Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Факультет «Вычислительная математика и информатика»
Кафедра «Экономико-математические методы и статистика»

РАБОТА ПРОВЕРЕНА
Рецензент, технический директор
ООО «Грид Инжиниринг»
_____Ю. В. Ситников
«___»_____2016 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. каф. ЭММиС, проф.,
д.ф.-м.н.
_____А. В. Панюков
«___»_____2016 г.

РАЗРАБОТКА ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МОНИТОРИНГА И АНАЛИЗА СЕРВЕРОВ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–Д.010400.62.2016.047.ПЗ (ВКР)

Руководитель работы,
доц. каф. ЭММиС, к.ф.-м.н.
_____В. А. Голодов
« »_____2016 г.

Автор работы
студент группы ВМИ-403
_____В. Д. Колногоров
« »_____2016 г.

Нормоконтролер,
доц. каф. ЭММиС, к.ф.-м.н.
_____Т. А. Макаровских
« »_____2016 г.

Челябинск, 2016

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Факультет «Вычислительная математика и информатика»
Кафедра «Экономико-математические методы и статистика»
Направление 010400.62–«Прикладная математика и информатика»

УТВЕРЖДАЮ
Зав. каф. ЭММиС, проф.,
д.ф.- м.н.
_____А. В. Панюков
«___»_____2016 г.

З А Д А Н И Е
на выпускную квалификационную работу студента
Колногорова Вячеслава Дмитриевича

Группа ВМИ-403

1 Тема работы: Разработка прикладного программного обеспечения для мониторинга и анализа серверов.
2 Утверждена приказом №_____ по университету от « »_____2016 г.
3 Срок сдачи студентом законченной работы « »_____2016 г.
4 Исходные данные к работе
  4.1 Данные о загруженности процессора серверов ООО «Грид Инжиниринг».
5 Перечень вопросов подлежащих к разработке
  5.1 Разработка и тестирование программного обеспечения для мониторинга и анализа серверов.
  5.2 Формирование выборки по загруженности процессора логируемого сервера.
  5.3 Анализ полученных данных с помощью разработанного программного обеспечения.
  5.4 Формирование выводов об оптимизации работы серверов.
6 Перечень графического материала

6.1 Введение в область – 1 л.
6.2 Цели и задачи – 1 л.
6.3 Методы прогнозирования загруженности процессора – 2 л.
6.4 Данные о загруженности процессора – 1 л.
6.5 Средняя загрузка по серверам в день (в процентах) – 1 л.
6.6 Анализ методов прогнозирования – 1 л.
6.7 Средства реализации ПО – 1 л.
6.8 Пользовательский интерфейс – 6 л.
6.9 Заключение – 1 л.

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломной работы	Срок выполнения этапов работы	Отметка руководителя о выполнении
1 Ознакомление с аналогичными разработками для мониторинга серверов		
2 Изучение литературы для написания дипломной работы и реализации поставленных задач		
3 Разработка логики программного обеспечения и пользовательского программного интерфейса		
4 Реализация программного обеспечения		
5 Тестирование программного обеспечения: формирование выборки по загруженности процессора логируемого сервера		
6 Анализ полученной выборки		
7 Подготовка пояснительной записки дипломной работы		
8 Оформление пояснительной записки		
9 Проверка работы руководителем, исправление полученных замечаний		
10 Подготовка графического материала и доклада		
11 Нормоконтроль		
12 Рецензирование, представление зав кафедрой		

Заведующий кафедрой _____/А.В. Панюков/
(подпись)

Руководитель работы _____/В.А.Голодов/
(подпись)

Студент_____/В.Д.Колногоров/
(подпись)

АННОТАЦИЯ

Колногоров, В.Д. Разработка прикладного программного обеспечения для мониторинга и анализа серверов/ В.Д. Колногоров. – Челябинск: ЮУрГУ, ВМИ-403, 2016. – 81 с.,29 ил., 2 табл., библ. список – 18 наим., 5 прил.

Разработано прикладное программное обеспечение для мониторинга и анализа серверов на операционных системах семейства Windows. Программно реализованы: мониторинг системной информации, статистический анализ полученных данных. В качестве инструментов разработки приложения использован язык программирования C#, а также, инструментарий управления Windows – WMI.

Практическая значимость разработанного приложения состоит в возможности использования организациями, работающими с недорогостоящими серверами без пакетов мониторинга.

Оглавление

Введение

В настоящее время всё больше организаций доверяют решения своих задач информационным технологиям. Зачастую эти задачи требуют объёмных и ресурсоёмких вычислений (экономическое прогнозирование, сейсмоанализ, рендер высокополигональных 3D-объектов), которые необходимо решать на вычислительных машинах большей мощности нежели обычный персональный компьютер. Не все организации могут позволить иметь у себя в постоянном распоряжении подобные суперкомпьютеры. В такой ситуации предприятия используют концепцию «облачных вычислений».

Организации, предоставляющие услуги по грид-вычислениям (форма распределённых вычислений, в которой суперкомпьютер представлен в виде кластеров, соединённых в сеть для выполнения ресурсоёмких задач), имеют у себя достаточно мощные вычислительные ресурсы, но и здесь, как и в любой сфере, нужен контроль за работой такой сложной системы. Корректная работа сервера зависит от загруженности его модулей и выполняемых на текущий момент задач. Поэтому мониторинг и анализ полученных данных являются актуальными задачами.

Подобные системы мониторинга обычно разрабатываются для конкретных суперкомпьютеров, например система оперативного мониторинга температуры и энергопотребления суперкомпьютера «Уран» института математики и механики Уральского отделения РАН (г. Екатеринбург) [1], система мониторинга суперкомпьютера «Ломоносов» Московского Государственного Университета [2], а также трёхуровневая система мониторинга суперкомпьютера «Торнадо ЮУрГУ» [3].

Однако многие сервера коммерческих организаций, обеспечивающие услугами грид-вычислений, не имеют своих предустановленных систем для мониторинга и проверка всех машин занимает у оператора достаточно много времени, которое тратится на переключение между удалёнными рабочими столами и сбор информации вручную. Для обеспечения оперативного мониторинга

необходимо предоставлять оператору всю системную информацию о серверах в один – два экрана.

Хранение данных о загруженности процессора позволяет осуществлять их статистическую обработку и прогнозировать загруженность сервера в будущем, а также выявить какие из серверов имеют слишком большие нагрузки, а какие, наоборот, простаивают, чтобы оптимизировать работу машин и сэкономить на электроэнергии.

В связи с вышесказанным, **целью работы** стало создание прикладного программного обеспечения для мониторинга, статистического анализа полученных данных.

Исходя из цели работы, поставлены и решены следующие **задачи**:

- обзор исследования загруженности процессора [6];
- выбор и анализ способов прогнозирования загруженности процессора;
- программная реализация системы мониторинга, статистического анализа полученных данных, согласно техническому заданию (приложение А);
- проектирование прикладного программного обеспечения: логика, пользовательский интерфейс.

Объект исследования – сервера, предназначенные для грид-вычислений.

Предмет исследования – оперативный мониторинг серверов.

Работа состоит из введения, 3 глав, заключения, 5 приложений и списка литературы. Объём работы составляет 81 страницу. Список литературы содержит 18 наименований.

В качестве теоритической базы исследования использовались пособие по программированию на языке C# В.В. Подбельского [4]; документация языка C# с сайта MSDN [5]; статья, посвящённая прогнозированию загруженности процессора Н.А. Кутепова, В.П. Горицкого, Ю.А. Бражникова Ю.С. Панкова [6].

В первой главе речь идёт о грид-вычислениях: истории создания, области применения, показывается необходимость мониторинга больших вычислительных систем.

Во второй главе рассматривается ранее проводимое исследование [6] прогнозирования загруженности процессора, анализируются результаты. Приводится альтернативные способы моделирования и прогнозирования данных о загруженности процессора, производится анализ способов, выбирается наиболее подходящий для реализации в ПО.

В третьей главе проводится обзор средств реализации ПО, программная реализация мониторинга и анализа серверов, демонстрируется пользовательский интерфейс программы.

В заключении перечислены основные результаты работы.

Практическая значимость разработанного приложения состоит в возможности использования организациями, работающими с недорогостоящими серверами без пакетов мониторинга.

1 Большие вычислительные системы: обзор, история, актуальность мониторинга

1.1 Коллективное использование больших вычислительных систем

Коллективное использование больших вычислительных систем на сегодняшний день является востребованным видом обеспечения мощными ресурсами многих конечных пользователей, которыми могут являться как индивидуальные пользователи, так и целые организации. Широко используется терминология кластерных, облачных, грид-вычислений; прочно вошли в обиход понятия виртуализации, частных и публичных облаков, термины "Compute Grids", "Data Grids", "Science Grids", "Access Grids", "Knowledge Grids", "Bio Grids", "Sensor Grids", "Cluster Grids", "Campus Grids", "Tera Grids", "Commodity Grids" и многие другие, связанные в первую очередь с повышением эффективности использования программно-аппаратных вычислительных структур [7].

В историческом аспекте можно проследить некоторое пересечение во времени возникновения понятий (технологий) виртуализации, грид-вычислений и их, в некотором смысле, агрегации – облачных вычислений. Без ограничения общности можно считать, постановка концепции грид-вычислений на фундамент виртуальной инфраструктуры с одновременным концептуальным упрощением (обоснованным более высоким уровнем изоляции) фактически определило возникновение понятия облачных вычислений [7].

1.2 Грид-вычисления

Основной теоретической предшественницей современных грид-проектов считается инициатива Metacomputing, предложенная в середине 80-х годов исследователями из Национального центра суперкомпьютерных приложений США. Ее главная идея состояла в объединении нескольких суперкомпьютеров для достижения большей производительности. Одной из первых инфраструк-

тур, реализующих эту идею, стала в 1995 году Wide Area Year (I-WAY). Йан Фостер (Аргоннская Национальная лаборатория Чикагского университета) и Карл Кессельман (Института информационных наук Университета Южной Калифорнии, США), участвовавшие в разработке проекта, в том же году опубликовали первые материалы [8]. Учёные дали следующее определение: "Грид - согласованная, открытая и стандартизованная среда, которая обеспечивает гибкое, безопасное, скоординированное разделение (общий доступ) ресурсов в рамках виртуальной организации".

Необходимо отметить, однако, что Metacomputing была не единственной инициативой такого рода. Среди схожих по концепции проектов конца 80-х – начала 90-х можно отметить:

- Condor (университета штата Висконсин, США);
- CODINE (Computing for Distributed Network Environments, немецкая компания Genias
- Software - позднее переименована в Gridware, в 1992 году куплена корпорацией Sun Microsystems);
- Legion (университет штата Вирджиния, позднее рабочая группа проекта выд лилась в компанию Avaki) [8].

Термин грид (в смысле одной из технологий распределенных вычислений - возник в середине 90-х годов) происходит от английского слова «grid» и его буквальным переводом на русский язык является «решетка». В русскоязычной литературе зачастую и используется этот буквальный перевод. Однако надо понимать, что такой буквальный перевод не вполне отражает смысл термина. Дело в том, что англоязычный термин grid произошел от «power grid», что соответствует русскому «электросеть» или «энергосистема». В этом и заключается смысл этого названия: подобно тому, как при использовании энергосистем мы не интересуемся, какой конкретный электрогенератор выработал ток, который мы потребляем (а просто платим за потребленное электричество энергетической компании, с которой мы заключили договор), при использовании компьютерного грида мы можем не заботиться о том - какой конкретно компьютер

(или устройство хранения/передачи данных) в грид-системе выполнил нашу задачу (при этом, возможно, придется тем или иным способом платить соответствующему провайдеру за использованные ресурсы) [8].

Создание таких систем стало возможным благодаря впечатляющим успехам, прежде всего, в четырех направлениях:

- повышению производительности микропроцессоров массового производства - современный персональный компьютер сравним по производительности с суперкомпьютерами десятилетней давности;
- появлению быстрых линий связи - сейчас осуществляется перевод основных магистралей на уровень нескольких Гигабит/сек;
- глобализации обмена информацией (Интернет/Веб);
- развитию методов метакомпьютинга - научной дисциплины по организации массовых и распределенных вычислительных процессов [8].

1.3 Облачные вычисления

Сама по себе концепция грид, несмотря на достаточно простое интуитивное понимание, оказалась весьма непростой с точки зрения технической реализации. Положение дел осложнялось тем фактом, что для грид необходимо разрабатывать программное обеспечение (ПО), которое удовлетворяло бы концепции грид как минимум с точки зрения взаимодействия компонент ПО [7].

Другим, вполне современным, подходом к более полному использованию вычислительных ресурсов является концепция облачных вычислений, основанная в первую очередь на технологиях виртуализации [7].

Архитектуры грид-систем и облаков заметно различаются, поскольку создавались исходя из разных предпосылок. На первые повлияло стремление как можно эффективнее использовать дорогостоящие распределенные вычислительные ресурсы, сделать их динамическими и однородными. Поэтому архитектура сфокусирована на интеграции уже существующих ресурсов, включая оборудование и программное обеспечение, операционные системы, локальные средства, обеспечивающие управление и безопасность. В результате создается

«виртуальная организация», ресурсы которой, переведенные в логическую форму, могут потребляться членами только этой организации. Существование этой организации поддерживается пятью уровнями протоколов, инструментами и сервисами, построенными поверх них (рис 1.1 а). Нижним является *инфраструктурный уровень* (fabric layer), объединяющий компьютеры, системы хранения, сети, репозитории кодов. Выше него расположен *уровень связности* (connectivity layer), на нем определены коммуникационные протоколы и протоколы аутентификации. *Ресурсный уровень* обеспечивает предоставление ресурсов, возможности управления ими, разделение между отдельными пользователями и оплату. *Коллективный уровень* (collective layer) дополняет ресурсный, позволяя оперировать наборами ресурсов. *Уровень приложений* (application layer) служит для поддержки приложений.

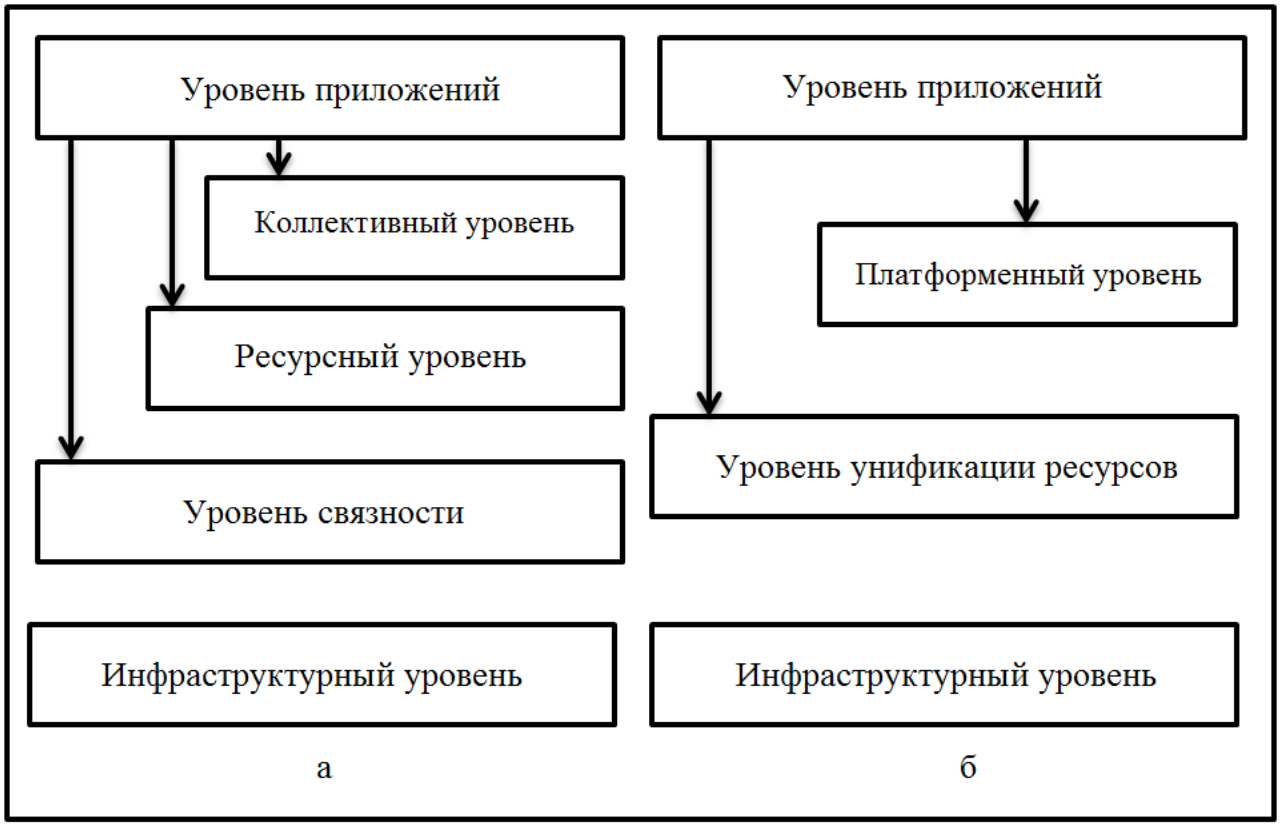


Рисунок 1.1 – Сравнение архитектуры грид-систем и облачных вычислений

Архитектура облаков открыта для доступа через Сеть, а не только в рамках grid. Обращение к пулам вычислительных ресурсов и системам хранения данных осуществляется по стандартным протоколам, например таким, как

WSDL и SOAP, или с помощью более продвинутых технологий Web 2.0 (REST, RSS, AJAX), а также через существующие технологии grid. Протоколы облаков можно разделить на четыре уровня (рис 1.1 б). *Инфраструктурный уровень* (fabric layer) содержит «сырые» компьютерные ресурсы (серверы, системы хранения, сети). *Уровень унификации ресурсов* (unified resource layer) содержит те же ресурсы, но в абстрагированном виде – они могут быть представлены пользователям и верхнему уровню как виртуализованные серверы, кластеры серверов, файловые системы и СУБД. *Уровень платформ* (platform layer) добавляет набор специализированных инструментов, связующее ПО и сервисы поверх универсальных ресурсов, образуя среду для разработки и внедрения приложений. *Уровень приложений* (application layer) содержит приложения, исполняемые в облачной среде.

С пользовательской позиции сервисы облаков можно разделить на три основных уровня – IaaS (Infrastructure as a Service), PaaS (Platform as a Service) и SaaS (Software as Service). Помимо них могут быть доступны и востребованы IdaaS (Identification as Service, управление идентификацией), HPCaaS (High Performance as a Service, высокопроизводительные вычисления) и другие специализированные сервисы.

IaaS обычно предоставляет унифицированные аппаратные и программные ресурсы, но в некоторых случаях и на инфраструктурном уровне для установки ПО с оплатой по мере использования. Заказанная инфраструктура может динамически масштабироваться. Типичные примеры – Amazon EC2 (Elastic Cloud Computing) Service и Amazon S3 (Simple Storage Service).

PaaS предоставляет более высокий уровень сервиса, позволяющий разрабатывать, тестировать и внедрять пользовательские приложения. Встроенная масштабируемость накладывает ограничения на тип разрабатываемых приложений. Пример – сервис Google App Engine, позволяющий внедрять Web-приложения на той же системе, на которой работают собственные приложения Google.

Software as a Service (SaaS) предлагает готовое специализированное ПО. Канонический пример – Salesforce и ее онлайновая система управления отношениями с клиентами [9].

Компьютерные инфраструктуры, основанные на концепции облачных вычислений, получают сегодня все большее распространение в самых различных областях. Облачные вычисления (cloud computing) — технология распределенных вычислений, в которой компьютерные ресурсы и мощности предоставляются пользователю как интернет-сервис. В соответствии с концепцией облачных вычислений, все, что нужно пользователю для решения его задачи, — это компьютер-клиент, на котором установлен интернет-обозреватель. Все остальное спрятано в облаке (рис. 1.2). Под облаком подразумевается не сам Интернет, а весь тот набор аппаратного и программного обеспечения, который обеспечивает решение его задачи. При этом пользователю не требуется никаких особых знаний об инфраструктуре облака или навыков управления облачными технологиями. Привлекательность облачных вычислений заключается в том, что пользователю нет необходимости обладать высокопроизводительной техникой и сложным программным обеспечением, ему достаточно обратиться через Интернет к соответствующему провайдеру и попросту оплатить услугу [10].

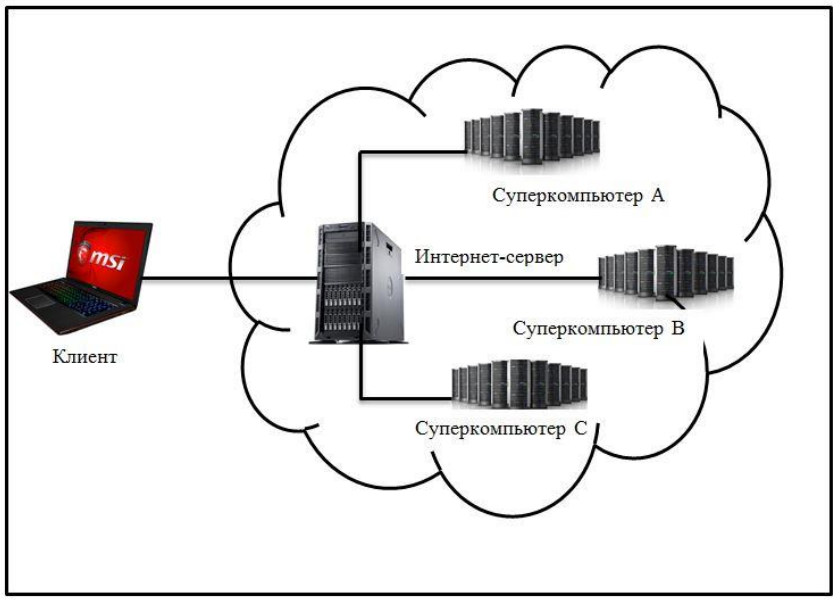


Рисунок 1.2 – Облачные вычисления

Сервера, предназначающиеся для грид и облачных вычислений, являются сложными по своему составу и принципу работы системами и должны подвергаться пристальному контролю со стороны обслуживающего персонала. Контроль должен осуществляться оперативными средствами слежения (мониторинг).

1.4 Мониторинг больших вычислительных систем

Системы мониторинга суперкомпьютерных комплексов – важная часть их программного обеспечения, призванная обеспечить максимально эффективное и бесперебойное функционирование, а в случае возникновения нештатных ситуаций – сохранность оборудования и оповещение персонала. Существующие системы мониторинга справляются с задачами обеспечения сохранности оборудования, хотя достижение нужного времени реакции для больших комплексов может быть непростой задачей. Кроме того, возникла еще одна важная область использования систем мониторинга – мониторинг производительности. Мониторинг производительности вычислительных комплексов в целом и отдельных программ – важная задача, решение которой позволяет оценить эффективность использования имеющихся ресурсов и предложить пути увеличения эффективности выполняемых программ [2].

Дорогостоящие сервера, используемые крупными организациями, уже имеют предустановленные пакеты мониторинга, однако, компании меньшего масштаба не могут позволить себе подобные системы. На серверах бюджетной ценовой категории, как правило, нет предустановленного программного обеспечения (ПО) для контроля за системой. Поэтому имеет смысл реализовывать своё ПО для мониторинга серверов.

