Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Системное программное обеспечение вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Серверная часть системы распределенного выполнения задач

БГУИР КП 1-40 02 01 01 016 ПЗ

Курсант: гр. 730501

Томашевич П.Д.

Руководитель: ассистент

кафедры ЭВМ Зайцев Ю.В

Минск 2019

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой

*––––––––––––––––––––––––*

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту    *Томашевичу Павлу Дмитриевичу –––––––––––––––––*

1. Тема проекта   *Серверная часть системы распределенного выполнения задач.* –

2. Срок сдачи студентом законченного проекта–––––*13. июня 2019 г.–––*

3. Исходные данные к проекту *алгоритм работы системы распределенного выполнения задач; среда разработки – Visual Studio 2017, язык программирования – C#, библиотека – MVVMLight.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение. 1. Постановка задачи. 2. Структура входных и выходных данных. 3. Описание алгоритмов. 4.Разработка программного обеспечения. 5. Результаты работы программы. 6. Руководство пользователя. Заключение. Литература. Приложения.*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. Диаграмма классов 2. Схема алгоритмов (2) шт – чертёж*

6. Консультант по проекту (с обозначением разделов проекта)   *Зайцев Ю. В.*

7. Дата выдачи задания –––––*15 февраля 2019 г.–––––––––––––––––––––––*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*раздел 1 к 10.03 – 15 %;–––––––––––––––––––––– –––––––––––––––––*

*раздел 2 к 01.04 – 15 %;–––––––––––––––––––––– –––––––––––––––––––*

*разделы 3,4 к 20.05 – 45 %;––––––––––––––––––––– –––––––––––––––––––*

*разделы 5, 6 к 01.06 – 10 %;––––––––––––––––––––– –––––––––––––––––––*

*оформление пояснительной записки и графического материала к 04.06 – 15 %*

*Защита курсового проекта с 10 по 15 июня––––––––––––––––––––––––––*

РУКОВОДИТЕЛЬ*––––––––– Ю.В. Зайцев*

(подпись)

Задание принял к исполнению *–––––– –\_\_ \_\_\_\_\_\_\_\_– П. Д. Томашевич*

(дата и подпись студента)

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

1. ОСНОВАНАЯ ЧАСТЬ 7

2. СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ 14

3. ОПИСАНИЕ АЛГОРИТМОВ 14

4. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 16

5. РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ 17

6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 18

ЗАКЛЮЧЕНИЕ 23

ЛИТЕРАТУРА 24

ПРИЛОЖЕНИЕ А 25

ПРИЛОЖЕНИЕ Б 26

ПРИЛОЖЕНИЕ В 27

ПРИЛОЖЕНИЕ Г 30

ПРИЛОЖЕНИЕ Д 39

**ВВЕДЕНИЕ**

На сегодняшний день учёные нуждаются в огромных вычислительных мощностях для решения широкого спектра задач, находящихся на фронте современной науки. Исследование свойств белка, исследование результатов работы адронного коллайдера, прогнозирование изменения климата, решение математических проблем, а также эффективный поиск лекарств, гравитационных волн и внеземного разума проводятся сегодня с использованием суперкомпьютеров. Производительность таких машин огромна. В качестве примера можно привести мощнейший на сегодня суперкомпьютер Tianhe-2, работающий в Национальном институте оборонных технологий в Китае.

Достигнута пиковая производительность Tianhe-2 – 33,86 петафлопс, что почти в 2 раза больше, чем у Titan, занимающего второе место. Время использования подобных машин строго ограничено и расписано на многие месяцы вперед, так как мощностями таких машин пользуется, как правило, несколько учреждений. Эти машины представляют собой большое количество мощных процессоров, соединенных локально для организации параллельных вычислений.

В настоящее время персональные компьютеры имеются почти у каждого человека, а нынешний уровень технологий в области распараллеливания задач таков, что позволяет использовать CPU, процессоры видеокарт, игровых приставок, и другие устройства для выполнения вычислений аналогично суперкомпьютерам.

В данном случае это не фиксированная система серверных компьютеров или объединенных в сеть процессоров, а легко масштабируемая система из большого количества компьютеров по всему миру. Такая система называется грид (англ. Grid – решётка, сеть). В основе ее лежат грид-вычисления — это форма [распределённых вычислений](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), в которой «виртуальный [суперкомпьютер](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80)» представлен в виде [кластеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2)), соединённых с помощью сети, слабосвязанных [гетерогенных](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D1%82%D0%B5%D1%80%D0%BE%D0%B3%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) компьютеров, работающих вместе для выполнения огромного количества заданий (операций, работ). Эта технология применяется для решения научных, математических задач, требующих значительных вычислительных ресурсов. Грид-вычисления используются также в коммерческой инфраструктуре для решения таких трудоёмких задач, как экономическое прогнозирование, сейсмоанализ, разработка и изучение свойств новых лекарств.

**1. ОСНОВНАЯ ЧАСТЬ**

**1.1 Краткие сведения о предметной области**

Грид с точки зрения сетевой организации представляет собой согласованную, открытую и стандартизованную [среду](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0), которая обеспечивает гибкое, безопасное, скоординированное разделение [вычислительных ресурсов](https://ru.wikipedia.org/w/index.php?title=%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81&action=edit&redlink=1) и ресурсов хранения информации, которые являются частью этой среды, в рамках одной виртуальной организации.

Грид-вычисления можно организовать на базе множества устаревших моделей [персональных компьютеров](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) объединённых в иерархическую [локальную вычислительную сеть](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C) Ethernet с присутствием [серверов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80). Эта сеть может иметь соединение с [интернетом](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82).

Концепция грид-систем проявляется в географически распределённой инфраструктурой, объединяющей множество ресурсов разных типов (процессоры, долговременная и оперативная память, хранилища и базы данных, сети), доступ к которым пользователь может получить из любой точки, независимо от места их расположения.

Идея грид-компьютинга возникла вместе с распространением персональных компьютеров, развитием [интернета](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82)и технологий пакетной передачи данных на основе оптического волокна ([SONET](https://ru.wikipedia.org/wiki/SONET), [SDH](https://ru.wikipedia.org/wiki/SDH) и [ATM](https://ru.wikipedia.org/wiki/ATM)), а также технологий [локальных сетей](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C) ([Gigabit Ethernet](https://ru.wikipedia.org/wiki/Ethernet)). Полоса пропускания коммуникационных средств стала достаточной, чтобы при необходимости привлечь ресурсы другого компьютера. Учитывая, что множество подключенных к глобальной сети компьютеров большую часть рабочего времени простаивает и располагает большими ресурсами, чем необходимо для решения их повседневных задач, возникает возможность применить их неиспользуемые ресурсы в другом месте.

