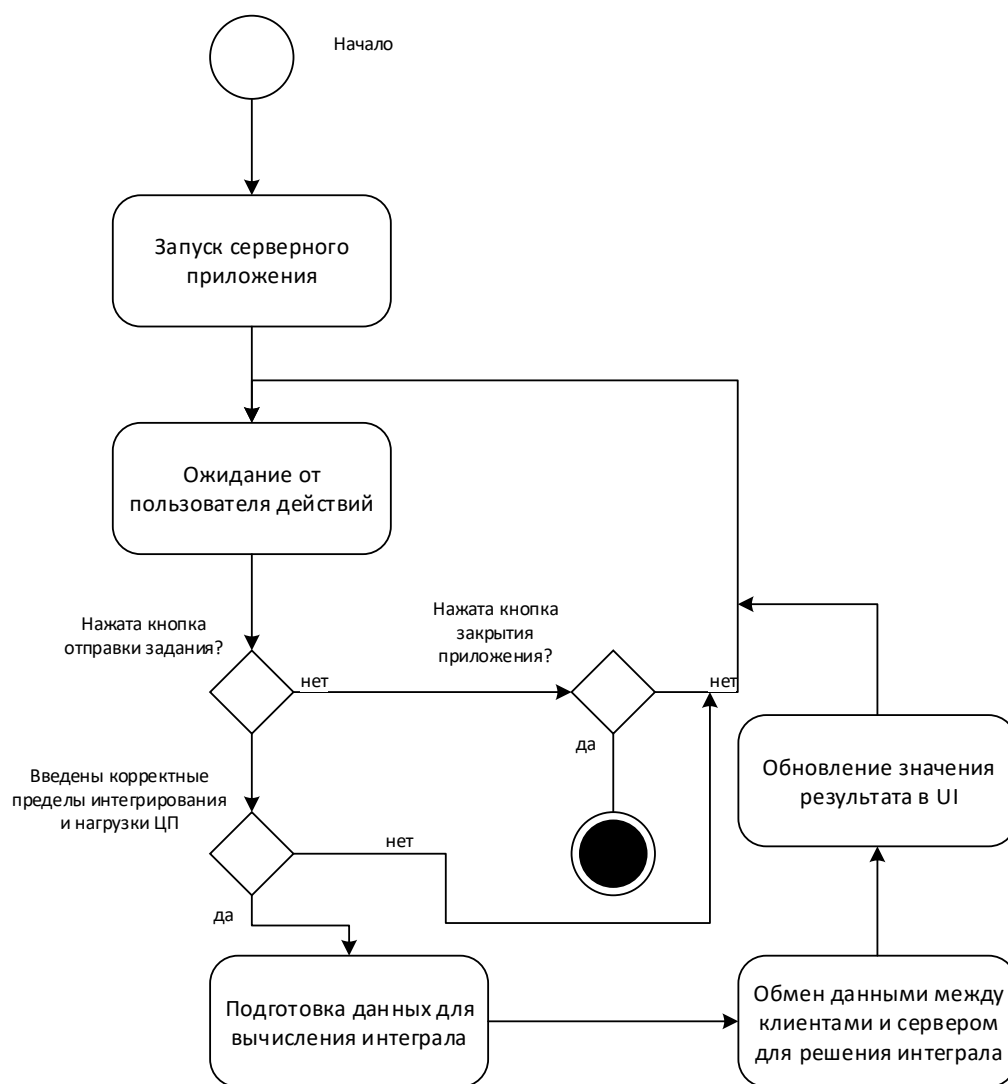


Рисунок 3. Диаграмма деятельности для серверного приложения

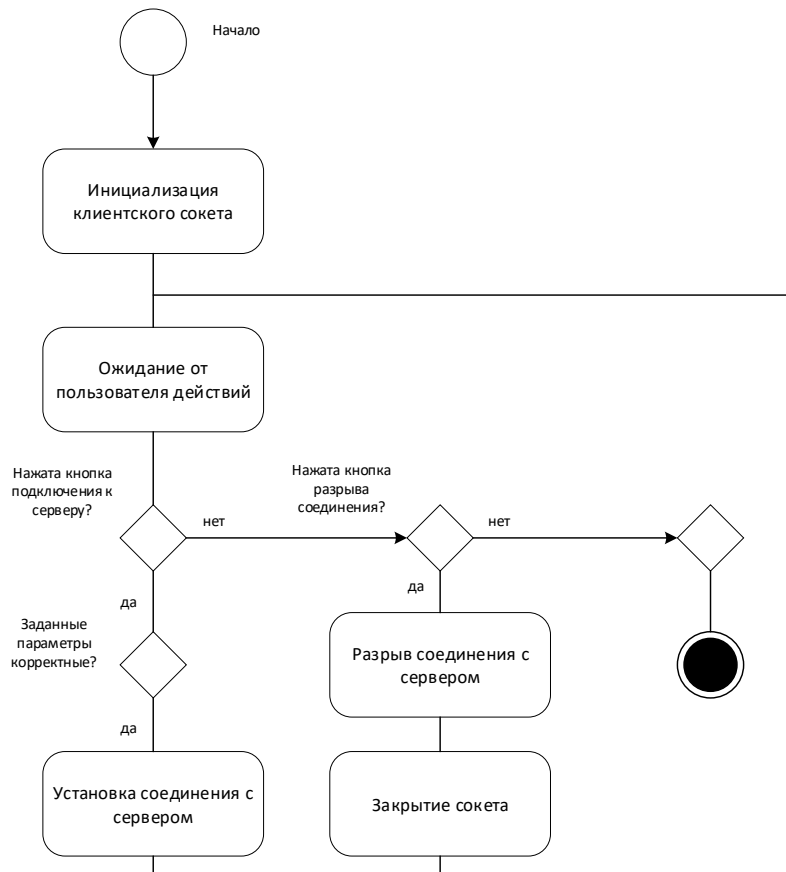


Рисунок 4. Диаграмма деятельности для клиентского приложения

С использованием библиотеки WinSock был реализован клиент-серверный код с поддержкой соединения по протоколу TCP/IP. Программный сокет создан с использованием функции `socket()` и структуры `SOCK_ADDR`, содержащей настройки серверного сокета.

Далее был реализован алгоритм вычисления определенного интеграла $\int \sqrt{x} dx$ методом средних прямоугольников с учетом требования многопоточности. Затем был добавлен механизм ограничения предельно допустимого уровня использования процессорного времени на стороне серверного приложения. Реализация вычисления определённого интеграла была выполнена с использованием уже проверенных ранее наработок.

Ожидание новых соединений было выделено в отдельную нить, сигнализация о начале вычислений – асинхронная.

Полученная диаграмма последовательностей программы представлена ниже (Рисунок 5).