Выводы по главе один

Мониторинг суперкомпьютеров – важная часть работы таких сложных систем. Реализация подобного программного обеспечения позволит контролировать такие процессы, как загруженность процессора, оперативной памяти, се-

тевого соединения. Так же реализация данного ПО будет актуальна для организаций с оборудованием, не имеющим подобного предустановленного обеспечения.

2 Способы реализации мониторинга и анализа загруженности серверов

2.1 Методы анализа загруженности серверов

Анализ загруженности серверов, а также дальнейший её прогноз, является актуальной задачей в области больших вычислительных систем любого размера.

Подобную задачу рассматривают Ю.С. Бражникова, Ю.А. Горицкий, В.П. Купетов, Н.А. Панков в своей работе «Исследование методов прогнозирования загруженности компьютеров и компьютерных систем» [6].

В данной работе сравниваются методы моделирования и прогнозирования рядов данных высокочастотной природы – загруженности процессора. Подобные методы применяются в сфере радиолокации [6].

В работе [6] используются данные о загруженности процессора с периодов в 1 секунду. Значения временного ряда прогнозируются на шаг вперёд, в силу нестационарности загруженности процессора. Для получения результатов применялись такие приложения, как MS Visual Studio, MathCad, MS Office. В работе [6] были рассмотрены следующие методы прогнозирования:

- метод экспоненциального сглаживания;
- метод медианного фильтра.

В исследовании [6] показано, что наиболее точным методом прогнозирования является метод медианной средней – он достаточно прост для реализации на практике, а также требует немного времени на определение прогноза загруженности.

2.2 Инструмент мониторинга серверов под управлением MS Windows

Первостепенной направленностью разрабатываемого программного обеспечения является реализация мониторинга серверов в реальном времени. Про-

грамма должна оперативно предоставлять пользователю следующую информацию:

- загруженность ядер и процессора в целом;
- количество свободной оперативной памяти;
- нагрузку на локальную сеть;
- информацию о текущих процессах.

Так как сервера, для которых разрабатывалось ПО находятся под управлением Microsoft Windows Server 2008 R2 Enterprise, то реализация мониторинга будет осуществляться с помощью инструментария управления Widows (**Windows Management Instrumentation – WMI**).

WMI – это одна из базовых технологий для централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows [11].

Технология WMI – это расширенная и адаптированная под Windows реализация стандарта WBEM (Web-Based Enterprise Management) [12], принятого многими компаниями, в основе которого лежит идея создания универсального интерфейса мониторинга и управления различными системами и компонентами распределенной информационной среды предприятия с использованием объектно-ориентированных идеологий и протоколов HTML и XML [11].

Архитектура WMI состоит из следующих частей (рисунок 2.1):

- Менеджер объектов CIM (Common Information Model Object Manager, CIMOM), который обеспечивает обработку всех запросов конечных приложений (WMI потребители) к WMI и доставку информации от WMI к конечным приложениям. Все провайдеры WMI (см. ниже) должны быть зарегистрированы с помощью CIMOM для правильного перенаправления полученных от конечного приложения запросов к нужному провайдеру. Функциональность CIMOM обеспечивает файл winmgmt.exe, который находится в каталоге%SystemRoot%\System32\Wbem\. Этот файл запускается как сервис.

- Репозиторий (хранилище классов) CIM. Объекты-экземпляры таких классов создаются провайдером WMI (см. ниже) по запросу потребителя. В Windows Server 2003 и Windows XP репозиторий физически располагается в каталоге%SystemRoot%\System32\Wbem\Repositiry\FS\ в файлах objects.data (репозиторий), index.btr (индексный файл), index.map и object.map (файлы контроля над транзакциями). В более ранних версиях Windows репозиторий располагается в файле cim.rep.

- Провайдеры WMI, которые скрывают детали внутренней реализации управляемых объектов, позволяя CIMOM обращаться к этим объектам единообразно, используя WMI API. Фактически провайдеры являются COM-серверами, которые представлены dll-библиотеками в каталоге %SystemRoot%\System32\Wbem\. WMI включает множество встроенных провайдеров, которые предназначены для получения данных из различных источников, например, журналов событий, системного реестра и т.д.

- Библиотека поддержки сценариев (WMI scripting library), которая располагается в файле wbemdisp.dll в каталоге %SystemRoot%\System32\Wbem\.

Для обращения к объектам WMI используется специфический язык запросов WMI Query Language (WQL), который является одной из разновидностей SQL. Основное его отличие от ANSI SQL — это невозможность изменения данных, то есть с помощью WQL возможна лишь выборка данных с помощью команды SELECT. Помимо ограничений на работу с объектами, WQL не поддерживает такие операторы как DISTINCT, JOIN, ORDER, GROUP, математические функции. Конструкции IS и NOT IS применяются только в сочетании с константой NULL [11].

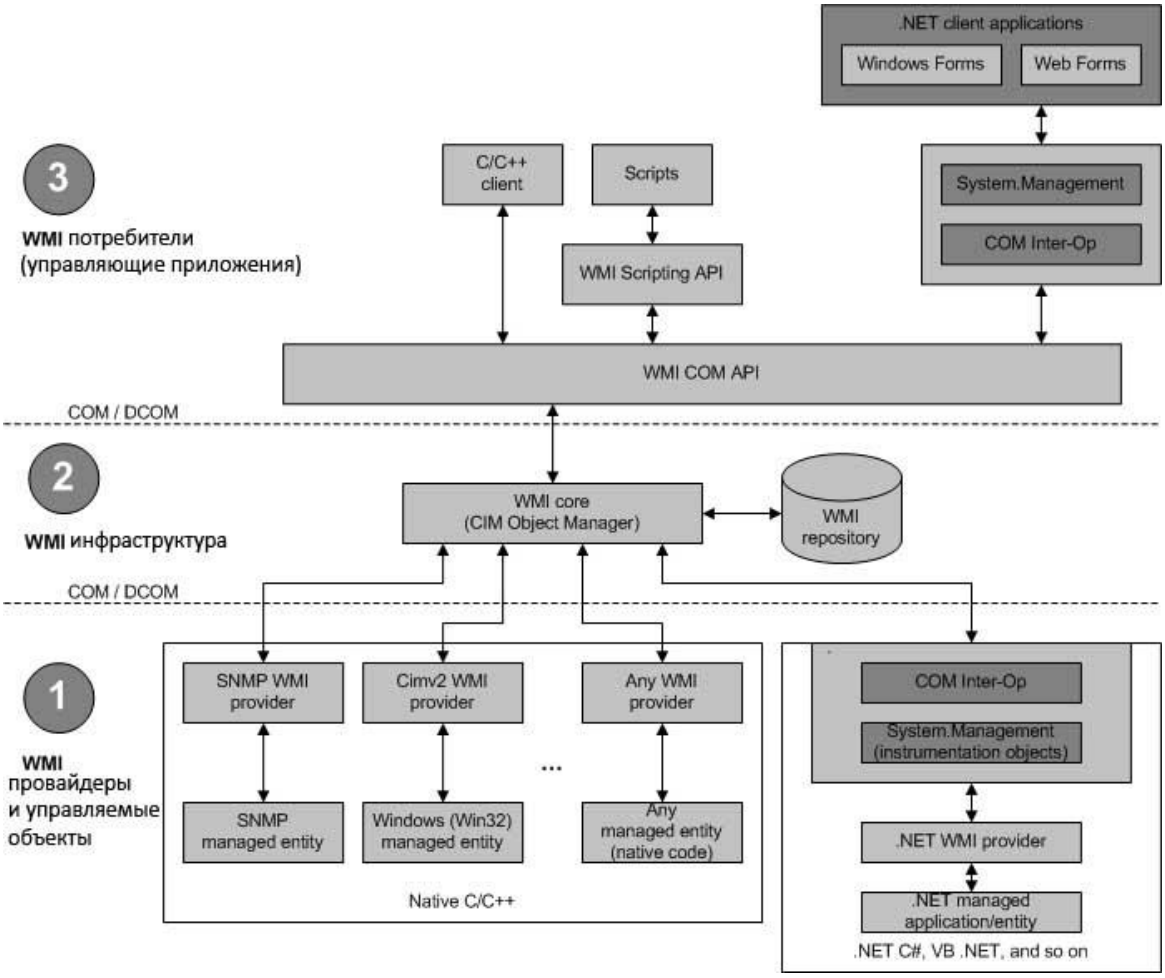


Рисунок 2.1 –Архитектура WMI

Для использования WQL в разрабатываемом ПО будут использоваться запросы с командой SELECT. Для безошибочного составления запросов, будем использовать ПО WMI Code Creator v1.0 (рисунок 2.2).

Данное ПО позволяет генерировать VB, C#, .NET код, который использует WMI для выполнения задач мониторинга, например, запрос данных, выполняя методы классов WMI или получение уведомления о событиях с помощью WMI [13].

Пример кода получения информации с помощью WMI представлен в листинге 2.1.

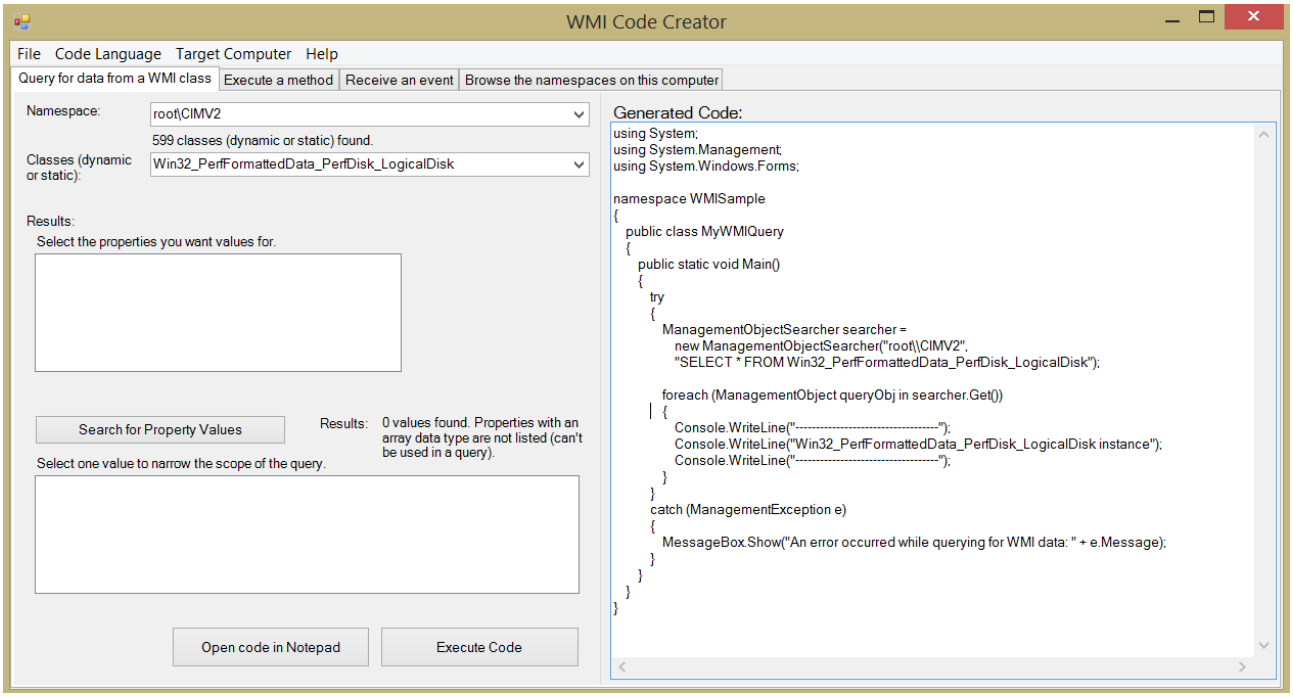


Рисунок 2.2 – Рабочая область WMI Code Creator v1.0

Листинг 2.1 – Пример получения информации с помощью WMI

```
1    ConnectionOptions options1 = new ConnectionOptions();
2    options1.Username = “Admin”;
3    options1.Password = “1234”;
4    ManagementScope myScope =
5    new ManagementScope("\\\\192.168.1.1\\root\\cimv2",
6    options1);
7    ObjectQuery query =
8    new ObjectQuery
9    ("select * from Win32_PerfRawData_PerfOS_Memory");
10   ManagementObjectSearcher searcher =
11   new ManagementObjectSearcher(myScope, query);
12   ManagementObjectCollection queryCollection =
13   searcher.Get();
14   foreach (ManagementObject m in queryCollection)
15   {
16   label.Text = string.Format("{0}", m["AvailableMBytes"]);
17   }
```

В строке 1 инициализируется объект ConnectionOptions, которой задает все параметры, обязательные для установки WMI-подключения.

В строках 2 и 3 заполняются поля объекта ConnectionOptions, содержащие логин и пароль соответственно.

В строках с 4 по 6 инициализируется объект типа ManagementScope, которой представляет область (пространство имен) для управляющих операций.

В строках с 7 по 9 инициализируется объект типа ObjectQuery, который будет возвращать объекты класса Win32_PerfRawData_PerfOS_Memory.

В строках с 10 по 11 инициализируется объект ManagementObjectSearcher возвращает коллекцию объектов на основе указанного запроса в указанном окружении.

В строках с 12 по 13 инициализируется объект типа ManagementObjectCollection, который представляет различные коллекции управляющих объектов, извлекаемых с помощью WMI.

В строках с 14 по 17, в цикле просматриваются все полученные объекты класса Win32_PerfRawData_PerfOS_Memory и извлекается необходимый объект с ключом "AvailableMBytes", который содержит количество доступной оперативной памяти подключённого сервера.

Доступ к инструментарию WMI предоставляется в пространстве имён System.Management.

2.3 Исследование методов анализа и прогнозирования загруженности процессора

Для повышения эффективности управления больших вычислительных систем, будет применяться расчёт аналитических показателей, а так же прогноз загруженности сервера.

Записи журнала (англ. log), предоставленные ООО «Грид инжиниринг» представляют собой информацию о выполняемых процессах на 10 серверах в период с 29.02.2016 по 10.03.2016.

Разобрав предоставленные данные, были получены следующие ряды данных (рисунки с 2.3 по 2.12). На оси абсцисс отмечено время, в секундах, от первой записи журнала. Но оси ординат отмечены показания загруженности процессора в процентах.