термин «грид-вычисления» появился в начале [1990-х годов](https://ru.wikipedia.org/wiki/1990-%D0%B5), как метафора, демонстрирующая возможность простого доступа к вычислительным ресурсам как и к электрической сети ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) power grid) в сборнике под редакцией Яна Фостера и Карла Кессельмана «The Grid: Blueprint for a new computing infrastructure».

Использование свободного времени процессоров и добровольного компьютинга стало популярным в конце 1990-х годов после запуска проектов [добровольных вычислений](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%B1%D1%80%D0%BE%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) [GIMPS](https://ru.wikipedia.org/wiki/GIMPS) в [1996 году](https://ru.wikipedia.org/wiki/1996_%D0%B3%D0%BE%D0%B4), [distributed.net](https://ru.wikipedia.org/wiki/Distributed.net) в [1997 году](https://ru.wikipedia.org/wiki/1997_%D0%B3%D0%BE%D0%B4) и [SETI@home](https://ru.wikipedia.org/wiki/SETI@home) в [1999 году](https://ru.wikipedia.org/wiki/1999_%D0%B3%D0%BE%D0%B4). Эти первые проекты добровольного компьютинга использовали мощности подсоединённых к сети компьютеров обычных пользователей для решения исследовательских задач, требующих больших вычислительных мощностей.

Идеи грид-системы (включая идеи из областей [распределённых вычислений](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), [объектно-ориентированного программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), использования [компьютерных кластеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2)), [веб-сервисов](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) и др.) были собраны и объединены [Иэном Фостером](https://ru.wikipedia.org/w/index.php?title=%D0%98%D1%8D%D0%BD_%D0%A4%D0%BE%D1%81%D1%82%D0%B5%D1%80&action=edit&redlink=1)[[en]](https://en.wikipedia.org/wiki/Ian_Foster), [Карлом Кессельманом](https://ru.wikipedia.org/w/index.php?title=%D0%9A%D0%B0%D1%80%D0%BB_%D0%9A%D0%B5%D1%81%D1%81%D0%B5%D0%BB%D1%8C%D0%BC%D0%B0%D0%BD&action=edit&redlink=1)[[en]](https://en.wikipedia.org/wiki/Carl_Kesselman) и [Стивом Тики](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%B8%D0%B2_%D0%A2%D0%B8%D0%BA%D0%B8&action=edit&redlink=1), которых часто называют отцами грид-технологии. Они начали создание набора инструментов для грид-компьютинга [Globus Toolkit](https://ru.wikipedia.org/w/index.php?title=Globus_Toolkit&action=edit&redlink=1)[[en]](https://en.wikipedia.org/wiki/Globus_Toolkit), который включает не только инструменты менеджмента вычислений, но и инструменты управления ресурсами хранения данных, обеспечения безопасности доступа к данным и к самому гриду, мониторинга использования и передвижения данных, а также инструментарий для разработки дополнительных грид-сервисов. В настоящее время этот набор инструментария является де факто стандартом для построения инфраструктуры на базе технологии грид, хотя на рынке существует множество других инструментариев для грид-систем как в масштабе предприятия, так и в глобальном.

Грид-технология применяется для моделирования и обработки данных в экспериментах на [Большом адронном коллайдере](https://ru.wikipedia.org/wiki/%D0%91%D0%BE%D0%BB%D1%8C%D1%88%D0%BE%D0%B9_%D0%B0%D0%B4%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BB%D0%BB%D0%B0%D0%B9%D0%B4%D0%B5%D1%80) (грид используется и в других задачах с интенсивными вычислениями). На платформе [BOINC](https://ru.wikipedia.org/wiki/BOINC) в настоящее время ведутся активные вычисления более 60 проектов. Например, проект Fusion (юг Франции, разработка метода получения электричества с помощью термоядерного синтеза на экспериментальном реакторе [ITER](https://ru.wikipedia.org/wiki/ITER)) также использует грид ([EDGeS@Home](https://ru.wikipedia.org/wiki/EDGeS@Home)). Под названием CLOUD начат проект коммерциализации грид-технологий, в рамках которого небольшие компании, институты, нуждающиеся в вычислительных ресурсах, но не могущие себе позволить по тем или иным причинам иметь свой суперкомпьютерный центр, могут покупать вычислительное время грида

В настоящее время выделяют три основных типа грид-систем:

Добровольные гриды — гриды на основе использования добровольно предоставляемого свободного ресурса персональных компьютеров;

Научные гриды — хорошо распараллеливаемые приложения программируются специальным образом (например, с использованием Globus Toolkit);

Гриды на основе выделения вычислительных ресурсов по требованию (коммерческий грид, [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) enterprise grid) — обычные коммерческие приложения работают на виртуальном компьютере, который, в свою очередь, состоит из нескольких физических компьютеров, объединённых с помощью грид-технологий.

**1.2 Примеры существующих грид-систем**

Существующие географически-разделенные грид-системы представлены рядом интереснейших проектов, в которых участвуют добровольцы со всего мира. Одной из платформ для таких проектов является BOINC (англ. Berkeley Open Infrastructure for Network Computing – открытая программная платформа (университета) Беркли для GRID вычислений). BOINC – это некоммерческий программный комплекс для организации распределенных вычислений, первоначально разработанный для проекта SETI@home (от англ. Search for Extra-Terrestrial Intelligence at Home – поиск внеземного разума на дому). Впоследствии платформа стала доступной для сторонних проектов.

Серверная часть BOINC состоит из HTTP-сервера с веб-сайтом проекта, базы данных MySQL и набора демонов (генератор заданий, планировщик, валидатор, ассимилятор результатов).

HTTP сервер представляет собой набор PHP-скриптов и необходим организаторам проектов для общего управления проектом: регистрация участников, распределение заданий для обработки, получение результатов, управление базами данных проекта. В базе данных хранятся пользователи, пароли, записи заданий, результатов, информация о хостах, программах проекта и прочее.

Для пользователей понятие BOINC чаще используется в контексте понятия BOINC-клиент — универсальный клиент для работы с различными (BOINC-совместимыми) проектами распределённых вычислений.  BOINC-клиент  позволяет  участвовать  одновременно  в  нескольких  проектах  с помощью одной общей программы управления (boinc или boinc.exe). Для визуализации процесса управления BOINC- клиентом можно использовать либо поставляемую по умолчанию официальную программу-менеджер (boincmgr или boincmgr.exe), либо воспользоваться «неофициальной» программой для мониторинга и управления BOINC-клиентом.

Следует отметить, что собственно BOINC-клиент в академическом понимании не имеет пользовательского интерфейса как такового, а представляет собой сервис, запускаемый при запуске системы, и управляется по протоколу TCP/IP. Однако для конечного пользователя это не имеет значения, поскольку дистрибутив программы комплектуется программой-менеджером, которая сразу по умолчанию устанавливается вместе с BOINC-клиентом как единое целое и абсолютно прозрачна для пользователя. В этом случае в качестве адреса управляемого программой менеджером BOINC-клиента указывается  адрес «localhost».

