

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

Клиент серверное приложение менеджера процессов Windows

Студент

(подпись, дата)

Куликов Егор Александрович

(ФИО полностью)

Руководитель

(подпись, дата)

Афонин А.Ю.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю.Козлов
«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

1. Студент Куликов Егор Александрович гр. 19ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Клиент-серверное приложение менеджера процессов Windows

Тема утверждена приказом ПГУ № _____ от "____" _____ 2023 г.

2. Руководитель работы _____ к.т.н., доцент кафедры «МОиПЭВМ» Афонин А.Ю.

3. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение: клиент серверное приложение предназначено для
для автоматизации работы сотрудников*

2) *Среда выполнения: операционная система Windows, начиная с версии 7.
Средства разработки: языки программирования C#,; инструментальные сред-
ства «Visual Studio»,
Технологии разработки: RUP, C#.*

3) *Основные функции автоматизированной информационной системы:*

а) Ведение учета пользователей, администраторов

б) Модификация пользователей

в) Просмотр данных о компьютере, процессах, и их состоянии.

г) Добавление и удаление пользователей управления процессами.

д) Обновление данных.

е) Авторизация.

4. Перечень подлежащих разработке вопросов

1) Анализ предметной области

2) Анализ требований

3) Проектирование

4) Реализация

5) Тестирование

6) План разработки и оценка бюджета

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем ра- боты	Срок выполнения	Подпись руко- водителя
1	Анализ предметной области и требований	10	06.06.23-07.06.23	
2	Проектирование	25	08.06.23-11.06.23	
3	Реализация	35	12.06.23-14.06.23	
4	Контроль качества	15	15.06.23-16.06.23	
5	План разработки бюджета	5	17.06.23-18.06.23	
6	Оформление пояснительной записки	10	19.06.23-20.06.23	

Дата выдачи задания «__» _____ 2023 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 79 листов, 15 рисунков, 4 таблицы, 20 использованных источника, 1 приложение.

RUP, C#, БАЗА ДАННЫХ, КЛИЕНТ СЕРВЕРНОЕ ПРИЛОЖЕНИЕ МЕНЕДЖЕРА ПРОЦЕССОВ WINDOWS, IDEF1X, UML, SQL, MYSQL, СУБД,

Объектом разработки является клиент-серверное приложение менеджера процессов Windows.

Цель выпускной квалификационной работы –разработать клиент серверное приложение менеджера процессов Windows, позволяющее автоматизировать работу.

Технология разработки – RUP, C#.

Результат работы – клиент серверное приложение«Менеджера процессов Windows», отвечающее всем требованиям технического задания.

Для разработки программного продукта применялись языки C#, SQL, редактор «Visual Studio», система управления базами данных MySQL и операционная система Windows 10.

					ПГУ 09.03.02 – 8БР191.15 ПЗ			
Изм.	Лист	№ докум.	Под-	Дата				
Разраб.	Куликов Е.А				Клиент серверное приложение менеджера процессов Windows	Лит.	Лист	Листов
Провер.	Афонин А.Ю.						3	79
Н. Контр.	Такташкин Д.В.				Пояснительная записка	Группа 19ВИ1		
Утверд.	Козлов А.Ю							

Содержание

Введение.....	7
1 Анализ предметной области и требований.....	8
1.1 Анализ предметной области и постановка задачи.....	8
1.1.1 Анализ программ аналогов	9
1.1.1.1 Анализ «Process hacker»	9
1.1.1.2 Анализ « AnVir Task Manager Free».....	10
1.1.1.3 Анализ « Task Manager Deluxe».....	12
1.1.2 Постановка задачи.....	13
1.1.2.1 Назначение	13
1.1.2.2 Предполагаемая аудитория	14
1.1.2.3 Взгляд на продукт	15
1.1.2.4 Приложения и зависимости	16
1.1.3 Выбор технологий разработки.....	17
1.1.3.1 Методология RUP	17
1.1.3.2 Технология .Net	18
1.1.3.3 Seq.....	19
1.1.3.4 Технология обмена сокетами (Socket programming) в C#.....	20
1.1.4 Выбор инструментальных средств.....	20
1.2 Анализ требований.....	21
1.2.1 Функциональные бе	21
1.2.2 Нефункциональные требования.....	22
2 Проектирование программного обеспечения	28
2.1 Базовая архитектура системы.....	28

2.2	Логическая модель базы данных	29
2.2.1	Первая нормальная форма (1NF)	29
2.2.2	Вторая нормальная форма (2NF)	30
2.2.3	Третья нормальная форма (3NF).....	30
2.2.4	Пользовательский интерфейс клиентского приложения	31
2.2.5	Планирование разработки и оценка бюджета	32
2.2.6	Планирование проекта	33
3	Реализация программного обеспечения.....	41
3.1	Серверная часть приложения	41
4	Контроль качества программного обеспечения.....	43
	Заключение	46
	Список использованных источников	47
	Приложение А	51

ВВЕДЕНИЕ

В современном информационном обществе компьютеры играют важнейшую роль во всех сферах деятельности, будь то бизнес, образование, наука или государственное управление. Управление и контроль за компьютерными системами и их производительностью это ключевая задача для обеспечения эффективной работы многих организаций. Контроль за активными процессами и ресурсами ОС является одним из основных средств для обеспечения безопасности и продуктивности компьютерных систем.

Для удобства мониторинга процессов ОС Windows и контроля за запущенными приложениями на удаленных машинах, можно использовать клиент-серверное приложение мониторинга процессов. Оно позволяет удаленно контролировать и управлять процессами на нескольких компьютерах подключенных к сети, предоставляет оперативную информацию о текущей нагрузке на систему и быстрый доступ к информации о состоянии процессов.

Целью данного проекта является создание клиент-серверного приложения мониторинга процессов ОС Windows, которое будет обеспечивать надежный и безопасный контроль за запущенными процессами, а также предоставлять пользователю возможность управления процессами на удаленной машине. Применение данного приложения позволит улучшить эффективность работы компьютерных систем, повысить безопасность и сохранность данных.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Предметная область мониторинга процессов связана с отслеживанием работы операционной системы и приложений, запущенных на компьютерах сети. Клиент-серверное приложение мониторинга процессов должно предоставлять возможность удаленного контроля и управления процессами на удаленных машинах. Оно должно обеспечивать быстрый доступ и удобный пользовательский интерфейс для отслеживания состояния процессов, а также сведения о нагрузке на систему.

Основная задача приложения мониторинга процессов – обеспечить надежный и безопасный доступ к удаленным машинам через сеть Интернет или локальной сети. Приложение должно иметь возможность собирать данные о процессах на удаленной машине и передавать их на клиента для дальнейшего анализа. Кроме того, клиент-серверное приложение мониторинга процессов должно предоставлять возможность запуска и остановки процессов на удаленной машине, управления приоритетами процессов, а также возможность посылать сообщения пользователям удаленной машины.

Для успешной реализации и использования приложения мониторинга процессов необходимо обеспечивать безопасность передачи данных, аутентификацию пользователей и защиту от вредоносных атак. Кроме того, приложение должно быть масштабируемым и готовым к работе с большим количеством машин и пользователями.

Таким образом, основная задача клиент-серверного приложения мониторинга процессов – обеспечение контроля и управления процессами на удаленных машинах, с возможностью выполнения различных операций с процессами и предоставления информации о состоянии системы и нагрузке на нее. Приложение должно быть безопасным, масштабируемым и гибким для настройки под конкретные требования пользователей.

1.1.1 Анализ программ аналогов

Клиент серверные приложения по менеджеру процессов, такие как «Process hacker», «AnVir Task Manager Free», «Task Manager Deluxe».

1.1.1.1 Анализ «Process hacker»

Process Hacker это бесплатное программное обеспечение с открытым исходным кодом, предназначенное для управления и мониторинга активных процессов в операционной системе Windows. Оно предоставляет пользователю удобный интерфейс для просмотра всех запущенных процессов, а также подробную информацию о каждом из них.

На рисунке 1 представлен интерфейс приложения «Process hacker»

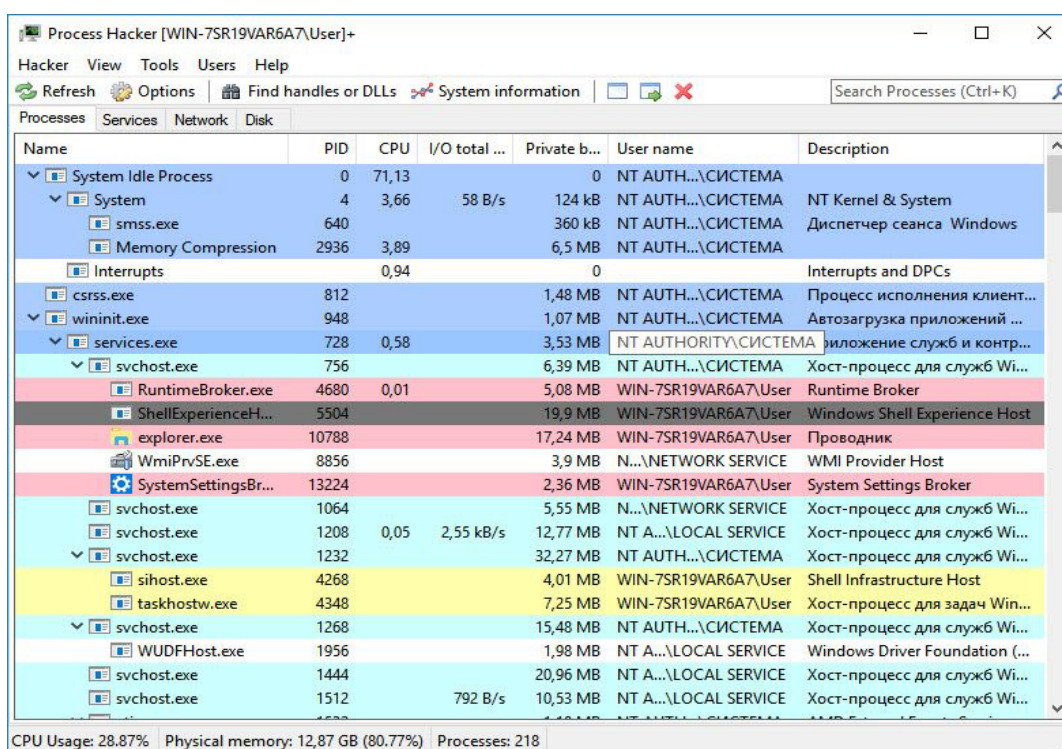


Рисунок 1 – Вкладка просмотра процессов компьютера на ОС Windows «Process hacker»

Основные возможности Process Hacker включают: Просмотр, управление и завершение процессов Мониторинг потребления ресурсов (память, процессор и т.д.) Просмотр и изменение сетевых соединений Анализ стека

вызовов для процессов и потоков Поддержка расширений и плагинов для повышения функционала

Преимуществами Process Hacker являются его открытый исходный код, хорошая поддержка операционных систем Windows, включая Windows 10, и возможность настройки интерфейса для удобства пользователя. Благодаря своим расширенным функциям, данный инструмент может стать полезным для анализа процессов и отслеживания некорректных работающих приложений.

Недостатком Process Hacker является необходимость изучения его возможностей и настроек для полноценного использования, что может потребовать времени и усилий. Также требуется осторожность при использовании данного программного обеспечения, чтобы не удалить или изменить важные процессы в системе.

1.1.1.2 Анализ « AnVir Task Manager Free»

AnVir Task Manager Free – это бесплатное приложение, предоставляющее пользователю возможность контролировать и оптимизировать нагрузку на систему. Приложение имеет много полезных функций, таких как мониторинг производительности, управление процессами и службами, а также анализ автозагрузки.

Окно просмотра запущенных процессов ОС Windows представлено на рисунке 2.

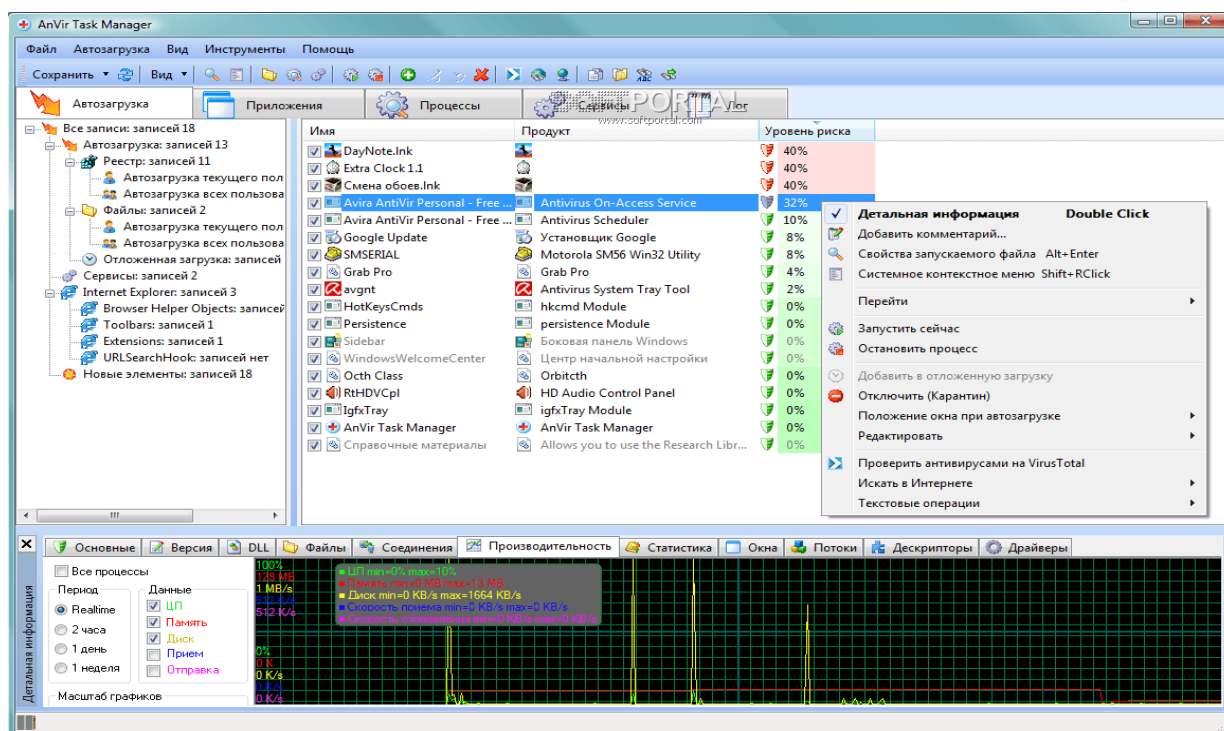


Рисунок 2 – Окно просмотра запущенных процессов «AnVir Task Manager Free»

Одна из главных особенностей AnVir Task Manager Free – это возможность просмотра всех запущенных процессов и их характеристик. Пользователь может увидеть список приложений и процессов, которые запущены в системе, а также количество используемой ими памяти и процессорного времени.