Рисунок 2.3 – Загруженность процессора сервера №1

Рисунок 2.4 – Загруженность процессора сервера № 2

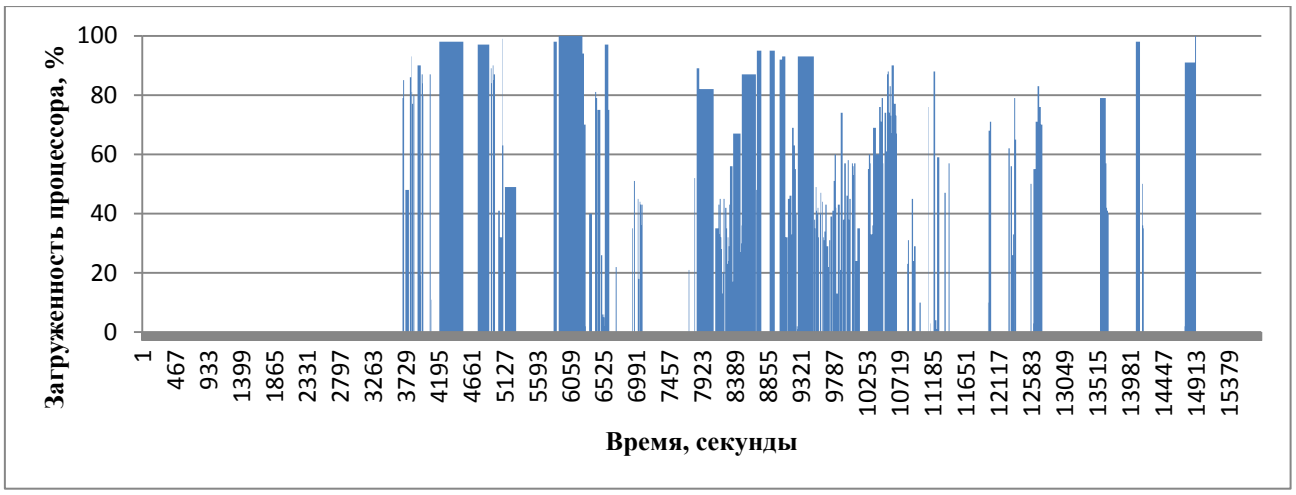


Рисунок 2.5 – Загруженность процессора сервера № 3

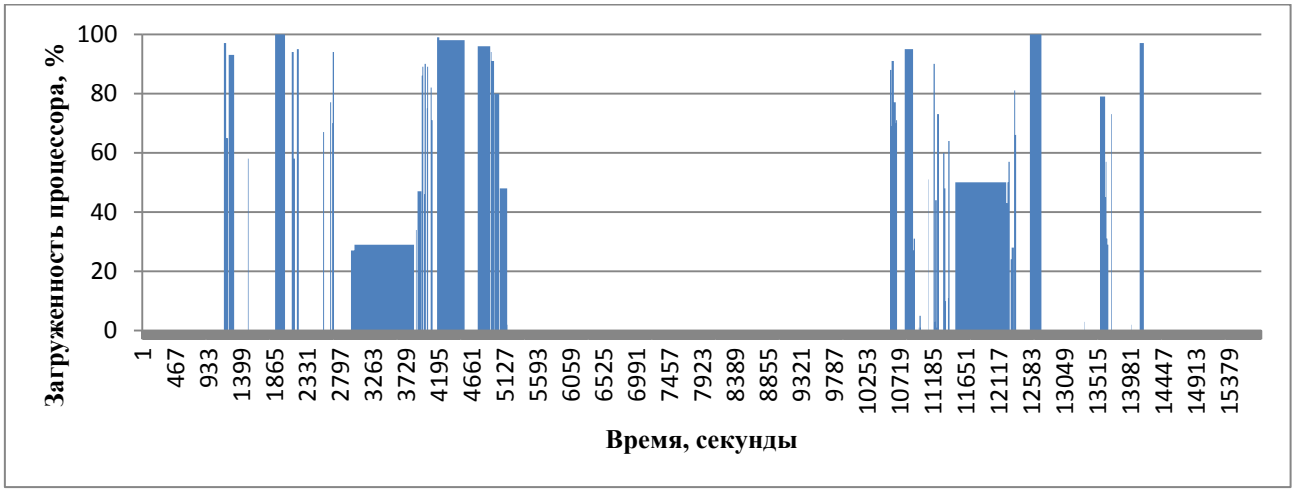


Рисунок 2.6 – Загруженность процессора сервера № 4

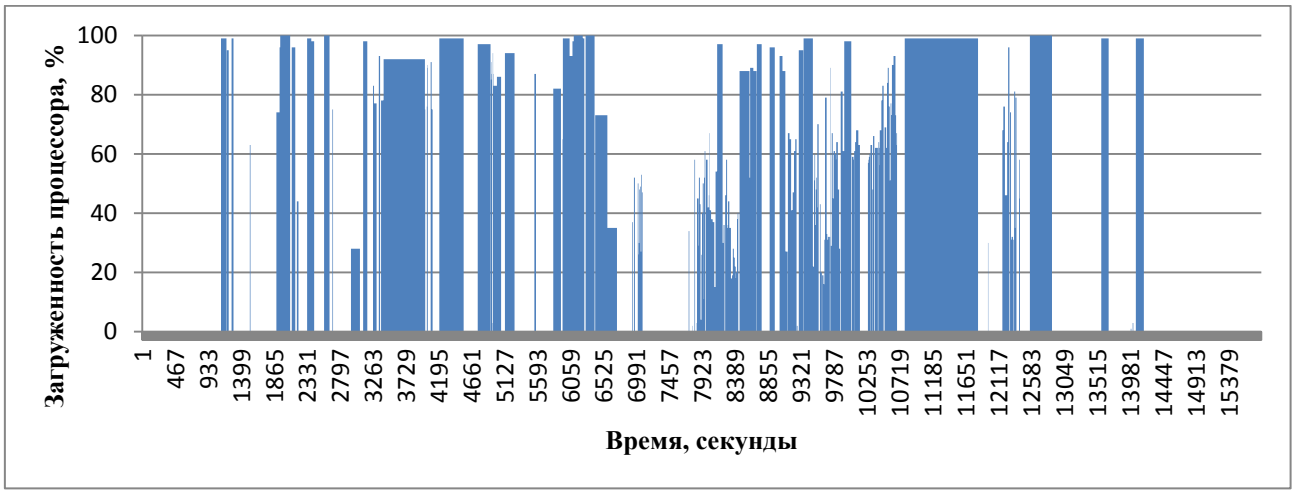


Рисунок 2.7 – Загруженность процессора сервера № 5

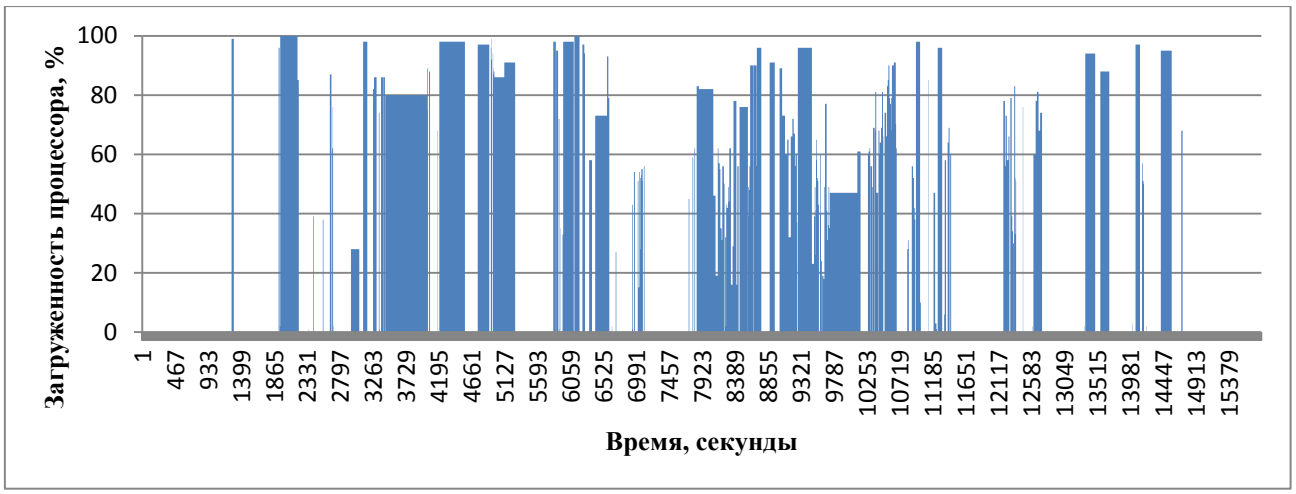


Рисунок 2.8 – Загруженность процессора сервера № 6

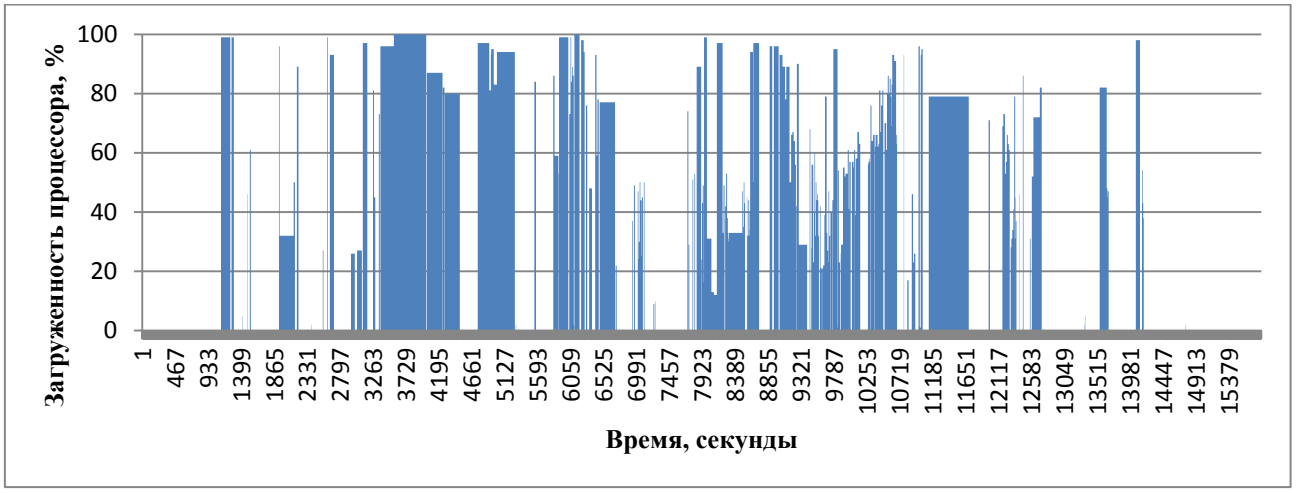


Рисунок 2.9 – Загруженность процессора сервера № 7

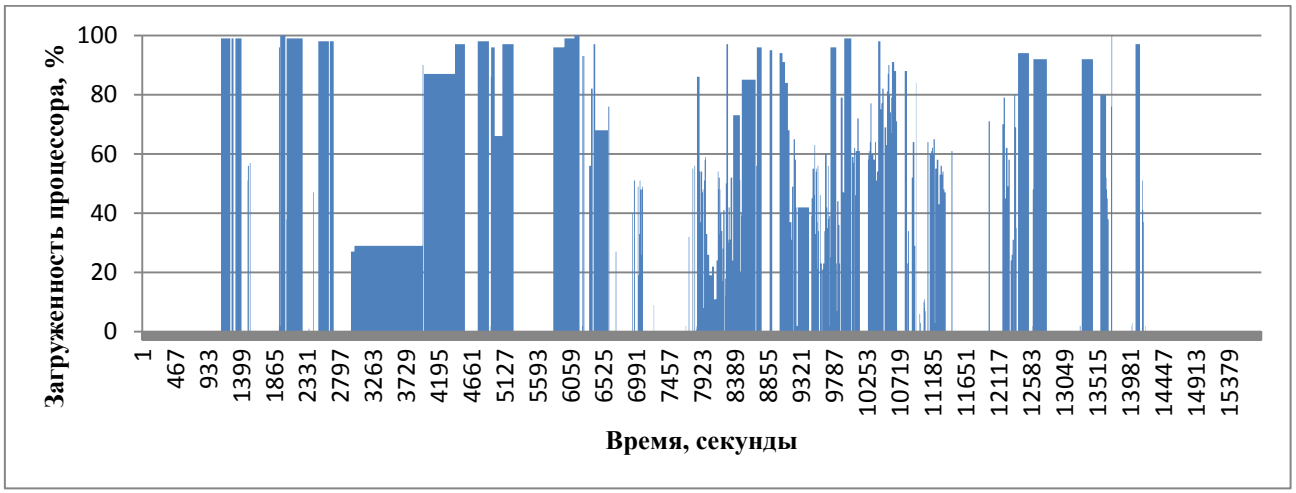


Рисунок 2.10 – Загруженность процессора сервера № 8

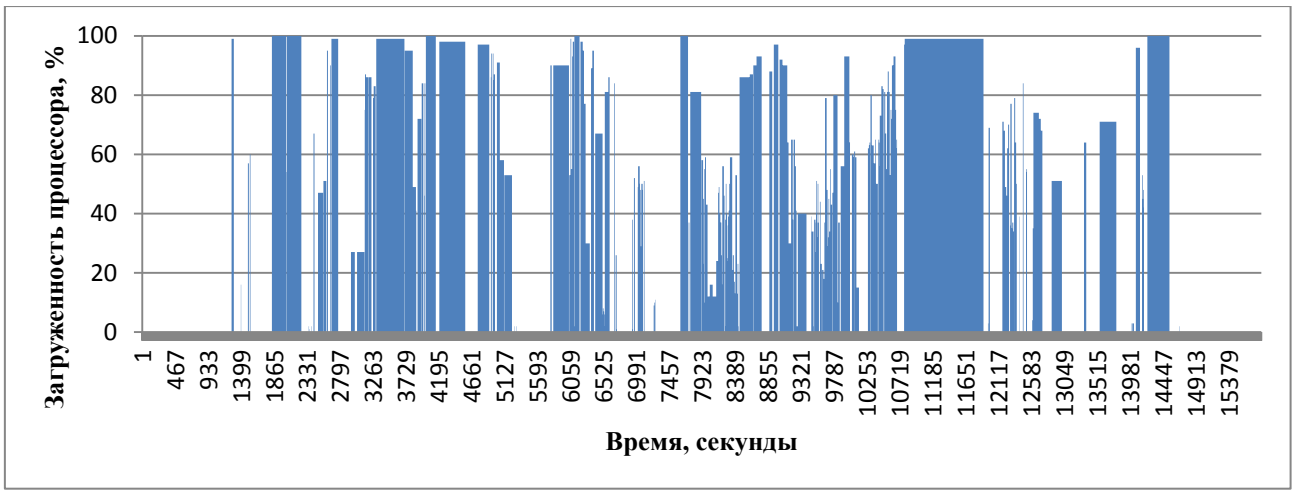


Рисунок 2.11 – Загруженность процессора сервера № 9

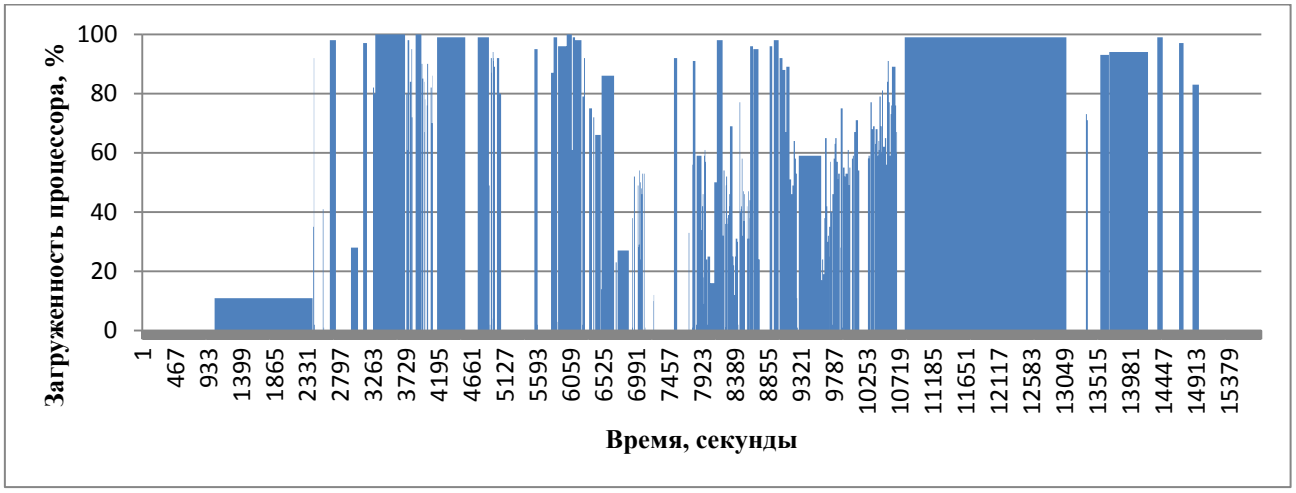


Рисунок 2.12 – Загруженность процессора сервера № 10

Из рисунков с 2.3 по 2.12 видно, что загруженность процессора представляет собой высокочастотный процесс, а интервалы измерения находятся в пределах одной секунды.

Чтобы адекватно анализировать и предсказывать значения подобных рядов данных можно, лишь применяя соответствующие методы сглаживания процесса или подавления высокочастотной составляющей [6].

В работе [6] были также использованы данные загруженности процессора с шагом в одну секунду, данные прогнозировались на шаг вперед. Исходя из структуры предоставленных записей журнала, для адекватного построения моделей полученных временных рядов, а также увеличения временного интервала прогноза, приведём данные к средней загруженности процессора по дням (таблица 2.1).

Прогноз изменения загруженности процессора имеет смысл делать на один шаг вперёд из-за нестационарности графиков загруженности процессора [6].

Следуя работе [6] для моделирования приведённых высокочастотных рядов будем использовать следующие фильтры:

- модель экспоненциального сглаживания;
- модель медианного фильтра.

Таблица 2.1 – Средняя загрузка по серверам в день (в процентах)

	Номер сервера									
Дни	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	3	12	0	7	8	2	10	16	2	3
2	0	0	0	15	27	20	10	34	39	13
3	0	19	17	33	58	53	68	41	67	58
4	16	36	35	37	44	46	44	45	38	36
5	3	44	38	0	48	34	41	40	42	50
6	0	24	33	0	27	32	25	27	37	27
7	0	49	47	0	54	52	44	46	46	49
8	12	30	24	18	70	32	47	35	69	71
9	20	21	11	39	49	12	23	29	44	99
10	0	14	10	9	14	20	10	20	31	54
11	0	6	9	0	0	10	0	0	9	11

Экспоненциальное сглаживание – один из простейших и распространенных приемов выравнивания временного ряда. Экспоненциальное сглаживание можно представить как фильтр, на вход которого последовательно поступают члены исходного ряда, а на выходе формируются текущие значения экспоненциальной средней.

При экспоненциальном сглаживании предыдущие значения параметров учитываются с убывающими по экспоненциальному закону весами [6].

В простейшем случае для нахождения очередного сглаженного значения используются только текущее измерение и предыдущее сглаженное значение. Формула прогнозирования имеет вид:

$$\hat{Y}_{n+1} = (1 - \varepsilon) * Y_n + \varepsilon * \hat{Y}_{n-1},$$

где $\hat{Y}_{n+1}$ – прогнозное значение на следующий период;

Y_n – значение исходного временного ряда в текущий период;

$\hat{Y}_{n-1}$ – прогнозное значение за предыдущий период;

ε – постоянная величина, имеющая смысл коэффициента сглаживания.

Медианный фильтр. Медианные фильтры достаточно часто применяются на практике для предварительной обработки цифровых данных как альтернатива средним арифметическим значениям отсчетов в оценке выборочных средних значений. Медианой числовой последовательности $x_1, x_2, \dots, x_n$ при нечетном n является средний по значению член ряда, получающегося при упорядо-

чивании этой последовательности по возрастанию (или убыванию). Для четных n медиану обычно определяют как среднее арифметическое двух средних отсчетов упорядоченной последовательности.

Медианный фильтр представляет собой оконный фильтр, последовательно скользящий по изменяющимся значениям сигнала и возвращающий на каждом шаге одно из значений, попавших в окно (апертуру) фильтра. Выходной сигнал y_k скользящего медианного фильтра шириной $2 * n + 1$ для текущего отсчета k формируется из входного временного ряда $\dots, x_{k-1}, x_k, x_{k+1}, \dots$ в соответствии с формулой:

$$y_k = med\left(x_{k-n}, x_{k-n-1}, \dots, x_{k-1,} x_k, x_{k+1}, \dots, x_{k+n-1}, x_{k+n}\right),$$

где $med\left(z_{(1)}, \dots, z_{(m)}, \dots, z_{(2n+1)}\right) = z_{(n+1)}, z_{(m)}$ – элементы вариационного ряда, ранжированные в порядке возрастания значений, $z_{(1)} \leq z_{(2)} \leq \dots \leq z_{(2*n+1)}$.

Медианная фильтрация реализуется в виде процедуры локальной обработки отсчетов в скользящем окне, которое включает определенное число отсчетов сигнала. Для каждого положения окна выделенные в нем отсчеты ранжируются по возрастанию или убыванию значений. Средний по своему положению отсчет в ранжированном списке называется медианой рассматриваемой группы отсчетов. Им заменяется центральный отсчет в окне для обрабатываемого сигнала. В силу этого медианный фильтр относится к числу нелинейных фильтров, заменяющих медианным значением аномальные точки и выбросы независимо от их амплитудных значений, и является устойчивым по определению, способным аннулировать даже бесконечно большие отсчеты.

Достоинством медианных фильтров является простота структуры, позволяющая легко реализовать фильтр как аппаратными, так и программными средствами. Медианный фильтр не применяется для ступенчатых и пилообразных функций, однако он хорошо подавляет одиночные импульсные помехи (случайные шумовые выбросы отсчетов и промахи) [6].

Сравним два типа сглаживания, для этого построим обе модели по каждому временному ряду и рассчитаем ошибку прогнозирования по формуле [6]:

$$\vartheta = \sqrt{\sum_{n=1}^{t} \frac{\left(\theta_n - \hat{\theta}_{n-1}\right)^2}{t}},$$

где θ_n – реальное значение временного ряда в текущий момент времени;

$\hat{\theta}_{n-1}$ – прогнозное значение ряда в предыдущий момент времени;

t – количество наблюдений.

Также рассчитаем коэффициент R^2 (коэффициент детерминации) – это доля дисперсии зависимой переменной, объясняемая рассматриваемой моделью зависимости, то есть объясняющими переменными. Коэффициент рассчитывается по формуле [14]:

$$R^2 = \frac{\sum_{n=1}^{t}(y_n - \hat{y}_n)^2}{\sum_{n=1}^{t}(y_n - \bar{y})^2},$$

где y_n – исходные значения временного ряда;

$\hat{y}_n$ – смоделированные значения;

$\bar{y}$ – среднее значение исходного временного ряда.

Для построения моделей используем язык программирования R (листинг 2.2).

Листинг 2.2 – Методы на языке R для построения моделей экспоненциального сглаживания и медианного фильтра

```
1    HoltWinters(midday, alpha = 0.5,beta = F, gamma = F )
2    medianFilter(md$V10, windowSize = 3)
```

В строке 1 находится функция построения модели экспоненциального сглаживания, где midday – исходный ряд данных, alpha – параметр сглаживания, коэффициенты beta и gamma не участвуют в построении модели.

В строке 2 находится функция построения модели медианного фильтра, где md$v10 – исходный ряд данных, windowSize – ширина окна.

Таблица 2.2 – Средние ошибки прогнозирования и коэффициенты R^2 для методов экспоненциального сглаживания и медианного фильтра

	Номер сервера									
	1	2	3	4	5	6	7	8	9	10
ϑ для модели экспоненциального сглаживания, %	9,48	17,91	19,71	21,26	29,94	23,25	25,93	16,79	28,79	34,84
ϑ для модели медианного фильтра, %	**7,50**	**8,53**	**9,34**	**14,68**	**16,73**	**13,64**	**19,90**	**8,89**	**14,55**	**22,44**
R^2 для модели экспоненциального сглаживания, %	6,10	8,26	16,37	3,83	0,08	0,50	0,26	1,61	3,07	4,15
R^2 для модели медианного фильтра, %	**70,17**	**57,33**	**91,94**	**81,96**	**80,88**	**79,92**	**81,34**	**79,44**	**70,23**	**69,29**

Как видно из таблицы 2.2 модель медианного фильтра показывает меньшую ошибку прогнозирования, чем модель экспоненциального сглаживания.

Также коэффициент R^2 для модели медианного фильтра значительно выше, чем у модели экспоненциального сглаживания, что говорит о лучшем качестве модели.

Следовательно, в разрабатываемом ПО будет предпочтительнее реализация прогнозирования методом медианного фильтра.

Выводы по главе два

Рассмотрено проведённое ранее исследование методов моделирования и прогнозирования загруженности процессора и, исходя из выводов работы [6], предложен и исследован новый подход к решению задачи прогнозирования загруженности процессора.

Проведено сравнение двух методов моделирования и прогнозирования высокочастотных временных рядов. Для реализации в ПО выбран метод медианного сглаживания.

Рассмотрен инструментарий мониторинга серверов под управлением MS Windows 2008 R2 Enterprise, который будет использован в реализации ПО.

3 Программная реализация прикладного обеспечения для мониторинга и анализа серверов

3.1 Обзор используемых средств разработки

Так как сервера, для которых предназначается реализуемое прикладное обеспечение (ПО) находятся под управлением Microsoft Windows Server 2008 R2 Enterprise, то было решено, что основой послужат программная платформа «.Net Framework», интерфейс программирования приложений (API) «Windows Forms», язык программирования «C#» и интегрирования среда разработки (IDE) «Visual Studio 2012». Для реализации метода прогнозирования будет использоваться библиотека RDotNet.

Microsoft .Net Framework (рисунок 3.1) является так называемой программной платформой. В общих чертах можно провести аналогию с видеофайлами, которые не будут воспроизводиться если в системе не установлен нужный кодек. В данном случае видеофайл — это программа, написанная с использованием технологии .Net, а кодек — это сама платформа Microsoft .Net Framework. Причем для работы приложения, написанного на конкретной версии фреймворка, требуется установка именно этой версии, чтобы разработчик мог максимально абстрагироваться от системного окружения на компьютере пользователя, таким образом, разработчика не интересует, операционная система, разрядность и архитектура процессора компьютера пользователя и т.д. Для запуска программы достаточно чтобы под данную систему существовала и была установлена реализация .Net Framework. Для операционных систем Windows разработкой платформы занимается её создатель, компания Microsoft. Существуют также независимые реализации, прежде всего это Mono и Portable.NET, позволяющие запускать программы .Net на других операционных системах, например на Linux [15].

Windows Forms позволяет разрабатывать интеллектуальные клиенты. *Интеллектуальный клиент* — это приложение с полнофункциональным графическим интерфейсом, простое в развертывании и обновлении, способное рабо-

тать при наличии или отсутствии подключения к Интернету и использующее более безопасный доступ к ресурсам на локальном компьютере по сравнению с традиционными приложениями Windows. Windows Forms — это технология интеллектуальных клиентов для .NET Framework (рисунок 3.1). Она представляет собой набор управляемых библиотек, упрощающих выполнение стандартных задач, таких как чтение из файловой системы и запись в нее. С помощью среды разработки типа Visual Studio можно создавать интеллектуальные клиентские приложения Windows Forms, которые отображают информацию, запрашивают ввод от пользователей и обмениваются данными с удаленными компьютерами по сети. При выполнении пользователем какого-либо действия с формой или одним из ее элементов управления создается событие. Приложение реагирует на эти события с помощью кода и обрабатывает события при их возникновении. Windows Forms включает широкий набор элементов управления, которые можно добавлять на формы: текстовые поля, кнопки, раскрывающиеся списки, переключатели и даже веб-страницы. В состав Windows Forms входят многофункциональные элементы пользовательского интерфейса, позволяющие воссоздавать возможности таких сложных приложений, как Microsoft Office [16].

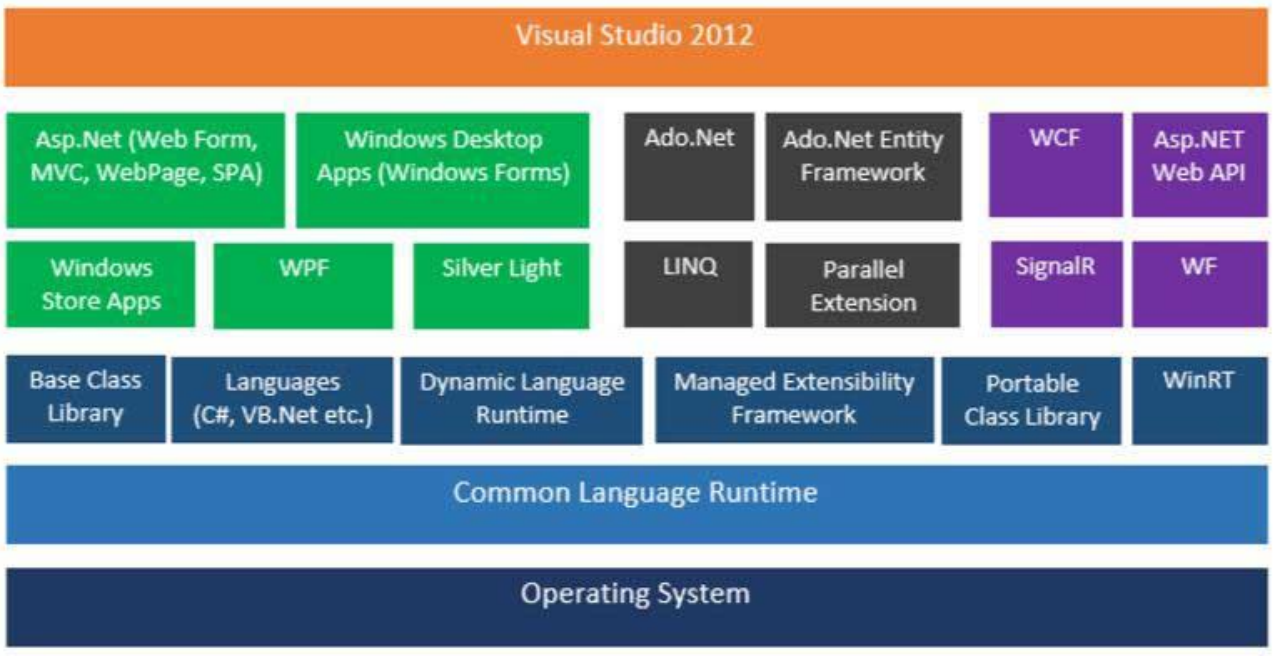


Рисунок 3.1 – Архитектура .NET Framework

Язык C# — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и, в особенности, Java — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов) [17].

Microsoft Visual Studio (рисунок 3.2) — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

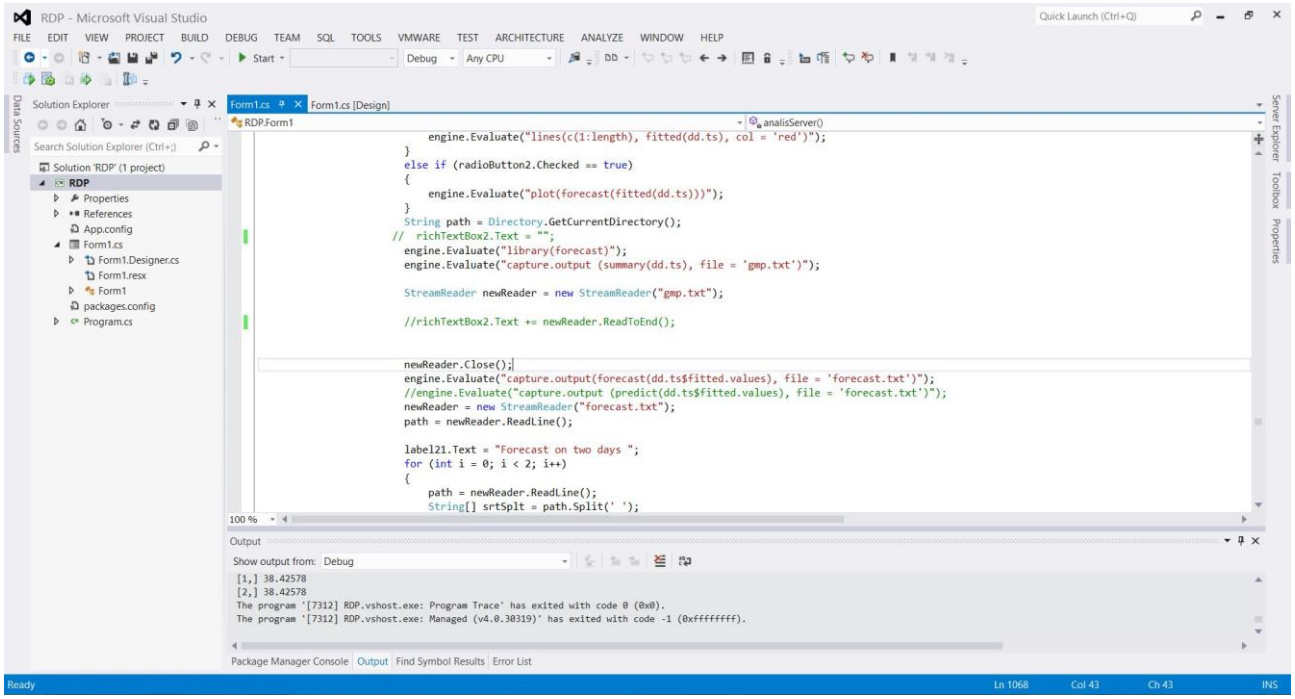


Рисунок 3.2 – Внешний вид рабочей области Microsoft Visual Studio 2012

Среда разработки Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server) [18].