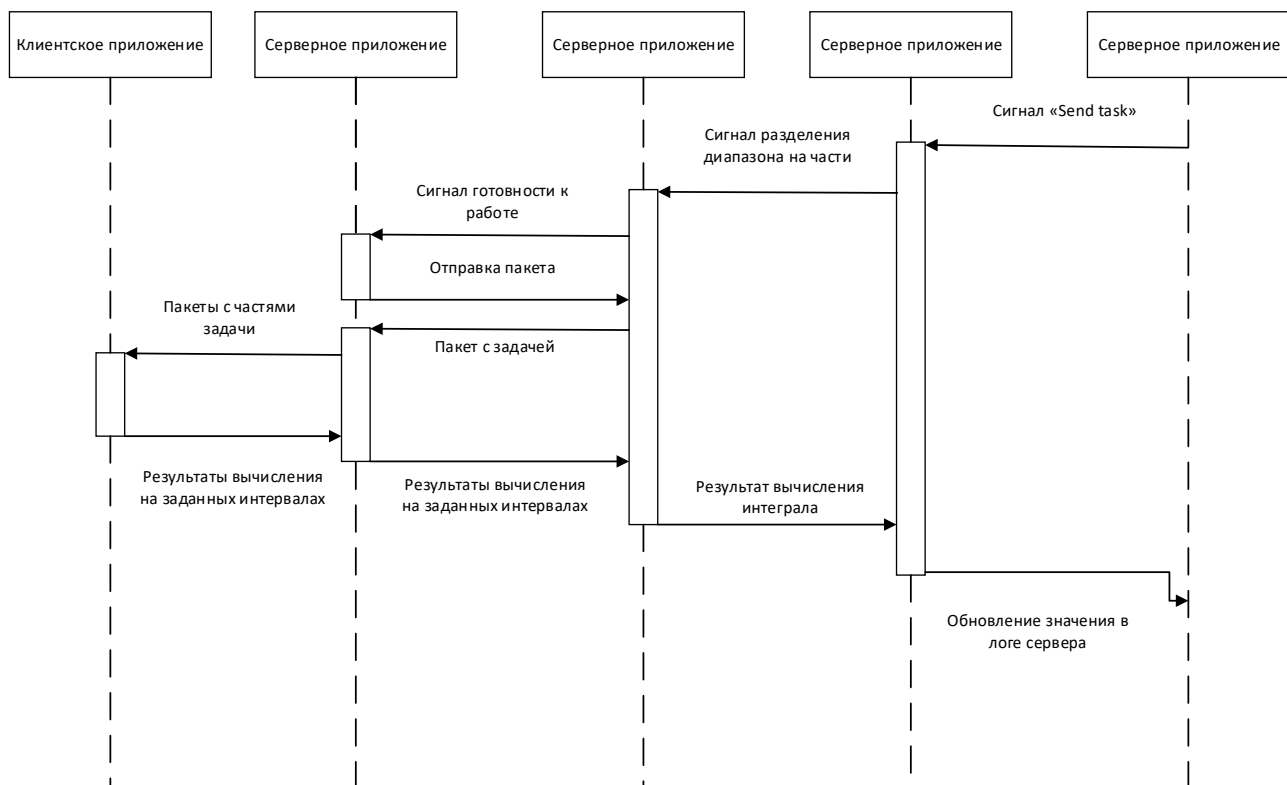


Рисунок 5. Диаграмма последовательностей клиент-серверной программы

4 Описание программы

После запуска вычислений сервер выдаёт сигнал о начале вычислений для распределителя задач. Распределитель задач собирает предустановленные в заголовочном файле параметры интеграла (шаг, количество пакетов в расчёте на одного клиента), диапазон читается из соответствующего поля в графическом интерфейсе. В том случае, если диапазон интегрирования корректный, передаётся сигнал отдельным соединениям для инициирования общения с клиентами.

Каждое соединение может забрать очередной пакет и вытеснить из очереди его номер. В том случае, если клиент не сможет просчитать данный пакет, сервер вернёт его номер в очередь у распределителя. То есть, при следующем обращении другое соединение получит доступ к данному пакету.

Вычисления будут считать оконченными, когда общее количество пакетов будет равно количеству посчитанных пакетов. После этого в лог сервера будет выведен полученный результат вычислений.

Клиентское приложение получает пакет данных в виде структуры из диапазона, уровня загрузки ЦП и кода команды (необходимо считать пакет и слать ответ или нет). Вычисление ведётся в параллельных нитях, результат собирается в родительской нити и отправляется серверу по сокету.

Графический интерфейс клиентского окна представляет из себя набор элементов для ввода текстовой информации (IP адрес сервера, порт сервера) и кнопку для подключения к серверу. В том случае, если необходимо отключиться от сервера, пользователь может нажать на данную кнопку повторно.

Диаграмма вариантов использования клиент-серверной программы
представлена ниже (Рисунок 9).

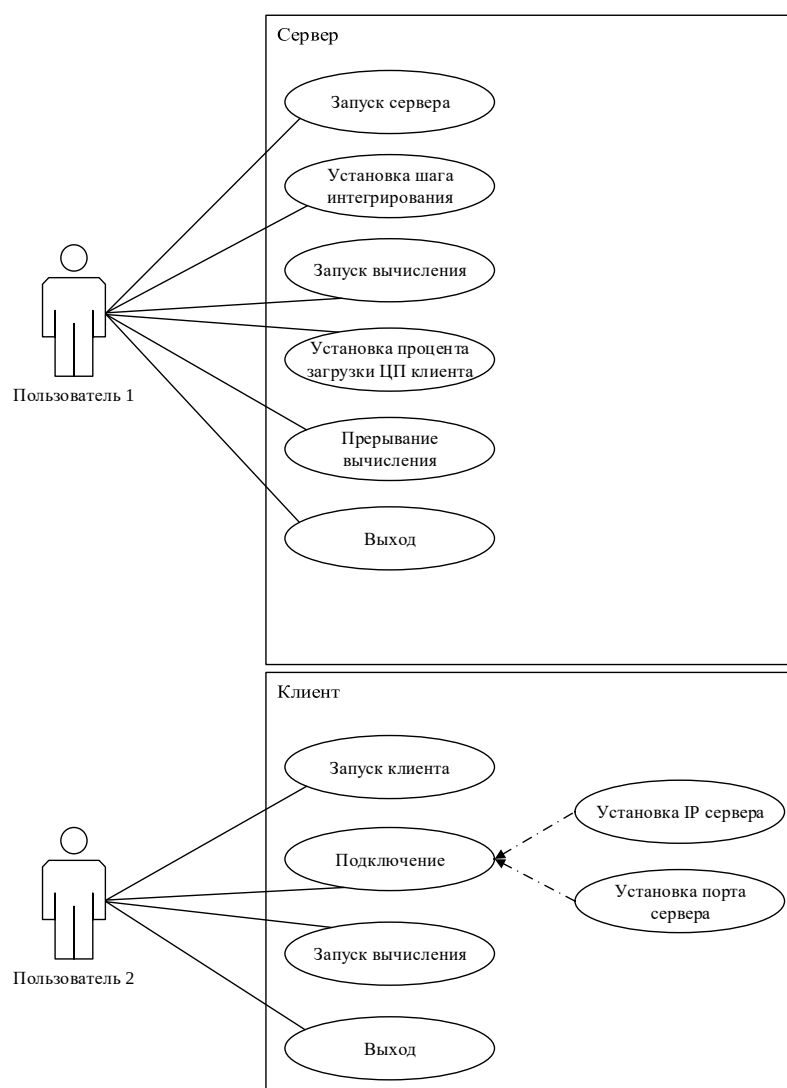


Рисунок 6. Диаграмма вариантов использования клиент-серверной программы

5 Отладка и тестирование

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2015, которая содержит в себе все необходимые средства для разработки и отладки модулей и программ. Для отладки программы использовались различные инструменты: точка остановки, пошаговое выполнение кода программы, анализ содержимого глобальных и локальных переменных, результаты работы некоторых функций.

Тестирование проводилось в рабочем порядке, в процессе разработки и после завершения написания программы. В процессе тестирования было выявлено и исправлено ошибки, связанные с подсчетом интеграла, ограничением загрузки процессора и подключением к серверу.

Ниже приведены результаты работы программы (Рисунок 6) и стороннего приложения (Рисунок 7). Если сравнивать результаты работы нашей программы и стороннего приложения, то можно сделать вывод, что результат верен.

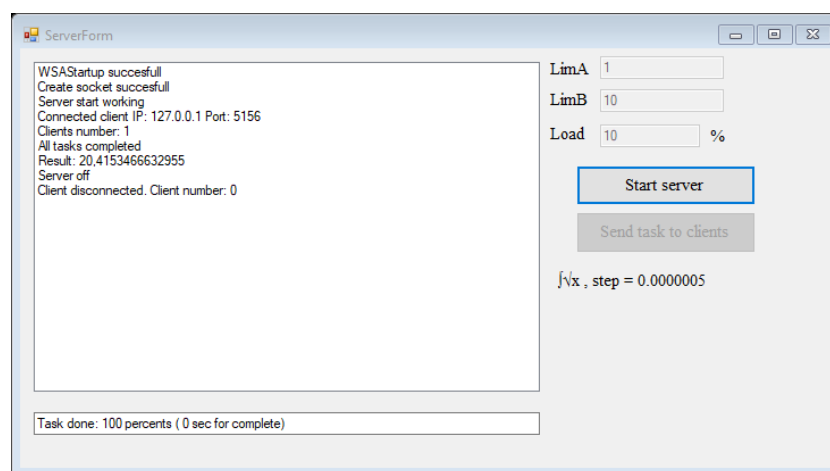


Рисунок 7. Результаты работы разработанной программы

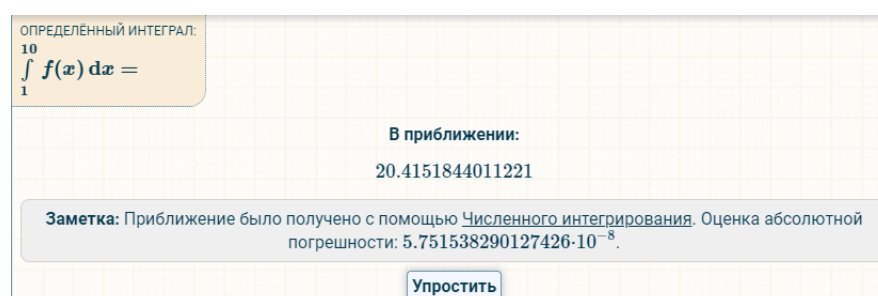


Рисунок 8. Результаты работы сторонней программы

Было проведено тестирование функционала по ограничению нагрузки на ЦП в пределах заданного значения, введённого на серверной стороне. Результаты тестирования приведены ниже (Рисунок 8).