AnVir Task Manager Free также позволяет контролировать запуск служб и программ при старте операционной системы. Пользователь может выборочно отключить запуск определенных служб и программ, что может ускорить загрузку системы и увеличить ее производительность.

Еще одна полезная функция AnVir Task Manager Free – это возможность мониторинга сетевой активности. Пользователь может увидеть список приложений, которые используют сеть, а также объем передаваемых данных.

AnVir Task Manager Free также имеет функцию отслеживания изменений в системе, что может помочь пользователю быстро обнаружить и устранить проблемы с программами и службами.

Несмотря на многие полезные функции, приложение имеет некоторые недостатки. Например, AnVir Task Manager Free не имеет возможности отключить автозагрузку программ для всех пользователей, также некоторые функции мониторинга доступны только в платной версии приложения.

1.1.1.3 Анализ «Task Manager Deluxe»

Task Manager Deluxe – это программное обеспечение для контроля за активностью системы, которое включает в себя множественные функции, включая мониторинг процессов, служб, сетевой активности, а также автозагрузки. Приложение предоставляет профессиональные инструменты управления ресурсами для оптимизации производительности компьютера.

Окно просмотра запущенных процессов ОС Windows представлено на рисунке 3

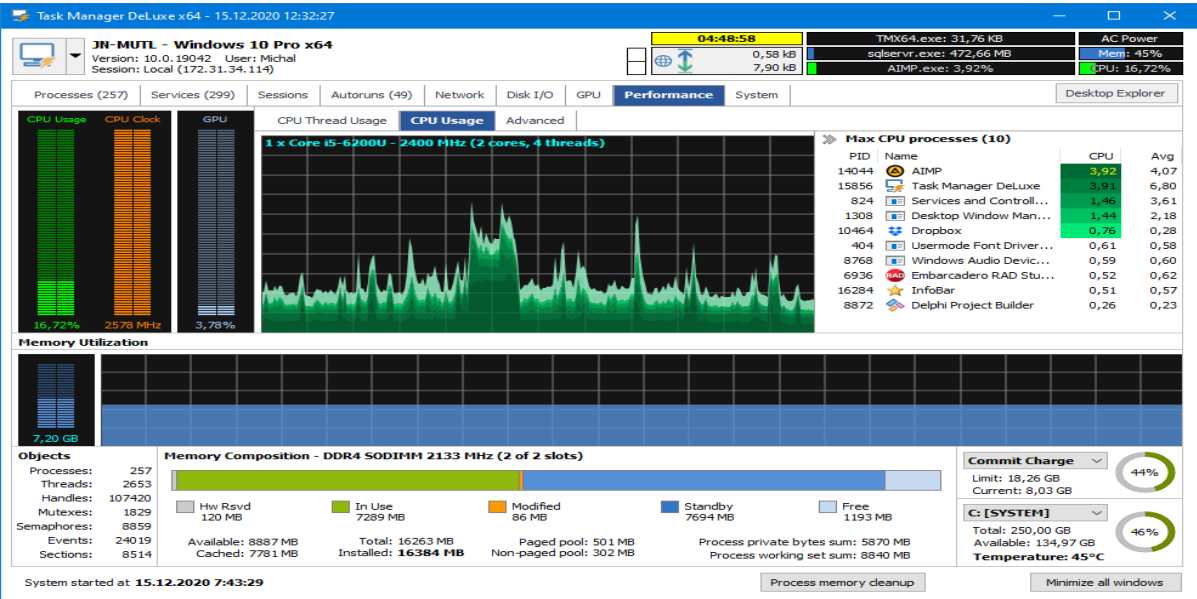


Рисунок 3 – Окно просмотра запущенных процессов ОС Windows «Task Manager Deluxe»

Одной из главных особенностей Task Manager Deluxe является отображение информации о запущенных процессах, включая имя процесса, процессорное время, память, количество потоков и т.д. Это поможет пользователям

определить процессы, использующие большое количество ресурсов, и убрать их для улучшения производительности системы.

Task Manager Deluxe также позволяет пользователю контролировать запуск программ и служб при старте системы. Это даст возможность пользователю выбирать, какие программы и службы должны быть запущены автоматически, и какие не должны.

Еще одной полезной функцией является мониторинг сетевой активности. Task Manager Deluxe отображает список приложений, которые используют сеть, объем передаваемых данных и скорость передачи данных. Это поможет пользователю отслеживать поток данных и улучшить скорость сетевого соединения.

Task Manager Deluxe также предоставляет графики производительности для отслеживания использования процессора, памяти, жесткого диска и сетевой активности. Графические отчеты помогут пользователям обнаружить какие-либо узкие места в системе и принять меры по их устранению.

Однако, Task Manager Deluxe в основном предназначен для профессионального использования и может быть сложен для новичков. Приложение также является платным, что может отпугнуть пользователей, которые ищут бесплатные решения для управления ресурсами компьютера.

1.1.2 Постановка задачи

1.1.2.1 Назначение

Данное клиент серверное приложение предназначена для просмотра запущенных процессов на персональном компьютере и автоматизации работы сотрудников на предприятии. Приложение должно показывать информацию о запущенных процессах. Приложение ориентирована на конечного пользователя, не имеющий навыков работой с компьютером. Поэтому клиент-серверное приложение должно обладать простым и интуитивным и понятным

интерфейсом. Также должно быть разработано подключение к компьютерам для просмотра запущенных процессов в локальной сети.

1.1.2.2 Предполагаемая аудитория

Аудитория клиент-серверного приложения менеджера процессов Windows может быть разнообразной, в зависимости от его предполагаемого назначения и функциональности. Вот некоторые возможные категории пользователей, которые могут быть заинтересованы в использовании такого приложения:

1. Системные администраторы: Данное приложение может быть полезным для системных администраторов, которые управляют сетевыми ресурсами и процессами в Windows-среде. Они могут использовать клиент-серверное приложение для мониторинга и управления процессами на удаленных компьютерах.

2. Разработчики программного обеспечения: Разработчики, работающие с Windows-приложениями, могут использовать данное приложение для мониторинга и управления запущенными процессами во время разработки и отладки.

3. ИТ-специалисты: ИТ-специалисты, отвечающие за обслуживание и поддержку рабочих станций и серверов в организации, могут использовать приложение для управления запущенными процессами, мониторинга ресурсов и решения проблем на удаленных компьютерах.

4. Администраторы баз данных: Администраторы баз данных могут использовать клиент-серверное приложение для мониторинга процессов, связанных с базами данных, и оптимизации их работы.

5. Системные аналитики: Аналитики, занимающиеся анализом производительности системы и оптимизацией ее работы, могут использовать приложение для сбора данных о запущенных процессах, времени их выполнения и потребляемых ресурсах.

Общий принцип состоит в том, что аудитория клиент-серверного приложения менеджера процессов Windows включает в себя технических специалистов и профессионалов, которым требуется более гибкий и детализированный контроль над запущенными процессами в Windows-среде.

1.1.2.3 Взгляд на продукт

Клиент-серверное приложение менеджера процессов – это инструмент, предназначенный для удобного мониторинга и управления процессами в операционной системе Windows. Он предоставляет возможность контролировать запущенные процессы, анализировать их использование ресурсов, управлять приоритетами и завершать ненужные процессы.

Взгляд на такой продукт может быть следующим:

1. Мониторинг процессов: Пользователь может видеть список всех текущих процессов, запущенных на компьютере, а также основную информацию о них, такую как идентификатор процесса, имя, потребление ресурсов и состояние процесса. Это позволяет быстро оценить текущую загрузку системы и определить проблемные процессы.

2. Управление процессами: Пользователь может выполнять действия с процессами, такие как изменение приоритета, замораживание (приостановка) или завершение нежелательных процессов. Это дает возможность оптимизировать загрузку системы, предотвращать зависания и устранять возникающие проблемы.

3. Детальная информация о процессах: Пользователь может получить подробную информацию о выбранном процессе, такую как список потоков, используемые ресурсы (память, процессор), загружаемые модули и другую информацию. Это позволяет осуществить более глубокий анализ процесса и определить его влияние на систему или другие процессы.

4. Удаленное управление: Если клиент-серверное приложение предоставляет возможность удаленного подключения к компьютерам, пользователь может управлять процессами на удаленных компьютерах. Это особенно

полезно для системных администраторов и ИТ-специалистов, управляющих сетями компьютеров.

В целом, клиент-серверное приложение менеджера процессов предоставляет удобный и гибкий интерфейс для мониторинга и управления процессами, позволяя эффективно управлять ресурсами системы и оптимизировать ее работу.

1.1.2.4 Приложения и зависимости

При разработке клиент-серверного приложения менеджера процессов Windows могут потребоваться следующие приложения и зависимости:

1. Язык программирования: Необходимо выбрать язык программирования, который поддерживает работу с сокетами и возможностями операционной системы Windows. Примеры таких языков включают Python, Java, C# и C++.

2. Библиотеки и фреймворки: В зависимости от выбранного языка программирования, могут потребоваться специфические библиотеки или фреймворки для работы с сокетами, обработки данных или взаимодействия с Windows API. Например, в Python вы можете использовать библиотеки, такие как `socket`, `asyncio`, `twisted`. Для C# можно использовать .NET Framework или ASP.NET.

3. Windows API: Многие действия связанные с управлением процессами и получением информации о них потребуют использования Windows API. Например, для получения списка процессов можно использовать функцию `EnumProcesses` из библиотеки `kernel32.dll`.

4. База данных (опционально): Если приложение должно сохранять информацию о процессах или предоставлять исторические данные, можно использовать базу данных для хранения и запросов данных. Для этого можно выбрать соответствующую базу данных, например, SQLite для локальных приложений или SQL Server, MySQL для удаленного доступа.

5. Графический интерфейс (опционально): Если требуется графический интерфейс для приложения, могут потребоваться инструменты разработки интерфейса, такие как Windows Forms, WPF или Qt.

6. Серверная инфраструктура (опционально): В случае, если вы разрабатываете распределенное приложение, могут потребоваться серверные компоненты, такие как веб-сервер или база данных на сервере, для обработки запросов от клиентов.

Важно помнить, что конкретные приложения и зависимости будут различаться в зависимости от выбранного языка программирования и требований вашего приложения.

1.1.3 Выбор технологий разработки

1.1.3.1 Методология RUP

RUP (Rational Unified Process) – это методология разработки программного обеспечения, созданная в Rational Software (приобретенная впоследствии IBM). RUP является итеративной и инкрементальной методологией, которая сфокусирована на улучшении качества разработки, повышении эффективности проекта и достижении бизнес-целей.

Методология RUP помогает командам разработчиков выполнять проекты в соответствии с планами, учитывая все требования заказчика и применяя наиболее эффективные методы разработки. RUP включает в себя циклы, состоящие из четырех основных фаз:

1. Начальная фаза (Inception) – включает в себя определение целей проекта, анализ требований, соответствующих бизнес-целям и создание предварительного плана проекта.

2. Фаза разработки (Elaboration) – включает в себя определение архитектуры проекта, создание более детальных планов проекта, разработку системы и ее тестирование.

3. Фаза конструкции (Construction) – включает в себя разработку функционала, создание документации, тестирование и интеграцию.

4. Фаза трансферта (Transition) – последняя фаза, которая осуществляет передачу готовой системы заказчику.

В каждой фазе проекта, методология RUP предоставляет множество артефактов, инструментов, шаблонов и библиотек. Это позволяет команде разработчиков сократить время разработки, улучшить качество продукта и снизить риски.

Методология RUP основана на гибкой системе управления, что позволяет ее применение как для крупномасштабных проектов, так и для более мелких. Она применяется во многих компаниях, включая IBM, Ericsson, HP, Sun Microsystems, и других.

1.1.3.2 Технология .Net

C# – это язык программирования, который используется для создания приложений на платформе .NET. Он был разработан компанией Microsoft в 2000 году и является одним из основных языков для создания Windows-приложений.

Основные принципы технологии C#:

1. Объектно-ориентированное программирование – это методология программирования, которая опирается на концепцию объектов, которые объединяют данные и методы по их обработке.