Таким образом, с одной стороны, ничто не мешает пользователю использовать альтернативную программу-менеджер для управления BOINC-клиентом, а с другой стороны, даёт возможность управлять несколькими BOINC-клиентами, находящимися на разных компьютерах из одной программы-менеджера. Такая организация управления BOINC-клиентом подразумевает возможность использовать BOINC-клиент в «невидимом» режиме, когда запускается исключительно сервис, без пользовательского интерфейса вообще.

POEM@Home – изучение белковых структур.

В проектах, базирующихся на BOINC, задействовано более миллиона машин. Общая производительность составляет порядка 8-10 петафлопс. В проектах BOINC участвуют команды волонтёров со всего света, в том числе и из России, однако в Казахстане и России развитие собственных грид-систем находится на невысоком уровне.  Россия  активно  использует  суперкомпьютеры,  Казахстан  начинает их использование (Казахский национальный технический университет и Казахстанско-Британский технический университет пользуются суперкомпьютерами невысокой производительности [8]), однако грид-системы пока не получили большого распространения.

На данный момент, в разработке находятся стандарты для Российских грид-систем, разрабатываемые с учётом интеграции в международную среду. Возможно, в ближайшем будущем в ходе освоения грид- систем в России и Казахстане станет возможным использование дешёвых высокопроизводительных систем в вузах и научных центрах для организации собственных масштабных проектов.

**1.3 Реализация программной части грид-системы**

Для реализации программной части грид-системы был выбран язык программирования C# в фрэймворке .NET 4.7.1 . C# — [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Разработан в [1998](https://ru.wikipedia.org/wiki/1998)—[2001 годах](https://ru.wikipedia.org/wiki/2001_%D0%B3%D0%BE%D0%B4) группой инженеров компании [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) под руководством [Андерса Хейлсберга](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D0%B9%D0%BB%D1%81%D0%B1%D0%B5%D1%80%D0%B3,_%D0%90%D0%BD%D0%B4%D0%B5%D1%80%D1%81) и Скотта Вильтаумота как язык разработки приложений для платформы [Microsoft .NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework). Впоследствии был стандартизирован как [ECMA](https://ru.wikipedia.org/wiki/ECMA)-334 и [ISO](https://ru.wikipedia.org/wiki/ISO)/[IEC](https://ru.wikipedia.org/wiki/IEC)23270.

C# относится к семье языков с [C-подобным синтаксисом](https://ru.wikipedia.org/wiki/C-%D0%BF%D0%BE%D0%B4%D0%BE%D0%B1%D0%BD%D1%8B%D0%B9_%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81), из них его синтаксис наиболее близок к [C++](https://ru.wikipedia.org/wiki/C%2B%2B) и [Java](https://ru.wikipedia.org/wiki/Java). Язык имеет [статическую типизацию](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), поддерживает [полиморфизм](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), [перегрузку операторов](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B0_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BE%D0%B2) (в том числе операторов явного и неявного приведения типа), [делегаты](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%B3%D0%B0%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), атрибуты, [события](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [свойства](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [обобщённые](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) типы и методы, [итераторы](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80), [анонимные функции](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%BE%D0%BD%D0%B8%D0%BC%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) с поддержкой [замыканий](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [LINQ](https://ru.wikipedia.org/wiki/Language_Integrated_Query), [исключения](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), [комментарии](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%80%D0%B8%D0%B8_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) в формате [XML](https://ru.wikipedia.org/wiki/XML).

Переняв многое от своих предшественников — языков [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Pascal](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Модула](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B0-2), [Smalltalk](https://ru.wikipedia.org/wiki/Smalltalk) и, в особенности, [Java](https://ru.wikipedia.org/wiki/Java) — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ и некоторых других языков, не поддерживает [множественное наследование](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) классов (между тем допускается [множественное наследование интерфейсов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)#%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BD%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B8_%D1%80%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%8)).

C# разрабатывался как язык программирования прикладного уровня для [CLR](https://ru.wikipedia.org/wiki/Common_Language_Runtime) и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает [BCL](https://ru.wikipedia.org/wiki/Base_Class_Library). Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем (однако, эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET). CLR предоставляет C#, как и всем другим [.NET](https://ru.wikipedia.org/wiki/.NET_Framework)-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, [сборка мусора](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0) не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на [VB.NET](https://ru.wikipedia.org/wiki/Visual_Basic_.NET), [J#](https://ru.wikipedia.org/wiki/Visual_J%E2%99%AF) и др.

Передача данных между вычислительными единицами грид-системы происходит на базе программного интерфейса сокет. Со́кет ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) socket — разъём) — название [программного интерфейса](https://ru.wikipedia.org/wiki/API) для обеспечения обмена данными между [процессами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)). Процессы при таком обмене могут исполняться как на одной [ЭВМ](https://ru.wikipedia.org/wiki/%D0%AD%D0%92%D0%9C), так и на различных ЭВМ, связанных между собой [сетью](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C). Сокет — [абстрактный](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F) объект, представляющий конечную точку соединения.

Для простой работы с кодом использовалась интегрированная среда разработки IDE Microsoft Visual Studio 2017.

**2. СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ**

Входными данным для программного обеспечения персонального компьютера является интерфейс управления приложением, который включает в себя выбор данных для обработки, всю остальную логику ПО выполняет автоматически, на основе алгоритмов поиска удаленных машин, способных обработать выбранные пользователем данные.

Выходными данными программного обеспечения персонального компьютера являются IP-адрес и порт, с которыми будет работать ПО, уже обработанные данные, сохраняющиеся на внутреннем либо внешнем накопителе (в зависимости, куда пользователь указал сохранять данные); log-файл, в который записываются все события произошедшие с GRID-системой в хронологическом порядке с временной меткой; состояние ПО.

**3. ОПИСАНИЕ АЛГОРИТМОВ**

GRID-система является ПО состоящим из двух модулей: серверного и клиентского. Пользователь, у которого установлен только клиентский модуль не может использовать возможности распределенных вычислений, он просто предоставляет доступ к вычислительным ресурсам своего персонального компьютера.

В свою очередь обладатели серверного модуля имеют возможность использовать вычислительные возможности удаленных персональных компьютеров.