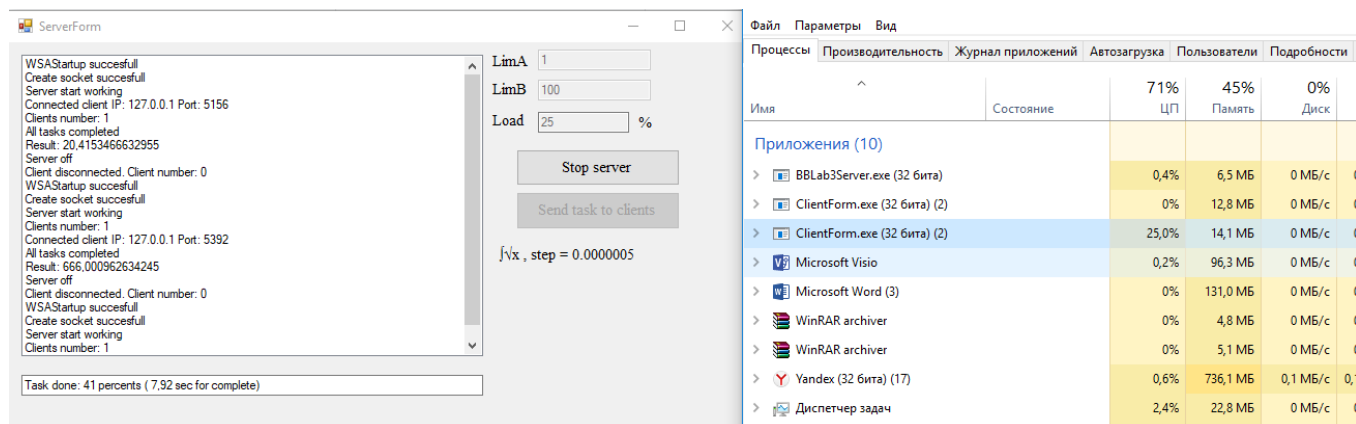


Рисунок 9. Демонстрация работы ограничения нагрузки на ЦП

6 Руководство пользователя

Программа представляет собой клиент-серверное приложение облачного типа для управления вычислительным процессом и состоит из двух частей: клиента и сервера. Она предназначена для вычисления определенного интеграла $\int \sqrt{x} dx$ с шагом интегрирования 0,0000005. Взаимодействие клиента и сервера организовано по протоколу TCP/IP. Диаграмма развертывания клиент-серверного приложения приведено ниже (Рисунок 10).

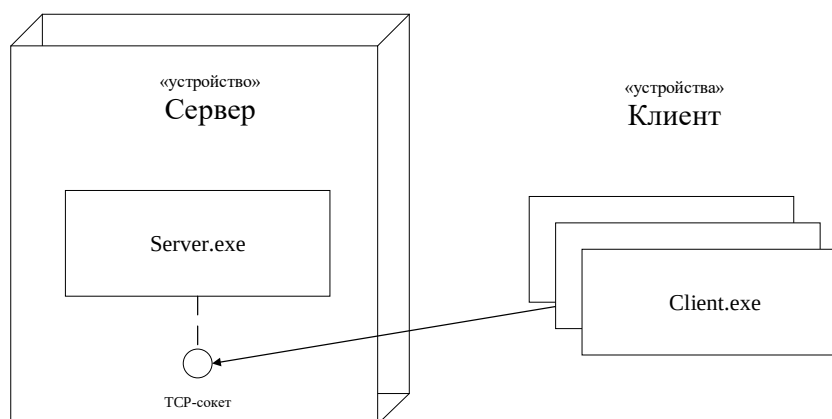


Рисунок 10. Диаграмма развертывания клиент-серверного приложения

Для начала работы необходимо запустить серверное приложение (Рисунок 11).

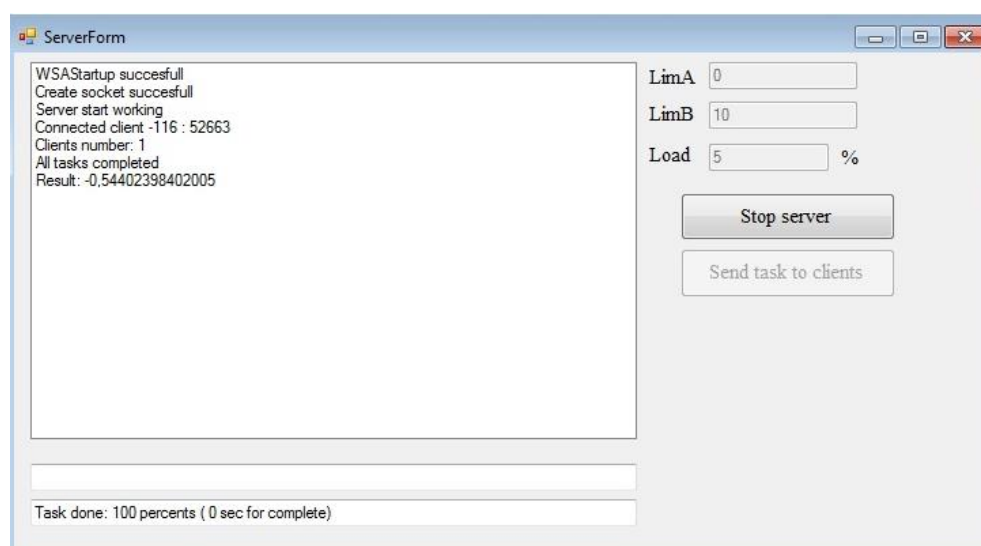


Рисунок 11. Окно серверного приложения

В случае успешного запуска в лог сервера будет выведено соответствующее сообщение «Server start working». В правой части окна содержатся следующие элементы:

1. Текстовое поле «LimA», требующее ввода значения нижнего предела интегрирования;
2. Текстовое поле «LimB», требующее ввода значения верхнего предела интегрирования;
3. Текстовое поле «Load», требующее ввода значения предельной загрузки клиентом ЦП;
4. Кнопка «Strat/Stop Server» для запуска/выключения сервера.
5. Кнопка «SendTasksToClients» для отправки задания клиентом для расчета заданного интеграла.

В левой части окна содержатся следующие элементы:

1. Текстовое поле «LogListBox» в котором отображаются сообщения о подключении и отключении клиентов, ошибки, а также сообщения об успешном запуске сервера;
2. Текстовое поле «ProcessStatusBox» в котором отображается процент завершения задачи и оставшееся время на её выполнение;

После успешного запуска сервера необходимо запустить клиентское приложение (Рисунок 12).

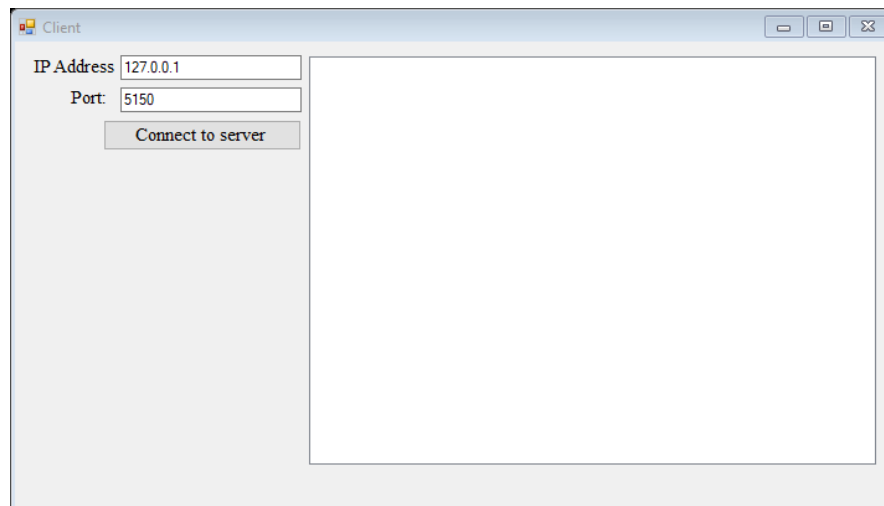


Рисунок 12. Окно клиентского приложения

Окно клиентского приложения содержит следующие графические элементы:

1. Текстовое поле «IP Address»;
2. Текстовое поле «Port»;
3. Кнопка «Connect/Disconnect from server ».

Далее необходимо установить соединение с сервером. Требуется ввести IP сервера (отображено в заголовке окна серверного приложения), порт (стандартный порт 5150). После этого требуется нажать на кнопку «Connect to server». Текст на кнопке изменится на «Disconnect from server», свидетельствующий о успешно проведенном соединении.

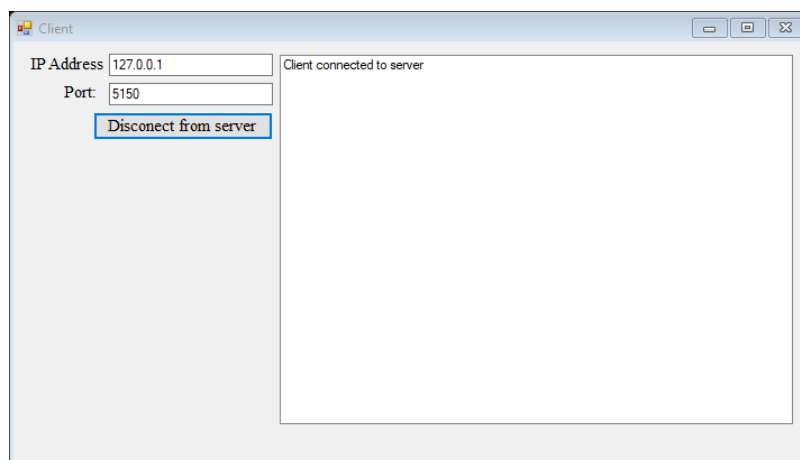


Рисунок 13. Активное окно клиента

При успешной установке соединения, на стороне сервера должно отобразиться появившееся соединение в логе (Рисунок 14).