2. Управляемый код – это способность C#-приложений работать в CLR (Common Language Runtime), который обеспечивает единый исполнительный код для всех языков платформы.

3. Сборка мусора – автоматический механизм, который освобождает память, занятую объектами, которые больше не используются.

4. Многопоточность – возможность одновременной работы нескольких потоков исполнения, что позволяет повысить эффективность приложения.

5. Лямбда-выражения – это анонимные функции, которые позволяют явно указывать аргументы и возвращаемые значения.

6. Async/await – используется для асинхронного программирования, что позволяет создавать быстрые и ресурсоэффективные приложения.

7. LINQ – это язык запросов на собственные объектные структуры, который обеспечивает более эффективный доступ к данным.

1.1.3.3 Seq

Seq – это лог-анализатор, который позволяет анализировать, мониторить и исследовать лог-файлы в реальном времени. Он поддерживает ряд языков запросов, а также имеет возможности для визуализации данных.

Основные возможности Seq:

1. Анализ и мониторинг журналов в режиме реального времени с помощью точного и эффективного движка.

2. Поддержка языка запросов Seq, который облегчает поиск конкретных сообщений в журнальных файлах.

3. Создание мощных и динамических дашбордов для мониторинга и визуализации журналов.

4. Интеграция с сервисами Azure DevOps, Slack, и Microsoft Teams для уведомления о критических событиях.

5. Централизованное хранение журнальных файлов, что облегчает анализ данных.

6. Расширяемость Seq с помощью плагинов и API.

Seq используется во многих компаниях для мониторинга и анализа журналов с целью выявления проблем и выработки стратегий предотвращения их возникновения в будущем. Seq также помогает компаниям быть более эффективными и экономичными в работе с журнальными файлами.

1.1.3.4Технология обмена сокетами (Socket programming) в C#

Позволяет создавать и управлять сетевыми соединениями между различными устройствами. С помощью сокетов можно обмениваться данными через Интернет, локальную сеть или даже на одной машине.

Для начала работы с сокетами необходимо создать экземпляр класса Socket, указав тип протокола (TCP или UDP), адрес и порт для соединения. Затем можно использовать методы Send и Receive для отправки и приема данных.

С помощью технологии обмена сокетами можно создавать различные сетевые приложения, например, клиент-серверные приложения для передачи файлов, игры, чаты и т д

1.1.4 Выбор инструментальных средств

Для хранения логов и просмотра ошибок клиент-серверного приложения, было принято решение использовать базу данных, в качестве СУБД выбран MicrosoftSqlServer, которая использует язык запросов SQL. Microsoft SQL Server (SQL Server) – это СУБД, разработанная корпорацией Microsoft, которая работает на платформе Windows. Она позволяет создавать и управлять базами данных, а также предоставляет множество инструментов для работы с данными, включая язык запросов SQL. Для разработки клиент-серверного приложения была выбрана программа «VisualStudio» – интегрированная среда разработки приложений (IDE), которая поддерживает множество языков программирования, включая C#, C++, Visual Basic, F #, Python, JavaScript, TypeScript и другие. Он разработан и поддерживается компанией Microsoft, и предоставляет мощные средства для разработки и тестирования программного обеспечения, включая отладку, профилирование, автоматизацию сборки и многие другие функции.

В качестве языка программирования был использован C#. Данный язык программирования позволяет создавать современные приложения, и использовать современные технологии и библиотеки.

В качестве языка моделирования был выбран UML. UML (Unified Modelling Language) – это спецификация графического языка моделирования, применяемого для описания и проектирования систем. UML включает множество графических символов и нотаций, которые помогают разработчикам ясно и точно определить, как работает их система или приложение.

Для проектирования UML – диаграмм был использован Microsoft Office Visio. Для распределения и планирования ресурсов был задействован Microsoft Project.

1.2 Анализ требований

Требования – это подробное описание функций и ограничений, которые должны быть реализованы в системе. Существует два основных типа требований:

- функциональные – описывают, какое поведение должна предоставлять система.
- нефункциональные – описывают специальные свойства или ограничения, накладываемые на систему.

1.2.1 Функциональные требования

Диаграммы вариантов использования помогают определить необходимые действия, чтобы пользователи смогли достичь своих целей взаимодействуя с системой. Эти диаграммы могут использоваться для определения потенциальных проблем, связанных с применением системы, а также для определения необходимых функций и технологий.

В соответствии с поставленной задачей были построена одна диаграмма вариантов использования.

Вариант использования «Найти пользователя» включает в себя варианты: «Привязать компьютер пользователя к локальной сети», «Отвязать пользователя».

Диаграмма вариантов использования представлены на рисунках 4 и 5

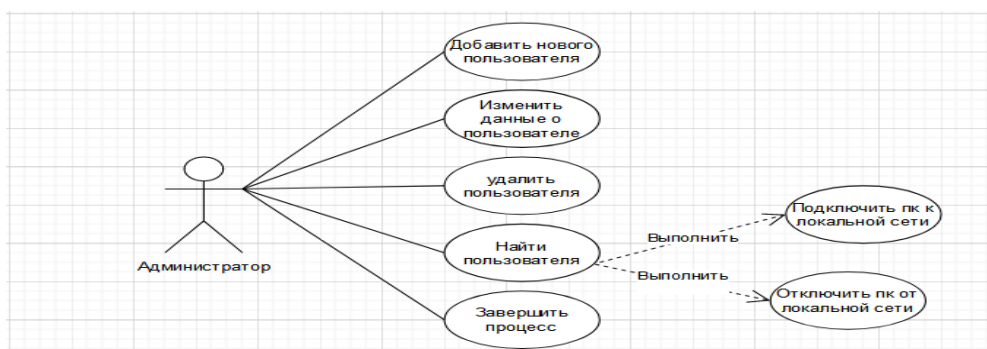


Рисунок 4 – Диаграмма вариантов использования (функции администратора)

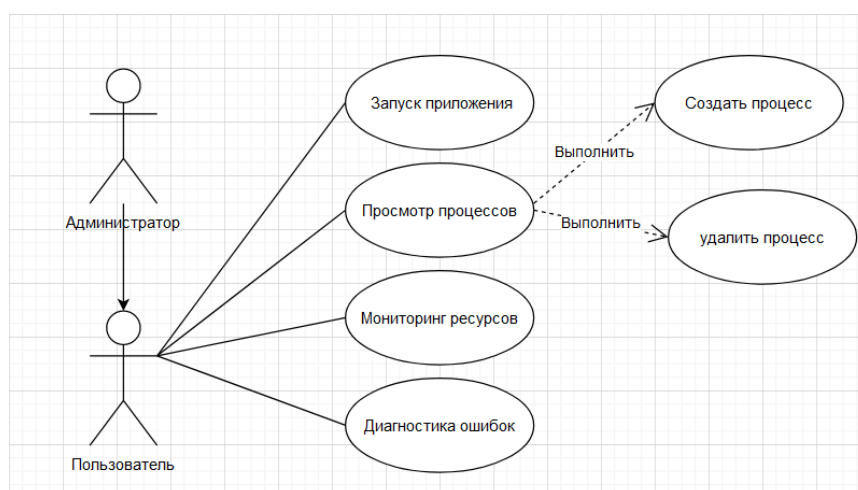


Рисунок 5– Диаграмма вариантов использования (функции пользователя)

1.2.2 Нефункциональные требования

Нефункциональные требования определяют параметры работы системы или программного продукта, а также ее характеристики и свойства

- Приложение должно быть совместимо с операционной системой Windows.

- Приложение должно обеспечивать работу с процессами системы Windows, включая создание, изменение и завершение процессов.
- Приложение должно предоставлять пользователю информацию о запущенных процессах, включая имя процесса, идентификатор процесса, использование ресурсов и другую статистику.
- Пользователь должен иметь возможность изменять приоритет и привилегии запущенных процессов.
- Пользователь должен иметь возможность завершать нежелательные или зависшие процессы.
- Приложение должно иметь возможность отображать иерархию процессов, связанных с запущенным процессом.
- Приложение должно иметь возможность отображать журнал системных сообщений и ошибок, связанный с запущенными процессами.
- Приложение должно предоставлять пользователю возможность управлять запуском и остановкой служб Windows.
- Приложение должно предоставлять пользователю возможность перезагружать или выключать компьютер.
- Пользователь должен иметь возможность выполнить поиск по запущенным процессам с помощью фильтров, таким как имя процесса или использование ресурсов.
- Приложение должно быть защищено от несанкционированного доступа, и пользователь должен иметь возможность установить пароль или другие меры безопасности для доступа к приложению.
- Приложение должно иметь возможность сохранять и восстанавливать состояние процессов после перезагрузки компьютера.

1.2.2.1 Интерфейсы пользователей

Клиент серверное приложение менеджера процессов Windows должно обладать удобным и интуитивно понятным графическим интерфейсом пользователя

Клиент-серверные приложения менеджера процессов Windows на языке C# могут иметь различные интерфейсы пользователя. Ниже приведены некоторые из распространенных типов интерфейсов пользователя для такого приложения:

1. Графический интерфейс Windows Forms: Windows Forms предоставляет набор элементов управления и возможностей для создания графического интерфейса пользователя в стиле Windows. Вы можете создавать окна, кнопки, текстовые поля, таблицы и другие элементы управления для взаимодействия с пользователями.

2. Графический интерфейс WPF (Windows Presentation Foundation): WPF предлагает более современный и гибкий подход к созданию графического интерфейса пользователя с использованием XAML (eXtensible Application Markup Language) и богатого набора визуальных элементов. Он предоставляет более гибкую компоновку, анимацию и стилизацию интерфейса пользователя.

1.2.2.2 Требования к производительности

. Количество одновременных подключений: Приложение должно поддерживать достаточное количество одновременных подключений от клиентов, чтобы обеспечить эффективное управление процессами на сервере.

2. Отзывчивость интерфейса: Интерфейс приложения должен реагировать быстро на действия пользователя и обновлять информацию о процессах в реальном времени. Это может потребовать оптимизированной работы сетевых запросов и эффективной обработки данных на сервере.

3. Производительность работы с базой данных: Приложение может использовать базу данных для хранения информации о процессах. Требуется, чтобы операции чтения и записи были оптимизированы, чтобы минимизировать время отклика приложения.

4. Масштабируемость: Приложение должно быть способно масштабироваться для работы с большим объемом процессов и большим количеством

пользователей. Это может потребовать использования распределенных вычислений, кластеризации или других технологий масштабирования.

5. **Безопасность:** Приложение должно обеспечивать безопасность данных и защищать их от несанкционированного доступа. Возможно, потребуется использование механизмов аутентификации, авторизации и шифрования данных.

6. **Оптимизация ресурсов сервера:** Приложение должно эффективно использовать ресурсы сервера, такие как процессорное время, память и сетевые ресурсы. Это может включать оптимизацию кода и алгоритмов, чтобы уменьшить нагрузку на сервер.

7. **Обработка ошибок и исключительных ситуаций:** Приложение должно обрабатывать ошибки и исключительные ситуации, такие как потеря связи с сервером или некорректные данные, чтобы предотвратить сбои и обеспечить надежную работу.

1.2.2.3 Требования к безопасности

Доступ к клиентской части приложения должен осуществляться при помощи аутентификации в локальной сети. А доступ аутентифицированного пользователя к определенным функциям системы должен осуществляться при помощи авторизации.

1. **Аутентификация пользователей:** Приложение должно иметь механизм аутентификации, чтобы убедиться, что только уполномоченные пользователи могут получить доступ к приложению. Это может включать использование паролей, двухфакторной аутентификации или сертификатов.

2. **Авторизация доступа:** Приложение должно определять уровень доступа и права пользователей на основе их ролей или иных параметров. Это позволяет предотвратить несанкционированный доступ пользователей к ограниченным ресурсам или функциям приложения.

3. **Защита данных:** Данные, хранящиеся на сервере, должны быть защищены от несанкционированного доступа и модификации. Это может включать

шифрование данных при их передаче и хранении, использование безопасных протоколов и методов хеширования паролей.

4. Защита от атак: Приложение должно быть устойчивым к различным видам атак, таким как переполнение буфера, инъекции кода или атаки словарными паролями. Это может потребовать применения соответствующих методов кодирования, фильтрации пользовательского ввода и обнаружения атак в реальном времени.

5. Журналирование и мониторинг: Приложение должно иметь систему журналирования, которая записывает важные события и действия пользователей для последующего анализа. Это помогает выявить потенциальные угрозы безопасности и предотвратить инциденты.

6. Обновление безопасности: Приложение должно иметь механизм обновления безопасности, чтобы устранять уязвимости и предотвращать возможные атаки. Это может включать регулярные патчи или обновления, а также проверку наличия уязвимостей в сторонних компонентах.

7. Физическая безопасность: Серверы, на которых работает приложение, должны быть физически защищены от несанкционированного доступа и повреждений. Это может включать использование контролируемого доступа в помещение серверной комнаты, регулярную проверку наличия физических уязвимостей и резервное копирование данных.

8. Обучение пользователей: Пользователи приложения должны быть обучены базовым принципам безопасности и правилам использования приложения. Это поможет предотвратить ошибки или несанкционированное использование, которые могут повлечь за собой угрозы для безопасности.

Учитывая изменчивость угроз и разнообразие архитектур и требований приложений, важно провести анализ рисков и конкретных требований безопасности, чтобы определить соответствующие меры защиты для конкретного клиент-серверного приложения менеджера процессов Windows.