RDotNet это мост функциональной совместимости языка R и .Net Framework. Для работы RDotNet требуется .NET Framework 4, библиотеки RDotNet, а

также установленная среда R. RDotNet работает под управлением ОС Windows, Linux, MacOS.

3.2 Реализация алгоритмов мониторинга и анализа данных сервера

В работе первостепенной задачей стала реализация подключения ПО к серверу. Данную задачу можно реализовать, используя инструментарий управления Windows (WMI).

Листинг 3.1 – Глобальные объекты для реализации подключения к серверу

```
1    ManagementScope myScope;
2    ConnectionOptions options;
3    List<ObjectQuery> query = new List<ObjectQuery>(6);
4    List<ManagementObjectSearcher> searcher = new
5    List<ManagementObjectSearcher>(6);
6    List<ManagementObjectCollection> queryCollection = new
7    List<ManagementObjectCollection>(6);
```

В строке 1 листинга 3.1 содержится объект типа ManagementScope, которой представляет область (пространство имен) для управляющих операций [5].

В строке 2 содержится объект ConnectionOptions, которой задает все параметры, обязательные для установки WMI-подключения [5].

В строке с 3 инициализируется объект типа List<>, содержащий в себе объекты типа ObjectQuery.

В строках с 4 по 5 содержится объект типа List<>, содержащий в себе объекты типа ManagementObjectSearcher, который извлекает коллекцию управляющих объектов в соответствии с заданным запросом.

В строках с 6 по 7 содержится объект типа List<>, содержащий в себе объекты типа ManagementObjectCollection, который представляет различные коллекции управляющих объектов, извлекаемых с помощью WMI .

Данные объекты необходимы, чтобы реализовать подключение к серверу.

Для подключения к серверу используются данные вводимые пользователем в файл cfg.txt. Для работы программы файл должен находиться в той же директории, что и исполняемый файл программы.

Листинг 3.2 – Алгоритм парсинга файла cfg.txt

```
1    if (File.Exists("cfg.txt") == true)
2    {
3    listBox2.Items.Clear();
4    StreamReader myReader = new StreamReader("cfg.txt");
5    String stringForSplit = null;
6    stringForSplit = myReader.ReadLine();
7    timer1.Interval = Convert.ToInt32(stringForSplit);
8    while ((stringForSplit = myReader.ReadLine()) != null{
9    String[] myMass = stringForSplit.Split(' ');
10   String forScope = "\\\\" + myMass[0] + "\\root\\cimv2";
11   ConnectionOptions options1 = new ConnectionOptions();
12   options1.Username = myMass[1];
13   options1.Password = myMass[2];
14   ManagementScope myScope1 =
15   new ManagementScope(forScope, options1);
16   listForScopes.Add(myScope1);
17   try{
18   myScope1.Connect();
19   listBox2.Items.Add(string.Format("{0} online",
20   myMass[0]));
21   }
22   catch
23   {
24   listBox2.Items.Add(string.Format("{0} offline",
25   myMass[0]));
26   }}
27   myReader.Close();            }
28   else
29   {
30   MessageBox.Show("File not found!",
31   "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);}
```

В строке 1, проверяем, присутствует ли файл cfg.txt в папке с исполняемым файлом программы. При его отсутствии сработают строки с 28 по 30 – пользователь получит соответствующее сообщение об ошибке.

В строке 3 удаляем предыдущие значения в ListBox, где выводится список серверов, доступных для подключения и анализа.

В строке 4 создаем новый поток StreamReader для чтения файла cfg.txt.

В строке 5 создаем объект String для записи строк из файла cfg.txt.

В строке 6 считываем первую строку из файла cfg.txt. Первое значение есть интервал обновления данных мониторинга (в миллисекундах). Это значение в строке 7 записывается в интервал объекта Timer.

Строки с 8 по 28 продолжают считывать оставшиеся строки в файле cfg.txt. В этих строках парсятся все ip-адресов подключаемых серверов, их логины и пароли:

- в строке 9 в массив типа String записываются элементы считываемой строки, разделённые точкой;
- в строке 10 в переменную типа String записывается с соответствующем форматированием ip-адрес для подкдючения к серверу;
- в строке 11 инициализируется объект типа ConnectionOptions для записи опций подключения к серверу, а именно логин и пароль;
- в строках 12 и 13 записываются в объект типа ConnectionOptions логин пароль соответственно;
- в строках 14 и 15 инициализируется объект типа ManagementScope, которой представляет область (пространство имен) для управляющих операций [5].
- в строке 16, новое окружение, созданное на предыдущем шаге записывается в объект типа List<>;
- в строках с 17 по 26 программа пробует подключиться к серверу: если текущий сервер доступен, то он добавится в ListBox со статусом online (строки с 19 по 20), иначе, сервер добавится в ListBox со статусом offline;
- в строке 27 закрывается поток на чтение, открытый в строке 4.

Так же логика алгоритма представлена на рисунке 3.3.

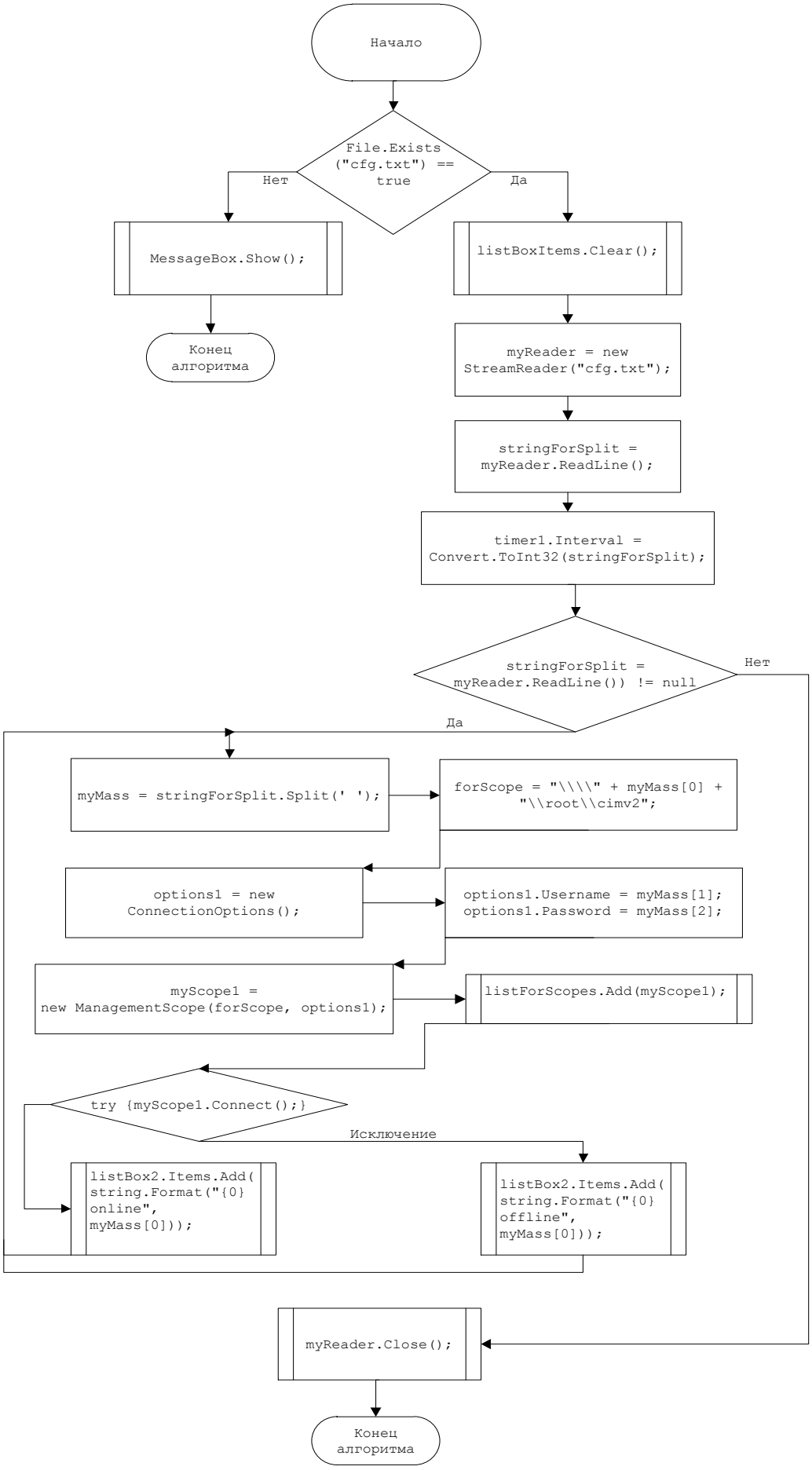


Рисунок 3.3 – Парсинг файла cfg.txt

После парсинга файла cfg.txt, станет доступен список серверов для подключения.

Листинг 3.3 – Алгоритм подключения к выбранному серверу

```
1    if (listBox2.SelectedIndex != -1)
2    {
3    String strForConnectButton =
4    listBox2.Items[listBox2.SelectedIndex].ToString();
5    String[] forCheck = strForConnectButton.Split(' ');
6    String[] whichServer = forCheck[0].Split('.');
7    if (forCheck[1] == "offline")                {
8    MessageBox.Show("Selected server is offline!
9    \nAnalisis avalible", "Error",
10   MessageBoxButtons.OK, MessageBoxIcon.Error);
11   selPath = whichServer[whichServer.Length - 1];
12   parserFolder();
13   }
14   else
15   {
16   selPath = whichServer[whichServer.Length - 1];
17   tabPage1.Show();
18   myScope = listForScopes[listBox2.SelectedIndex];
19   try
20   {
21   myScope.Connect();
22   timer1.Enabled = true;
23   query[1] = new ObjectQuery("select * from
24   Win32_PerfFormattedData_PerfProc_Process");
25   query[3] = new ObjectQuery("select * fro
26   Win32_Processor");
27   searcher[1] = new ManagementObjectSearcher(myScope,
28   query[1]);
29   searcher[3] = new ManagementObjectSearcher(myScope,
30   query[3]);
31   queryCollection[1] = searcher[1].Get();
32   queryCollection[3] = searcher[3].Get();
33   foreach (ManagementObject m in queryCollection[1])
34   {
35   listBox1.Items.Add(string.Format("{0}.exe", m["Name"]));
36   }
37   foreach (ManagementObject m in queryCollection[3])
38   {
39   label2.Text = string.Format("{0}", m["Name"]);
40   NumOfCores =
41   Convert.ToInt32(m["NumberOfLogicalProcessors"]);}
42   parserFolder();}
43   catch{
44   MessageBox.Show("Cannot connect to server!", "Error",
45   MessageBoxButtons.OK, MessageBoxIcon.Error);
46   }}}
```

Алгоритм описанный в листинге 3.3 находится в событии «Click» кнопки «Connect».

В строке 1 проверяется исключение «ArgumentOutOfRangeException» поля «ListBox.SelectedIndex», это даёт гарантию того, что индекс выбранного элемента в ListBox не выйдет за границы массива, содержащегося в ListBox.

В поле Items объекта ListBox, содержится статус сервера (online/offline), поэтому выбранный элемент парсится в строках с 4 по 6 и проверяется в строке 7. Если выбранный сервер онлайн, то пользователь получит соответствующее сообщение об ошибке. Однако функция анализа выбора сервера будет также доступна после вызова функции в строке 12.

Иначе, вкладка автоматически переключится с Servers на Info с помощью метода «Show()» в строке 17.

В строке 18, глобальной переменной myScope класса ManagementScope присваивается именно те настройки окружения, что соответствуют выбранному серверу.

Со строки 19 по 42, в блоке try происходит подключении к выбранному серверу, если в блоке произойдёт исключение, то сработает блок catch в строках с 43 по 46, и пользователь получит соответствующее сообщение об ошибке.

Также логика алгоритма представлена на рисунке 3.4.

После подключения к серверу, для анализа его логов, должна пропарситься соответствующая папка для последующего анализа сервера. Данный алгоритм представлен в приложении 1.

В строке со 2 по 3 инициализируется массив типа string, содержащий все именя файлов с логами выбранного сервера.

Со строки 9 по 103, в цикле, проводится парсинг каждого файла лога, находящегося в папке, принадлежащей определённому серверу. Цель парсинга – вычленить дату лога, загрузку процессора и время исполнения задачи. Пример лога представлен в листинге 3.4.

Так же, в приложении 2, представлена блок-схема алгоритма парсинга папки с логами.

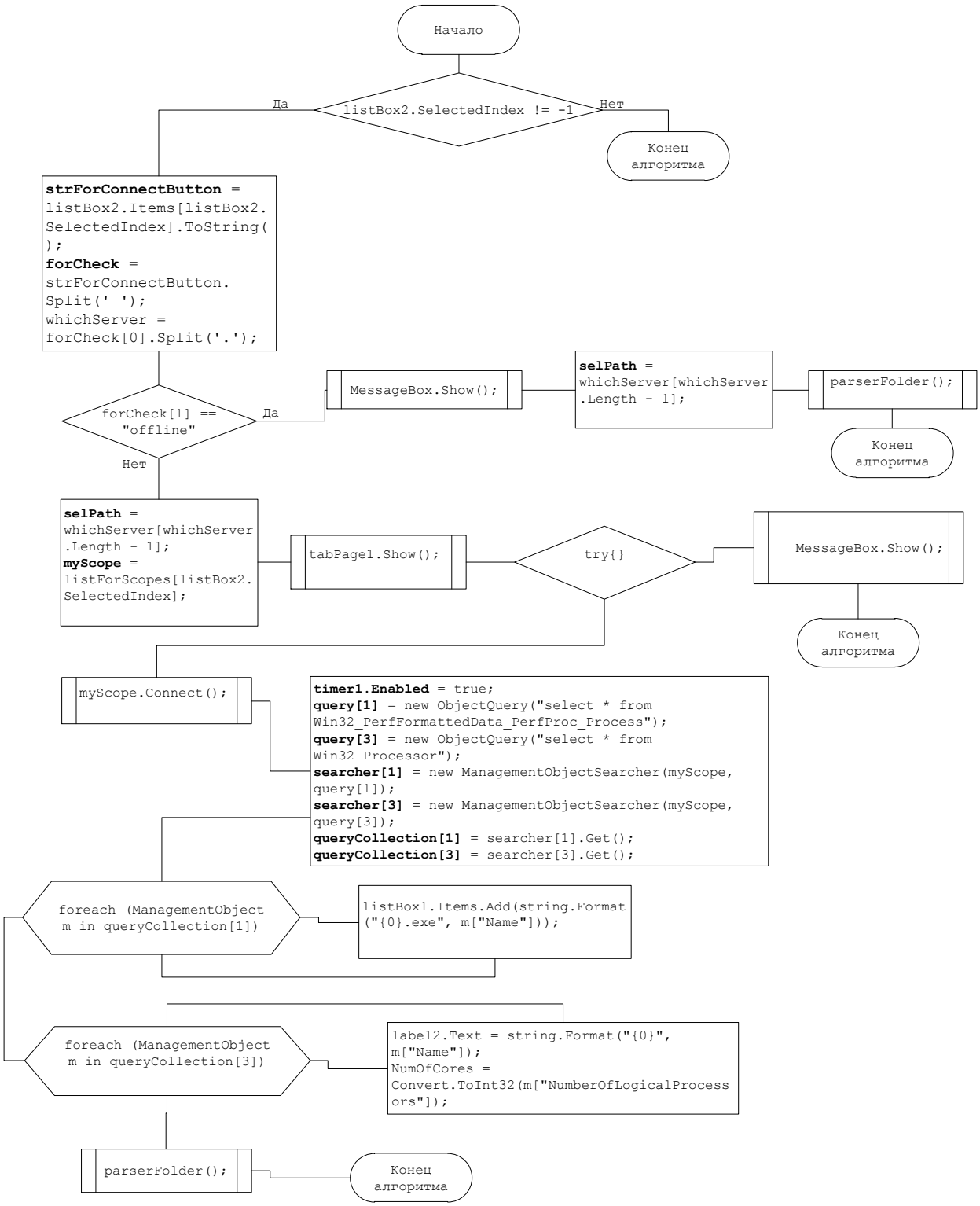


Рисунок 3.4 – Блок-схема алгоритма подключения к выбранному серверу

Листинг 3.4 – Пример лога

```
1    === F20152631_8a858084-530d8065-0153-3f9dc35c-0c17_0-6,
2    Chunk
3    1 - 3/4/2016 3:14:01 AM ==============
4    Scenes:   P:\ftpuser\F20152631\kuhna\kuhna.max
5    Section:  Frames: 0.000 - 0.000, Region: Entire image
6    Renderer: 3ds Max/2014
7    === Started
8    "C:\Program Files\Autodesk\3ds Max 2014\3dsmaxcmd.exe" -
9    v:4 -
10   preRenderScript:P:/tasks/8a858084-530d8065-0153-3f9dc35c-
11   0c17/preRenderScript.ms -
12   workPath:P:/ftpuser/F20152631/kuhna
13   -outputName:P:/tasks/8a858084-530d8065-0153-3f9dc35c-
14   0c17/8a858084-530d8065-0153-3f9dc35c-
15   0c17_result/scene.EXR -
16   start:0 -end:0 -nthFrame:1 -frames:0-0 -continueOnError -
17   gammaCorrection:1 -gammaValueIn:2.200 -gammaValueOut:1 -
18   rfw:0
19   P:/ftpuser/F20152631/kuhna/kuhna.max
20   3/4/2016 3:14:02 AM;  1 frames initialized
21   3/4/2016 3:14:02 AM;
22   Max install location: C:\Program Files\Autodesk\3ds Max
23   2014\
24   3/4/2016 3:14:02 AM;  Max file being rendered:
25   P:/ftpuser/F20152631/kuhna/kuhna.max
26   3/4/2016 3:14:02 AM;  Renderer: Corona 1.3
27   3/4/2016 3:14:55 AM;  Error rendering frame 0: An unex
28   pected
29   exception has occurred in the network renderer and it is
30   ter
31   minating.
32   3/4/2016 3:14:56 AM;  Job Completed with Error(s) - see
33   above
34   3/4/2016 3:14:56 AM;  Max is down
35   3/4/2016 3:14:56 AM;  Error occured while rendering job.
36   === F20152631_8a858084-530d8065-0153-3f9dc35c-0c17_0-6,
37   Chunk
38   1 - 3/4/2016 3:14:57 AM ==============
39   Duration: 55 sec         Avg. CPU usage: 2%       CPU
40   idle
41   time: 54 sec
42   Result:   Erroneous
43   Error:    The renderer returned an error-code (0x3)
44   === Done
```

Также, в приложении 2, представлена блок-схема парсинга папки с логами.

После подключения к серверу станет доступен анализ сервера:

• получение аналитических показателей о работе сервера;

• прогнозирование загруженности процессора на один шаг вперёд от выбранного промежутка.

В приложении 3 представлена программная реализация расчёта аналитических показателей о работе выбранного сервера и прогноза загруженности процессора.

В приложении 4 представлена блок-схема работы алгоритма расчёта аналитических показателей о работе выбранного сервера и прогноза загруженности процессора.

3.3 Пользовательский интерфейс

ПО для пользователя представляет из себя форму с тремя вкладками:

• Analysis – вкладка с элементами управления для просмотра аналитической информации и вывода графика с прогнозом загруженности процессора.

• Info – вкладка с обновляемой информацией в реальном времени информации о загруженности процессора, текущих задачах, количестве свободной оперативной памяти, загруженности локальной сети выбранного сервера.

• Servers – вкладка для выбора желаемого сервера, с которого будет собираться информация мониторинга.

Для пользователя работа с программой осуществляется следующим образом:

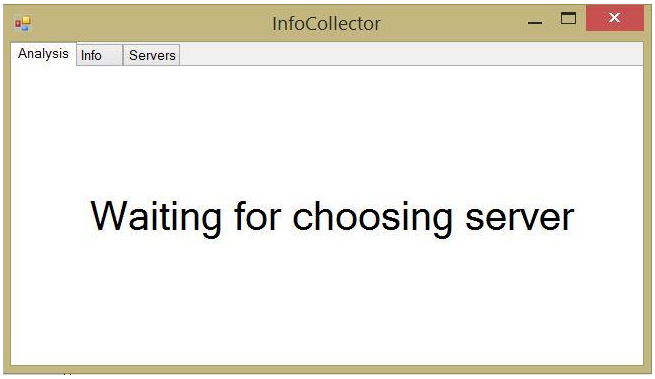


Рисунок 3.5 – Вкладка Analysis до подключения к серверу

На вкладке Analysis до подключения к серверу пользователь видит надпись «Waiting for choosing server» (ожидание выбора сервера) (рисунок 3.3).

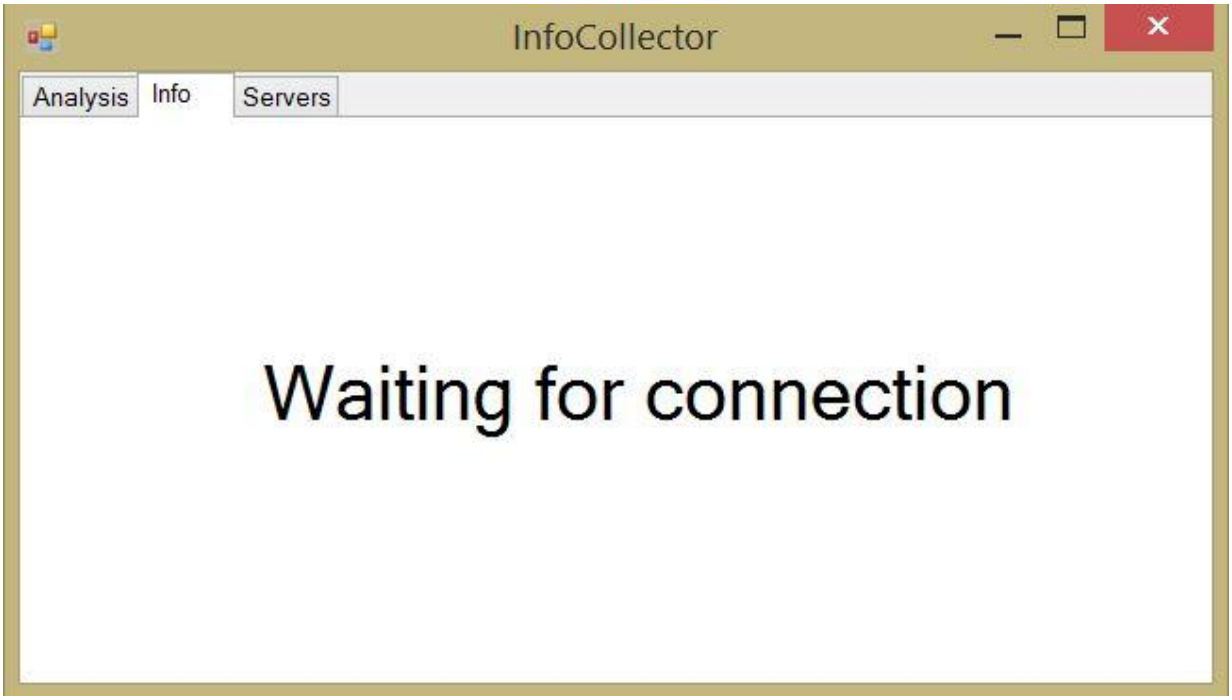


Рисунок 3.4 – Вкладка Info до подключения к серверу

На вкладке Info до подключения к серверу пользователь видит сообщение «Waiting for connection» (ожидание соединения) (рисунок 3.4).