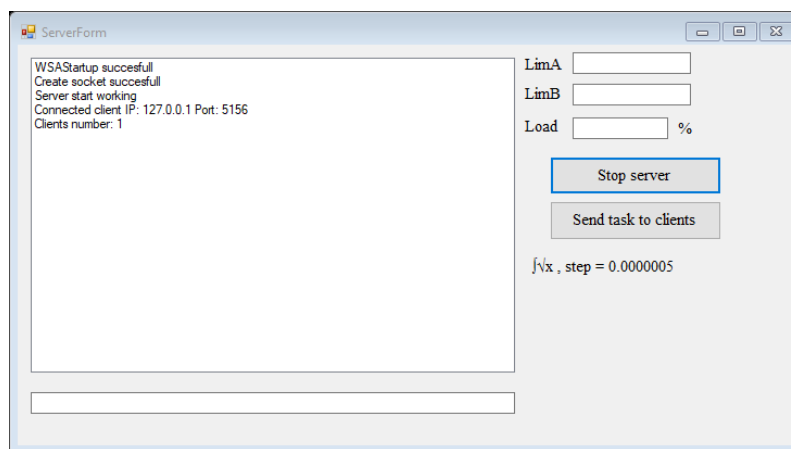


Рисунок 14. Сообщение об установке соединения

Для запуска вычислений пользователю необходимо ввести пределы интегрирования в соответствующие поля и нажать на кнопку «Send task to clients». (Рисунок 15).

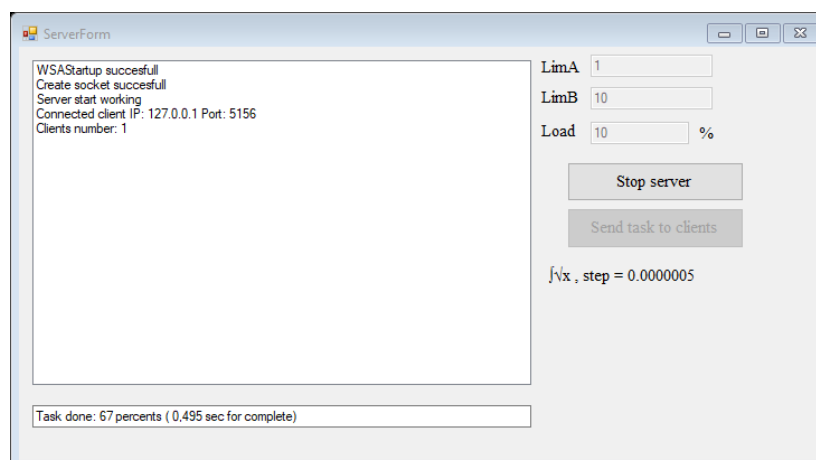


Рисунок 15. Запуск вычислений

В момент вычисления на клиенте высвечиваются сообщения о прогрессе вычислений, различных ошибках (Рисунок 16).

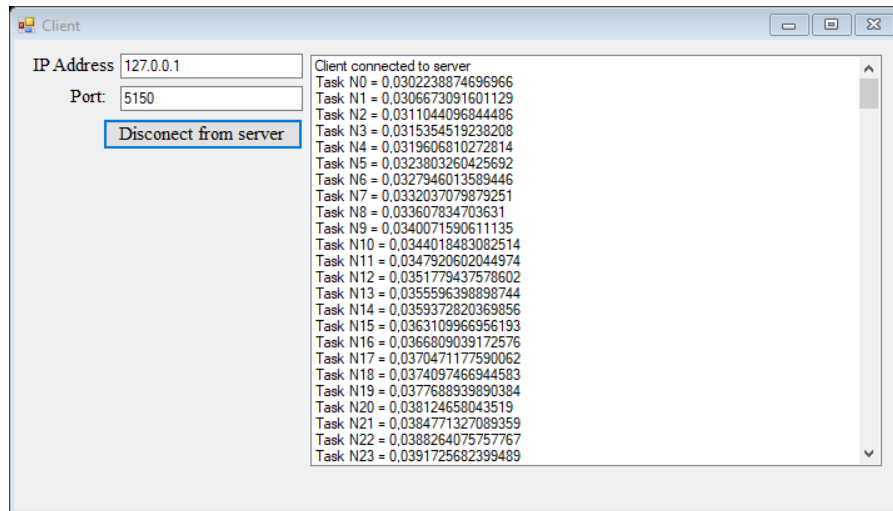


Рисунок 16. Готовая к работе клиент-серверная система

После завершения вычислений в логе сервера появится сообщение «All tasks completed», отобразится результат интегрирования, в логе сохранятся сообщения о выполнении интегрирования (Рисунок 17).

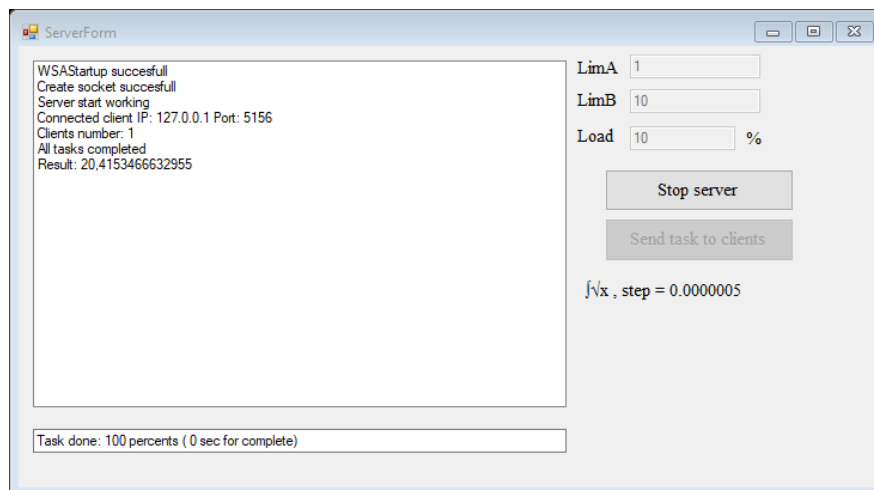


Рисунок 17. Завершение вычислений

Завершить работу сервера и клиентов можно по щелчку элемента «Stop Server». Соединение с сервером, если оно было активно на момент отключения, будет разорвано автоматически.

Заключение

При выполнении данной курсовой работы были получены навыки разработки параллельной программы, усвоены механизмы синхронизации. Получены базовые навыки работы с библиотекой WinSock и организации обмена данных по протоколу TCP/IP.

В рамках выполнения курсового проекта было создано клиент-серверное приложение, позволяющее производить расчет значения определенного интеграла. Пользовательский интерфейс интуитивно понятен и прост. Имеется возможность ограничения потребления клиентами ресурсов ЦП.

В дальнейшем программу можно улучшить различными способами в соответствии с потребностями пользователя. Можно добавить возможность вводить другие подынтегральные функции, а также менять диапазон вычислений.

Список используемых источников

1. MSDN
2. Антонов А. С. Технологии параллельного программирования MPI и OpenMP – Изд. МГУ, 2012.
3. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы. – СПб., Питер, 2007.
4. Хортон А. Visual C++ 2010. Полный курс, Издательство «Вильямс», 2011.
5. Определенный интеграл онлайн [Электронный ресурс]. – Электрон.дан. – URL: https://math24.biz/integral_def – 27.04.2018

Приложение А. Листинг программы

Приложение А.1 clientForm.cpp

```
#include "Form1.h"
using namespace ClientForm;

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    Application::Run(gcnew Form1());
    return 0;
}
```