1.2.2.4 Атрибуты качества ПО

Таблицы, открытые в программном обеспечении, обновляются автоматически. Когда в системе регистрируется новая запись в журнале учета, новый работник, то система автоматически обновляет таблицу с данными в каждом открытом экземпляре клиент серверного приложения администратора.

Надежность: Важным атрибутом является надежность, чтобы приложение было стабильным и не наносило вреда операционной системе или другим процессам. Надежность может быть достигнута путем тщательного обработки ошибок, обработки исключений и контроля некорректных входных данных.

Производительность: Приложение должно обеспечивать высокую производительность для обеспечения быстрого и отзывчивого взаимодействия с процессами Windows. Нужно оптимизировать процессы передачи данных между клиентом и сервером, обеспечивая эффективное использование сетевых ресурсов и минимизируя задержки.

2 Проектирование программного обеспечения

2.1 Базовая архитектура системы

Рассматриваемое клиент серверное приложение имеет трехуровневую архитектуру клиент-сервер-сервер.

Архитектура клиент-серверного приложения представлена на рисунке

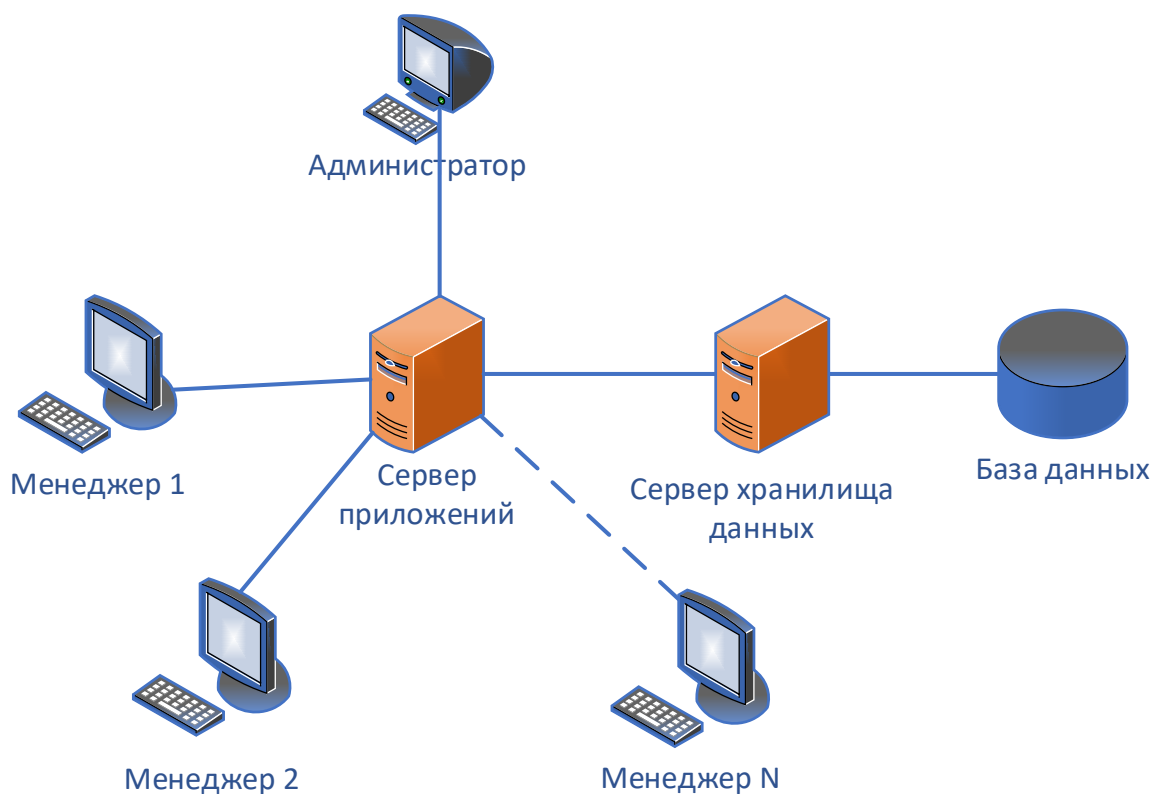


Рисунок 6 – Архитектура клиент-серверного приложения

Описана системная архитектура, состоящая из трех основных компонентов. Первый компонент – клиентская станция, на которой запускаются отдельные компоненты приложения. Второй компонент – приложение, размещенное на сервере или на компьютере администратора, к которому обращаются удаленные пользователи системы. Третий компонент – сервер базы данных, размещенный на отдельной станции и содержащий базу данных, с которой постоянно взаимодействует приложение для управления данными.

2.2 Логическая модель базы данных

Логическая модель базы данных создается на основе концептуальной модели и выбранной модели данных, независимо от СУБД. Отношения в модели декомпозируются с целью избежать избыточности, оставляя в каждом отношении только первичные факты. Этот процесс называется нормализацией, которая направлена на уменьшение избыточности, что позволяет предотвратить потенциальную противоречивость информации, хранящейся в БД. Нормальная форма отношения в реляционной модели данных характеризует его с точки зрения избыточности, которая может привести к ошибочным результатам выборки или изменения данных.

2.2.1 Первая нормальная форма (1НФ)

Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение. Что же касается различных таблиц, то они могут не быть правильными представлениями отношений и, соответственно, могут не находиться в 1НФ.

В 1НФ все атрибуты в таблице должны быть одноатомарными, то есть не должны иметь повторяющихся или составных значений. Например, таблица с информацией о клиентах не должна иметь столбца "телефон", содержащего списки телефонных номеров, а должна иметь столбец для каждого телефонного номера.

Также в 1НФ каждая строка в таблице должна быть уникальной, то есть не допускается наличие дублирующих строк.

Приведение таблицы к 1НФ включает в себя разделение составных атрибутов на отдельные столбцы, удаление повторяющихся атрибутов и добавление первичного ключа для обеспечения уникальности строк.

После приведения таблицы к 1НФ, можно переходить к следующим уровням нормализации для дальнейшего улучшения структуры базы данных и избавления от аномалий в данных.

2.2.2 Вторая нормальная форма (2NF)

Вторая нормальная форма (2НФ) - это уровень нормализации базы данных, который достигается после приведения таблицы к первой нормальной форме (1НФ). 2НФ устраняет проблемы, связанные с функциональными зависимостями атрибутов в нереляционных таблицах.

В 2НФ каждый неключевой атрибут зависит только от целого первичного ключа, а не от его части. Другими словами, 2НФ требует, чтобы все атрибуты, которые не являются частью ключа, полностью зависели от всего ключа, а не от его частей.

При нарушении 2НФ в таблице могут возникать аномалии в данных. Например, если в таблице "Покупки" есть столбцы "Имя_товара", "Цена_товара" и "Количество_товара", причем ключ состоит только из столбца "Номер_покупки", то возможны некорректные данные, такие как повторяющиеся значения "Имя_товара" и несогласованные данные о цене и количестве для каждого товара.

Чтобы привести таблицу к 2НФ, необходимо:

1. Выделить все функционально зависимые от ключа атрибуты в отдельные таблицы.
2. Создать связи между этими таблицами с помощью внешних ключей, связывающих их с основной таблицей.

Таким образом, применение 2НФ улучшает структуру базы данных, устраняя аномалии и повышая ее эффективность и надежность.

2.2.3 Третья нормальная форма (3NF)

Третья нормальная форма (3NF) – это стандарт базы данных, который помогает устранить избыточность данных и улучшить их целостность.

В соответствии с третьей нормальной формой, каждый атрибут в таблице должен быть функционально зависимым от первичного ключа таблицы, а не зависеть от других не первичных атрибутов. Другими словами, каждый атрибут должен относиться к одному и только одному ключу.



Рисунок 7 – Логическая модель базы данных

2.2.4 Пользовательский интерфейс клиентского приложения

Исходя из диаграммы вариантов использования и этапа анализа требований был спроектирован простой и интуитивно понятный графический пользовательский интерфейс.

Графический пользовательский интерфейс был разработан при помощи специальной графической библиотеки Visual Studio. На рисунке представлен графический интерфейс главного окна

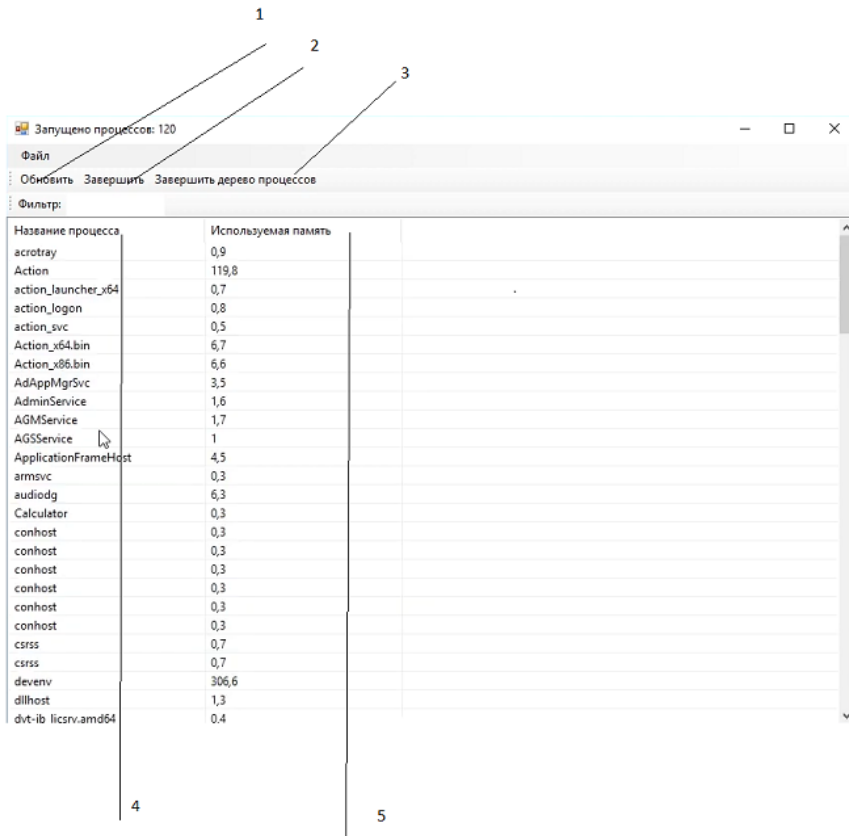


Рисунок 8 – Графический интерфейс главного окна приложения

- 1 – кнопка для обновления списка запущенных процессов;
- 2 – кнопка для завершения выбранного процесса ;
- 3 – кнопка для завершения всех процессов;
- 4 – таблица с информацией об запущенных процессах;
- 5 – таблица с информацией об количестве используемой оперативной памяти;

2.2.5 Планирование разработки и оценка бюджета

Управление проектами охватывает все этапы жизненного цикла проекта – от начальной фазы идеи до завершения проекта и передачи его результатов в эксплуатацию. Оно включает в себя такие процессы, как определение целей и задач, планирование работы и распределение ресурсов, управление временем и бюджетом, контроль за ходом работ и управление рисками и изменениями.

В проектном управлении кроме технических аспектов учитываются социальные и коммуникационные аспекты, взаимодействие с заинтересованными сторонами, командообразование и управление командой. Это помогает достичь качественного результата в соответствии с требованиями и ограничениями проекта, а также удовлетворить интересы всех заинтересованных лиц – заказчика, исполнителей, пользователей и др.

Знания и опыт, используемые в управлении проектами, постоянно совершенствуются и развиваются. Это обеспечивает возможность улучшения процессов управления и получения наилучших результатов в проектах.

2.2.6 Планирование проекта

В ходе планирования проекта было выделено 12 задач. Список задач проекта, их виды, длительность и предшественники приведены в таблице 1.

Таблица 1 – Задачи проекта

№	Название	Вид Задачи	Предшественники	Длительность
1	Начало проекта	Веха	-	-
2	Анализ предметной области	Фаза	-	-
3	Просмотр рынка клиент серверных приложений	Задача	-	5
4	Сбор информации о существующих клиент серверных приложениях	Задача	4	7
5	Анализ требований	Фаза	-	-
6	Анализ функциональных требований	Задача	5	14
7	Анализ используемых технологий	Задача	5;6	9
8	Проектирование	Фаза	-	-
9	Проектирование базовой архитектуры	Задача	6;7	4
10	Проектирование логической модели базы данных	Задача	6	2,5
11	Проектирование интерфейса пользователя	Задача	6;7;9	14
12	Реализация	Фаза	-	-
	Реализация физической модели базы данных	Задача	10	4

Продолжение таблицы 1

14	Реализация серверной части приложения	Задача	13	8
15	Реализация клиентской части приложения	Задача	9;13;11	22
16	Тестирование	Фаза	-	-
17	Функциональное тестирование	Задача	16;17	10
18	Завершение проекта	Фаза	-	-
19	Оформление документации	Задача	17	10
20	Конец проекта	Веха	-	-

В MS Project 2013 создадим проект с начальной датой 01.12.22. Занесем выше перечисленные задачи в созданный проект.

Выделим необходимые ресурсы для выполнения каждой из задач данного проекта. Ресурсы и затраты представлены в таблице 2.

Таблица 2 – Ресурсы проекта

Название ресурса	Тип	Затраты			Затраты на исп.
		Таблица норм	Станд. Ставка	Ставка сверхур.	
Программист 1	Т	А	25000 р./мес	350 р./ч	-
Программист 2	Т	А	20000 р./мес	280 р./ч	-
Аналитик	Т	А	15000 р./мес	250 р./ч	-
Тестировщик	Т	А	15000 р./мес	250 р./ч	-
Компьютер	М	А	18000 р.		-
Принтер	М	А	4000 р.		-
Бумага	М	А	250 р.		-
Интернет	З				-

Распределение ресурсов по задачам представлено в таблице 3.