Алгоритм работы серверного модуля:

1. При запуске GRID-система получает IP-адрес от провайдера, поставляющий услуги выхода в интернет, либо локальный IP-адрес, при работе по локальной сети и свободный порт.
2. Инициализирует все поля необходимые поля статического класса GRIDCore.
3. Созданный объект класса SOCKET начинает прослушивание выделенного порта на входящие подключения (асинхронно).
4. Пользователь вводит директорию, куда сохранять обработанные файлы и какие файлы нужно обработать.
5. При наличии подключенных удаленных машин, ПО опрашивает каждую подключенную машину на загруженность ее, по полученным данным выбирает наименее загруженную машину.
6. Отправляет ей данные для обработки, получает обработанные данные и сохраняет их на внутреннем (внешнем) носителе информации.

UML— [язык](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [графического](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) описания для [объектного моделирования](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) в области [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), [моделирования бизнес-процессов](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1), [системного проектирования](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и отображения [организационных структур](https://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0).

UML позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий (таких как класс, компонент, обобщение, агрегация и поведение) и больше сконцентрироваться на проектировании и архитектуре.

UML использует графические обозначения для создания [абстрактной модели](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) [системы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем.

Разработанная диаграмма классов находится в Приложении А.

Описание алгоритмов(блок-схемы) проверки соединения с подключенными удаленными персональными компьютерами public static void GridCore::CheckConnection() и поиска наименее загруженных машин public static int GridCore::GetLightyLoadedMachine(out Socket socket) в приложении Б.

Описание алгоритмов по шагам находятся в Приложении В.

**4. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Для разработки программного обеспечения для персонального компьютера использовалась среда разработки Microsoft Visual Studio 2017 (рисунок 4.1).

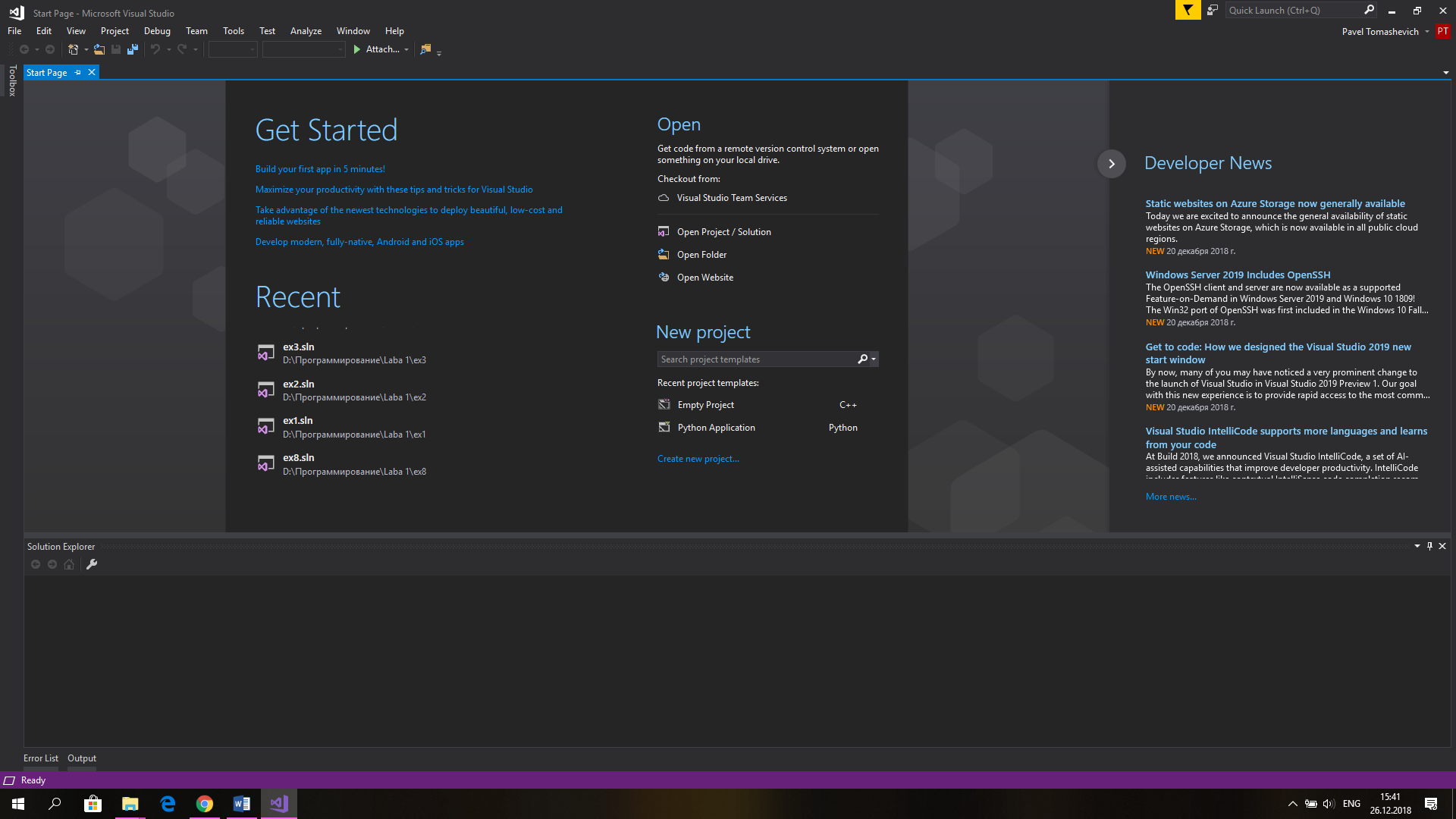


Рисунок 4.1 – Среда разработки Visual Studio 2017

Исходный код написан на C# для платформы .NET Framework 4.7, с использованием библиотеки MVVMLight.

До начала разработки ПО нужно было выбрать: каким образом будут передаваться данные между машинами. Использование непосредственно низкоуровневый TCP-протокол затруднил бы разработку ПО и отвлек от реализации логики программы. По этой причине было решено использование более высокой абстракции – сокеты.

Для большей производительности некоторое время затратные алгоритмы обрабатываются в отдельных потоках, также во вторичном потоке происходит прослушивание порта для подключения удаленных машин, что позволяет без длительных временных задержек принимать подключения.

Но использование нескольких потоков приводит к проблемам изменения состояния пользовательского интерфейса. Решением этого стало использование паттерна MVVM (Model-View-ViewModel). Он предполагает реализацию пользовательского интерфейса независимо от логики самой программы и модели, как прослойки между интерфейсом и логикой программы, связывающей эти компоненты.

Это возможно реализовать за счет возможностей языка C#, таких как события, и системы для построения клиентских приложений Windows – WPF, которая поддерживает возможность привязать элементы управления к свойствам класса.

Этот паттерн решил проблемы реализации многопоточного приложения с графическим интерфейсом.

Листинг программного обеспечения для персонального компьютера находится в Приложении Г.

**5. РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ**

Результатом работы программы являются обработанные файлы, сохраненные в указанной пользователем директории; запись информации о состоянии GRID-системы.