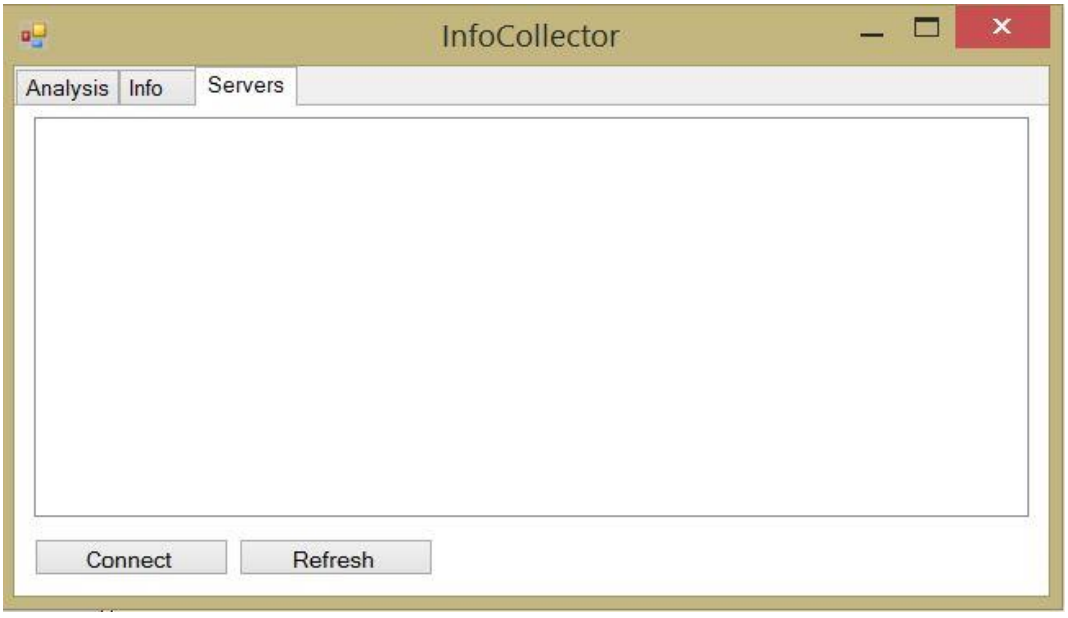


Рисунок 3.5 – Вкладка Servers до обновления списка серверов

После первого запуска программы вкладка Servers остаётся пустой до обновления списка серверов (рисунок 3.5).

Список серверов обновляется с помощью заполненного пользователем файла cfg.txt.

Листинг 3.1 – Файл cfg.txt

```
1    4000
2    192.168.141.1 Администратор1 aA1234
3    192.168.141.2 Администратор2 aA1234
4    192.168.141.3 Администратор3 aA1234
5    192.168.141.4 Администратор4 aA1234
6    192.168.141.5 Администратор5 aA1234
7    192.168.141.6 Администратор6 aA1234
8    192.168.141.7 Администратор7 aA1234
9    192.168.141.8 Администратор8 aA1234
10   192.168.141.9 Администратор9 aA1234
11   192.168.141.10 Администратор10 aA1234
```

В файле cfg.txt (листинг 3.1) находится следующая информация:

- время обновления информации мониторинга в миллисекундах (строка 1);
- в строках с 2 по 11 содержатся параметры для подключения к серверам:
  - первое, до пробела, значение в строке содержит ip-адрес сервера;
  - второе значение содержит логин;
  - третье значение содержит пароль для входа на сервер.

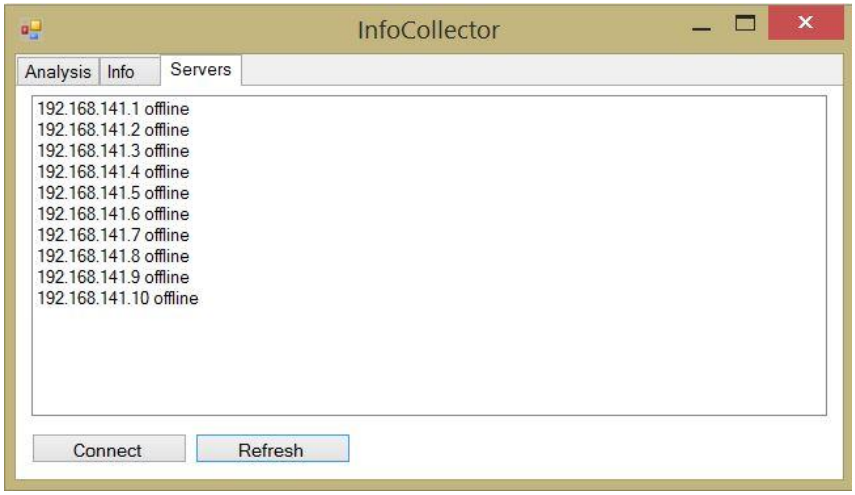


Рисунок 3.6 – Вкладка Servers после обновления

После обновления вкладки Servers, все записанные сервера в файле cfg.txt отобразятся на вкладке (рисунок 3.6).

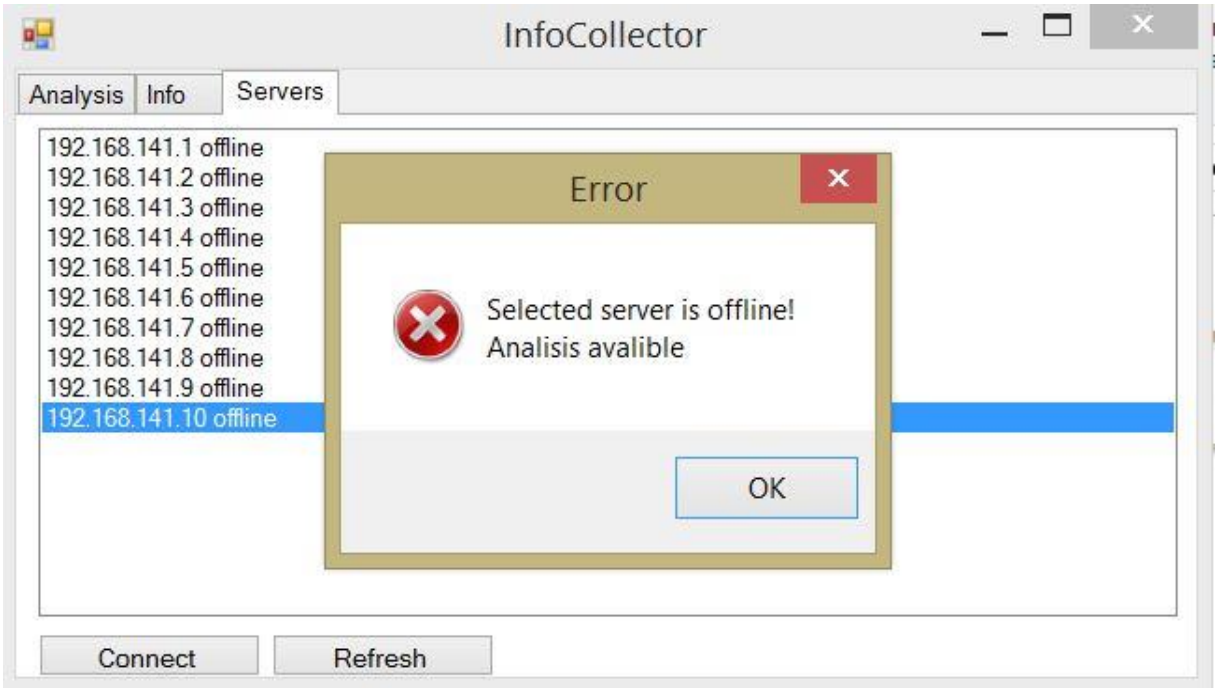


Рисунок 3.7 – Сообщение о статусе «offline» выбранного сервера

Если выбранный сервер offline (не в сети), то пользователь получит соответствующие сообщение (рисунок 3.7), в таком случае мониторинг сервера будет недоступен, однако, аналитическая информация всё также будет предоставляется пользователю на соответствующей вкладке.

Рисунок 3.8 – Сообщение об отсутствии файла cfg.txt в рабочей директории

Если файл cfg.txt отсутствует в рабочей директории программа выдаст соответствующую ошибку (рисунок 3.8).

Выбор сервера осуществляется одинарным кликом по желаемому серверу, затем кликом по кнопке «Connect» или же просто двойным кликом по желаемому серверу.

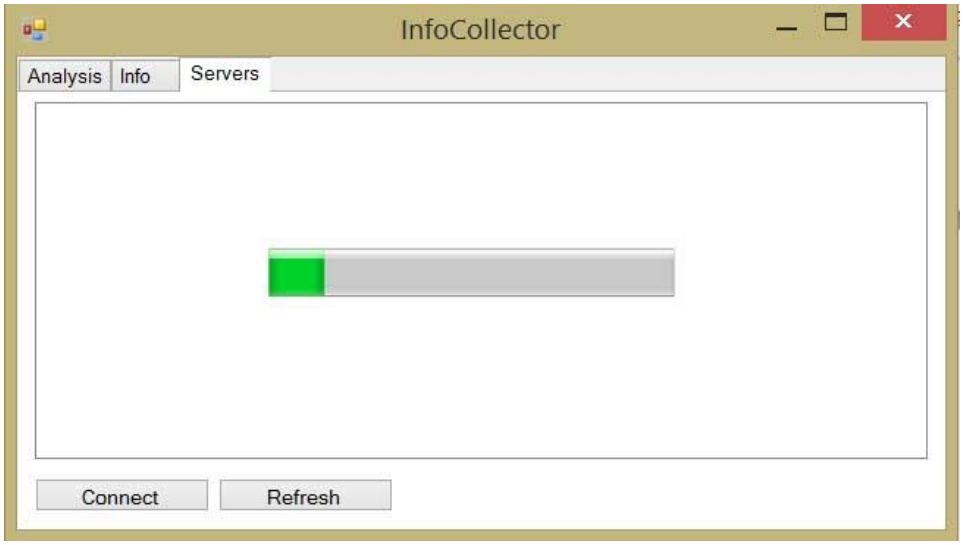


Рисунок 3.9 – Загрузка информации о выбранном сервере

После выбора сервера пользователь увидит полосу загрузки информации (рисунок 3.9).

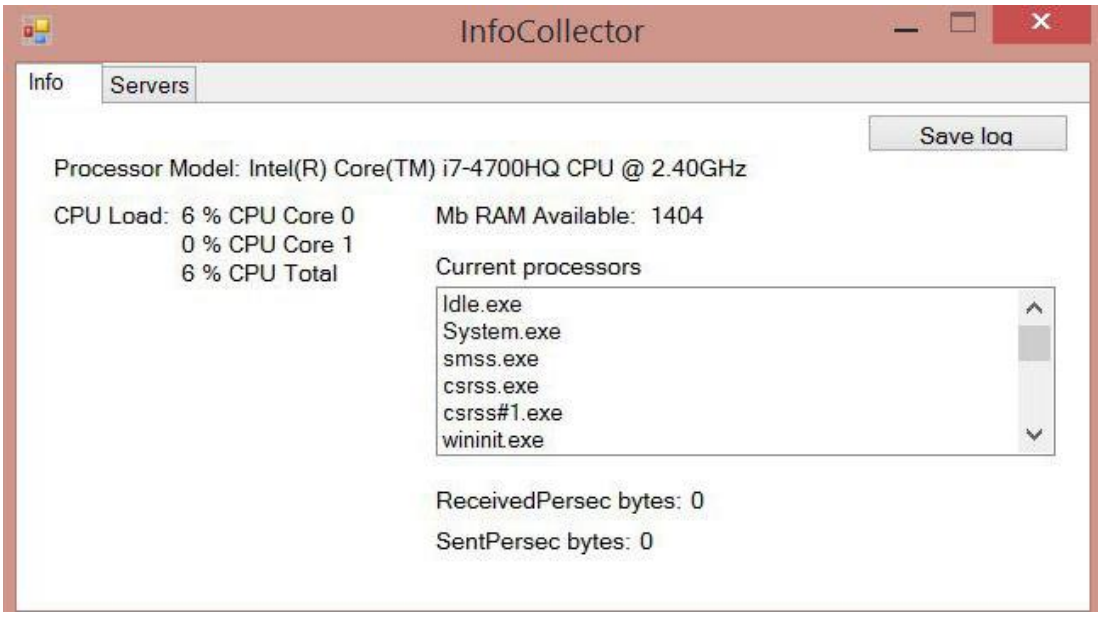


Рисунок 3.10 – Вкладка Servers после подключения к серверу

После подключения к выбранному серверу на вкладке Servers появится информация мониторинга обновляемая с соответствующей частотой (в миллисекундах), указанной в первой строке файла cfg.txt (строка 1 листинг 3.1).

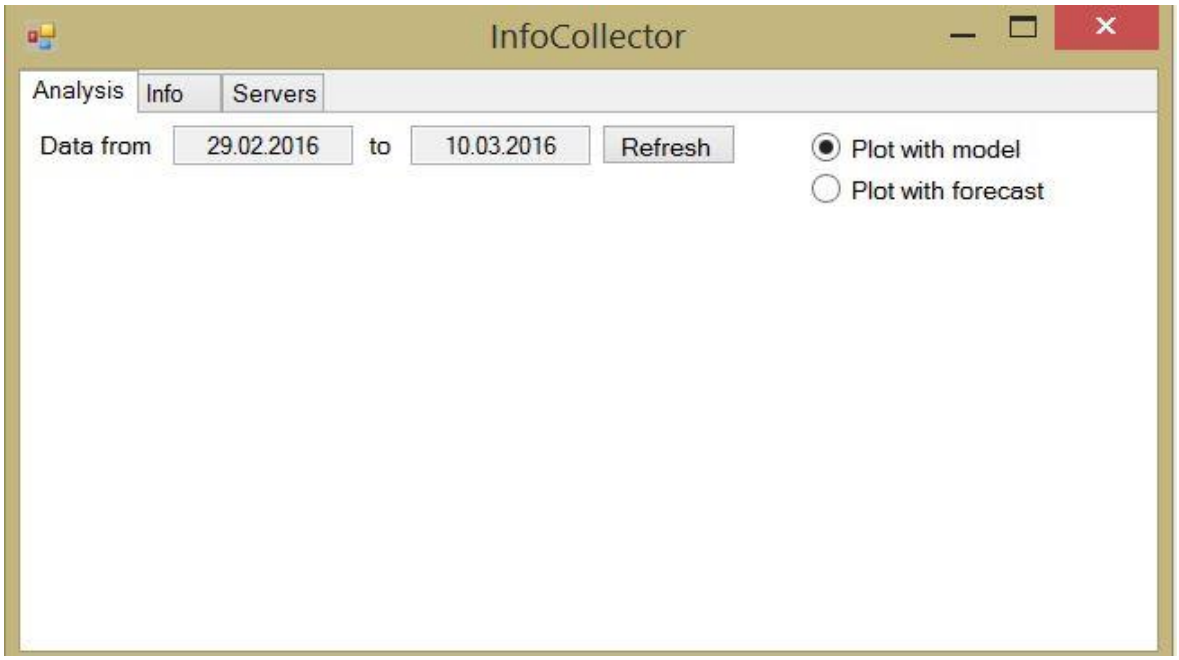


Рисунок 3.11 – Вкладка Servers до обновления аналитической информации

После выбора желаемого сервера, на вкладке Analysis появляются поля для выбора временного промежутка (в днях), который требуется проанализировать (рисунок 3.11).

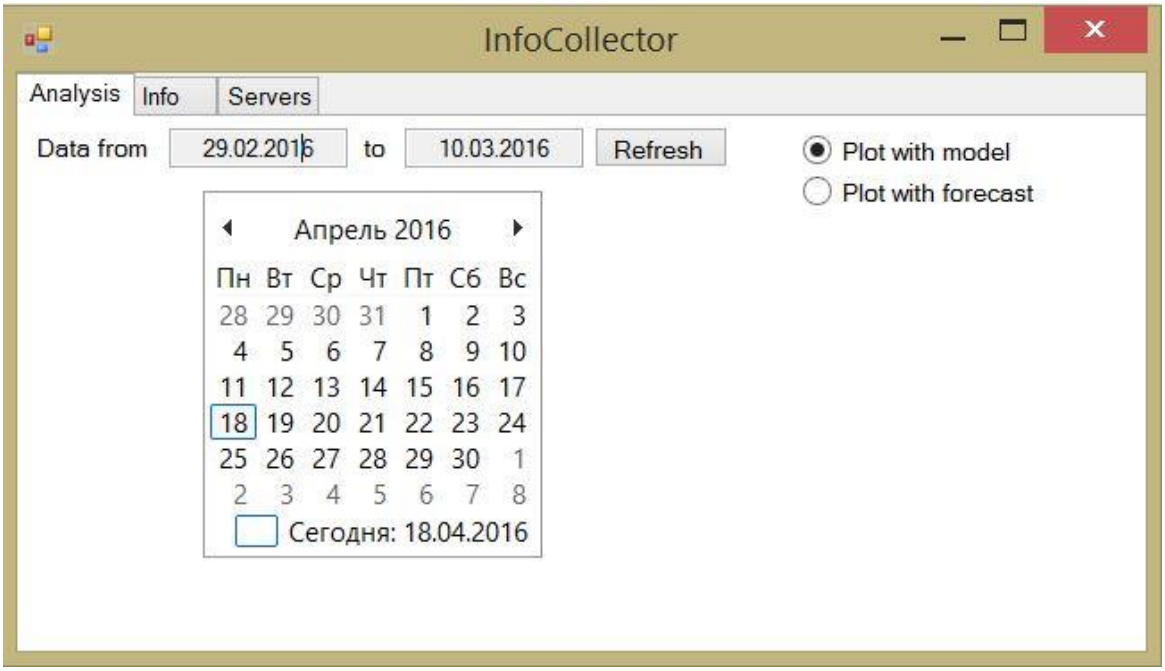


Рисунок 3.12 Календарь для выбора временного промежутка

Для удобства поля для выбора временного промежутка заполняются с помощью календаря, вызываемого по клику на поле выбора даты (рисунок 3.12).

По умолчанию поля выбора даты заполняются минимальной и максимальной датами существующих логов.

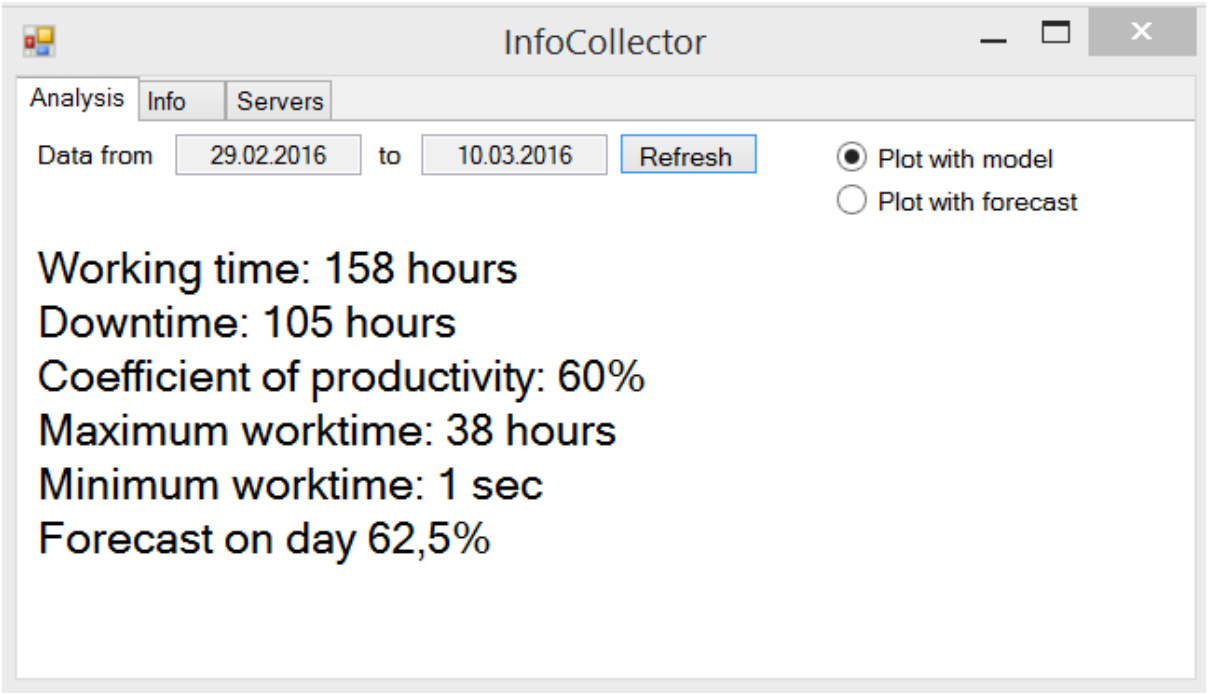


Рисунок 3.13 – Обновлённая аналитическая информация о выбранном сервере

После обновления аналитической информации, на вкладке Analysis появляется следующая информация (рисунок 3.13):

- Working time – информация о времени работы сервера;
- Downtime – информация о времени простоя сервера;
- Coefficient of productivity –коэффициент продуктивности сервера, который вычисляется по формуле:

$$\text{Коэффициент продуктивности} = \frac{\text{Время работы}}{\text{Общее время работы}} * 100\%;$$

- Maximum worktime – максимальное время работы процессора;
- Minimum worktime – минимальное время работы сервера;
- Forecast on two days – прогнозные значения на два дня вперёд.

Также, на вкладке имеются кнопки Plot with model (график с моделью) и Plot with forecast (график с прогнозом) для вывода соответствующих графиков.