Таблица 3 – Распределение ресурсов по задачам

Название	Ресурс	Единицы (затраты)	Таблица норм затрат
Начало проекта	Компьютер	4	А
Просмотр рынка информационных систем	Аналитик, Интернет	100	А 250
Сбор информации о существующих информационных системах	Аналитик, Интернет	100	А 350
Анализ функциональных требований	Аналитик, Программист 1	100 100	А А

Продолжение таблицы 3

Проектирование базовой архитектуры	Программист 2	100	A
Проектирование логической модели базы данных	Программист 1	100	A
Проектирование интерфейса пользователя	Программист 2	100	A
Реализация физической модели базы данных	Программист 2	100	A
Реализация серверной части приложения	Программист 2	100	A
Реализация клиентской части приложения	Программист 1, Программист 2	100 100	A A
Функциональное тестирование	Тестировщик, Программист 2	50 100	A A
Название	Ресурс	Единицы (затраты)	Таблица норм затрат
Оформление документации	Программист 1, Тестировщик, Принтер, Бумага	100 100 1 1	A A A A

Занесем выше перечисленные ресурсы в проект. Результаты представлены на рисунке 12.

	Название ресурса ▼	Тип ▼	Единицы измерения материалов ▼	Краткое название ▼	Макс. единиц ▼	Стандартная ставка ▼	Ставка сверхурочных ▼	Начисление ▼	Базовый календарь ▼
1	Программист1	Трудовой		ПроГ1	100%	25 000,00р./мес	350,00р./ч	Пропорциональное	Стандартный
2	Программист2	Трудовой		ПроГ2	100%	20 000,00р./мес	280,00р./ч	Пропорциональное	Стандартный
3	Аналитик	Трудовой		Аналит	100%	15 000,00р./мес	250,00р./ч	Пропорциональное	Стандартный
4	Тестировщик	Трудовой		Тестир	100%	15 000,00р./мес	250,00р./ч	Пропорциональное	Стандартный
5	Компьютер	Материальный	Штука	Комп		18 000,00р.		Пропорциональное	
6	Принтер	Материальный	Штука	Принт		4 000,00р.		Пропорциональное	
7	Бумага	Материальный	Пачка	Бум		250,00р.		Пропорциональное	
8	Интернет	Затраты		Инет				Пропорциональное	

Рисунок 12 – Ресурсы проекта

Далее закрепим за задачами соответствующие ресурсы. После, MS Project 2013 предложит наглядную иллюстрацию календарного плана в виде диаграммы Ганта. Полученная диаграмма Ганта приведена на рисунке 9.

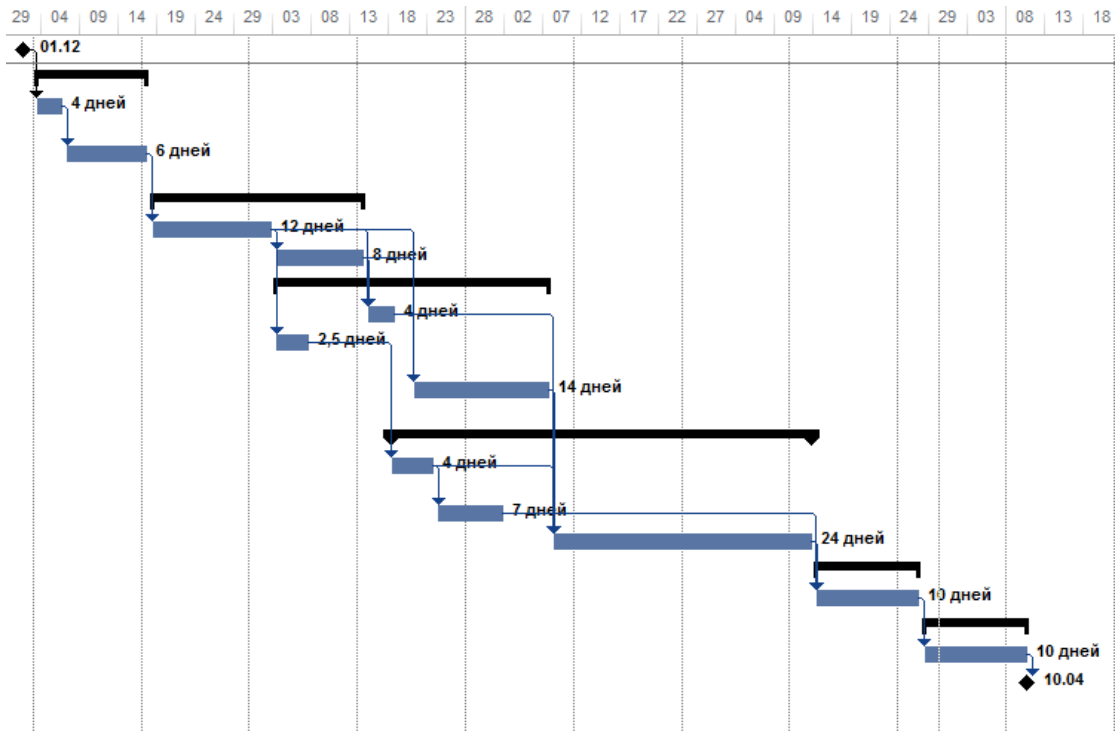


Рисунок 9 – Диаграмма Ганта

Сетевой график данного проекта представлен на рисунке 10. Красным цветом на рисунке 14 показан критический путь.

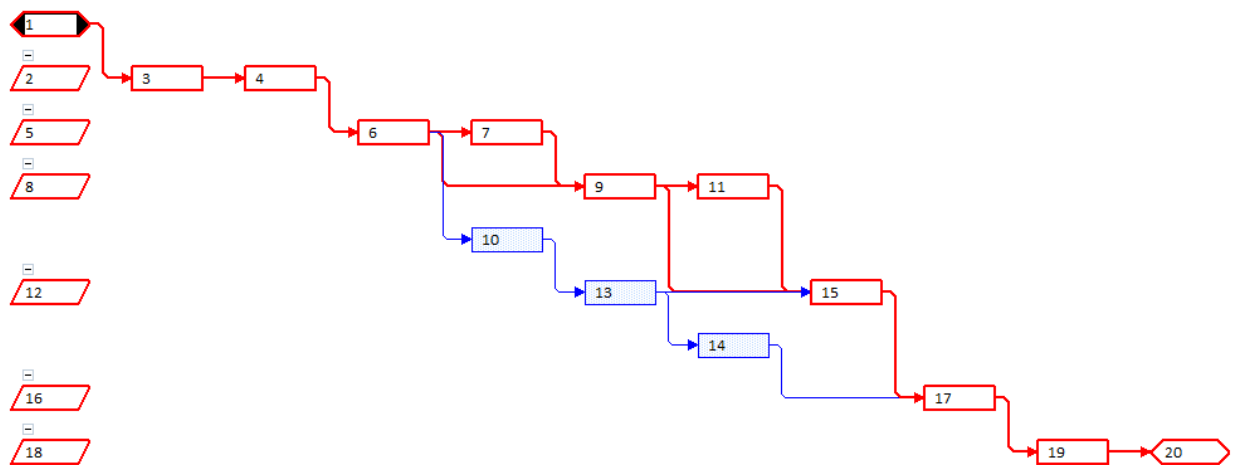


Рисунок 10 – Сетевой график

Графики загрузки трудовых ресурсов программиста 1, программиста 2, аналитика и тестировщика представлены на рисунках .