**6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

При включении ПО пользователь увидит главное окно с элементами управления Grid System и информацию о ПО (рис. 1).

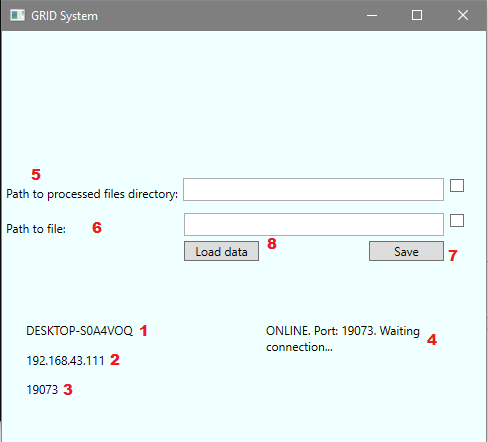


Рисунок 1. Главное окно ПО.

Элементы управления:

1. Имя персонального компьютера.
2. IP-адрес, по которому должны подключаться удаленные персональные компьютеры.
3. Порт, по которому происходит прослушивание входящих подключений.
4. Статус Grid-системы.
5. Путь, к директории, куда сохранять обработанные файлы.
6. Путь к файлу, который нужно обработать.
7. Кнопка сохранения введенных значений.
8. Начать обработку файла на удаленном ПК.

При попытке сохранить неверный путь к директории или файлу (рис. 2) (ввести запрещенные символы (рис. 3)) появится предупреждение.

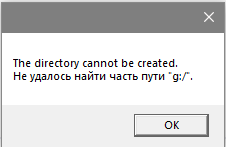


Рис. 2 Предупреждение о неверном пути.

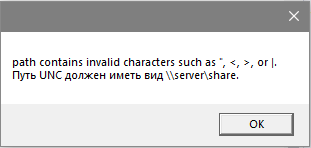


Рис. 3 Предупреждение о использовании запрещенных символов.

Если файл, для обработки не существует, пользователь увидит предупреждение (рис. 4).

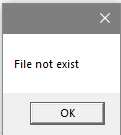


Рис. 4 Предупреждение о отсутствии запрошенного файла.

При попытке выполнить обработать файл без корректно веденных путях появится предупреждение (рис. 5).

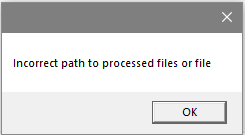


Рис. 5 Предупреждение о некорректном пути.

Если все требования ввода директорий и пути к файлу, для обработки соблюдены пользователь увидит галочки напротив путей, которые сигнализируют о успешном сохранении введенных путей (рис. 6)

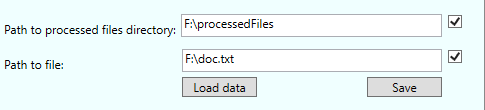


Рис. 6 Оповещение о успешном сохранении путей.

При попытке запросить обработать файл без подключенных удаленных машин, появится предупреждение (рис. 7).

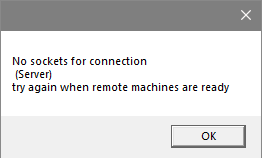


Рис. 7 Предупреждение о отсутствии подключенных сокетов.

Если удаленный хост разорвет соединение, то система предупредит пользователя перед попыткой обработки данных (рис. 8).

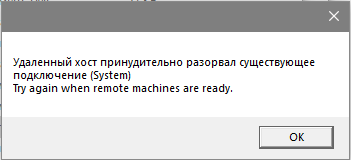


Рис. 8 Предупреждение, что удаленный хост разорвал соединение.

ПО в директории, в которой оно установлено создает лог-файл. В нем записывается важная информация о состоянии GRID-системы с хронологическими метками (рис. 9)

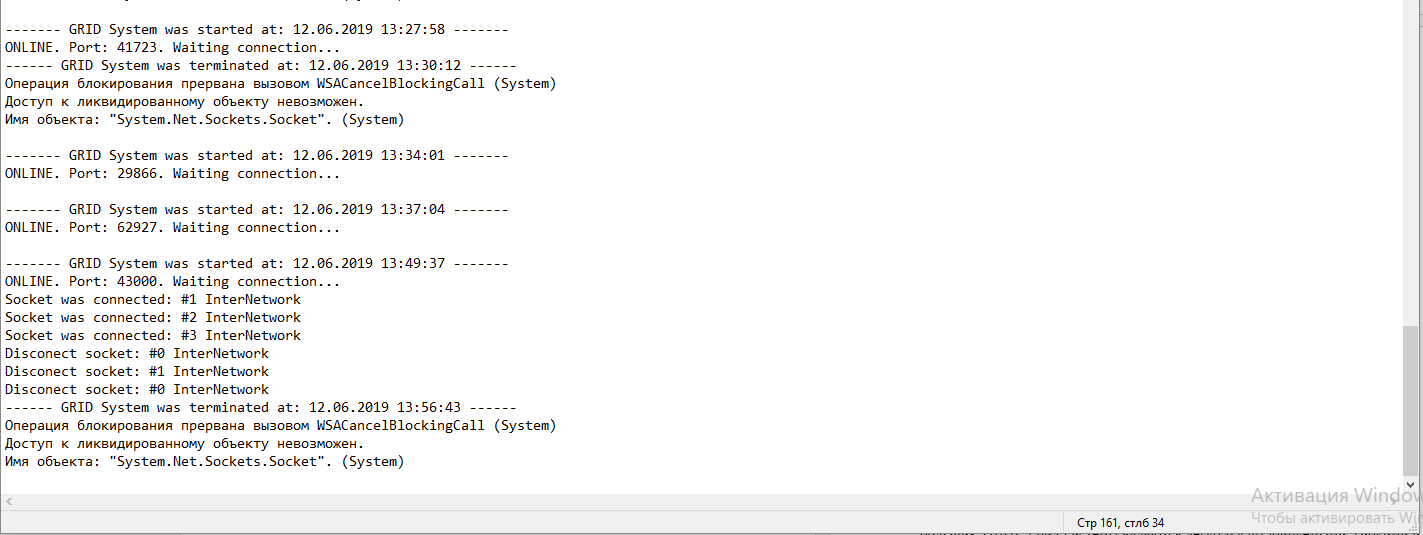


Рис. 9 Лог-файл.

В указанную директорию будут сохранятся обработанные файлы (рис. 10).