Приложение А.2 Form1.h

```
#pragma once
#include <iostream>
#include <stdio.h>
#include <tchar.h>
#include <WinSock2.h>
#include <WS2tcpip.h>
#include <conio.h>
#include <process.h>
#include <list>

#include "Func.h"

#pragma comment (lib, "wsock32.lib")
#pragma comment (lib, "ws2_32.lib")

namespace ClientForm {

    using namespace System;
    using namespace System::Runtime::InteropServices;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Threading;
    using namespace System::Collections;
    using namespace System::IO;
    using namespace System::Net;
    using namespace System::Net::Sockets;
```

```

    /// <summary>
    /// Сводка для Form1
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();
            portTextBox->Text = "5150";
            ipAddressTexBox->Text = "127.0.0.1";
        }

    protected:
        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        ~Form1()
        {
            if (components)
            {
                delete components;
            }
        }

    private: System::Windows::Forms::Label^ label1;
    protected:
    private: System::Windows::Forms::Label^ label2;
    private: System::Windows::Forms::TextBox^ ipAddressTexBox;
    private: System::Windows::Forms::TextBox^ portTextBox;
    private: System::Windows::Forms::Button^
connectWihServerButton;
    private: System::Windows::Forms::ListBox^ logListBox;

        delegate void setTextCallback(System::String^
text);

    private: bool connectedToServer;

    private: Thread^ recvThread;

    private:
        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
        void InitializeComponent(void)
        {
            this->label1 = (gcnew
System::Windows::Forms::Label());
            this->label2 = (gcnew
System::Windows::Forms::Label());

```

```

        this->ipAddressTextBox = (gcnew
System::Windows::Forms::TextBox());
        this->portTextBox = (gcnew
System::Windows::Forms::TextBox());
        this->connectWihServerButton = (gcnew
System::Windows::Forms::Button());
        this->logListBox = (gcnew
System::Windows::Forms::ListBox());
        this->SuspendLayout();
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label1->Location = System::Drawing::Point(12,
12);

        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(73, 17);
        this->label1->TabIndex = 0;
        this->label1->Text = L"IP Address";
        //
        // label2
        //
        this->label2->AutoSize = true;
        this->label2->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label2->Location = System::Drawing::Point(42,
37);

        this->label2->Name = L"label2";
        this->label2->Size = System::Drawing::Size(36, 17);
        this->label2->TabIndex = 1;
        this->label2->Text = L"Port:";
        //
        // ipAddressTextBox
        //
        this->ipAddressTextBox->Location =
System::Drawing::Point(85, 11);
        this->ipAddressTextBox->Name = L"ipAddressTextBox";
        this->ipAddressTextBox->Size =
System::Drawing::Size(146, 20);
        this->ipAddressTextBox->TabIndex = 2;
        //
        // portTextBox
        //
        this->portTextBox->Location =
System::Drawing::Point(85, 37);

```

```

        this->portTextBox->Name = L"portTextBox";
        this->portTextBox->Size = System::Drawing::Size(146,
20);

        this->portTextBox->TabIndex = 3;
        //
        // connectWihServerButton
        //
        this->connectWihServerButton->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
        System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
        this->connectWihServerButton->Location =
System::Drawing::Point(71, 63);
        this->connectWihServerButton->Name =
L"connectWihServerButton";
        this->connectWihServerButton->Size =
System::Drawing::Size(160, 25);
        this->connectWihServerButton->TabIndex = 4;
        this->connectWihServerButton->Text = L"Connect to
server";

        this->connectWihServerButton-
>UseVisualStyleBackColor = true;
        this->connectWihServerButton->Click += gcnew
System::EventHandler(this, &Form1::connectWihServerButton_Click);
        //
        // logListBox
        //
        this->logListBox->FormattingEnabled = true;
        this->logListBox->Location =
System::Drawing::Point(237, 12);
        this->logListBox->Name = L"logListBox";
        this->logListBox->Size = System::Drawing::Size(457,
329);

        this->logListBox->TabIndex = 5;
        //
        // Form1
        //
        this->AutoScaleDimensions =
System::Drawing::SizeF(6, 13);
        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
        this->ClientSize = System::Drawing::Size(706, 376);
        this->Controls->Add(this->logListBox);
        this->Controls->Add(this->connectWihServerButton);
        this->Controls->Add(this->portTextBox);
        this->Controls->Add(this->ipAddressTextBox);
        this->Controls->Add(this->label2);
        this->Controls->Add(this->label1);
        this->Name = L"Form1";
        this->Text = L"Client";

```

```

        this->FormClosed += gnew
System::Windows::Forms::FormClosedEventHandler(this,
&Form1::Form1_FormClosed);
        this->ResumeLayout(false);
        this->PerformLayout();
    }

```

```

#pragma endregion

```

```

    private: System::Void setTextInLogListBox(System::String^
text)
    {
        try
        {
            if (this->logListBox->InvokeRequired)
            {
                setTextCallback^ d = gnew
setTextCallback(this, &Form1::setTextInLogListBox);
                this->Invoke(d, gnew array<
System::Object^>(1) { text });
            }
            else
            {
                this->logListBox->Items->Add(text);
            }
        }
        catch (System::Exception^ ex)
        {
            setTextInLogListBox("Error: " + ex->Message);
            return;
        }
    }

```

```

    private: System::Void
setTextInConnectWihServerButton(System::String^ text)
    {
        if (this->connectWihServerButton->InvokeRequired)
        {
            setTextCallback^ d = gnew setTextCallback(this,
&Form1::setTextInConnectWihServerButton);
            this->Invoke(d, gnew array< System::Object^>(1) {
text });
        }
        else
        {
            this->connectWihServerButton->Text = text;
        }
    }

```

```

    private: System::Void recvProc()

```

```

{
    int nsize;

    while ((nsize = recv(s, &buff[0], sizeof(buff), 0)) !=
SOCKET_ERROR && connectedToServer)
    {
        integral_info* integral_value =
(integral_info*)&buff[0];

        per = (double)integral_value->load_percentage / 100;

        double x = integral_value->limA;
        int start = 0, stop = 0, workTime = 0, sleepTime =
0, rest = 0;

        int startTask = 0, stopTask = 0, resultTask = 0;
        double result = 0;

        startTask = clock();
        while (x < integral_value->limB) {
            int counter = 0;
            start = clock();
            while (counter < integral_value->numpoint && x
< integral_value->limB) {
                result += (sqrt(x) + sqrt(x +
integral_value->step)) * 0.5 * integral_value->step;
                x += integral_value->step;
                counter++;
            }
            stop = clock();
            workTime += stop - start;

            start = clock();
            rest = workTime * (1 / per) - workTime -
sleepTime;

            if (rest >= 0) {
                Sleep(rest);
            }
            stop = clock();
            sleepTime += stop - start;
        }
        stopTask = clock();
        resultTask = stopTask - startTask;

        setTextInLogListBox("Task N" + integral_value-
>task_index.ToString() + " = " + result.ToString());
        result_from_client result_data;

        result_data.result = result;
        result_data.execute_time = resultTask ;
        result_data.index = integral_value->task_index;

        //После нахождения результата - отсылаем его серверу

```

```

        send(s, (char*)&result_data, sizeof(result_data),
0);

    }
    setTextInLogListBox("Client disconnected");
}

private: System::Void
connectWihServerButton_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (!connectedToServer)
    {
        if (WSAStartup(MAKEWORD(2, 0), &wsd) != 0) //
инициализировать библиотеку WinSock
        {
            setTextInLogListBox("Error: WSAStartup!");
            setTextInLogListBox("Error code: " +
WSAGetLastError().ToString());
            return;
        }

        s = socket(AF_INET, SOCK_STREAM, 0); // создать
гнездо
        if (s == INVALID_SOCKET)
        {
            setTextInLogListBox("Error: socket!");
            setTextInLogListBox("Error code: " +
WSAGetLastError().ToString());
            WSACleanup();
            return;
        }

        char* nativeString;

        try
        {
            String^ str = ipAddressTextBox->Text;
            IntPtr ptrToNativeString =
Marshal::StringToHGlobalAnsi(str);
            nativeString =
static_cast<char*>(ptrToNativeString.ToPointer());

        }
        catch (System::Exception^ ex)
        {
            System::Windows::Forms::MessageBox::Show("Wrong
adress", "Error", System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
            return;
        }
    }
}

```

```

        addr = inet_addr(nativeString);    // преобразовать
в 32-битное число

        if (addr == INADDR_NONE)          // при ошибке
        {
            setTextInLogListBox("Wrong address");
            setTextInLogListBox("Error: address!");
            setTextInLogListBox("Error code: " +
WSAGetLastError());
            closesocket(s);
            WSACleanup();
            return;
        }

        try
        {
            port = Convert::ToInt32(portTextBox->Text);
        }
        catch (System::Exception^ ex)
        {
            System::Windows::Forms::MessageBox::Show("Wrong
port", "Error", System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
            return;
        }

        if (!(port <= 49151 && port >= 1024))
        {
            System::Windows::Forms::MessageBox::Show("Wrong
port", "Error", System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
            return;
        }

        srv_address.sin_family = AF_INET;    // заполнить
структуру данных гнезда
        srv_address.sin_port = htons(port);
        srv_address.sin_addr.s_addr = addr;

        if (connect(s, (sockaddr *)&srv_address,
sizeof(srv_address)))
        {
            setTextInLogListBox("Connect error: \n" +
WSAGetLastError());
            closesocket(s);
            WSACleanup();
            return;
        }
        setTextInLogListBox("Client connected to server\n");

        connectedToServer = true;