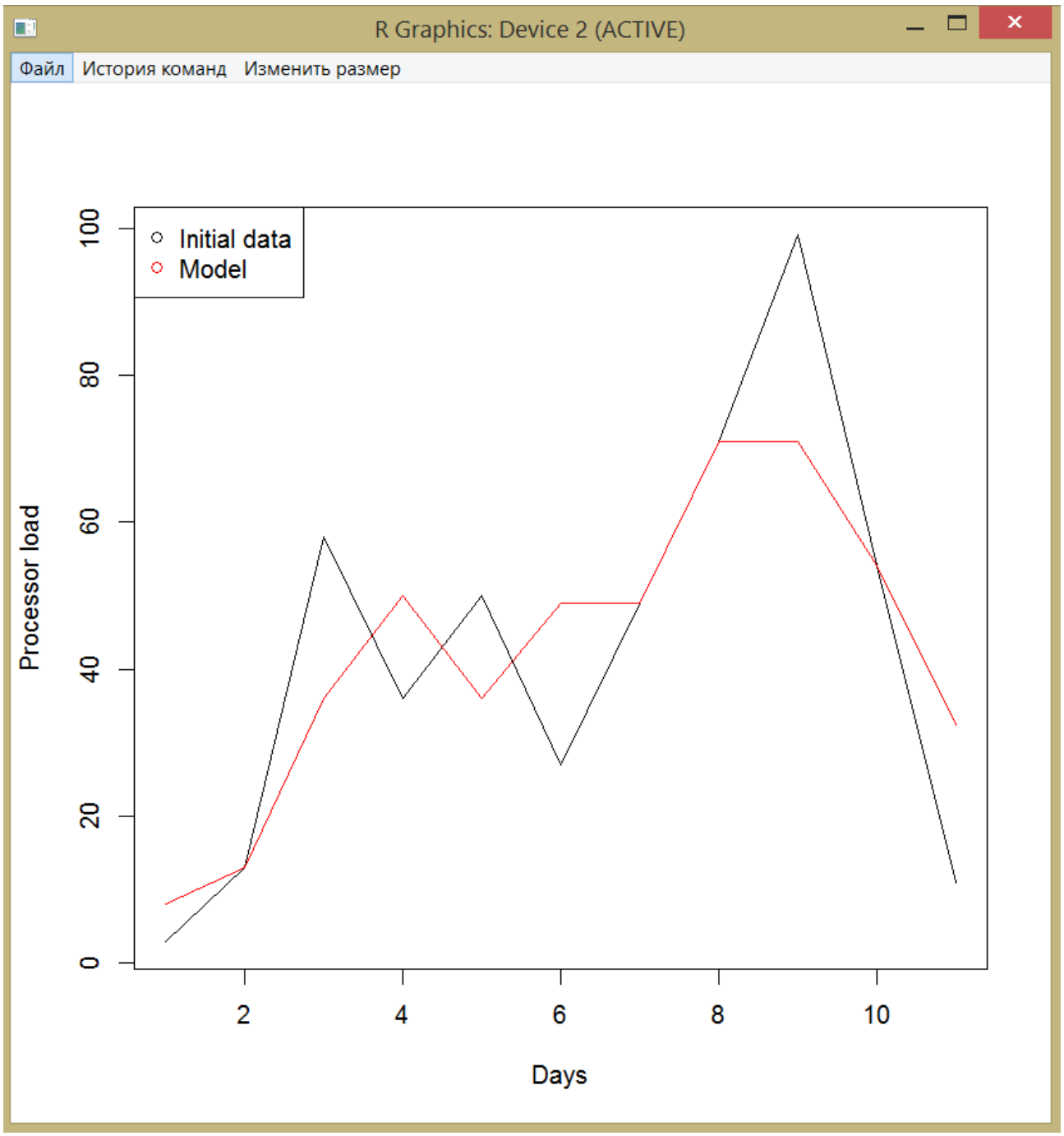


Рисунок 3.14 – График с исходными данными и модельными значениями загруженности выбранного сервера

После обновления аналитической информации о выбранном сервере, появится график с исходными данными и модельными значениями загруженности процессора выбранного сервера (рисунок 3.14).

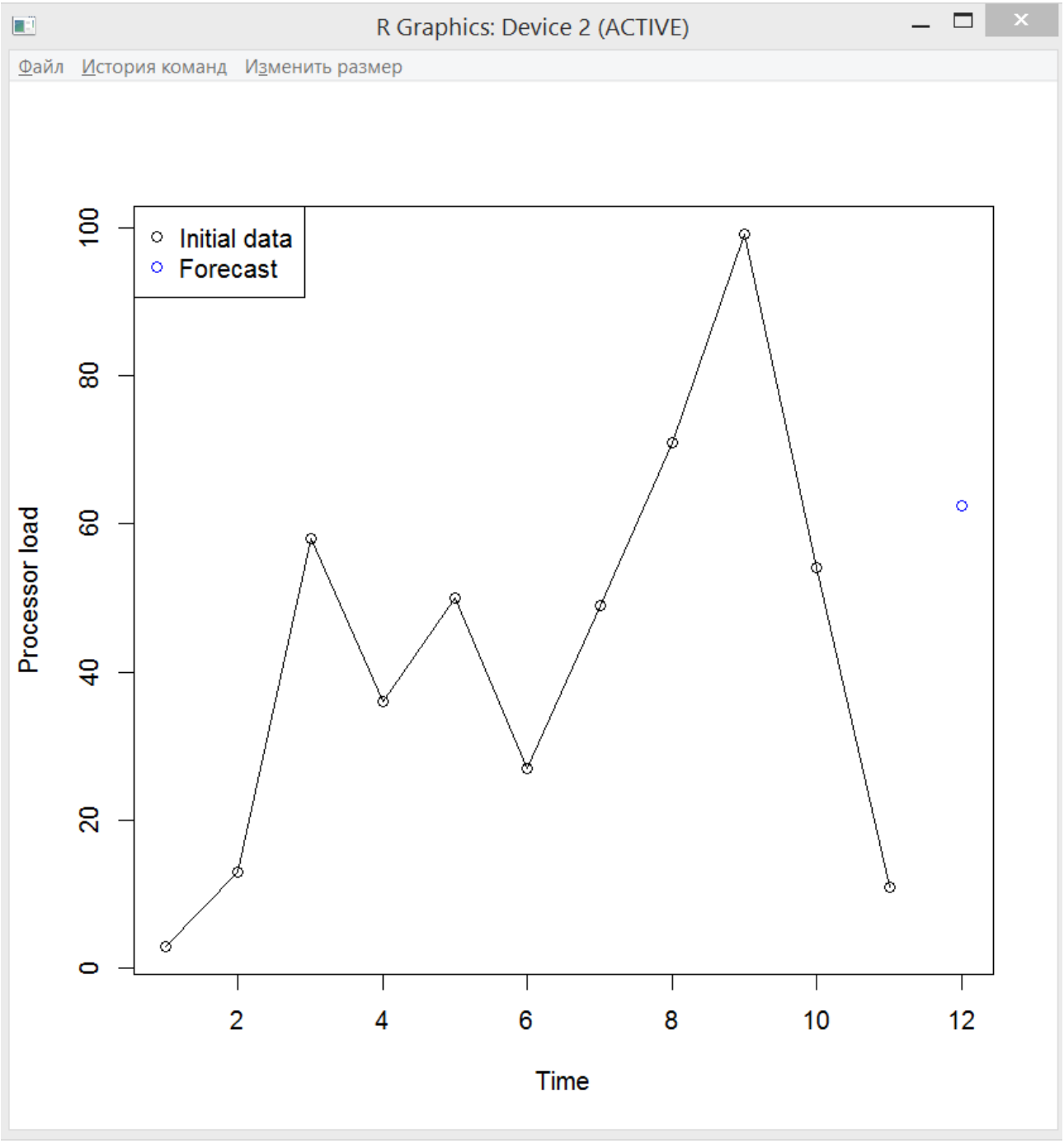


Рисунок 3.15 – График с прогнозом загруженности процессора выбранного сервера

После нажатия на кнопку Plot with forecast появится график с прогнозом загруженности процессора (рисунок 3.15).

Выводы по главе три

Описаны использованные средства разработки ПО. Реализован весь необходимый функционал согласно техническому заданию (приложение 5). Проведён обзор пользовательского интерфейса разработанной программы.

Заключение

В дипломной работе представлена реализация прикладного программного обеспечения со следующей функциональностью: мониторинг серверов, анализ и прогнозирование загруженности сервера.

В работе решены следующие **задачи**:

- был сделан обзор ранее проведённого исследования [6] прогнозирования загруженности процессора;
- проанализированы два способа прогнозирования загруженности процессора, для выявления наиболее точного и подходящего способа для реализации;
- согласно техническому заданию (приложение А) программно реализованы: мониторинг серверов, прогнозирование загруженности процессора, графический интерфейс пользователя.

Реализованное ПО получило практическое применение в ООО «Грид инжиниринг», что подтверждено актом внедрения.

В качестве направлений дальнейшего усовершенствования прикладного программного обеспечения следует рассматривать следующие:

- реализация одновременного подключения к нескольким серверам;
- повышение качества прогноза;
- реализация формирования записей журнала непосредственно самим ПО.

Список литературы

1. Созыкин, А.В. Система оперативного мониторинга температуры и энергопотребления суперкомпьютера «УРАН» / А.В. Созыкин, М.Л. Гольдштейн, М.А Черноскутов // Журнал ППС. – 2011. – № 4. – С. 120–123.

2. Стефанов, К.С. Система мониторинга производительности суперкомпьютеров / К.С. Стефанов // Вестник ПНИПУ. Аэрокосмическая техника. – 2014. – № 39. – С. 17–34.

3. Суперкомпьютер «Торнадо ЮУрГУ» [Электронный ресурс] / Лаборатория суперкомпьютерного моделирования ЮУрГУ. – Режим доступа http://supercomputer.susu.ru/computers/tornado/ – Дата обращения 06.05.2016.

4. Подбельский, В.В. Язык C#. Базовый курс: Учебник. / В.В. Подбельский. – М.: Финансы и статистика, Инфра-М, 2013. – 2-е изд. – 384 с.

5. MSDN [Электроный ресурс] / Microsoft – Режим доступа https://msdn.microsoft.com/ru-ru/default.aspx. – Дата обращения 06.05.2016.

6. Бражникова, Ю.С. Исследование методов прогнозирования загруженности компьютеров и компьютерных систем / Ю.С. Бражникова, Ю.А. Горицкий, В.П. Купетов, Н.А. Панков // Программные продукты и системы. – 2015. – № 2. – С. 135–139.

7. Хашковский, В.В. Применение облачных вычислений и GRID-технологий для организации коллективного использования вычислительных ресурсов в научно-исследовательской и учебной работе / В.В. Хашковский, И.Г. Данилов // Известия Южного федерального университета. Технические науки. – 2011. – № 1. – С. 139–143.

8. Демичев, А.П. Введение в грид технологии: Препринт. / А.П. Демичев, В.А. Ильин, А.П. Крюков. – НИИЯФ МГУ, 2007. – 87 с.

9. Черняк, Л. Облака: три источника и три составных части / Л. Черняк // Открытые системы. – 2010. – № 1.

10. Радченко, Г.И. Грид-технологии и суперкомпьютерный инжиниринг. / Г.И. Радченко, Л.Б. Соколинский // Суперкомпьютерные технологии в науке, образовании и промышленности / Под ред. В.А. Садовничего. – М.: Издательство Московского университета, 2009. – 232 с.

11. Windows Management Instrumentation [Электронный ресурс] / Режим доступа https://ru.wikipedia.org/wiki/WMI. – Дата обращения 06.05.2016.

12. About Windows Management Instrumentation [Электронный ресурс] / Microsoft – Режим доступа https://msdn.microsoft.com/en-us/library/aa384642(VS.85).aspx. – Дата обращения 06.05.2016.

13. WMI Code Creator v1.0 [Электронный ресурс] / Microsoft – Режим доступа https://www.microsoft.com/en-us/download/details.aspx?id=8572.– Дата обращения 06.05.2016.

14. Коэффициент детерминации [Электронный ресурс] / Режим доступа http://www.machinelearning.ru/wiki/index.php?title=Коэффициент_детерминации. – Дата обращения 06.05.2016.

15. Что такое Microsoft.Net Framework [Электронный ресурс] / Режим доступа http://net-framework.ru/article/chto-takoe. – Дата обращения 06.05.2016.

16. Общие сведения о Windows Forms [Электронный ресурс] / Microsoft – Режим доступа https://msdn.microsoft.com/ru-ru/library/8bxxy49h(v=vs.110).aspx. – Дата обращения 06.05.2016.

17. C Sharp [Электронный ресурс] / Режим доступа https://ru.wikipedia.org/wiki/C_Sharp. – Дата обращения 06.05.2016.

18. Microsoft Visual Studio [Электронный ресурс] / Режим доступа https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio. – Дата обращения 06.05.2016.

ПРИЛОЖЕНИЯ

Приложение А

ПРИКЛАДОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ МОНИТРИНГА И АНАЛИЗА СЕРВЕРОВ

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

ЮУрГУ–Д.010400.62.2016.047.ТЗ (ВКР)

Руководитель работы,
доц. каф. ЭММиС, к.ф.-м.н.
_____ В. А. Голодов
« » _____2016 г.

Автор работы
студент группы ВМИ-403
_____В. Д. Колногоров
« » _____2016 г.

Нормоконтролер,
доц. каф. ЭММиС, к.ф.-м.н.
_____Т. А. Макаровских
« » _____2016 г.

Челябинск, 2016

АННОТАЦИЯ

В данном программном документе приведено техническое задание на разработку прикладного программного обеспечения для мониторинга и анализа серверов на операционных системах семейства Windows. Программно реализованы мониторинг системной информации, статистический анализ полученных данных. В качестве инструментов разработки приложения использован язык программирования C#, а также, инструментарий управления Windows – WMI.

Практическая значимость разработанного приложения состоит в возможности использования организациями, работающими с недорогостоящими серверами без пакетов мониторинга.

Введение

Наименование программы

Наименование – «Прикладное программное обеспечение для мониторинга и анализа серверов».

Краткая характеристика области применения программы

Программа предназначена к применению мониторинга и анализа серверов ООО «Грид Инжиниринг».

1 Назначение разработки

1.1 Функциональное назначение программы

Функциональным назначением программы является мониторинг и анализ серверов.

1.2 Эксплуатационное назначение программы

Программа должна эксплуатироваться в организации ООО «Грид Инжиниринг».

2 Требование к программе

2.1 Требования к составу выполняемых функций

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

1. Загрузка параметров подключения из файла конфигурации;
2. Подключение к определённому серверу;
3. Подгрузка в программу порций данных записей журнала;
4. Анализ данных за указанный пользователем временной промежуток и расчёт анаитический показателей:
    4.1. время работы сервера;
    4.2. время простоя сервера;
    4.3. коэффициент продуктивности сервера;
    4.4. расчёт прогноза на день вперёд;
5. Мониторинг в реальном времени загруженности процессора, текущих процессов, свободной оперативной памяти, нагрузки на сеть;

2.2 Требования к организации входных данных

Входные данные должны быть организованы в виде (листинг 2.1):

Листинг 2.1 – Файл cfg.txt

```
12  4000
13  192.168.141.1 Администратор1 aA1234
14  192.168.141.2 Администратор2 aA1234
15  192.168.141.3 Администратор3 aA1234
16  192.168.141.4 Администратор4 aA1234
17  192.168.141.5 Администратор5 aA1234
18  192.168.141.6 Администратор6 aA1234
19  192.168.141.7 Администратор7 aA1234
20  192.168.141.8 Администратор8 aA1234
21  192.168.141.9 Администратор9 aA1234
22  192.168.141.10 Администратор10 aA1234
```

В файле cfg.txt (листинг 3.1) находится следующая информация:

- время обновления информации мониторинга в миллисекундах (строка 1);
- в строках с 2 по 11 содержатся параметры для подключения к серверам:
    - первое, до пробела, значение в строке содержит ip-адрес сервера;
    - второе значение содержит логин;
    - третье значение содержит пароль для входа на сервер.

Файл cfg.txt должен находиться в папке с исполняемым файлом программы.

Записи журнала, подгружаемые в программу должны находиться в каталоге C:\logs.

2.3 Требования к обеспечению надёжного функционирования программы

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением совокупности организационно-технических мероприятий, перечень которых приведен ниже:

а) организацией бесперебойного питания технических средств;

б) регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;

в) регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов;

г) необходимым уровнем квалификации сотрудников профильных подразделений.

2.4 Время восстановления после отказа

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать времени, необходимого на перезагрузку операционной системы и запуск программы, при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

2.5 Отказы из-за некорректных действий оператора

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий.

2.6 Требования к составу и параметрам технических средств

В состав технических средств должен входить персональный компьютер, на базе операционной системы Windows Server 2008 R2 и выше, включающий в себя:

- процессор Pentium 4 с тактовой частотой не менее 1.2 ГГц;
- оперативную память объёмом не менее 128 Мб;
- жестки диск объёмом 40 Гб и выше;
- оптический манипулятор типа «мышь»;

Приложение Б

Листинг парсинга папки с записями журнала выбранного сервера.

```
1  private void parserFolder() {
2  string[] file_list = Directory.GetFiles(@"C:\logs\" +
3   selPath);
4  bool countForEach = false;
5  progressBar1.Visible = true;
6  progressBar1.Maximum = file_list.Length;
7  progressBar1.Step = 1;
8  xForPlot.Clear();
9  foreach (string file_to_read in file_list)
10 {
11 ArrayList fMass = new ArrayList();
12 fMass.Clear();
13 if (File.Exists(file_to_read) == true)
14 {
15 StreamReader myReader = new StreamReader(file_to_read);
16 String myStr = null;
17 while ((myStr = myReader.ReadLine()) != null)
18 {
19 String[] splitMass = myStr.Split(' ');
20 if ((splitMass[0] == "===") && (splitMass[1] != "Started") &&
21  (splitMass[1] != "Done"))
22 {
23 fMass.Add(splitMass[5]);
24 fMass.Add(splitMass[6]);
25 fMass.Add(splitMass[7]);
26 }
27 if ((splitMass.Length > 1) && (splitMass[1] == "Duration:"))
28 {
29 int i = 0;
30 foreach (string str in splitMass)
31 {
32 if (str == "usage:")
33 fMass.Add(splitMass[i + 1]);
34 i++;
35 }
36 }if (fMass.Count > 3)
37 {
38 string[] datas = Convert.ToString(fMass[0]).Split('/');
39 if (countForEach == false) mindate =
40 Convert.ToString(fMass[0]);
41 else maxdate = Convert.ToString(fMass[0]);
42 int whichDay = Convert.ToInt32(datas[1], 10);
43 string[] sHours = Convert.ToString(fMass[1]).Split(':');
44 int intSec = 0;
45 for (int i = 0; i < sHours.Length; i++)
46 {
```

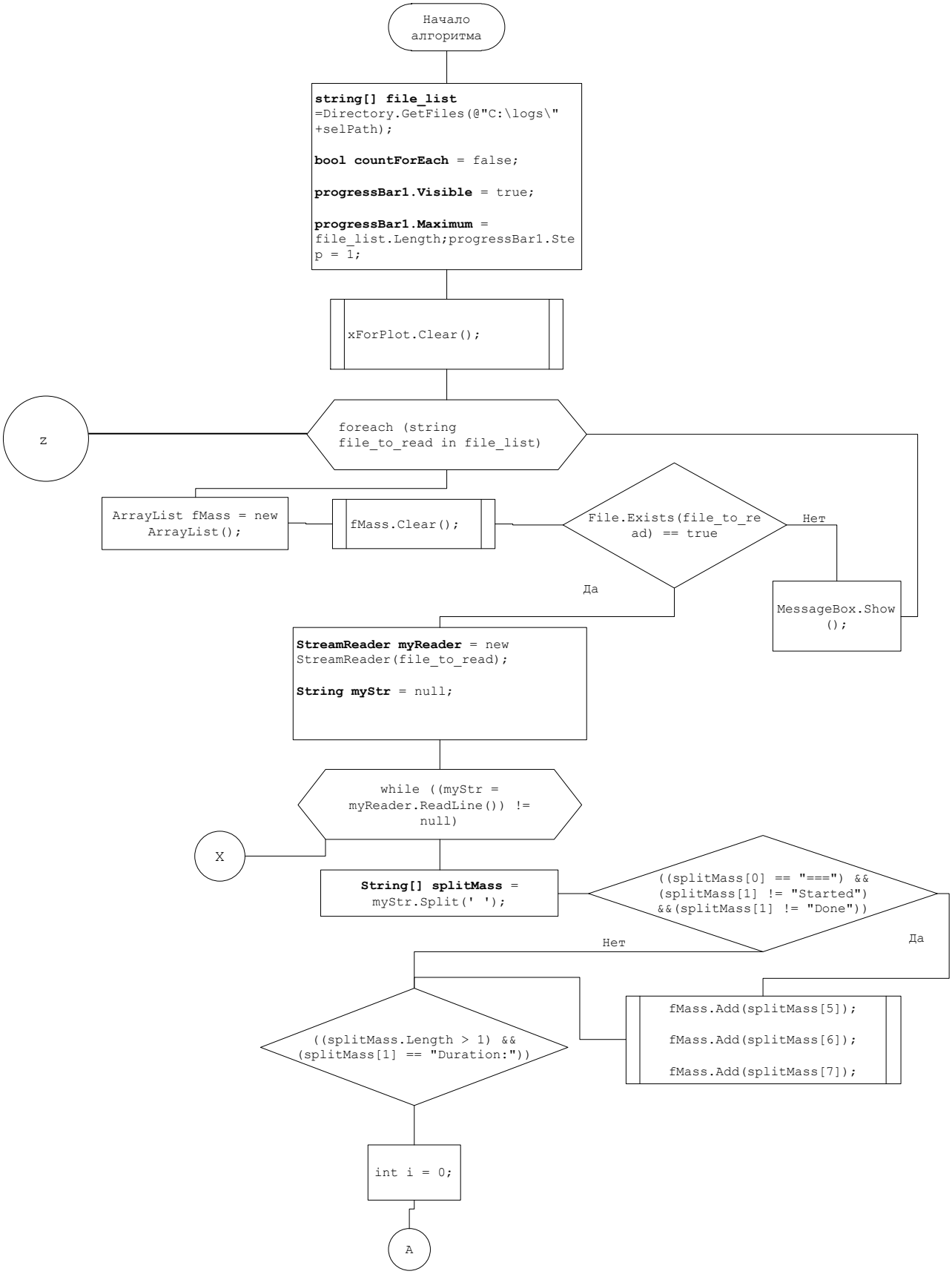
```
47 if (i == 0)
48 {
49 if ((Convert.ToInt32(sHours[i]) == 12) &&
50 (Convert.ToString(fMass[2]) == "AM")) { }
51 else intSec += Convert.ToInt32(sHours[i], 10) * 60 * 60;
52 }
53 if (i == 1) intSec += Convert.ToInt32(sHours[i], 10) * 60;
54 if (i == 2) intSec += Convert.ToInt32(sHours[i], 10);
55 }
56 if ((Convert.ToString(fMass[2]) != "AM") &&
57 (Convert.ToInt32(sHours[0]) != 12))
58 {
59 intSec += 43200;
60 }
61 if (whichDay != 29)
62 {
63 intSec += whichDay * 86400;
64 }
65 xForPlot.Add(intSec);
66 sHours = Convert.ToString(fMass[4]).Split(':');
67 datas = Convert.ToString(fMass[3]).Split('/');
68 whichDay = Convert.ToInt32(datas[1], 10);
69 intSec = 0;
70 for (int i = 0; i < sHours.Length; i++)
71 {
72 if (i == 0)
73 {
74 if ((Convert.ToInt32(sHours[i]) == 12) &&
75 (Convert.ToString(fMass[5]) == "AM")) { }
76 else intSec += Convert.ToInt32(sHours[i], 10) * 60 * 60;
77 }
78 if (i == 1) intSec += Convert.ToInt32(sHours[i], 10) * 60;
79 if (i == 2) intSec += Convert.ToInt32(sHours[i], 10);
80 }
81 if ((Convert.ToString(fMass[5]) != "AM") &&
82 (Convert.ToInt32(sHours[0]) != 12))
83 {
84 intSec += 43200;
85 }
86 if (whichDay != 29)
87 {
88 intSec += whichDay * 86400;
89 }
90 xForPlot.Add(intSec);
91 sHours = Convert.ToString(fMass[6]).Split('%');
92 intSec = Convert.ToInt32(sHours[0], 10);
93 xForPlot.Add(intSec);
94 countForEach = true;
95 }
96 }
97 else
98 {
99 MessageBox.Show("File not found!", "Error!",
```