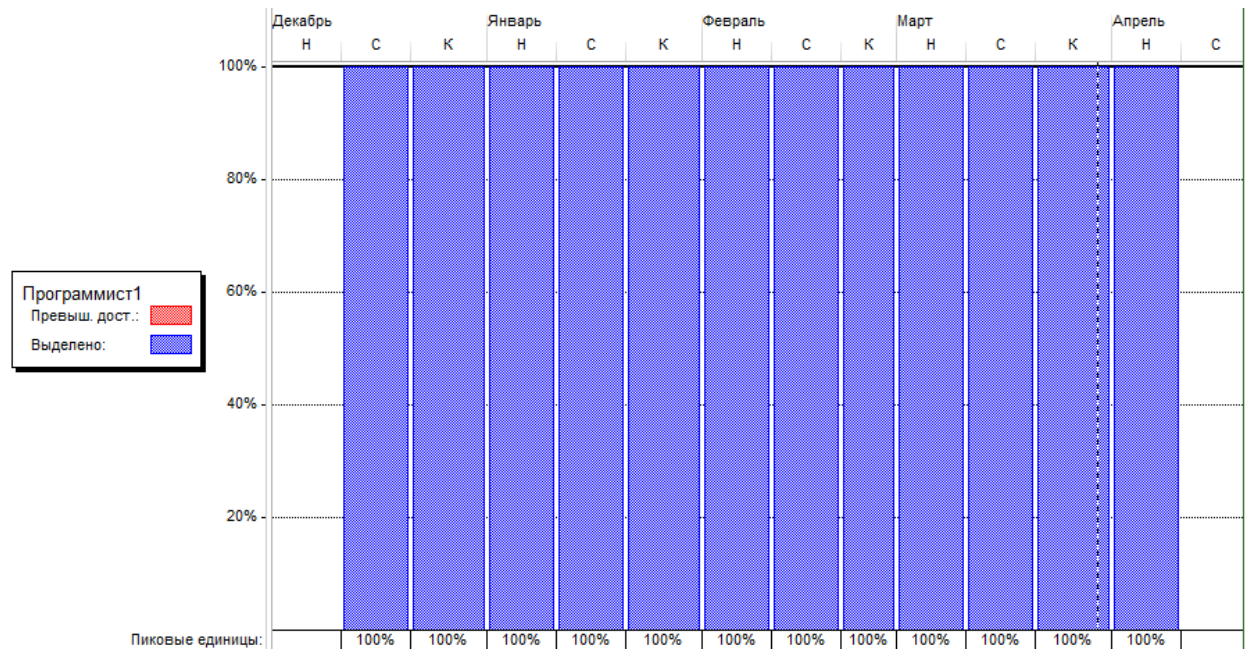


Рисунок 11 – График загрузки программиста 1

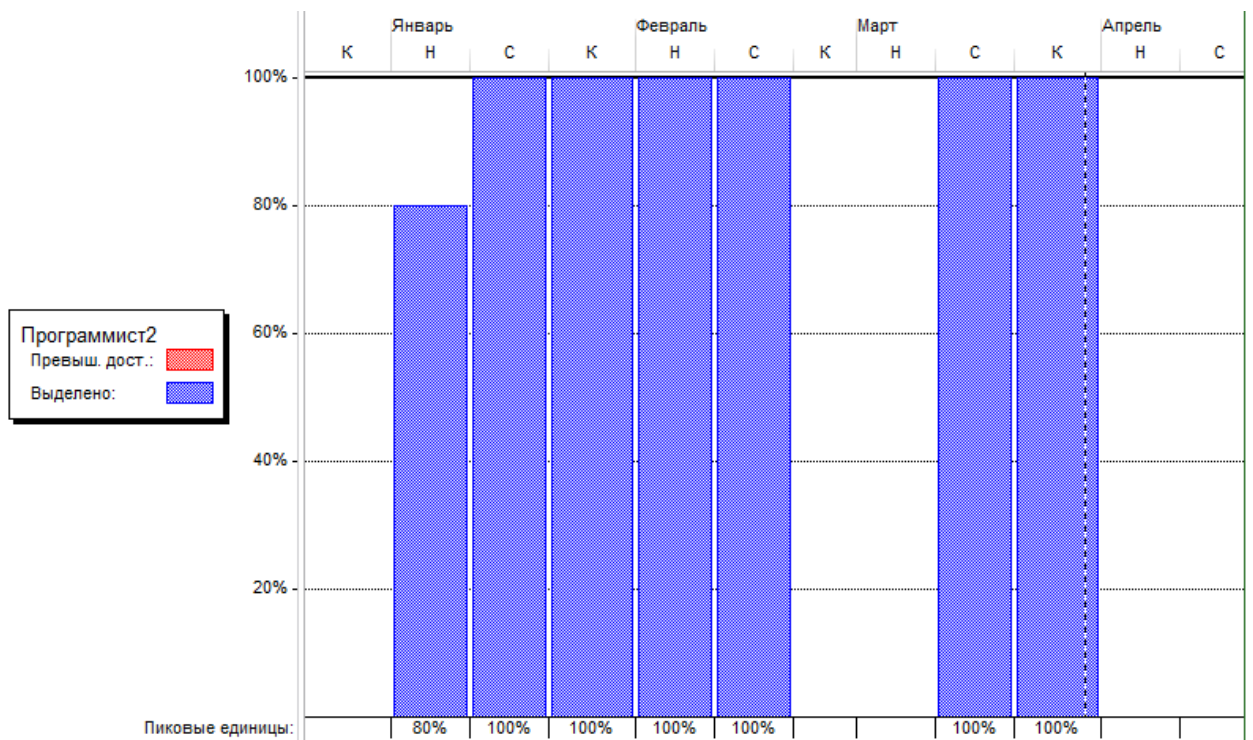


Рисунок 12 – График загрузки программиста 2

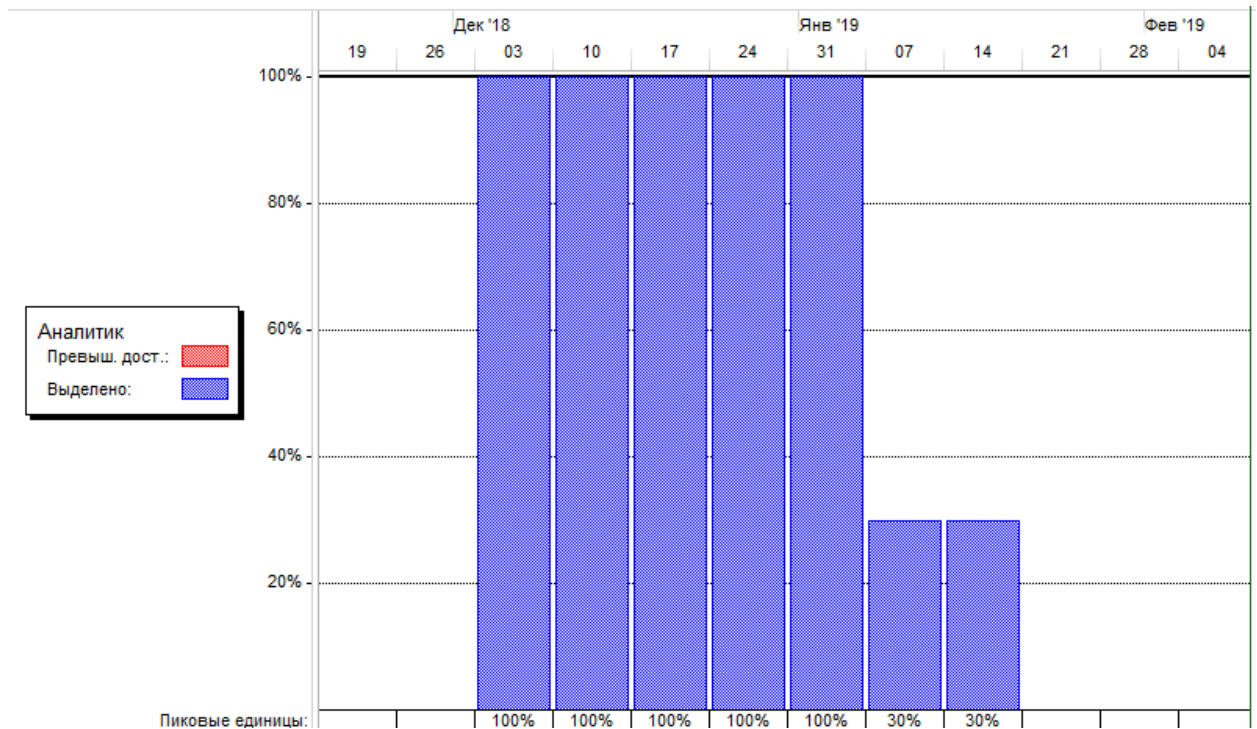


Рисунок 13 – График загрузки аналитика

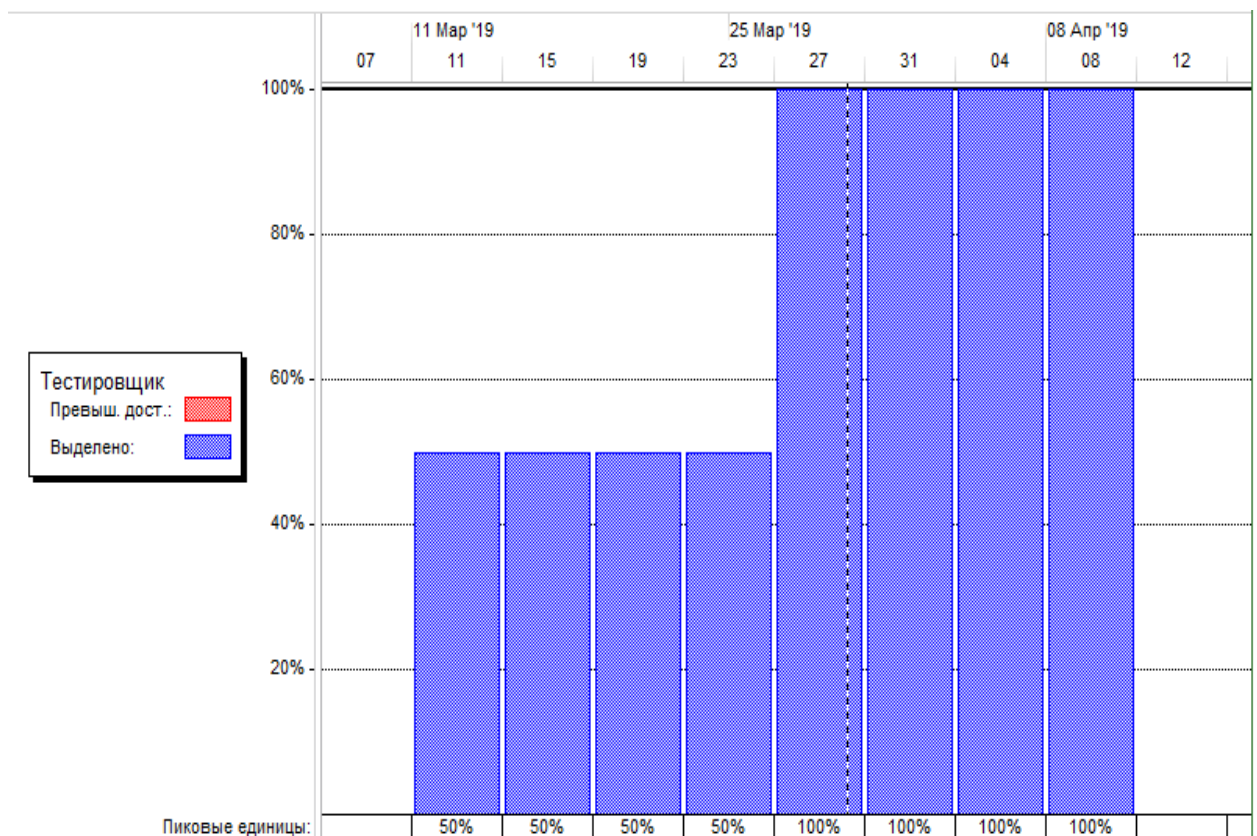


Рисунок 14 – График загрузки тестировщика

Из графиков можно увидеть, что первый программист работает на полную загрузку, второй программист в большей части времени работает на

максимальной загрузке, аналитик также в большей части времени работает на максимальной загрузке, тестировщик в среднем работает на 75% загрузки.

Важно отметить, что главный программист выполняет основную работу по анализу, разработке и кодированию программного обеспечения. Помощник главного программиста занимается проектированием и кодированием серверной части приложения.

Кроме того, на сводке по проекту, представленной на рисунке 15, указана дата начала и окончания работ, продолжительность в днях, общие затраты и график затрат по фазам проекта.

ОБЗОР ПРОЕКТА



Рисунок 15 – Сводка по проекту

На рисунке 15 видно, что проект разделен на 5 фаз:

- «Анализ предметной области»;

- «Анализ требований»;
- «Проектирование»;
- «Реализация»;
- «Тестирование».

Также на рисунке 15 представлены две вехи: «Начало проекта» и «Завершение проекта».

Длительность выполнения проекта составила 93 дня, а его бюджет составил 153 169 рублей.

3 Реализация программного обеспечения

В ходе разработки клиент серверного приложения менеджера процессов было принято решение описать подключение 2 ПК с помощью сокетов. Серверная часть приложения представляет собой вторую часть приложения на языке программирования C#.

3.1 Серверная часть приложения

В ходе разработки клиент серверного приложения менеджера процессов было принято решение описать подключение 2 ПК с помощью сокетов. Серверная часть приложения представляет собой вторую часть приложения на языке программирования C#.

Листинг 1 – подключение с помощью сокетов

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class Server
{
    public static void Main()
    {
        // задаем IP-адрес и порт для сервера
        IPAddress ipAddress = IPAddress.Any;
        int port = 12345;

        // создаем серверный сокет
        TcpListener serverSocket = new TcpListener(ipAddress,
port);

        //запускаем сервер
        serverSocket.Start();

        Console.WriteLine("Сервер запущен. Ожидание
подключения...");

        //бесконечный цикл принятия подключений
        while (true)
        {
            //подключаем клиента
            TcpClient clientSocket =
serverSocket.AcceptTcpClient();
```

```

        //получаем IP-адрес и порт клиента
        IPEndPoint clientEndpoint =
(IPPEndPoint)clientSocket.Client.RemoteEndPoint;

        Console.WriteLine($"Клиент
{clientEndpoint.Address}:{clientEndpoint.Port} подключен.");

        //получаем данные от клиента
        byte[] bytesFrom = new
byte[clientSocket.ReceiveBufferSize];

        clientSocket.Client.Receive(bytesFrom);

        string dataFromClient =
Encoding.ASCII.GetString(bytesFrom).Trim();

        Console.WriteLine($"Получено сообщение от клиента:
{dataFromClient}");

        //отправляем ответ клиенту
        string replyMessage = "Добро пожаловать на сервер!";

        byte[] bytesTo =
Encoding.ASCII.GetBytes(replyMessage);

        clientSocket.Client.Send(bytesTo);