```

```

        setTextInConnectWihServerButton("Disconect from
server");

        recvThread = gcnew Thread(gcnew ThreadStart(this,
&Form1::recvProc));/////
        recvThread->Start();/////
        //цикл приема сообщений
    }
    else
    {
        connectedToServer = false;
        setTextInConnectWihServerButton("Connect to
server");

        closesocket(s);
        WSACleanup();
    }
}

private: System::Void Form1_FormClosed(System::Object^
sender, System::Windows::Forms::FormClosedEventArgs^ e) {
    exit(1);
}

};
}

```

Приложение A.3 Form1.h

```

#include <iostream>
#include <stdio.h>
#include <tchar.h>
#include <WinSock2.h>
#include <WS2tcpip.h>
#include <conio.h>
#include <process.h>
#include <time.h>

#pragma once
#pragma comment (lib, "wsock32.lib")
#pragma comment (lib, "ws2_32.lib")

double per = 0;

struct integral_info
{
    double limA;                // нижний предел интегрирования
    double limB;                // верхний предел интегрирования
    int numpoint;               // количество шагов интегрирования
    double step;
    int task_index;             // индекс задачи
    double load_percentage;     // процент загрузки клиента
};

```

```

struct result_from_client
{
    double result;
    double execute_time;
    int index;                // индекс решенной задачи
};

WSADATA          wsd;
SOCKET           s;
sockaddr_in      client_address;    // структура данных
гнззд
sockaddr_in      srv_address;

unsigned short    port = 5150;      // порт серверной
программы
int               server_addr_size;
char              ip_address[16];
unsigned long     addr;             // адрес сервера

char buff[1024], str[100];

```

Приложение A.4 serverForm.cpp

```

#include "ServerForm.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThread]
void Main(array<String^>^args)
{
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    BBLab3Server::ServerForm serverForm;
    Application::Run(%serverForm);
}

```

Приложение A.5 serverForm.h

```

#pragma once
#include <iostream>
#include <stdio.h>
#include <tchar.h>
#include <WinSock2.h>
#include <WS2tcpip.h>
#include <conio.h>
#include <process.h>

```

```

#include <list>

#include "Func.h"

#pragma comment (lib, "wsock32.lib")
#pragma comment (lib, "ws2_32.lib")
namespace BBLab3Server {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Threading;
    using namespace System::Collections;
    using namespace System::IO;
    using namespace System::Net;
    using namespace System::Net::Sockets;

    /// <summary>
    /// Сводка для ServerForm
    /// </summary>
    public ref class ServerForm : public
System::Windows::Forms::Form
    {
    public:
        ServerForm(void)
        {
            InitializeComponent();
            //Состояние элементов управления по умолчанию (при
запуске приложения сервера)
            SendToClientsButton->Enabled = false;
            LimATextBox->Enabled = false;
            LimBTextBox->Enabled = false;
            loadTextBox->Enabled = false;

        }

    protected:
        ~ServerForm()
        {
            if (components)
            {
                delete components;
            }
        }

    public: value class client_info
    {
    public:
        SOCKET s;
        bool is_free;
    };
};
}

```

```

};

private: bool serverEnabled;
private: int loadPercentage;
private: double limA;
private: double limB;
private: Thread^ acceptThread;
private: Thread^ clientThread;
private: Thread^ mainThread;
private: System::Windows::Forms::Button^
StartStopServerButton;
        delegate void setTextCallback(System::String^
text);
private: System::Windows::Forms::Button^
SendToClientsButton;

private: System::Windows::Forms::ListBox^   LogListBox;
private: System::Windows::Forms::TextBox^   ResultTextBox;
private: System::Windows::Forms::TextBox^   LimATextBox;
private: System::Windows::Forms::TextBox^   LimBTextBox;
private: System::Windows::Forms::Label^     LimALabel;
private: System::Windows::Forms::Label^     LimBLabel;
private: System::Windows::Forms::Label^     LoadLabel;
private: System::Windows::Forms::TextBox^   loadTextBox;
private: System::Windows::Forms::Label^     PercentLabel;
private: System::Windows::Forms::Label^     InfoLabel;
protected:

private:
        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
        void InitializeComponent(void)
        {
                this->StartStopServerButton = (gcnew
System::Windows::Forms::Button());
                this->SendToClientsButton = (gcnew
System::Windows::Forms::Button());
                this->LogListBox = (gcnew
System::Windows::Forms::ListBox());
                this->ResultTextBox = (gcnew
System::Windows::Forms::TextBox());
                this->LimATextBox = (gcnew
System::Windows::Forms::TextBox());
                this->LimBTextBox = (gcnew
System::Windows::Forms::TextBox());
                this->LimALabel = (gcnew
System::Windows::Forms::Label());
                this->LimBLabel = (gcnew
System::Windows::Forms::Label());
                this->LoadLabel = (gcnew
System::Windows::Forms::Label());

```

```

        this->loadTextBox = (gcnew
System::Windows::Forms::TextBox());
        this->PercentLabel = (gcnew
System::Windows::Forms::Label());
        this->InfoLabel = (gcnew
System::Windows::Forms::Label());
        this->SuspendLayout();
        //
        // StartStopServerButton
        //
        this->StartStopServerButton->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
        System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
        this->StartStopServerButton->Location =
System::Drawing::Point(490, 103);
        this->StartStopServerButton->Name =
L"StartStopServerButton";
        this->StartStopServerButton->Size =
System::Drawing::Size(158, 35);
        this->StartStopServerButton->TabIndex = 0;
        this->StartStopServerButton->Text = L"Start server";
        this->StartStopServerButton->UseVisualStyleBackColor
= true;

        this->StartStopServerButton->Click += gcnew
System::EventHandler(this,
&ServerForm::StartStopServerButton_Click);
        //
        // SendToClientsButton
        //
        this->SendToClientsButton->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->SendToClientsButton->Location =
System::Drawing::Point(490, 144);
        this->SendToClientsButton->Name =
L"SendToClientsButton";
        this->SendToClientsButton->Size =
System::Drawing::Size(158, 36);
        this->SendToClientsButton->TabIndex = 1;
        this->SendToClientsButton->Text = L"Send task to
clients";
        this->SendToClientsButton->UseVisualStyleBackColor =
true;

        this->SendToClientsButton->Click += gcnew
System::EventHandler(this,
&ServerForm::SendToClientsButton_Click);
        //
        // LogListBox
        //

```

```

        this->LogListBox->FormattingEnabled = true;
        this->LogListBox->HorizontalScrollbar = true;
        this->LogListBox->Location =
System::Drawing::Point(12, 12);
        this->LogListBox->Name = L"LogListBox";
        this->LogListBox->Size = System::Drawing::Size(446,
290);

        this->LogListBox->TabIndex = 2;
        //
        // ResultTextBox
        //
        this->ResultTextBox->Location =
System::Drawing::Point(12, 320);
        this->ResultTextBox->Name = L"ResultTextBox";
        this->ResultTextBox->Size =
System::Drawing::Size(446, 20);
        this->ResultTextBox->TabIndex = 4;
        //
        // LimATextBox
        //
        this->LimATextBox->Location =
System::Drawing::Point(511, 7);
        this->LimATextBox->Name = L"LimATextBox";
        this->LimATextBox->Size = System::Drawing::Size(109,
20);

        this->LimATextBox->TabIndex = 6;
        //
        // LimBTextBox
        //
        this->LimBTextBox->Location =
System::Drawing::Point(511, 36);
        this->LimBTextBox->Name = L"LimBTextBox";
        this->LimBTextBox->Size = System::Drawing::Size(109,
20);

        this->LimBTextBox->TabIndex = 7;
        //
        // LimALabel
        //
        this->LimALabel->AutoSize = true;
        this->LimALabel->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->LimALabel->Location =
System::Drawing::Point(464, 10);
        this->LimALabel->Name = L"LimALabel";
        this->LimALabel->Size = System::Drawing::Size(42,
17);

        this->LimALabel->TabIndex = 8;
        this->LimALabel->Text = L"LimA";
        //
        // LimBLabel

```

```

        //
        this->LimBLabel->AutoSize = true;
        this->LimBLabel->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->LimBLabel->Location =
System::Drawing::Point(464, 37);
        this->LimBLabel->Name = L"LimBLabel";
        this->LimBLabel->Size = System::Drawing::Size(41,
17);

        this->LimBLabel->TabIndex = 9;
        this->LimBLabel->Text = L"LimB";
        //
        // LoadLabel
        //
        this->LoadLabel->AutoSize = true;
        this->LoadLabel->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->LoadLabel->Location =
System::Drawing::Point(464, 67);
        this->LoadLabel->Name = L"LoadLabel";
        this->LoadLabel->Size = System::Drawing::Size(38,
17);

        this->LoadLabel->TabIndex = 10;
        this->LoadLabel->Text = L"Load";
        //
        // loadTextBox
        //
        this->loadTextBox->Location =
System::Drawing::Point(511, 67);
        this->loadTextBox->Name = L"loadTextBox";
        this->loadTextBox->Size = System::Drawing::Size(88,
20);

        this->loadTextBox->TabIndex = 11;
        //
        // PercentLabel
        //
        this->PercentLabel->AutoSize = true;
        this->PercentLabel->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->PercentLabel->Location =
System::Drawing::Point(605, 69);
        this->PercentLabel->Name = L"PercentLabel";
        this->PercentLabel->Size = System::Drawing::Size(21,
17);