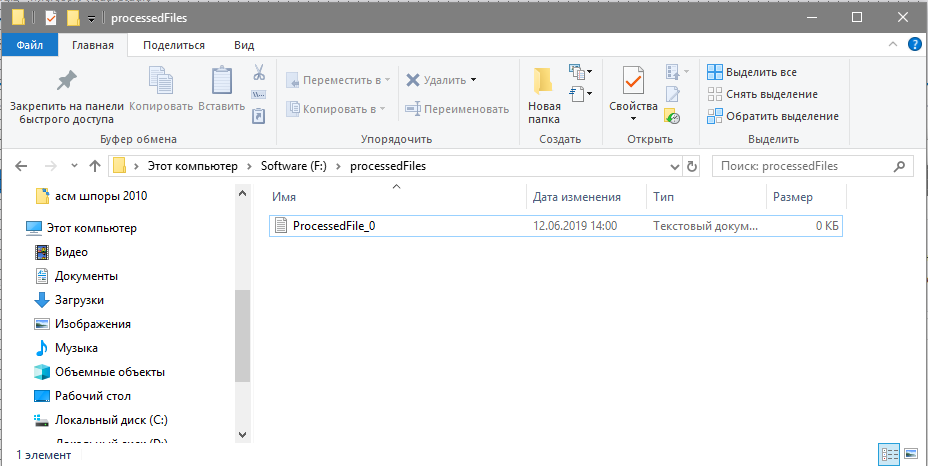


Рис. 10 Директория с обработанными файлами.

**ЗАКЛЮЧЕНИЕ**

Грид-системы являются отличным инструментом для решения задач, требующих высоких вычислительных  мощностей,  не  требуя  при этом больших затрат. Грид-системы являются легко масштабируемыми, гибкими и имеют возможность быстрой интеграции беспроводных устройств, что говорит о значительном потенциале их использования, а с возможностью привлечения волонтёров это открывает большой простор для научного использования грид-систем.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Грид-системы и потенциал их использования [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://articlekz.com/article/13129>

[2] Грид [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs.pdf>

[3] MSDN – сеть разработчиков Microsoft [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://msdn.microsoft.com/ru-ru>

[4] Джонсон М. Харт Системное программирование в среде Windows ­– 3-e изд. – 514 с.

[5] Джеффри Рихтер Windows для профессионалов. Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows – 4-e изд. – Питер : Русская редакция, 2008 – 720 с.

**ПРИЛОЖЕНИЕ А**

*(обязательное)*

Диаграмма классов

**ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Описание алгоритмов (блок- схемы)

**ПРИЛОЖЕНИЕ В**

*(обязательное)*

Описание алгоритмов (по шагам)

Алгоритм проверки соединения с подключенными удаленными персональными компьютерами public static void GridCore::CheckConnection()

Шаг 0. Начало.

Шаг 1. Если количество подключенных сокетов равно 0 перейти на шаг 11, иначе – на шаг 2.

Шаг 2. Объявление и определение переменной int i.

Шаг 3. Если i < количества элементов в контейнере \_socket(кол-во подключенных компьютеров) перейти на шаг 4, иначе – на шаг 11.

Шаг 4. Если метод Poll вернул true и значение байт информации доступных для чтения равно 0 или значение поля Connected равно fasle перейти на шаг 5, иначе – на шаг 9.

Шаг 5. Присвоение полю \_hostStatus (свойству HostStatusGc) строки сообщающей о отключении удаленного ПК.

Шаг 6. Вызов метода Shutdown(SocketShutdown.Both) для i-го элемента контейнера \_sockets, который закрывает соединение с отключенным сокетом удаленной машины.

Шаг 7. Вызов метода Close() для i-го элемента контейнера \_sockets, который завершает выполнение работы серверного сокета, соединенного с сокетом удаленной машины.

Шаг 8. Удаление i-го элемента из массива \_sockets.

Шаг 9. Увеличение i на 1

Шаг 10. Переход на шаг 3.

Шаг 11. Конец.

Алгоритм поиска наименее загруженных машин public static int GridCore::GetLightyLoadedMachine(out Socket socket)

Шаг 1. Вызов метода public static void GridCore::CheckConnection().

Шаг 2. Если количество подключенных сокетов равно 0, то создается исключение Exception("No sockets for connection\n"), иначе – на шаг 3;

Шаг 3. Объявление и определение переменной int workload = 100.

Шаг 4. Объявление и определение переменной int index = -1.

Шаг 5. Объявление и определение переменной int bytesRec = 0.

Шаг 6. Объявление и определение массива byte[] msg = Encoding. Default.GetBytes(\_WorkLoad).

Шаг 7. Объявление массива byte[] answer = new byte[56];

Шаг 8. Объявление и определение переменной int i = 0.

Шаг 9. Если i < количество подключенных сокетов, то перейти на шаг 10, иначе – на шаг 17.

Шаг 10. Отправляем i-тому сокету массив байтов msg.

Шаг 11. Принимаем набор байтов от i-го подключенного сокета и записываем их в массив байтов answer.

Шаг 12. При помощи метода Int32.Parse(Encoding.Default.GetString (answer, 0, bytesRec)) преобразуем массив байтов в число.

Шаг 13. Если полученное число <= workload, то перейти на шаг 14, иначе – на шаг 16.

Шаг 14. Присваиваем index = i;

Шаг 15. Присваиваем workload значение полученное от удаленного компьютера.

Шаг 16. Увеличить i на 1.

Шаг 17. Перейти на шаг 9.

Шаг 17. Если index равен -1, перейти на шаг 18, иначе – на шаг 20.

Шаг 18. Аргументу метода socket, переданному по ссылке присваивается нулевой элемент контейнера \_sockets.

Шаг 19. Перейти на шаг 21.

Шаг 20. Аргументу метода socket, переданному по ссылке присваивается элемент контейнера \_sockets[index].

Шаг 21. Возврат управления из метода public static int GetLightyLoadedMachine(out Socket socket) и значение переменной index.

Шаг 22. Конец.

**ПРИЛОЖЕНИЕ Г**

*(обязательное)*

Листинг программы для персонального компьютера

Статический класс GridCore.

using System;

using System.Collections.Generic;

using System.Net.Sockets;

using System.Net;

using System.Text;

using System.IO;

namespace Server

{

public static class GRIDCore

{

// initialize IP host, IP address, end point & create socket-listner & bind it

// \_ipHost = Dns.GetHostEntry("localhost");

public static void Initialize()

{

while (true)

{

try

{

HostStatusGc = Environment.NewLine + "------- GRID System was started at: " +

DateTime.Now + " -------";

Random random = new Random();

\_hostPort = random.Next(1, 65000);

\_ipHost = Dns.GetHostByName(Dns.GetHostName());

\_ipAddr = \_ipHost.AddressList[0];

\_ipEndPoint = new IPEndPoint(\_ipAddr, \_hostPort);

\_sockListner = new Socket(\_ipAddr.AddressFamily, SocketType.Stream,

ProtocolType.Tcp);

\_sockListner.Bind(\_ipEndPoint); // связываем прослушивающий сокет с конечной точкой

if (\_ipHost.AddressList[0].ToString() == "127.0.0.1")