        //закрываем соединение с клиентом
        clientSocket.Close();
    }
}

```

Этот листинг серверного соединения принимает входящие подключения от клиента, получает данные от клиента, отправляет ответ и закрывает соединение с клиентом. Чтобы обработать запросы от клиентов, необходимо дополнительно написать код, который будет принимать и обрабатывать внешние запросы. Например, можно использовать цикл `while` и методы `Receive()` и `Send()` для чтения входящих данных и отправки ответов.

4 Контроль качества программного обеспечения

Тестирование – способ обеспечения качества программного продукта.

Качество программного продукта характеризуется набором свойств, определяющих, насколько продукт «хорош» с точки зрения заинтересованных сторон. Тестирование является одним из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения и входит в набор эффективных средств современной системы обеспечения качества программного продукта.

С технической точки зрения тестирование заключается в выполнении приложения на некотором множестве исходных данных и сверке получаемых результатов с заранее известными (эталонными) с целью установить соответствие различных свойств и характеристик приложения заказанным свойствам. Как одна из основных фаз процесса разработки программного продукта, тестирование характеризуется достаточно большим вкладом в суммарную трудоемкость разработки продукта .

В данной выпускной квалификационной работе была выбрана методология функционального тестирования

4.1 Функциональное тестирование

Функциональное тестирование относится категории тестирования «черного ящика». Каждая функция программы тестируется ввода её исходных данных и анализа выходных. При этом внутренняя структура программы учитывается очень редко.

Функциональное тестирование предполагает составление плана тестирования, который строится на основе вариантов использования и включает в себя описание тестов и их результаты.

Таблица 4 – Тестирование

№	Входные данные	Выходные данные
1	IP: Статус подключения:	Сообщение: «IP не введено» Сообщение: «IP не корректно» Сообщение: «Статус подключения отсутствует» Сообщение: «Статус подключения не корректен»
2	IP: A157.193.237.151 Статус подключения:	Сообщение: «IP не корректно» Сообщение: «Статус подключения не введен» Сообщение: «Статус подключения корректен»
3	IP: 157.193.237.151 Статус подключения:	Сообщение: «Статус подключения не введен» Сообщение: «не корректен»
4	IP: Статус подключения: casda	Сообщение: «IP не введено» Сообщение: «IP не корректно» Сообщение: «Статус подключения не корректен»
5	Ip: Статус подключения: connected	Сообщение: «Ip не введено» Сообщение: «Ip не корректно»
6	Ip: 157.193.237.151a Статус подключения: casda	Сообщение: «Ip не корректно» Сообщение: «Статус подключения не корректен»
7	Ip: 157.193.237.151a Статус подключения: connected	Сообщение: «Название не корректно»
8	Ip: 157.193.237.151	Сообщение: «Статус подключения не корректен»

	Статус подключения: asd	
9	Ip: 157.193.237.151 Статус подключения: connected	Сообщение: «Устройство успешно подключено»

На основе анализа результатов тестирования можно сделать вывод, что разработанное программное обеспечение клиент серверное приложение менеджера процессов Windows работает корректно.

Заключение

При выполнении выпускной квалификационной работы было разработано клиент серверное приложение менеджера процессов Windows, состоящее из клиентского приложения и серверного приложения.

Было проведено логическое и физическое проектирование реляционной базы данных, проектирование графического интерфейса.

Клиентская часть приложения реализована при помощи языка JavaScript, серверная часть приложения реализована при помощи языка программирования C# в среде разработки «Visual Studio». В СУБД с помощью языка SQL была реализована база данных, которая включает в себя таблицы и представление. База данных была реализована в СУБД Microsoft SQL Server.

На этапе контроля качества программного обеспечения было проведено тестирование. На основе анализа результатов тестирования можно сделать вывод, что разработанное программное обеспечение работает корректно.

В ходе выполнения работы также было выполнено планирование проекта. Длительность выполнения проекта составила 93 дня, а бюджет равен 153 169 рублей. Оценка сроков и бюджета говорит о том, что данная разработка экономически выгодна.

Таким образом, в результате выполнения выпускной квалификационной работы было разработано программное обеспечение, отвечающее всем требованиям технического задания.

Список использованных источников

1. Павловская Т.А. Программирование на языке высокого уровня С# [Электронный ресурс] / Т.А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 с.
2. Прохоренок Н. HTML, JavaScript, PHP и MySQL. Джентельменский набор Web-мастера / Прохоренок Н. — СПб.: БХВ-Петербург, 2009. — 840 с.
3. Путилин А. Б., Юрагов Е. А. Компонентное моделирование и программирование на языке UML. — М.: НТ Пресс, 2005.-662 с.
4. Подбельский В. В. Язык С#. Базовый курс [Текст]: учебное пособие / В. В. Подбельский — 2-е изд. — Москва: Финансы и статистика, 2013. — 408 с.
5. Полное руководство по языку программирования С# 6.0 и платформе .NET 4.6 [Электронный ресурс]. — Режим доступа: <http://metanit.com/sharp/tutorial/> (дата обращения: 10.12.2016).
6. Полный Курс С# Base [Электронный ресурс]. — Режим доступа: <https://glamcoder.ru/video/c-sharp-base-video/> (дата обращения: 06.12.2022).
7. Практическое руководство. Создание проекта приложения Windows Forms [Электронный ресурс]. — Режим доступа: [https://msdn.microsoft.com/ru-ru/library/42wc9kk5\(v=vs.110\).aspx/](https://msdn.microsoft.com/ru-ru/library/42wc9kk5(v=vs.110).aspx/) (дата обращения: 26.12.2022).
8. Программирование на С, С# и Java [Электронный ресурс]. — Режим доступа: <http://vscode.ru/category/prog-lessons/c-sharp/> (дата обращения: 10.2.2017).
9. Программирование на языке С# [Электронный ресурс] : практикум для студентов направлений 270800.62, 280700.62 всех профилей подготовки, всех форм обучения / А. В. Алешкин ; ВятГУ, ФСА, каф. ТиСМ. - Киров : [б. и.], 2013. - 64 с.. - Библиогр.: с. 65
10. Программирование на С# [Электронный ресурс] : учебно-метод. пособие для студентов направлений 38.03.05 и 01.03.02 всех профилей подготовки,

всех форм обучения / Н. М. Козьминых; ВятГУ, ФЭМ, каф. БИ. - Киров: [б. и.]

11. Программирование | Интересные публикации | Хабрахабр [Электронный ресурс]. — Режим доступа: <http://habrahabr.ru/posts/programming/> (дата обращения: 25.02.2017).
12. Рубанцев В. Тотальный тренинг по Си-шарпу [Текст]: учебное пособие / В. Рубанцев — RVGames, 2014. — 205 с.
13. Руководство по программированию в Windows Forms [Электронный ресурс]. — Режим доступа: <https://metanit.com/sharp/windowsforms/> (дата обращения: 10.12.2016).
14. Румянцев П. В. Азбука программирования в Win32 API. — М: Горячая Линия - Телеком, 2000. — 310 с.
15. Румянцев П. В. Работа с файлами в Win32 API. — М: Горячая Линия - Телеком, 2002. — 197 с.
16. Савельев А.О. Проектирование и разработка веб-приложений на основе технологий Microsoft [Электронный ресурс] / А.О. Савельев, А.А. Алексеев. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 419 с.
17. Сафонов, В.О. Основы современных операционных систем / В.О. Сафонов. - М. : Интернет-Университет Информационных Технологий, 2011. - 584 с. - (Основы информационных технологий).
18. Смирнов, А.А. Технологии программирования : учебно-практическое пособие / А.А. Смирнов. - М. : Евразийский открытый институт, 2011. - 192 с.
19. Создание электронных учебников [Электронный ресурс]. — Режим доступа: <http://koi.tspu.ru/ebook/> (дата обращения: 15.03.2017).
20. Столбовский Д.Н. Основы разработки Web-приложений на ASP.NET [Электронный ресурс] / Д.Н. Столбовский. — Электрон. текстовые

- данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 375 с.
21. Суслова И. А. МУ по выполнению и оформлению ВКР [Текст] / И. А. Суслова, Е. В. Чубаркова — Екатеринбург: ФГАОУ ВО «Рос. гос. проф.-пед. ун-т», 2014. — 41 с.
 22. Терехов А.Н. Технология программирования [Электронный ресурс] : учебное пособие / А.Н. Терехов. — Электрон. текстовые данные. Москва, 23. Саратов: Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 152 с. —
 24. Технологии создания электронных обучающих средств [Электронный ресурс]. — Режим доступа — <http://www.ido.edu.ru/open/technology> (дата обращения: 29.01.2017). 53
 25. Уотсон К. Visual C# 2010: полный курс [Текст]: учебник / К. Уотсон, К. Нейгел, Я. Х. Педерсен, Дж. Рид, М. Скинер — под. ред. Артеменко Ю. Н. — Москва: Вильямс, 2011. — 955 с.
 26. Фаронов В. Программирование на языке C # / Фаронов В. — СПб.: Питер, 2007. — 240 с.
 27. Фаронов В. В. Создание приложений с помощью C#: Руководство программиста [Текст]: учебник / В. В. Фаронов — Москва: Эксмо, 2008. — 576 с.
 28. Финогенов, К.Г. Win32. Основы программирования / К.Г. Финогенов. - М. : Диалог-МИФИ, 2006. - 411 с.
 29. Фленов М. Е. Библия C# [Текст]: учебное пособие / М. Е. Фленов — 2-е изд. — Санкт-Петербург: БХВ-Петербург, 2012. — 560 с.
 30. Хайруллин Р.С. Программирование на C#: учебное пособие / Р.С. Хайруллин. — Казань: Изд-во Казанск. гос. архитектур.-строит. ун-та, 2017. — 153 с.
 31. Хорев П. Б. Объектно-ориентированное программирование с примерами на C# [Текст]: учебное пособие / П. Б. Хорев — 3-е изд. — Москва: Форум, Инфра-М, 2016. — 200 с.

32. Шилдт Г. Полное руководство C# 4.0 [Текст]: учебное пособие / Г. Шилдт — пер. с англ. Берштейн И. В. — Москва: Вильямс, 2012. — 1051 с.

Приложение А
ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ
(рекомендуемое)

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using Microsoft.VisualBasic;
using System.Management;
using System.Drawing.Text;

namespace diplom
{
    public partial class Form1 : Form
    {
        private List<Process>? processes = null;

        public Form1()
        {
            InitializeComponent();

            private void menuStrip1_ItemClicked(object sender, Tool-
StripItemClickedEventArgs e)
            {

```

```

private void Form1_Load(object sender, EventArgs e)
{
    processes = new List<Process>();
    Getprocesses();
    RefreshProcessesList();
}

private void listView1_SelectedIndexChanged(object
sender, EventArgs e)
{
    GetProcesses();
    RefreshProcessesList();
}
private void Getprocesses()
{
    processes.Clear();
    processes = Process.GetProcesses().ToList<Process>();
}
private void RefreshProcessesList()
{
    listView1.Items.Clear();
    double memSize = 0;
    foreach (Process p in processes)
    {
        memSize = 0;
        PerformanceCounter pc = new PerformanceCounter();
        pc.CategoryName = "Process";
        pc.CounterName = "Working set-private";
        pc.InstanceName = p.ProcessName;
        memSize = (double)pc.NextValue() / (1000 * 1000);
        string[] row = new string[] { p.Process-
Name.ToString(), Math.Round(memSize, 1).ToString() };
        listView1.Items.Add(new ListViewItem(row));
        pc.Close();
        pc.Dispose();
    }
}

```

```

        }
        Text = "Запущенно процессов:" + processes.Count.ToString();
    }
    private void RefreshProcessesList(List<Process> processes, string keyword)
    {
        listView1.Items.Clear();
        double memSize = 0;
        foreach (Process p in processes)
        {
            memSize = 0;
            PerformanceCounter pc = new PerformanceCounter();
            pc.CategoryName = "Process";
            pc.CounterName = "Working set - private";
            pc.InstanceName = p.ProcessName;
            memSize = (double)pc.NextValue() / (1000 * 1000);
            string[] row = new string[] { p.ProcessName.ToString(), Math.Round(memSize, 1).ToString() };
            listView1.Items.Add(new ListViewItem(row));
            pc.Close();
            pc.Dispose();
        }
        Text = $"Запущенно процессов'{keyword}': " + processes.Count.ToString();
    }

    private void Killprocess(Process process)
    {
        process.Kill();
        process.WaitForExit();
    }

    private void Killprocessandchildren(int pid)
    {

```

```

        if (pid == 0)
        {
            return;
        }

        ManagementObjectSearcher searcher = new ManagementObjectSearcher("select * From Win32_Process Where ParentID=" + pid);

        ManagementObjectCollection objectCollection = searcher.Get();

        foreach (ManagementObject obj in objectCollection)
        {
            Killprocessandchildren(Convert.ToInt32(obj["ProcessId"]));
        }
    {

        try
        {

            Process p = Process.GetProcessById(pid);
            p.Kill();
            p.WaitForExit();

        }

        catch (ArgumentException) { }
    }

```

```

    }

}

private static int GetParentProcessId1(Process p)
{
    int parentId = 0;
    try
    {
        ManagementObject managementObject = new Manage-
mentObject("win32_process.handle = ' " + p.Id + "'");
        managementObject.Get();
        parentId = Convert.ToInt32(managemen-
tObject["ParentProcessId"]);
    }
    catch (Exception) { }
    return parentId;
}

private void ToolStripButton1_Click(object sender, Even-
tArgs e)
{
    Getprocesses();
    RefreshProcessesList();
}

private IEnumerable<Process>? GetProcesses()
{
    return processes;
}

private void ToolStripButton2_Click(object sender, Even-
tArgs e, IEnumerable<Process>? processes)
{

```

```

        try
        {
            if (listView1.SelectedItems[0] != null)
            {
                Process processToKill = processes.Where((x)
=>      x.ProcessName      ==      listView1.SelectedItems[0].SubI-
tems[0].Text).ToList()[0];

                Killprocess(processToKill);
                Getprocesses();
                RefreshProcessesList();
            }
        }
        catch (Exception) { }
    }

    private void toolStripButton3_Click(object sender, Even-
tArgs e)
    {
        try
        {
            if (listView1.SelectedItems[0] != null)
            {
                Process processToKill = processes.Where((x)
=>      x.ProcessName      ==      listView1.SelectedItems[0].SubI-
tems[0].Text).ToList()[0];

                Killprocessandchildren(GetParentProces-
sId(processToKill));
                Getprocesses();
                RefreshProcessesList();
            }
        }
        catch (Exception) { }
    }

```

```

private int GetParentProcessId(Process processToKill)
{
    throw new NotImplementedException();
}

private void завершитьДеревоПроцес-
совToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        if (listView1.SelectedItems[0] != null)
        {
            Process processToKill = processes.Where((x)
=> x.ProcessName == listView1.SelectedItems[0].SubI-
tems[0].Text).ToList()[0];

            Killprocessandchildren(GetParentProces-
sId(processToKill));

            Getprocesses();
            RefreshProcessesList();

        }
    }
    catch (Exception) { }
}

private void toolStripButton4_Click(object sender, Even-
tArgs e)
{
}

}

// Licensed to the .NET Foundation under one or more agreements.

```

```
// The .NET Foundation licenses this file to you under the MIT
license.
```

```
using System.ComponentModel;
using System.Globalization;
using System.Runtime.CompilerServices;
using System.Threading;
```

```
namespace System.Diagnostics
```

```
{
    /// <summary>
    ///     Performance Counter component.
    ///     This class provides support for NT Performance coun-
    ters.
    ///     It handles both the existing counters (accessible by
    Perf Registry Interface)
    ///     and user defined (extensible) counters.
    ///     This class is a part of a larger framework, that
    includes the perf dll object and
    ///     perf service.
    /// </summary>
    public sealed class PerformanceCounter : Component, ISup-
    portInitialize
    {
        private string _machineName;
        private string _categoryName;
        private string _counterName;
        private string _instanceName;
        private PerformanceCounterInstanceLifetime _instanceLife-
        time = PerformanceCounterInstanceLifetime.Global;

        private bool _isReadOnly;
        private bool _initialized;
        private string _helpMsg;
        private int _counterType = -1;
```

```

// Cached old sample
private CounterSample _oldSample = CounterSample.Empty;

// Cached IP Shared Performance counter
private SharedPerformanceCounter _sharedCounter;

[ObsoleteAttribute("PerformanceCounter.Default-
FileMappingSize has been deprecated and is not used. Use ma-
chine.config or an application configuration file to set the size
of the PerformanceCounter file mapping instead.")]
public static int DefaultFileMappingSize = 524288;

private object _instanceLockObject;
private object InstanceLockObject
{
    get
    {
        if (_instanceLockObject == null)
        {
            object o = new object();
            Interlocked.CompareExchange(ref _in-
stanceLockObject, o, null);
        }
        return _instanceLockObject;
    }
}

/// <summary>
///     The default constructor. Creates the perf counter
object
/// </summary>
public PerformanceCounter()
{
    _machineName = ".";

```

```

        _categoryName = string.Empty;
        _counterName = string.Empty;
        _instanceName = string.Empty;
        _isReadOnly = true;
        GC.SuppressFinalize(this);
    }

    /// <summary>
    ///     Creates the Performance Counter Object
    /// </summary>
    public PerformanceCounter(string categoryName, string
counterName, string instanceName, string machineName)
    {
        MachineName = machineName;
        CategoryName = categoryName;
        CounterName = counterName;
        InstanceName = instanceName;
        _isReadOnly = true;
        Initialize();
        GC.SuppressFinalize(this);
    }

    internal PerformanceCounter(string categoryName, string
counterName, string instanceName, string machineName, bool
skipInit)
    {
        MachineName = machineName;
        CategoryName = categoryName;
        CounterName = counterName;
        InstanceName = instanceName;
        _isReadOnly = true;
        _initialized = true;
        GC.SuppressFinalize(this);
    }