```

```

        this->PercentLabel->TabIndex = 12;
        this->PercentLabel->Text = L"%";
        //
        // InfoLabel
        //
        this->InfoLabel->AutoSize = true;
        this->InfoLabel->Font = (gcnew
System::Drawing::Font(L"Times New Roman", 11.25F,
System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->InfoLabel->Location =
System::Drawing::Point(471, 196);
        this->InfoLabel->Name = L"InfoLabel";
        this->InfoLabel->Size = System::Drawing::Size(152,
17);

        this->InfoLabel->TabIndex = 13;
        this->InfoLabel->Text = L"√x dx , step = 0.0000005";
        //
        // ServerForm
        //
        this->AutoScaleDimensions =
System::Drawing::SizeF(6, 13);
        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
        this->ClientSize = System::Drawing::Size(717, 369);
        this->Controls->Add(this->InfoLabel);
        this->Controls->Add(this->PercentLabel);
        this->Controls->Add(this->loadTextBox);
        this->Controls->Add(this->LoadLabel);
        this->Controls->Add(this->LimBLabel);
        this->Controls->Add(this->LimALabel);
        this->Controls->Add(this->LimBTextBox);
        this->Controls->Add(this->LimATextBox);
        this->Controls->Add(this->ResultTextBox);
        this->Controls->Add(this->LogListBox);
        this->Controls->Add(this->SendToClientsButton);
        this->Controls->Add(this->StartStopServerButton);
        this->Name = L"ServerForm";
        this->Text = L"ServerForm";
        this->FormClosed += gcnew
System::Windows::Forms::FormClosedEventHandler(this,
&ServerForm::ServerForm_FormClosed);
        this->ResumeLayout(false);
        this->PerformLayout();

    }
#pragma endregion
    private: System::Void
StartStopServerButton_Click(System::Object^ sender,
System::EventArgs^ e)
    {
        try

```

```

        {
            if (!serverEnabled)
            {

InitializeCriticalSection(&critsect_for_task_list);

InitializeCriticalSection(&critsect_for_clients_list);

                // Инициализация библиотеки WinSock

                if (WSAStartup(MAKEWORD(2, 0), &wsd))
                {
                    setTextInLogListBox("Error WSAStartup:
%d\n" + WSAGetLastError().ToString());
                    return;
                }
                setTextInLogListBox("WSAStartup succesfull\n");

                // Создание сокета
                if ((sck = socket(AF_INET, SOCK_STREAM, 0)) ==
INVALID_SOCKET)
                {
                    setTextInLogListBox("Error SOCKET: %d\n" +
WSAGetLastError().ToString());
                    WSACleanup();
                    return;
                }
                setTextInLogListBox("Create socket
succesfull\n");

                // Заполнение структуры данных

                srv_address.sin_family = AF_INET;
                srv_address.sin_port = htons(port);
                srv_address.sin_addr.s_addr =
htonl(INADDR_ANY);

                if (bind(sck, (sockaddr*)&srv_address,
sizeof(srv_address)) == SOCKET_ERROR)
                {
                    // В случае неудачи - выдать сообщение,
закрыть гнездо и выйти
                    setTextInLogListBox("Error bind: " +
WSAGetLastError().ToString());
                    closesocket(sck);
                    WSACleanup();
                    return;
                }
                printf("Bind socket succesfull\n");

                if (listen(sck, 4))
                {
                    setTextInLogListBox("Error listen: " +
WSAGetLastError().ToString());

```

```

        closesocket(sck);
        WSACleanup();
        return;
    }

    //
    acceptThread = gcnew Thread(gcnew
ThreadStart(this, &ServerForm::AcceptProc));
    acceptThread->Start();

    changeStartStopServerButton("Stop server");

    setTextInLogListBox("Server start working");

    serverEnabled = true;

    SendToClientsButton->Enabled = true;
    LimATextBox->Enabled = true;
    LimBTextBox->Enabled = true;
    loadTextBox->Enabled = true;

}
else
{
    setTextInLogListBox("Server off");
    changeStartStopServerButton("Start server");

    serverEnabled = false;
    SendToClientsButton->Enabled = false;
    LimATextBox->Enabled = false;
    LimBTextBox->Enabled = false;
    loadTextBox->Enabled = false;

    closesocket(sck);
    WSACleanup();
}
}
catch (System::Exception^ ex)
{
    setTextInLogListBox("Error: " + ex->Message);
    return;
}

}

private: System::Void AcceptProc()
{
    sockaddr_in client_address;
    SOCKET client_socket;
    int client_addr_size;

    SOCKET socket = (SOCKET)sck;
    client_addr_size = sizeof(client_address);

```

```

        // Цикл обработки входящих соединений
        while ((client_socket = accept(socket, (sockaddr
*)&client_address, &client_addr_size)) && client_socket !=
SOCKET_ERROR)
        {

            client_info currentClient;

            currentClient.s = client_socket;
            currentClient.is_free = true;

            // Для каждого соединения (клиента) создаем свою
НИТЬ

            clientThread = gcnew Thread(gcnew
ParameterizedThreadStart(this, &ServerForm::ClientProc));
            clientThread->Start(currentClient);

            char *some_addr;
            some_addr = inet_ntoa(client_address.sin_addr);
            std::string str = std::string(some_addr);
            System::String^ strS = gcnew
System::String(str.c_str());

            setTextInLogListBox("Connected client IP: " + strS +
" Port: " + Convert::ToString(ntohs(client_address.sin_port)));
        }
        return;
    }

private: System::Void ClientProc(Object^ data)
{
    client__info current_client;

    client_info currentClient = client_info(data);

    current_client.socket = currentClient.s;
    current_client.is_free = currentClient.is_free;

    Client_Proc(&current_client);
}

public: DWORD WINAPI Client_Proc(client__info *data)
{
    SOCKET client_socket;
    sockaddr_in client_addr;
    int bytes_recv;

    client__info *info = (client__info*)data;

```

```

        client_socket = info->socket;

        char buff[1024];

        clients_list.push_back(*info);

        std::list<client__info>::iterator this_client =
clients_list.end();
        this_client--;

        setTextInLogListBox("Clients number: " +
clients_list.size().ToString());

        // Ждем ответов от клиента по решению
        while (1)
        {
            bytes_recv = recv(client_socket, &buff[0],
sizeof(buff), 0);

            if (bytes_recv == SOCKET_ERROR)
            {
                break;
            }

            EnterCriticalSection(&critsect_for_task_list);

            result__from__client* result_data =
(result__from__client*)&buff[0];
            results_list.push_back(*result_data);

            remove_id = result_data->index;
            tasks_list.remove_if(rem_func);

            this_client->is_free = true;

            changeResultTextBox("Task done: " +
(Math::Round((100 * (((double)TASKSNUM -
(double)tasks_list.size()) / (double)TASKSNUM))))).ToString() + "
percents" + " ( " + (double)((result_data->execute_time *
tasks_list.size()) / clients_list.size()) / 1000).ToString() + "
sec for complete)");

            LeaveCriticalSection(&critsect_for_task_list);
        }

        // Если вышли сюда - значит клиент разорвал соединение

        EnterCriticalSection(&critsect_for_task_list);

        remove_id = this_client->execute_task_index;

```

```

        for (std::list<task__info>::iterator iter =
tasks_list.begin(); iter != tasks_list.end(); iter++)
        {
            if (iter->task_id == remove_id)
            {
                iter->status = task__status::active;
            }
        }

        LeaveCriticalSection(&critsect_for_task_list);

        EnterCriticalSection(&critsect_for_clients_list);

        clients_list.erase(this_client);

        LeaveCriticalSection(&critsect_for_clients_list);

        setTextInLogListBox("Client disconnected. Client number:
" + clients_list.size().ToString());

        closesocket(client_socket);
        return 0;
    }

    std::list<client__info>::iterator
Found_Free_Client()
    {
        while (1)
        {

            EnterCriticalSection(&critsect_for_clients_list);

                for (std::list<client__info>::iterator
iter = clients_list.begin(); iter != clients_list.end(); iter++)
                {
                    if (iter->is_free == true)
                    {

                        LeaveCriticalSection(&critsect_for_clients_list);
                        return iter;
                    }
                }

            LeaveCriticalSection(&critsect_for_clients_list);
            Sleep(20);
        }
    }

    bool Is_All_Tasks_Completed()
    {
        if (tasks_list.size() > 0)
        {