HostStatusGc = "OFFLINE. Check internet connection...";

else

HostStatusGc = ($"ONLINE. Port: {\_ipEndPoint.Port}. Waiting connection...");

return;

}

catch (SocketException ex)

{

HostStatusGc = ex.Message + " " + "(" + ex.Source + ")";

continue;

}

catch(Exception ex)

{

HostStatusGc = ex.Message + " " + "(" + ex.Source + ")";

throw new Exception(ex.Message + " " + "(" + ex.Source + ")");

}

}

}

// this method must be run in separate thread & you need call initialize() before

public static void WaitSocket()

{

while (true)

{

try

{

\_sockListner.Listen(10);

while (true)

{

\_sockets.Add(\_sockListner.Accept());

//\_sockets[\_sockets.Count - 1].ReceiveTimeout = 5000;

HostStatusGc = ($"Socket was connected: #" + $"{\_sockets.Count} " +

$"{\_sockets[\_sockets.Count - 1].AddressFamily.ToString()}");

}

}

catch(InvalidOperationException ex)

{

HostStatusGc = ex.Message + " " + "(" + ex.Source + ")";

return;

// throw new Exception(ex.Message + " " + "(" + ex.Source + ")");

}

catch (Exception ex)

{

HostStatusGc = ex.Message + " " + "(" + ex.Source + ")";

continue;

}

}

}

public static int GetLightyLoadedMachine(out Socket socket)

{

CheckConnection();

if (\_sockets.Count == 0)

throw new Exception("No sockets for connection\n");

int workload = 100;

int index = -1;

int bytesRec = 0;

byte[] msg =Encoding.Default.GetBytes(\_WorkLoad);

byte[] answer = new byte[56];

for (int i = 0; i < \_sockets.Count; i++)

{

\_sockets[i].Send(msg);

bytesRec = \_sockets[i].Receive(answer);

if (Int32.Parse(Encoding.Default.GetString(answer, 0, bytesRec)) <= workload)

{

index = i;

workload = Int32.Parse(Encoding.Default.GetString(answer, 0, bytesRec));

}

}

if (index == -1)

{

socket = \_sockets[0];

return index;

}

else

{

socket = \_sockets[index];

return index;

}

}

// index of prepared socket to proccess data

public static void SendDataToMachine(string filePath, string pathToDir, ref Socket socket)

{

CheckConnection();

if (\_sockets.Count == 0)

throw new Exception("No sockets for connection\n");

byte[] bytes = Encoding.Default.GetBytes(\_ProcessData);

socket.Send(bytes);

bytes = new byte[2];

int bytesRec = socket.Receive(bytes);

if (Encoding.Default.GetString(bytes, 0, bytesRec) == "ok")

Console.WriteLine(Encoding.Default.GetString(bytes, 0, bytesRec));

// если не будет работать попробовать Socket.ReceiveTimeout или Socket.ShutDown(SocketShutdown.Both)

socket.SendFile(filePath);

\_file = new FileStream(pathToDir, FileMode.OpenOrCreate, FileAccess.ReadWrite,

FileShare.None);

int offset = 0;

while (socket.Available != 0)

{

bytes = new byte[1024];

bytesRec = socket.Receive(bytes);

if (bytesRec == 0)

break;

\_file.Write(bytes, 0, bytesRec);

offset += bytesRec;

}

\_file.Close();

}

public static void CheckConnection()

{

if (\_sockets.Count == 0)

return;

for (int i = 0; i < \_sockets.Count; i++)

{

if ((\_sockets[i].Poll(1000, SelectMode.SelectRead) &&

(\_sockets[i].Available == 0)) ||

!\_sockets[i].Connected)

{

HostStatusGc = ($"Disconect socket: #" + $"{i} " +

$"{\_sockets[i].RemoteEndPoint.AddressFamily.ToString()}");

\_sockets[i].Shutdown(SocketShutdown.Both);

\_sockets[i].Close();

\_sockets.RemoveAt(i);

}

}

}

public static void TerminateGridCore()

{

if (\_sockets.Count != 0)

{

do

{

\_sockets[0].Shutdown(SocketShutdown.Both);

\_sockets[0].Close();

\_sockets.RemoveAt(0);

} while (\_sockets.Count != 0);

//\_sockListner.Shutdown(SocketShutdown.Both);

// \_sockListner.Disconnect(false);

}

\_sockListner.Close();

HostStatusGc = "------ GRID System was terminated at: " +

DateTime.Now + " ------";

}

public static string HostNameGc

{

get { return \_ipHost.HostName.ToString(); }

}

public static int HostPortGc

{

get { return \_hostPort; }

}

public static string HostIpGc

{

get { return \_ipAddr.ToString(); }

}

public static string HostStatusGc

{

set

{

lock (\_locker)

{

\_hostStatus = value;

\_logFile = new FileStream(@"GRID\_LOG.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite,

FileShare.None);

\_logFile.Seek(0, SeekOrigin.End);

byte[] log = System.Text.Encoding.Default.GetBytes(\_hostStatus + Environment.NewLine);

\_logFile.Write(log, 0, log.Length);

\_logFile.Close();

}

}

get { return \_hostStatus; }

}

private static IPHostEntry \_ipHost;

private static IPAddress \_ipAddr;

private static IPEndPoint \_ipEndPoint;

private static Socket \_sockListner;

private static List <Socket> \_sockets = new List<Socket>();

private static int \_hostPort = 8005;

private static String \_hostStatus;

private static FileStream \_file;

private static FileStream \_logFile;

private static string \_WorkLoad = "load";

private static string \_ProcessData = "data";

private static object \_locker = new object();

}

}

Класс-прослойка, связывающий View и Model

using GalaSoft.MvvmLight;

using GalaSoft.MvvmLight.Command;

using System;

using System.ComponentModel;

using System.IO;

using System.Net.Sockets;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

namespace Server

{

public class MainWindowVM : ViewModelBase

{

public MainWindowVM()

{

try

{

GRIDCore.Initialize();

}

catch(Exception ex)

{

MessageBox.Show(ex.Message + " " + "(" + ex.Source + ")" +

"\nError of host initializing. Reload the GRID System");

GRIDCore.TerminateGridCore();

}

\_hostName = GRIDCore.HostNameGc;

\_hostIP = GRIDCore.HostIpGc;

\_hostPort = GRIDCore.HostPortGc;

\_hostStatus = GRIDCore.HostStatusGc;

\_waitSockThread = new Thread(GRIDCore.WaitSocket);

\_waitSockThread.Start();

}

private void CheckAndSendData()

{

if(\_pathToDir == null || \_pathToFile == null)

{

MessageBox.Show("Incorrect path to processed files or file\n");

return;