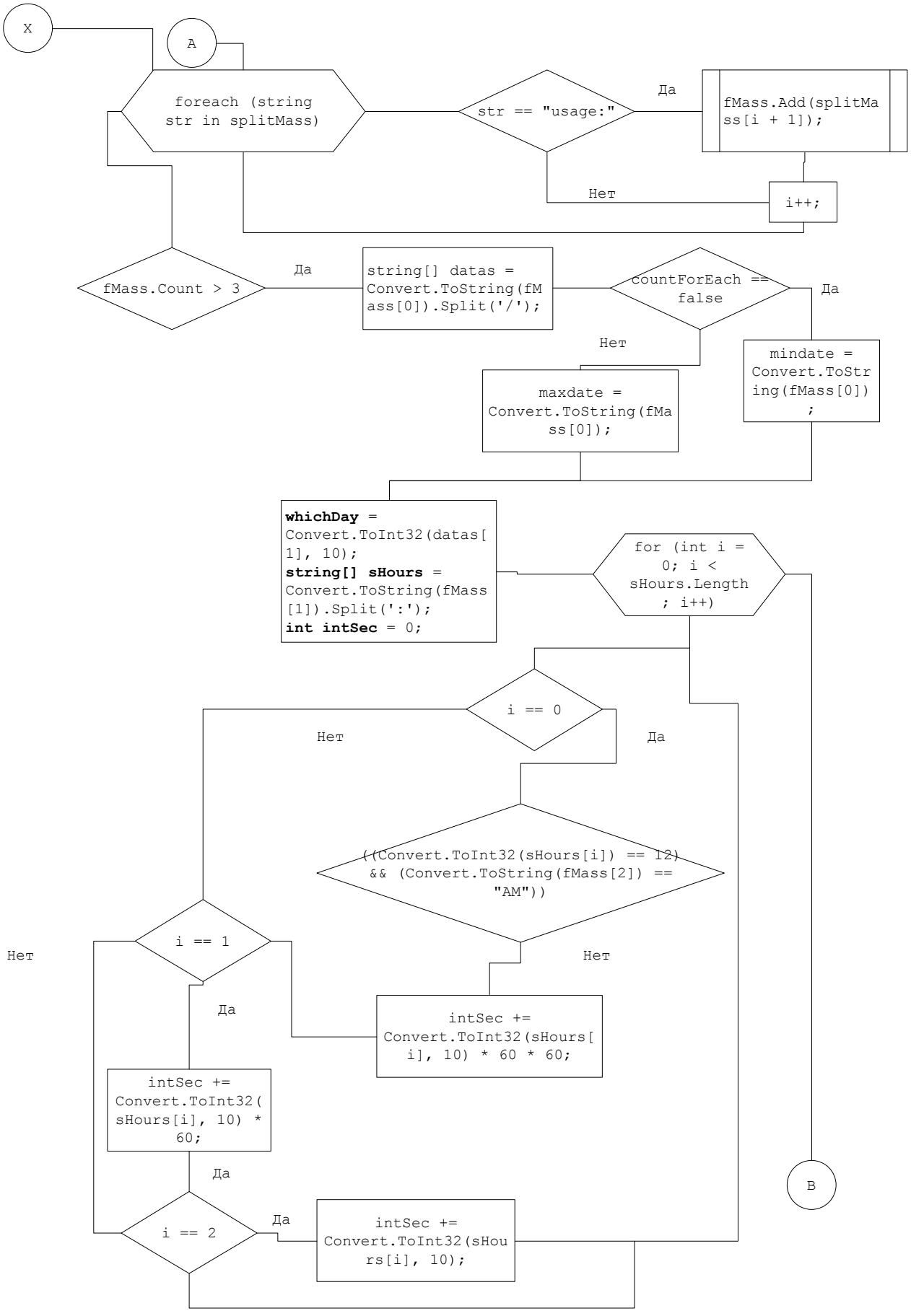
```
100     MessageBoxButtons.OK, MessageBoxIcon.Error);
101     }
102     progressBar1.PerformStep();
103     }
104     for (int i = 0; i < xForPlot.Count - 3; i += 3)
105     {
106     for (int j = i; j < xForPlot.Count - 3; j += 3)
107     {
108     if (Convert.ToInt32(xForPlot[j]) >
109     Convert.ToInt32(xForPlot[j + 3]))
110     {
111     object tmp = xForPlot[j];
112     xForPlot[j] = xForPlot[j + 3];
113     xForPlot[j + 3] = tmp;
114     tmp = xForPlot[j + 1];
115     xForPlot[j + 1] = xForPlot[j + 4];
116     xForPlot[j + 4] = tmp;
117     tmp = xForPlot[j + 2];
118     xForPlot[j + 2] = xForPlot[j + 5];
119     xForPlot[j + 5] = tmp;
120     }
121     }
122     }
123     progressBar1.Visible = false;
124     string[] checkMax = maxdate.Split('/');
125     string tmp1 = checkMax[0];
126     checkMax[0] = checkMax[1];
127     checkMax[1] = tmp1;
128     string[] check = mindate.Split('/');
129     tmp1 = check[0];
130     check[0] = check[1];
131     check[1] = tmp1;
132     textBox1.Text = check[0] + ".0" + check[1] + "." +
133     check[2];
134     textBox2.Text = checkMax[0] + ".0" + checkMax[1] + "." +
135      checkMax[2];
136     secondDate = new DateTime(Convert.ToInt32(checkMax[2]),
137     Convert.ToInt32(checkMax[1]),
138     Convert.ToInt32(checkMax[0]));
139     firstDate = new DateTime(Convert.ToInt32(check[2]),
140     Convert.ToInt32(check[1]), Convert.ToInt32(check[0]));
141     check = textBox1.Text.Split('.');
142     if (Convert.ToInt32(check[0]) == 29) { startAnalisisSec =
143     0; }
144     else
145     {
146     startAnalisisSec = Convert.ToInt32(check[0]) * 86400;
147     }
148     string[] ch = textBox2.Text.Split('.');
149     if (Convert.ToInt32(ch[0]) == 29) { endAnalisisSec =
150     86400; }
151     else
152     {
```

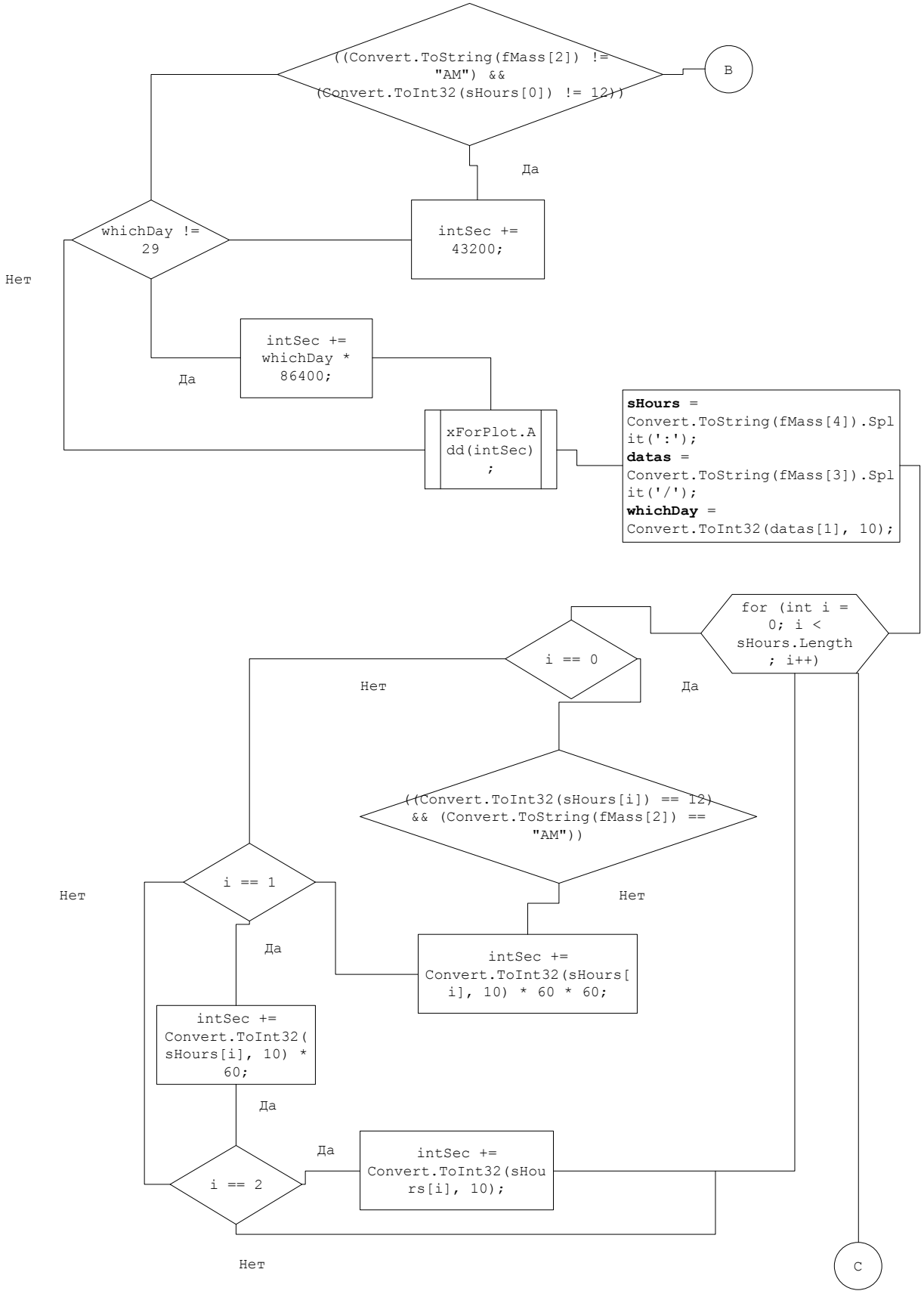
```
153     endAnalisisSec = Convert.ToInt32(ch[0]) * 86400 + 86400;
154     }
```

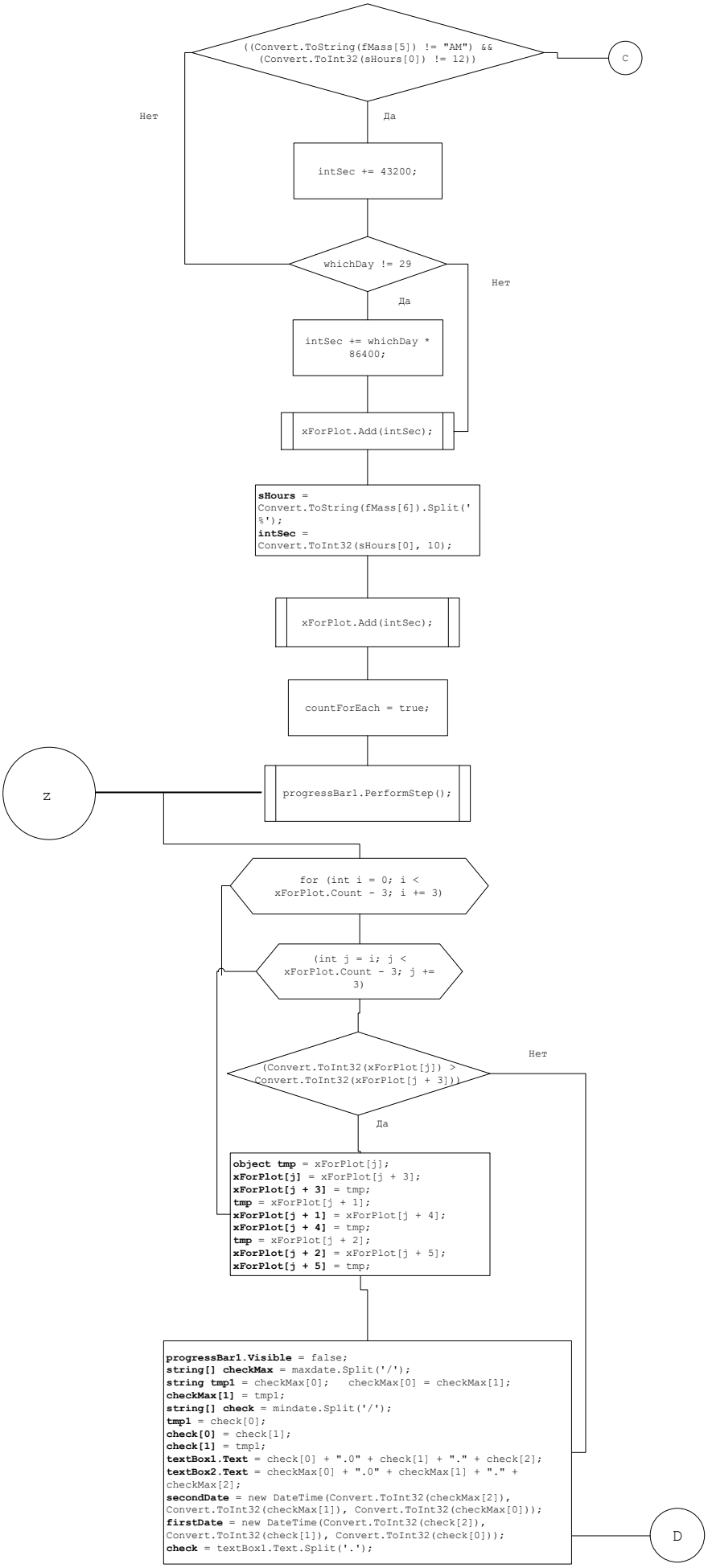
Приложение В

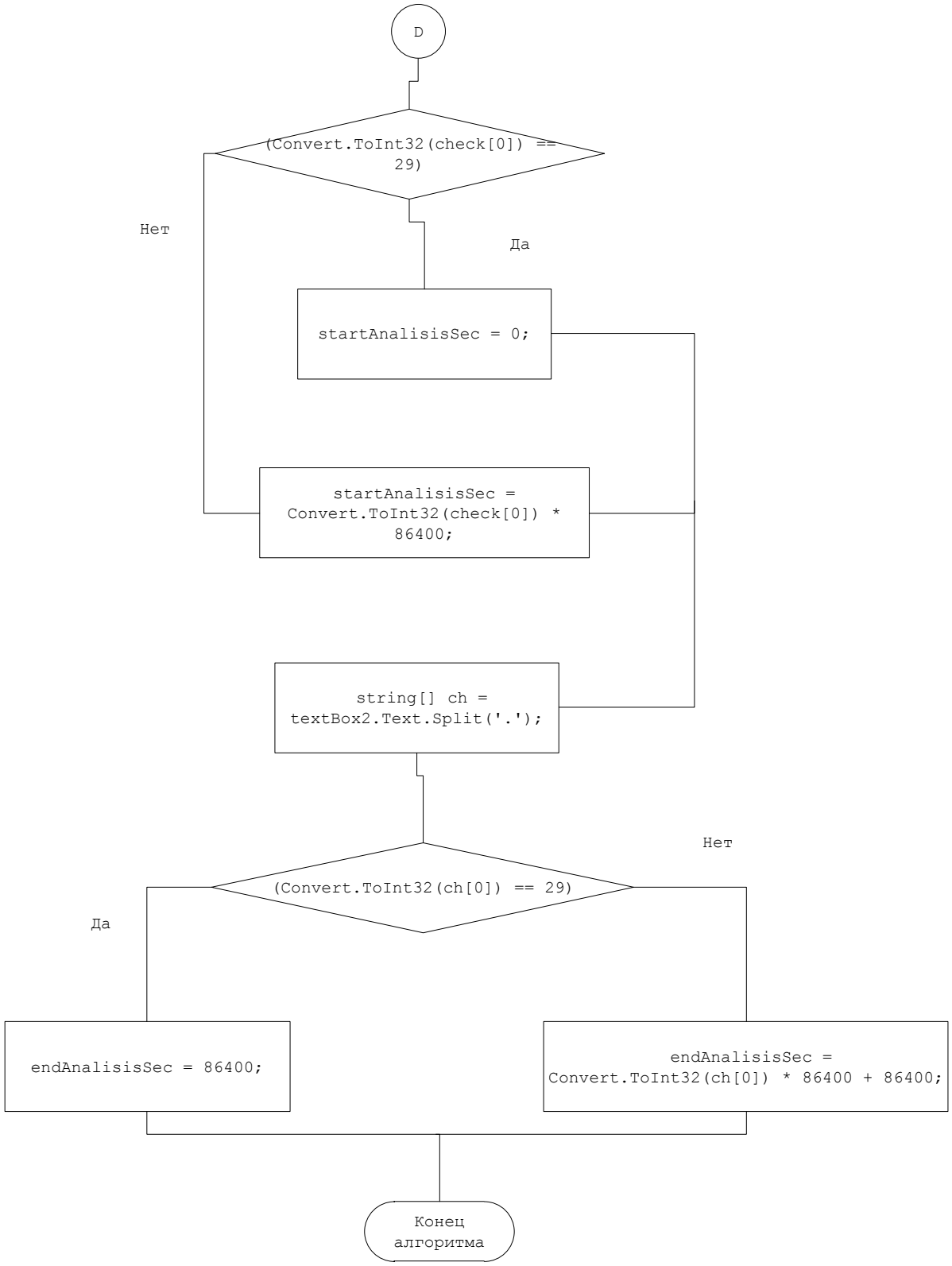
Блок-схема парсинга папки с записями журнала.




X
A
foreach (string str in splitMass)
str == "usage:"
Да
fMass.Add(splitMass[i + 1]);
Нет
i++;
fMass.Count > 3
Да
string[] datas = Convert.ToString(fMass[0]).Split('/');
countForEach == false
Да
Нет
mindate = Convert.ToString(fMass[0]);
maxdate = Convert.ToString(fMass[0]);
whichDay = Convert.ToInt32(datas[1], 10);
string[] sHours = Convert.ToString(fMass[1]).Split(':');
int intSec = 0;
for (int i = 0; i < sHours.Length ; i++)
i == 0
Нет
Да
((Convert.ToInt32(sHours[i]) == 12) && (Convert.ToString(fMass[2]) == "AM"))
Нет
Нет
i == 1
Да
intSec += Convert.ToInt32(sHours[i], 10) * 60 * 60;
intSec += Convert.ToInt32(sHours[i], 10) * 60;
Да
i == 2
Да
intSec += Convert.ToInt32(sHours[i], 10);
B


((Convert.ToString(fMass[2]) != "AM") && (Convert.ToInt32(sHours[0]) != 12))
B
Да
intSec += 43200;
whichDay != 29
Нет
Да
intSec += whichDay * 86400;
xForPlot.Add(intSec);
sHours = Convert.ToString(fMass[4]).Split(':');
datas = Convert.ToString(fMass[3]).Split('/');
whichDay = Convert.ToInt32(datas[1], 10);
for (int i = 0; i < sHours.Length ; i++)
i == 0
Нет
Да
((Convert.ToInt32(sHours[i]) == 12) && (Convert.ToString(fMass[2]) == "AM"))
Нет
intSec += Convert.ToInt32(sHours[i], 10) * 60 * 60;
i == 1
Нет
Да
intSec += Convert.ToInt32(sHours[i], 10) * 60;
Да
i == 2
Да
intSec += Convert.ToInt32(sHours[i], 10);
Нет
C


((Convert.ToString(fMass[5]) != "AM") &&
(Convert.ToInt32(sHours[0]) != 12))
C
Нет
Да
intSec += 43200;
whichDay != 29
Нет
Да
intSec += whichDay *
86400;
xForPlot.Add(intSec);
sHours =
Convert.ToString(fMass[6]).Split('
%');
intSec =
Convert.ToInt32(sHours[0], 10);
xForPlot.Add(intSec);
countForEach = true;
Z
progressBar1.PerformStep();
for (int i = 0; i <
xForPlot.Count - 3; i += 3)
(int j = i; j <
xForPlot.Count - 3; j +=
3)
(Convert.ToInt32(xForPlot[j]) >
Convert.ToInt32(xForPlot[j + 3]))
Нет
Да
object tmp = xForPlot[j];
xForPlot[j] = xForPlot[j + 3];
xForPlot[j + 3] = tmp;
tmp = xForPlot[j + 1];
xForPlot[j + 1] = xForPlot[j + 4];
xForPlot[j + 4] = tmp;
tmp = xForPlot[j + 2];
xForPlot[j + 2] = xForPlot[j + 5];
xForPlot[j + 5] = tmp;
progressBar1.Visible = false;
string[] checkMax = maxdate.Split('/');
string tmp1 = checkMax[0]; checkMax[0] = checkMax[1];
checkMax[1] = tmp1;
string[] check = mindate.Split('/');
tmp1 = check[0];
check[0] = check[1];
check[1] = tmp1;
textBox1.Text = check[0] + ".0" + check[1] + "." + check[2];
textBox2.Text = checkMax[0] + ".0" + checkMax[1] + "." +
checkMax[2];
secondDate = new DateTime(Convert.ToInt32(checkMax[2]),
Convert.ToInt32(checkMax[1]), Convert.ToInt32(checkMax[0]));
firstDate = new DateTime(Convert.ToInt32(check[2]),
Convert.ToInt32(check[1]), Convert.ToInt32(check[0]));
check = textBox1.Text.Split('.');
D


D
(Convert.ToInt32(check[0]) == 29)
Нет
Да
startAnalisisSec = 0;
startAnalisisSec = Convert.ToInt32(check[0]) * 86400;
string[] ch = textBox2.Text.Split('.');
Нет
(Convert.ToInt32(ch[0]) == 29)
Да
endAnalisisSec = 86400;
endAnalisisSec = Convert.ToInt32(ch[0]) * 86400 + 86400;
Конец алгоритма

Приложение Г

Алгоритм анализа записей журнала.