```

```

    /// <summary>
    ///     Creates the Performance Counter Object on local
machine.
    /// </summary>
    public PerformanceCounter(string categoryName, string
counterName, string instanceName) :
        this(categoryName, counterName, instanceName, true)
    {
    }

    /// <summary>
    ///     Creates the Performance Counter Object on local
machine.
    /// </summary>
    public PerformanceCounter(string categoryName, string
counterName, string instanceName, bool readOnly)
    {
        if (!readOnly)
        {
            VerifyWriteableCounterAllowed();
        }
        MachineName = ".";
        CategoryName = categoryName;
        CounterName = counterName;
        InstanceName = instanceName;
        _isReadOnly = readOnly;
        Initialize();
        GC.SuppressFinalize(this);
    }

    /// <summary>
    ///     Creates the Performance Counter Object, assumes
that it's a single instance
    /// </summary>

```

```

        public PerformanceCounter(string categoryName, string
counterName) :
        {
            this(categoryName, counterName, true)
        }

        /// <summary>
        ///     Creates the Performance Counter Object, assumes
that it's a single instance
        /// </summary>
        public PerformanceCounter(string categoryName, string
counterName, bool readOnly) :
        {
            this(categoryName, counterName, "", readOnly)
        }

        /// <summary>
        ///     Returns the performance category name for this
performance counter
        /// </summary>
        public string CategoryName
        {
            get
            {
                return _categoryName;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException(
nameof(value));

                if (_categoryName == null || !string.Equals(_cat-
egoryName, value, StringComparison.OrdinalIgnoreCase))
                {

```

```

        _categoryName = value;
        Close();
    }
}

/// <summary>
///     Returns the description message for this perfor-
mance counter
/// </summary>
public string CounterHelp
{
    get
    {
        string currentCategoryName = _categoryName;
        string currentMachineName = _machineName;

        Initialize();

        if (_helpMsg == null)
            _helpMsg = PerformanceCounterLib.GetCounter-
Help(currentMachineName, currentCategoryName, _counterName);

        return _helpMsg;
    }
}

/// <summary>
///     Sets/returns the performance counter name for this
performance counter
/// </summary>
public string CounterName
{
    get
    {

```

```

        return _counterName;
    }
    set
    {
        if (value == null)
            throw new ArgumentNullException(nameof(value));

        if (_counterName == null || !string.Equals(_counterName, value, StringComparison.OrdinalIgnoreCase))
        {
            _counterName = value;
            Close();
        }
    }
}

/// <summary>
///     Sets/Returns the counter type for this performance
counter
/// </summary>
public PerformanceCounterType CounterType
{
    get
    {
        if (_counterType == -1)
        {
            string currentCategoryName = _categoryName;
            string currentMachineName = _machineName;

            // This is the same thing that NextSample
            // does, except that it doesn't try to get the actual counter
            // value. If we wanted the counter value, we
            // would need to have an instance name.
            Initialize();

```

```

        using (CategorySample categorySample = Per-
formanceCounterLib.GetCategorySample(currentMachineName,    cur-
rentCategoryName))
        {
            CounterDefinitionSample counterSample =
categorySample.GetCounterDefinitionSample(_counterName);
            _counterType = counterSample._counter-
Type;
        }
    }

    return (PerformanceCounterType)_counterType;
}
}

public PerformanceCounterInstanceLifetime InstanceLife-
time
{
    get { return _instanceLifetime; }
    set
    {
        if (value > PerformanceCounterInstanceLife-
time.Process || value < PerformanceCounterInstanceLife-
time.Global)
            throw new ArgumentOutOfRangeException(nameof(value));

        if (!_initialized)
            throw new InvalidOperationException(SR.CantSetLifetimeAfterInitialized);

        _instanceLifetime = value;
    }
}

```

```

    /// <summary>
    ///     Sets/returns an instance name for this performance
counter
    /// </summary>
    public string InstanceName
    {
        get
        {
            return _instanceName;
        }
        set
        {
            if (value == null && _instanceName == null)
                return;

            if ((value == null && _instanceName != null) ||
                (value != null && _instanceName == null) ||
                !string.Equals(_instanceName, value,
StringComparison.OrdinalIgnoreCase))
            {
                _instanceName = value;
                Close();
            }
        }
    }

    /// <summary>
    ///     Returns true if counter is read only (system
counter, foreign extensible counter or remote counter)
    /// </summary>
    public bool ReadOnly
    {
        get
        {
            return _isReadOnly;

```

```

    }

    set
    {
        if (value != _isReadOnly)
        {
            if (value == false)
            {
                VerifyWriteableCounterAllowed();
            }
            _isReadOnly = value;
            Close();
        }
    }
}

/// <summary>
///     Set/returns the machine name for this performance
counter
/// </summary>
public string MachineName
{
    get
    {
        return _machineName;
    }
    set
    {
        if (!SyntaxCheck.CheckMachineName(value))
            throw new ArgumentException(SR.Format(SR.InvalidProperty, nameof(MachineName), value), nameof(value));

        if (_machineName != value)
        {
            _machineName = value;

```

```

        Close();
    }
}

/// <summary>
///     Directly accesses the raw value of this counter.
If counter type is of a 32-bit size, it will truncate
///     the value given to 32 bits. This can be signifi-
cantly more performant for scenarios where
///     the raw value is sufficient. Note that this only
works for custom counters created using
///     this component, non-custom counters will throw an
exception if this property is accessed.
/// </summary>
public long RawValue
{
    get
    {
        if (ReadOnly)
        {
            //No need to initialize or Demand, since
NextSample already does.

            return NextSample().RawValue;
        }
        else
        {
            Initialize();

            return _sharedCounter.Value;
        }
    }
    set
    {
        if (ReadOnly)

```

```

        ThrowReadOnly();

        Initialize();

        _sharedCounter.Value = value;
    }
}

/// <summary>
/// </summary>
public void BeginInit()
{
    Close();
}

/// <summary>
///     Frees all the resources allocated by this counter
/// </summary>
public void Close()
{
    _helpMsg = null;
    _oldSample = CounterSample.Empty;
    _sharedCounter = null;
    _initialized = false;
    _counterType = -1;
}

/// <summary>
///     Frees all the resources allocated for all perfor-
mance
    ///     counters, frees File Mapping used by extensible
counters,
    ///     unloads dll's used to read counters.
/// </summary>
public static void CloseSharedResources()

```

```

{
    PerformanceCounterLib.CloseAllLibraries();
}

/// <internalonly/>
/// <summary>
/// </summary>
protected override void Dispose(bool disposing)
{
    // safe to call while finalizing or disposing
    if (disposing)
    {
        //Dispose managed and unmanaged resources
        Close();
    }

    base.Dispose(disposing);
}

/// <summary>
///     Decrements counter by one using an efficient atomic
operation.
/// </summary>
public long Decrement()
{
    if (ReadOnly)
        ThrowReadOnly();

    Initialize();

    return _sharedCounter.Decrement();
}

/// <summary>
/// </summary>

```

```

public void EndInit()
{
    Initialize();
}

/// <summary>
///     Increments the value of this counter.  If counter
type is of a 32-bit size, it'll truncate
///     the value given to 32 bits. This method uses a
mutex to guarantee correctness of
///     the operation in case of multiple writers. This
method should be used with caution because of the negative
///     impact on performance due to creation of the mutex.
/// </summary>
public long IncrementBy(long value)
{
    if (_isReadOnly)
        ThrowReadOnly();

    Initialize();

    return _sharedCounter.IncrementBy(value);
}

/// <summary>
///     Increments counter by one using an efficient atomic
operation.
/// </summary>
public long Increment()
{
    if (_isReadOnly)
        ThrowReadOnly();

    Initialize();
}

```

```

        return _sharedCounter.Increment();
    }

    private void ThrowReadOnly()
    {
        throw new InvalidOperationException(SR.ReadOnly-
Counter);
    }

    private static void VerifyWriteableCounterAllowed()
    {
        if (EnvironmentHelpers.IsAppContainerProcess)
        {
            throw new NotSupportedException(SR.PCNotSup-
portedUnderAppContainer);
        }
    }

    private void Initialize()
    {
        // Keep this method small so the JIT will inline it.
        if (!_initialized && !DesignMode)
        {
            InitializeImpl();
        }
    }

    /// <summary>
    ///     Intializes required resources
    /// </summary>
    private void InitializeImpl()
    {
        bool tookLock = false;
        try
        {

```

```

        Monitor.Enter(InstanceLockObject, ref tookLock);

        if (!_initialized)
        {
            string currentCategoryName = _categoryName;
            string currentMachineName = _machineName;

            if (currentCategoryName.Length == 0)
                throw new InvalidOperationException(
                    SR.CategoryNameMissing);
            if (_counterName.Length == 0)
                throw new InvalidOperationException(
                    SR.CounterNameMissing);

            if (ReadOnly)
            {
                if (!PerformanceCounterLib.CounterExists(
                    currentMachineName, currentCategoryName, _counterName))
                    throw new InvalidOperationException(
                        SR.Format(SR.CounterExists, currentCategoryName, _counterName));

                PerformanceCounterCategoryType categoryType =
                    PerformanceCounterLib.GetCategoryType(currentMachineName,
                        currentCategoryName);
                if (categoryType == PerformanceCounterCategoryType.MultiInstance)
                {
                    if (string.IsNullOrEmpty(_instanceName))
                        throw new InvalidOperationException(
                            SR.Format(SR.MultiInstanceOnly, currentCategoryName));
                }
                else if (categoryType == PerformanceCounterCategoryType.SingleInstance)

```

```

        {
            if (!string.IsNullOrEmpty(_instanceName))

                throw new InvalidOperationException(
                    SR.Format(SR.SingleInstanceOnly, categoryName));
        }

        if (_instanceLifetime != PerformanceCounterInstanceLifetime.Global)

            throw new InvalidOperationException(
                SR.InstanceLifetimeProcessonReadOnly);

        _initialized = true;
    }
    else
    {
        if (currentMachineName != "." &&
            !string.Equals(currentMachineName, PerformanceCounterLib.ComputerName,
                StringComparison.OrdinalIgnoreCase))

            throw new InvalidOperationException(
                SR.RemoteWriting);

        if (!PerformanceCounterLib.IsCustomCategory(currentMachineName, categoryName))

            throw new InvalidOperationException(
                SR.NotCustomCounter);

        // check category type
        PerformanceCounterCategoryType categoryType =
            PerformanceCounterLib.GetCategoryType(currentMachineName,
                categoryName);

        if (categoryType == PerformanceCounterCategoryType.MultiInstance)
        {

```

```

        if (string.IsNullOrEmpty(_instanceName))

            throw new InvalidOperationException(
                SR.Format(SR.MultiInstanceOnly, currentCategoryName));
        }
        else if (categoryType == PerformanceCounterCategoryType.SingleInstance)
        {
            if (!string.IsNullOrEmpty(_instanceName))

                throw new InvalidOperationException(
                    SR.Format(SR.SingleInstanceOnly, currentCategoryName));
        }

        if (string.IsNullOrEmpty(_instanceName)
            && InstanceLifetime == PerformanceCounterInstanceLifetime.Process)

            throw new InvalidOperationException(
                SR.InstanceLifetimeProcessforSingleInstance);

        _sharedCounter = new SharedPerformanceCounter(
            currentCategoryName.ToLowerInvariant(),
            _counterName.ToLowerInvariant(),
            _instanceName.ToLowerInvariant(),
            _instanceLifetime);

        _initialized = true;
    }
}

finally
{
    if (tookLock)
        Monitor.Exit(InstanceLockObject);
}
}

```

```

        // Will cause an update, raw value
        /// <summary>
        ///     Obtains a counter sample and returns the raw value
for it.
        /// </summary>
public CounterSample NextSample()
{
    string currentCategoryName = _categoryName;
    string currentMachineName = _machineName;

    Initialize();

    using (CategorySample categorySample = Perfor-
manceCounterLib.GetCategorySample(currentMachineName, currentCat-
egoryName))
    {
        CounterDefinitionSample counterSample = catego-
rySample.GetCounterDefinitionSample(_counterName);
        _counterType = counterSample._counterType;
        if (!categorySample._isMultiInstance)
        {
            if (_instanceName != null && _in-
stanceName.Length != 0)
                throw new InvalidOperationException(SR.Format(SR.InstanceNameProhibited, _instanceName));

            return counterSample.GetSingleValue();
        }
        else
        {
            if (_instanceName == null || _in-
stanceName.Length == 0)

```

```

        throw new InvalidOperationException(
            SR.InstanceNameRequired);

        return counterSample.GetInstanceValue(_instanceName);
    }
}

/// <summary>
///     Obtains a counter sample and returns the calculated
value for it.
///     NOTE: For counters whose calculated value depend
upon 2 counter reads,
///         the very first read will return 0.0.
/// </summary>
public float NextValue()
{
    //No need to initialize or Demand, since NextSample
already does.
    CounterSample newSample = NextSample();
    float retVal = 0.0f;

    retVal = CounterSample.Calculate(_oldSample, newSample);

    _oldSample = newSample;

    return retVal;
}

/// <summary>
///     Removes this counter instance from the shared
memory
/// </summary>
public void RemoveInstance()

```

```
{
    if (_isReadOnly)
        throw new InvalidOperationException(SR.ReadOnlyRemoveInstance);

    Initialize();
    _sharedCounter.RemoveInstance(_instanceName.ToLowerInvariant(), _instanceLifetime);
}
}
```