}

try

{

Socket socket;

if (GRIDCore.GetLightyLoadedMachine(out socket) == -1)

{

MessageBox.Show("No unloaded machines.\n");

return;

}

string pathToProcessedFile = \_pathToDir + @"\Processed\_File\_" + processedFileCount.ToString() + ".txt";

GRIDCore.SendDataToMachine(\_pathToFile, pathToProcessedFile, ref socket);

processedFileCount++;

}

catch(Exception ex)

{

MessageBox.Show(ex.Message + " " + "(" + ex.Source + ")" +

"\n Reply load the data");

return;

}

}

public ICommand SendDataToMachine

{

get

{

return \_loadFileDir ?? (\_loadFileDir = new RelayCommand(() =>

{

Task.Factory.StartNew(() =>

{

CheckAndSendData();

});

}));

}

}

public ICommand WindowClosing

{

get

{

return new RelayCommand<CancelEventArgs>(

(args) => {

MessageBox.Show("Closing Grid System(");

GRIDCore.TerminateGridCore();

});

}

}

public string HostName{ get { return \_hostName; } }

public string HostIP { get { return \_hostIP; } }

public int HostPort { get { return \_hostPort; } }

public string HostStatus

{

get { return \_hostStatus; }

set

{

\_hostStatus = value;

RaisePropertyChanged(() => HostStatus);

}

}

public string PathToDir

{

get { return \_pathToDir; }

set

{

\_pathToDir = @"" + value;

if (\_pathToDir == null)

{

\_validDirectory = false;

return;

}

try

{

DirectoryInfo dirInfo = new DirectoryInfo(\_pathToDir);

if (!dirInfo.Exists)

dirInfo.Create();

MessageBox.Show(\_pathToDir);

ValidDirectory = true;

}

catch(ArgumentException ex)

{

MessageBox.Show("path contains invalid characters such as \", <, >, or |.\n" +

ex.Message);

\_pathToDir = null;

ValidDirectory = false;

}

catch (Exception ex)

{

MessageBox.Show("The directory cannot be created.\n" +

ex.Message);

\_pathToDir = null;

ValidDirectory = false;

}

}

}

public bool ValidDirectory

{

get { return \_validDirectory; }

set

{

\_validDirectory = value;

RaisePropertyChanged(() => ValidDirectory);

}

}

public string PathToFile

{

get { return \_pathToFile; }

set

{

\_pathToFile = @"" + value;

if (\_pathToFile == null)

{

ValidPathFile = false;

return;

}

try

{

FileInfo fileInfo = new FileInfo(\_pathToFile);

if(!fileInfo.Exists)

{

MessageBox.Show("File not exist");

\_pathToFile = null;

ValidPathFile = false;

return;

}

MessageBox.Show(\_pathToFile);

ValidPathFile = true;

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

\_pathToFile = null;

ValidPathFile = false;

}

}

}

public bool ValidPathFile

{

get { return \_validPathFile; }

set

{

\_validPathFile = value;

RaisePropertyChanged(() => ValidPathFile);

}

}

private string \_hostName;

private int \_hostPort;

private string \_hostIP;

private string \_hostStatus;

private string \_pathToDir;

private string \_pathToFile;

private bool \_validDirectory = false;

private bool \_validPathFile = false;

private int processedFileCount = 0;

private Thread \_waitSockThread;

private ICommand \_loadFileDir;

}

}

<Window x:Class="Server.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Server"

xmlns:i="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"

xmlns:command="http://www.galasoft.ch/mvvmlight"

mc:Ignorable="d"

Title="GRID System" Height="450" Width="500">

<Window.DataContext>

<local:MainWindowVM/>

</Window.DataContext>

<i:Interaction.Triggers>

<i:EventTrigger EventName="Closing">

<command:EventToCommand Command="{Binding WindowClosing}" PassEventArgsToCommand="True" />

</i:EventTrigger>

</i:Interaction.Triggers>

<Grid HorizontalAlignment="Right" Width="490" Height="490" VerticalAlignment="Bottom" Background="Azure">

<TextBlock x:Name="UI\_hostPort" HorizontalAlignment="Left" Margin="30,429,0,0" TextWrapping="Wrap" Text="{Binding HostPort}" VerticalAlignment="Top"/>

<TextBlock x:Name="UI\_hostIP" HorizontalAlignment="Left" Margin="30,400,0,0" TextWrapping="Wrap" Text="{Binding HostIP}" VerticalAlignment="Top" RenderTransformOrigin="-3.085,2.562"/>

<TextBlock x:Name="UI\_hostName" HorizontalAlignment="Left" Margin="30,370,0,0" TextWrapping="Wrap" Text="{Binding HostName}" VerticalAlignment="Top"/>

<TextBlock x:Name="UI\_hostStatus" HorizontalAlignment="Left" Margin="270,370,0,0" TextWrapping="Wrap" Text="{Binding HostStatus}" VerticalAlignment="Top"/>

<TextBlock x:Name="TextBlock" HorizontalAlignment="Left" TextWrapping="Wrap" Text="Path to processed files directory:" VerticalAlignment="Top" Margin="10,233,0,0"/>

<TextBlock x:Name="textBlock1" HorizontalAlignment="Left" Margin="10,268,0,0" TextWrapping="Wrap" Text="Path to file:" VerticalAlignment="Top" RenderTransformOrigin="0.952,1.773"/>

<TextBox x:Name="filesDir" HorizontalAlignment="Left" Height="23" Margin="187,226,0,0" TextWrapping="Wrap" Text="{Binding PathToDir}" VerticalAlignment="Top" Width="261"/>

<TextBox x:Name="pathToFile" HorizontalAlignment="Left" Height="23" Margin="188,261,0,0" TextWrapping="Wrap" Text="{Binding PathToFile}" VerticalAlignment="Top" Width="260"/>

<Button x:Name="save" Content="Save" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75" Margin="373,289,0,0"/>

<Button x:Name="load" Content="Load data" HorizontalAlignment="Left" Margin="188,289,0,0" VerticalAlignment="Top" Width="75" RenderTransformOrigin="0.275,-0.296" Command="{Binding SendDataToMachine}"/>

<CheckBox x:Name="validPathDir" HorizontalAlignment="Left" Margin="453,226,0,0" VerticalAlignment="Top" IsChecked="{Binding Path=ValidDirectory, UpdateSourceTrigger=PropertyChanged}"/>

<CheckBox x:Name="validPathFile" HorizontalAlignment="Left" Margin="453,261,0,0" VerticalAlignment="Top" IsChecked="{Binding Path=ValidPathFile, UpdateSourceTrigger=PropertyChanged}"/>

</Grid>

</Window>

**ПРИЛОЖЕНИЕ Д**

*(обязательное)*

Ведомость документов