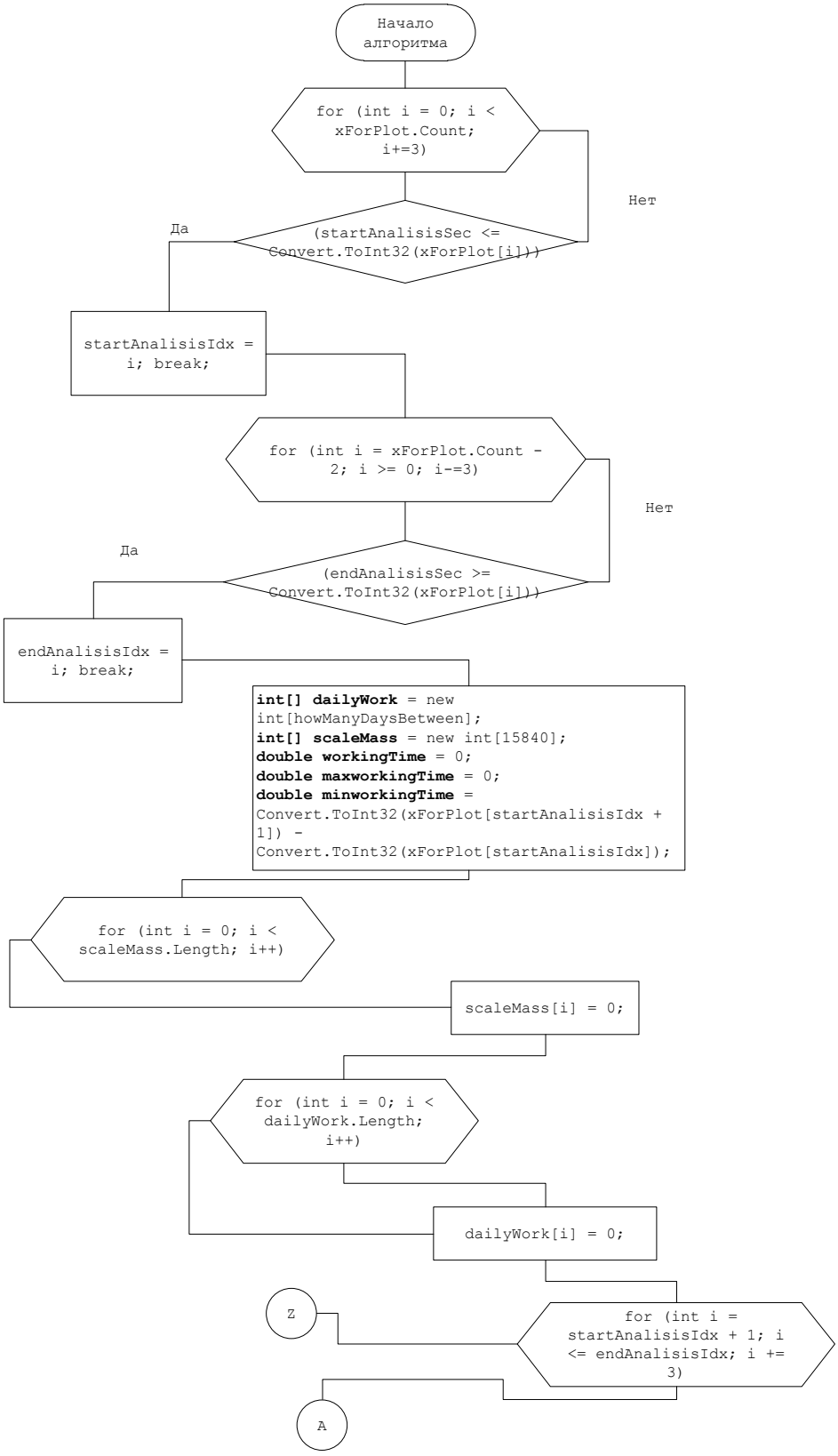
```
1    private void analisServer() {
2    for (int i = 0; i < xForPlot.Count; i+=3) {
3    if (startAnalisisSec <= Convert.ToInt32(xForPlot[i]))
4    { startAnalisisIdx = i;
5    break; } }
6    for (int i = xForPlot.Count - 2; i >= 0; i-=3)
7    {
8    if (endAnalisisSec >= Convert.ToInt32(xForPlot[i]))
9    { endAnalisisIdx = i;
10   break; }
11   }
12   int[] dailyWork = new int[howManyDaysBetween];
13   int[] scaleMass = new int[15840];
14   double workingTime = 0;
15   double maxworkingTime = 0;
16   double minworkingTime = Convert.ToInt32
17   (xForPlot[startAnalisisIdx + 1]) -
18   Convert.ToInt32(xForPlot[startAnalisisIdx]);
19   for (int i = 0; i < scaleMass.Length; i++)
20    scaleMass[i] = 0;
21   for (int i = 0; i < dailyWork.Length; i++)
22    dailyWork[i] = 0;
23   for (int i = startAnalisisIdx + 1; i <=
24    endAnalisisIdx; i += 3)
25   {
26   if ((Convert.ToInt32(xForPlot[i]) -
27   Convert.ToInt32(xForPlot[i - 1])) >
28    maxworkingTime) maxworkingTime =
29   Convert.ToInt32(xForPlot[i]) -
30   Convert.ToInt32(xForPlot[i - 1]);
31   if (((Convert.ToInt32(xForPlot[i]) -
32   Convert.ToInt32(xForPlot[i - 1])) <
33   minworkingTime) &&
34   (Convert.ToInt32(xForPlot[i]) -
35   Convert.ToInt32(xForPlot[i - 1])) != 0)
36   minworkingTime = Convert.ToInt32(xForPlot[i]) -
37   Convert.ToInt32
38   (xForPlot[i - 1]);
39   workingTime += Convert.ToInt32(xForPlot[i]) -
40   Convert.ToInt32
41   (xForPlot[i - 1]);
42   for (int j = Convert.ToInt32(xForPlot[i - 1]) / 60; j <
43   Convert.ToInt32(xForPlot[i]) / 60; j++)
44   {
45   scaleMass[j] = Convert.ToInt32(xForPlot[i + 1]);
46   }
47   }
48   DateTime tn = new DateTime(2016, 2, 29);
```

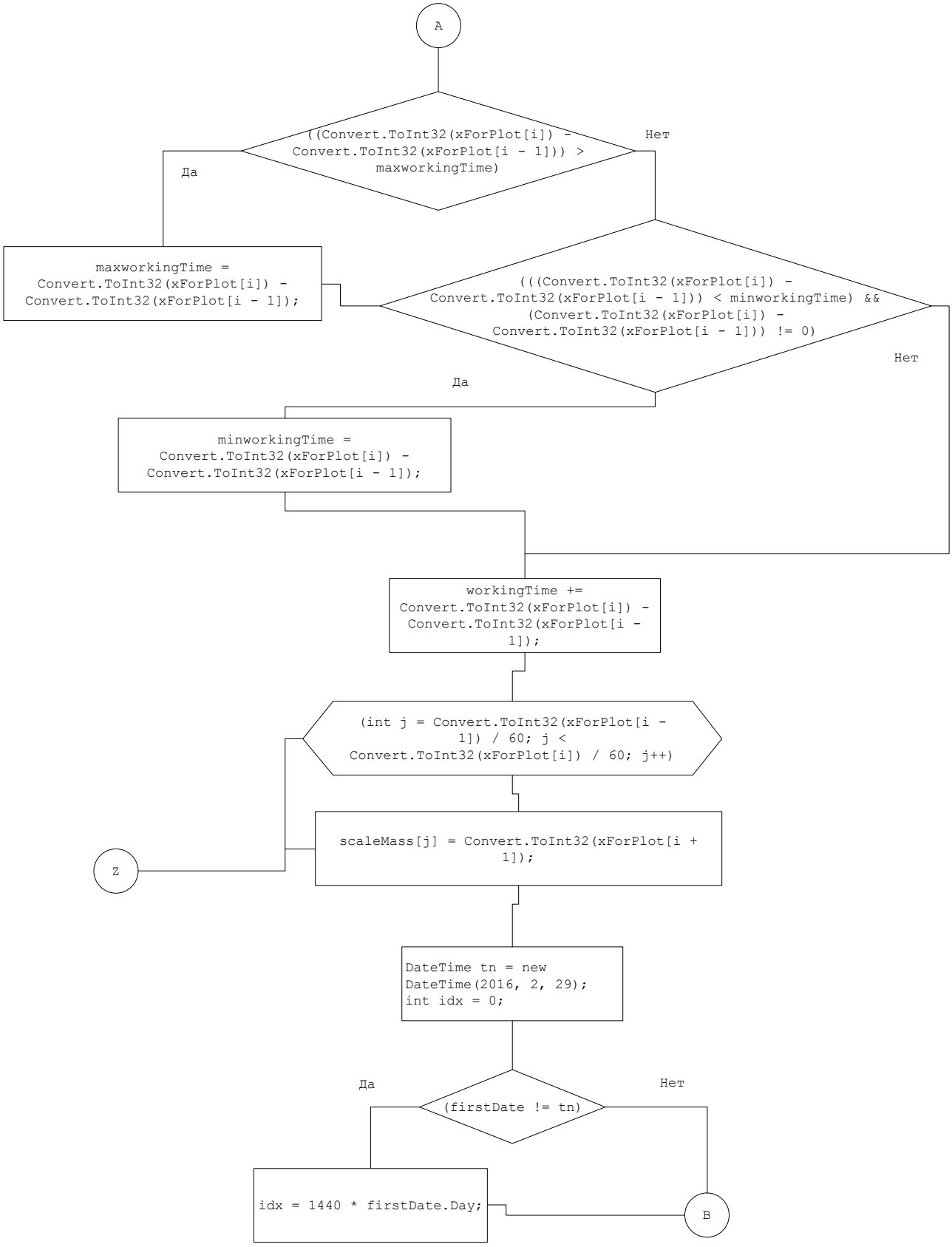
```
49   int idx = 0;
50   if (firstDate != tn) idx = 1440 * firstDate.Day;
51   for (int i = 0; i < dailyWork.Length; i++)
52   {
53   for (int j = idx + 1440 * i; j < 1440 * i + 1440 +
54   idx; j++)
55   {
56   dailyWork[i] += scaleMass[j];
57   }
58   dailyWork[i] /= 1440;
59   }
60   double downtime = (endAnalisisSec - startAnalisisSec -
61   workingTime);
62   double coefOfProd = workingTime / (endAnalisisSec -
63   startAnalisisSec) *
64    100;
65   label16.Visible = true;
66   label17.Visible = true;
67   label18.Visible = true;
68   label19.Visible = true;
69   label20.Visible = true;
70   secMinHour(workingTime, label16, "Working time: ");
71   secMinHour(downtime, label17, "Downtime: ");
72   label18.Text = string.Format
73   ("Coefficient of productivity: {0}%\n", (int)coefOfProd);
74   secMinHour(maxworkingTime, label19,
75   "Maximum worktime: ");
76   secMinHour(minworkingTime, label20,
77   "Minimum worktime: ");
78   string typeOfModel = toolStripComboBox1.Text;
79   REngine.SetEnvironmentVariables();
80   REngine engine = REngine.GetInstance();
81   engine.Initialize();
82   double[] forR = new double[dailyWork.Length];
83   for (int i = 0; i < forR.Length; i++) {
84   forR[i] = dailyWork[i];
85   mSW.WriteLine(Convert.ToString(forR[i]) + "    "
86   +Convert.ToString(dailyWork[i] + "\n"));
87   }
88   switch(typeOfModel){
89   case (""): { MessageBox.Show("Choose type of model!",
90    "Warning!",
91    MessageBoxButtons.OK, MessageBoxIcon.Asterisk); break; }
92   case ("Holt Winters"):
93   {
94   NumericVector dd = engine.CreateNumericVector(forR);
95   engine.SetSymbol("dd", dd);
96   try
97   {
98   engine.Evaluate("library(FBN)");
99   engine.Evaluate
100  ("fit <- medianFilter(dd, windowSize = 3)");
101  engine.Evaluate("library(forecast)");
```

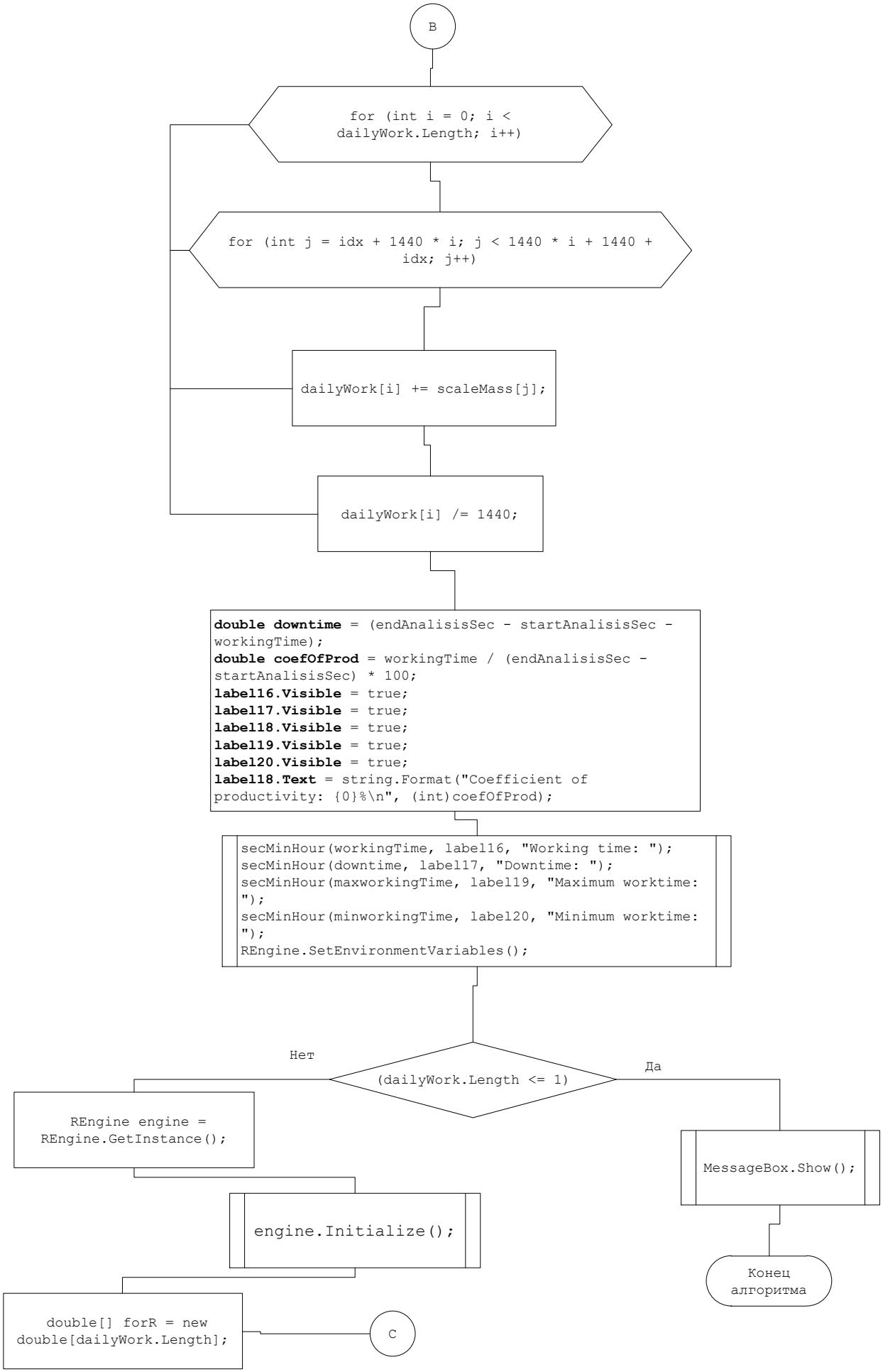
```
102  if (radioButton2.Checked == true)
103  {
104  engine.Evaluate("plot(c(dd, median(dd[" +
105  " (length(dd)-3):length(dd)])), type = 'p', " +
106  "col = 'blue', xlab = 'Time', ylab = 'Processor load')");
107  engine.Evaluate("lines(c(dd, NA), type = 'o')");
108  engine.Evaluate("legend('topleft', c('Initial data', "+
109  "'Forecast'), col = c('black','blue'), pch = c(1,1))");}
110  else if (radioButton1.Checked == true)
111  {
112  engine.Evaluate("plot(dd, xlab = 'Days', ylab = 'Proces-
sor load', type = 'l')");
113  engine.Evaluate("lines(fit, col = 'red', type = 'l')");
114  engine.Evaluate("legend('topleft', c('Initial data', "+
115  "'Model'), col = c('black','red'), pch = c(1,1))");
116  }
117  NumericVector ddRes = engine.Evaluate
118  ("predict(HoltWinters
119  (dd, alpha = 0.5 ,beta = F ,gamma = F ), " +
120  " n.ahead = 1)").AsNumeric();
121  label21.Text = string.Format
122  ("Forecast on day {0}%", string.Join
123  (", ", ddRes));
124  label21.Visible = true;
125  }
126  catch
127  {
128  MessageBox.Show("Try again", "Warning",
129  MessageBoxButtons.OK,
130   MessageBoxIcon.Asterisk);
131  }
132  break;}
133  }
134  }
```

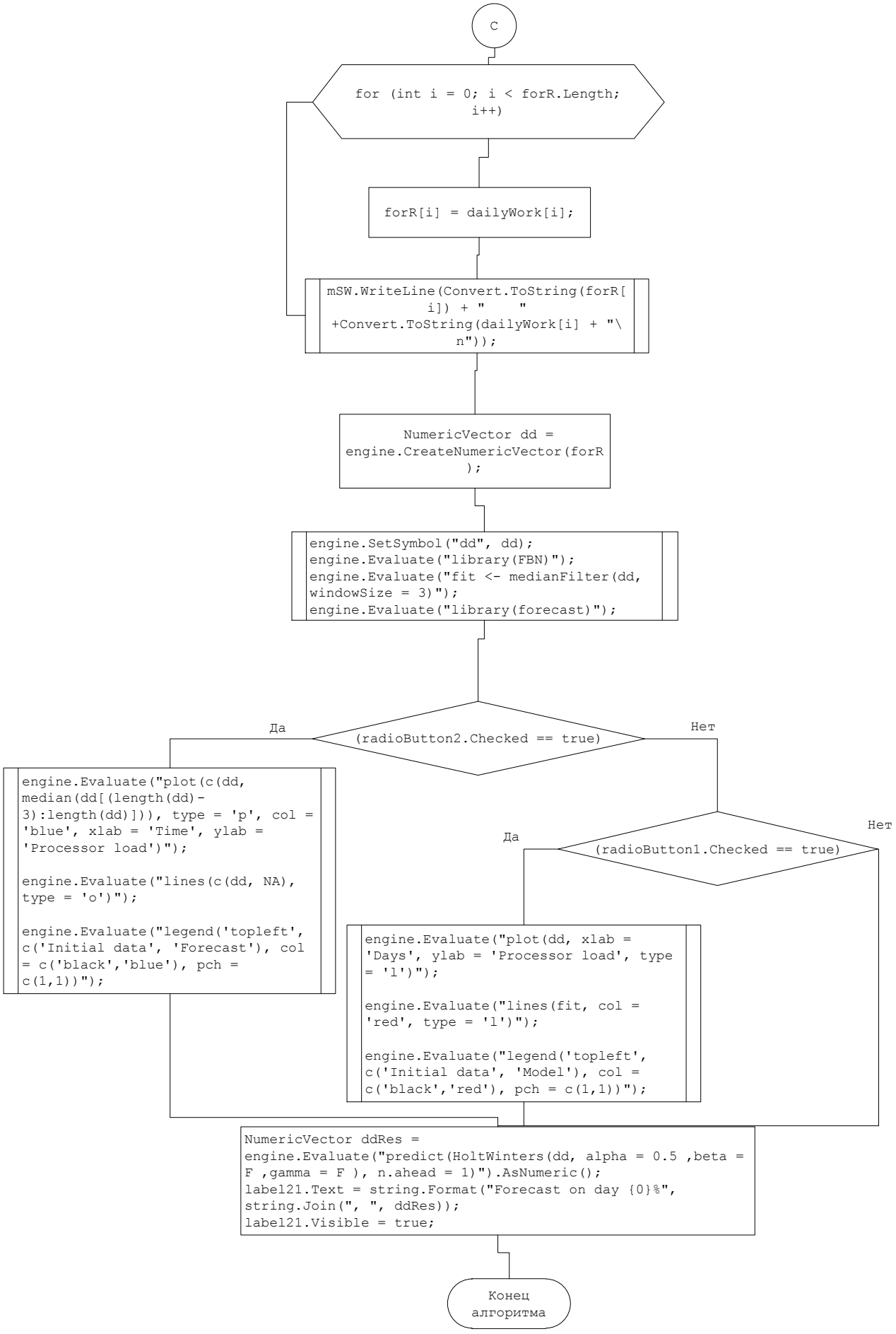
Приложение Д

Блок-схема анализа записей журнала.




A
((Convert.ToInt32(xForPlot[i]) - Convert.ToInt32(xForPlot[i - 1])) > maxworkingTime)
Нет
Да
maxworkingTime = Convert.ToInt32(xForPlot[i]) - Convert.ToInt32(xForPlot[i - 1]);
(((Convert.ToInt32(xForPlot[i]) - Convert.ToInt32(xForPlot[i - 1])) < minworkingTime) && (Convert.ToInt32(xForPlot[i]) - Convert.ToInt32(xForPlot[i - 1])) != 0)
Нет
Да
minworkingTime = Convert.ToInt32(xForPlot[i]) - Convert.ToInt32(xForPlot[i - 1]);
workingTime += Convert.ToInt32(xForPlot[i]) - Convert.ToInt32(xForPlot[i - 1]);
(int j = Convert.ToInt32(xForPlot[i - 1]) / 60; j < Convert.ToInt32(xForPlot[i]) / 60; j++)
scaleMass[j] = Convert.ToInt32(xForPlot[i + 1]);
Z
DateTime tn = new DateTime(2016, 2, 29);
int idx = 0;
Да
(firstDate != tn)
Нет
idx = 1440 * firstDate.Day;
B


B
for (int i = 0; i < dailyWork.Length; i++)
for (int j = idx + 1440 * i; j < 1440 * i + 1440 + idx; j++)
dailyWork[i] += scaleMass[j];
dailyWork[i] /= 1440;
double downtime = (endAnalisisSec - startAnalisisSec - workingTime);
double coefOfProd = workingTime / (endAnalisisSec - startAnalisisSec) * 100;
label16.Visible = true;
label17.Visible = true;
label18.Visible = true;
label19.Visible = true;
label20.Visible = true;
label18.Text = string.Format("Coefficient of productivity: {0}%\n", (int)coefOfProd);
secMinHour(workingTime, label16, "Working time: ");
secMinHour(downtime, label17, "Downtime: ");
secMinHour(maxworkingTime, label19, "Maximum worktime: ");
secMinHour(minworkingTime, label20, "Minimum worktime: ");
REngine.SetEnvironmentVariables();
Нет
(dailyWork.Length <= 1)
Да
REngine engine = REngine.GetInstance();
MessageBox.Show();
engine.Initialize();
Конец алгоритма
double[] forR = new double[dailyWork.Length];
C


C
for (int i = 0; i < forR.Length; i++)
forR[i] = dailyWork[i];
mSW.WriteLine(Convert.ToString(forR[i]) + " " +Convert.ToString(dailyWork[i] + "\ n"));
NumericVector dd = engine.CreateNumericVector(forR);
engine.SetSymbol("dd", dd);
engine.Evaluate("library(FBN)");
engine.Evaluate("fit <- medianFilter(dd, windowSize = 3)");
engine.Evaluate("library(forecast)");
(radioButton2.Checked == true)
Да
Нет
engine.Evaluate("plot(c(dd, median(dd[(length(dd)-3):length(dd)])), type = 'p', col = 'blue', xlab = 'Time', ylab = 'Processor load')");
engine.Evaluate("lines(c(dd, NA), type = 'o')");
engine.Evaluate("legend('topleft', c('Initial data', 'Forecast'), col = c('black','blue'), pch = c(1,1))");
(radioButton1.Checked == true)
Да
Нет
engine.Evaluate("plot(dd, xlab = 'Days', ylab = 'Processor load', type = 'l')");
engine.Evaluate("lines(fit, col = 'red', type = 'l')");
engine.Evaluate("legend('topleft', c('Initial data', 'Model'), col = c('black','red'), pch = c(1,1))");
NumericVector ddRes = engine.Evaluate("predict(HoltWinters(dd, alpha = 0.5 ,beta = F ,gamma = F), n.ahead = 1)").AsNumeric();
label21.Text = string.Format("Forecast on day {0}%", string.Join(", ", ddRes));
label21.Visible = true;
Конец алгоритма