```

```

        return false;
    }
    else return true;
}

private: System::Void
SendToClientsButton_Click(System::Object^ sender,
System::EventArgs^ e)
{
    // Проверки
    if (clients_list.size() == 0)
    {
        System::Windows::Forms::MessageBox::Show("No
connected clients", "Error",
System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
        return;
    }

    if (!(LimATextBox->Text->Length > 0 && LimATextBox->Text-
>Length < 11) || !(LimBTextBox->Text->Length > 0 && LimBTextBox-
>Text->Length < 11))
    {
        System::Windows::Forms::MessageBox::Show("LimA or
LimB incorrect", "Error",
System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
        return;
    }

    try
    {
        limA = Convert::ToDouble(LimATextBox->Text);
        limB = Convert::ToDouble(LimBTextBox->Text);
    }
    catch (System::Exception^ ex)
    {
        System::Windows::Forms::MessageBox::Show("LimA or
LimB incorrect", "Error",
System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
        return;
    }

    if (limA > limB)
    {
        double buf;
        buf = limA;
        limA = limB;
        limB = buf;
    }

    if (limA < 0 || limB < 0)

```

```

        {
            System::Windows::Forms::MessageBox::Show("LimA and
LimB can not be less than 0", "Error",
System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
            return;
        }

        if (!(loadTextBox->Text->Length > 0 && loadTextBox->Text-
>Length < 4))
        {
            System::Windows::Forms::MessageBox::Show("Load
incorrect", "Error",
System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
            return;
        }

        try
        {
            loadPercentage = Convert::ToInt32(loadTextBox-
>Text);
        }
        catch (System::Exception^ ex)
        {
            System::Windows::Forms::MessageBox::Show("Load
incorrect", "Error",
System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
            return;
        }

        if (!(loadPercentage <= 100 && loadPercentage >= 5))
        {
            System::Windows::Forms::MessageBox::Show("Load
incorrect", "Error",
System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
            return;
        }

        // Конец проверок

        // Блокируем элементы управления, т.к. сейчас начнется
решаться задача
        SendToClientsButton->Enabled = false;
        LimATextBox->Enabled = false;
        LimBTextBox->Enabled = false;
        loadTextBox->Enabled = false;

        mainThread = gcnew Thread(gcnew ThreadStart(this,
&ServerForm::MainProc));
        mainThread->Start();

```

```

    }

private: System::Void MainProc()
{
    integral__info* integrals =
Prepare_Integral_Func(TASKSNUM, limA, limB, loadPercentage, STEP);

    // Формирование списка задач
    for (int i = 0; i < TASKSNUM; i++)
    {
        task__info task;
        task.status = task__status::active;
        task.task_id = i;
        tasks_list.push_back(task);
    }

    while (1)
    {
        if (clients_list.size() == 0)
        {
            setTextInLogListBox("No connected clients");
            System::Windows::Forms::MessageBox::Show("No
connected clients", "Error",
System::Windows::Forms::MessageBoxButtons::OK,
System::Windows::Forms::MessageBoxIcon::Error);
            return;
        }
        // Нахождение свободного (не занятого работой
клиента) из списка клиентов clients_list
        std::list<client__info>::iterator free_client =
Found_Free_Client();

        EnterCriticalSection(&critsect_for_task_list);

        if (tasks_list.size() >= 1)
        {
            // Указатель на первую задачу
            std::list<task__info>::iterator this_task =
tasks_list.begin();

            // Указатель на последнюю задачу
            std::list<task__info>::iterator end_task_list =
tasks_list.end();
            end_task_list--;

            while (this_task->status ==
task__status::execute_now && this_task != end_task_list)
            {
                this_task++;
            }
        }
    }
}

```

```

        // Когда попали на конец списка, так и не найдя
задачу для выполнения - проверяем эту задачу, если она еще ни кем
не выполняется - выполняем ее
        if (this_task == end_task_list)
        {
            if (this_task->status ==
task__status::active)
            {
                if (send(free_client->socket,
(char*)&integrals[this_task->task_id], sizeof(*integrals), 0) !=
SOCKET_ERROR)
                {
                    // Теперь статус этой задачи -
выполняется

                    free_client->is_free = false;
                    free_client->execute_task_index
= this_task->task_id;

                    this_task->status =
task__status::execute_now;
                }
            }
        }
        else
        {
            if (send(free_client->socket,
(char*)&integrals[this_task->task_id], sizeof(*integrals), 0) !=
SOCKET_ERROR)
            {
                // Теперь статус этой задачи -
выполняется

                free_client->is_free = false;
                free_client->execute_task_index =
this_task->task_id;

                this_task->status =
task__status::execute_now;
            }
        }
    }
    else
    {
        setTextInLogListBox("All tasks completed\n");
        LeaveCriticalSection(&critsect_for_task_list);
        break;
    }

    LeaveCriticalSection(&critsect_for_task_list);
}

result = 0;

```

```

        for (std::list<result__from__client>::iterator
this_result = results_list.begin(); this_result !=
results_list.end(); this_result++)
        {
            result += this_result->result;
        }

        setTextInLogListBox("Result: " + result.ToString());

        //после того, как вычисления выполнены - очищаем
результаты
        results_list.clear();
        result = 0;
    }

private: System::Void setTextInLogListBox(System::String^
text)
{
    try
    {
        if (this->LogListBox->InvokeRequired)
        {
            setTextColor^ d = gcnew
setTextCallback(this, &ServerForm::setTextInLogListBox);
            this->Invoke(d, gcnew array<
System::Object^>(1) { text });
        }
        else
        {
            this->LogListBox->Items->Add(text);
        }
    }
    catch (System::Exception^ ex)
    {
        setTextInLogListBox("Error: " + ex->Message);
        return;
    }
}

private: System::Void changeResultTextBox(System::String^
text)
{
    if (this->ResultTextBox->InvokeRequired)
    {
        setTextColor^ d = gcnew setTextColor(this,
&ServerForm::changeResultTextBox);
        this->Invoke(d, gcnew array< System::Object^>(1) {
text });
    }
    else

```

```

        {
            this->ResultTextBox->Text = text;
        }
    }

    private: System::Void
changeStartStopServerButton(System::String^ text)
    {
        if (this->StartStopServerButton->InvokeRequired)
        {
            setTextColor^ d = gcnew setTextColor(this,
&ServerForm::changeStartStopServerButton);
            this->Invoke(d, gcnew array< System::Object^>(1) {
text });
        }
        else
        {
            this->StartStopServerButton->Text = text;
        }
    }

    private: System::Void ServerForm_FormClosed(System::Object^
sender, System::Windows::Forms::FormClosedEventArgs^ e) {
        exit(1);
    }

};
}

```

Приложение A.6 Func.h

```

#pragma once
#include <iostream>
#include <stdio.h>
#include <tchar.h>
#include <WinSock2.h>
#include <WS2tcpip.h>
#include <conio.h>
#include <process.h>
#include <list>

#pragma once
#pragma comment (lib, "wsock32.lib")
#pragma comment (lib, "ws2_32.lib")

enum task__status
{
    execute_now,
    active
};

```

```

struct client__info
{
    SOCKET socket;
    bool is_free;
    int execute_task_index;
};

struct task__info
{
    int task_id;
    task__status status;
};

struct integral__info
{
    double limA;
    double limB;
    int numpoint;           //количество шагов до проверки
    double step;
    int task_index;
    int load;
};

struct result__from__client
{
    double result;
    double execute_time;
    int index;
};

WSADATA wsd;
SOCKET sck;
sockaddr_in srv_address;
unsigned short port = 5150;

// Количество шагов до проверки
#define NUMPOINT 2000

// Число подзадач
#define TASKSNUM 300

#define STEP 0.0000005

// Критическая секция для операций со списком tasks_list
CRITICAL_SECTION critsect_for_task_list;

// Критическая секция для операций со списком clients_list
CRITICAL_SECTION critsect_for_clients_list;

// Список клиентов
std::list<client__info> clients_list;

```

```

// Список подзадач (главная задача разбивается на эти подзадачи)
std::list<task__info> tasks_list;

// Список результатов, элемент списка - результат от какого-либо
клиента
std::list<result__from__client> results_list;

// Индекс задачи, которая удалится из списка задач
int remove_id;

double result = 0;

bool rem_func(const task__info& value) { return (value.task_id ==
remove_id); }

integral__info* Prepare_Integral_Func(int integral_num, int
left_border, int right_border, int load, double step_f)
{
    double stepbuf, dlina_diapazona, numpoint;

    double limA = left_border;

    double limB = right_border;
    double loadBuf = load;

    stepbuf = step_f;

    dlina_diapazona = (limB - limA) / integral_num;

    numpoint = NUMPOINT;

    integral__info* integrals = new integral__info[integral_num];

    for (int i = 0; i < integral_num; i++)
    {
        integrals[i].limA = i * dlina_diapazona + limA;
        integrals[i].limB = i * dlina_diapazona + dlina_diapazona
+ limA;
        integrals[i].numpoint = numpoint;
        integrals[i].step = stepbuf;
        integrals[i].task_index = i;
        integrals[i].load = loadBuf;
    }

    return integrals;
}

```