Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Архитектуры вычислительных систем»

|  |  |
| --- | --- |
|  | *К защите допустить:* |
|  | И. о. заведующего кафедрой информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. И. Сиротко |
|  |  |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**ВИДЖЕТ МОНИТОРИНГА СОСТОЯНИЯ АППАРАТНОГО ОБЕСПЕЧЕНИЯ КОМПЬЮТЕРА И ПРОГНОЗ ВРЕМЕНИ ЕГО РАБОТЫ**

БГУИР КП 1-40 04 01 023 ПЗ

Студент             А. К. Хрищанович

Руководитель             А. Н. Марков

Нормоконтролер             А. А. Калиновская

Минск 2023

СОДЕРЖАНИЕ

[Введение 3](#_Toc152116482)

[1 Архитектура вычислительной системы 4](#_Toc152116483)

[1.1 Структура и архитектура вычислительной системы 4](#_Toc152116484)

[1.2 История, версии и достоинства 6](#_Toc152116485)

[1.3 Обоснование выбора вычислительной системы 11](#_Toc152116486)

[1.4 Анализ выбранной вычислительной системы 15](#_Toc152116487)

[2 Платформа программного обеспечения 17](#_Toc152116488)

[2.1 Структура и архитектура платформы 17](#_Toc152116489)

[2.2 История, версии и достоинства 19](#_Toc152116490)

[2.3 Анализ программного обеспечения для написания программы и       обоснование выбора платформы 22](#_Toc152116491)

[3 Теоретическое обоснование разработки программного продукта 23](#_Toc152116492)

[3.1 Обоснование необходимости разработки 23](#_Toc152116493)

[3.2 Технологии программирования, используемые для решения                           поставленных задач 23](#_Toc152116494)

[3.3 Связь архитектуры вычислительной системы с разрабатываемым                  программным обеспечением 26](#_Toc152116495)

[4 Функциональные возможности программы 27](#_Toc152116496)

[4.1 Обоснование и описание функций программного обеспечения 27](#_Toc152116497)

[5 Архитектура разрабатываемое программы 35](#_Toc152116498)

[5.1 Общая структура программы 35](#_Toc152116499)

[5.2 Описание функциональной схемы программы 35](#_Toc152116500)

[5.3 Описание блок схемы алгоритма программы 36](#_Toc152116501)

[Заключение 37](#_Toc152116502)

[Список литератул источников 38](#_Toc152116503)

[Приложение А (обязательное) Листинг исходного кода 40](#_Toc152116504)

[Прилоежние Б (обязательное) Функциональная схема алгоритма,                                                 реализующего программное средство 62](#_Toc152116505)

[Приложение В (обязательное) Блок схема алгоритма,                                                                     реализующего программное средства 63](#_Toc152116506)

[Приложение Г (обязательное) Графический интерфейс пользователя 64](#_Toc152116507)

[Приложение Д (обязательное) Ведомость документов 65](#_Toc152116508)

# **ВВЕДЕНИЕ**

В современном мире компьютеры играют неотъемлемую роль в повседневной жизни. Они используются в различных сферах, начиная от домашних рабочих станций и заканчивая критически важными системами, такими как медицинские системы и серверы данных. Поэтому актуальность темы «Виджет мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы» трудно переоценить, что означает, что данная тема имеет высокую актуальность, так как связана с повседневной жизнью многих людей и организаций, которые зависят от надежной работы своих технических средств.

Цель данного курсового проекта заключается в разработке и реализации виджета мониторинга состояния аппаратного обеспечения компьютера, который способен предоставить пользователю информацию о текущем состоянии компьютера, а также прогнозировать оставшееся время работы устройства. Для достижения данной цели были поставлены следующие задачи:

1 Анализ существующих решений: провести обзор существующих программных средств и решений для мониторинга аппаратного обеспечения компьютера и определить их преимущества и недостатки.

2 Разработка виджета мониторинга состояния аппаратного обеспечения компьютера: разработать программный виджет, который будет собирать информацию о состоянии различных аппаратных компонентов компьютера, таких как центральный процессор, графический процессор, оперативная память, жесткий диск и аккумулятор.

3 Прогнозирование работы компьютера: разработать алгоритм прогнозирования работы компьютера на основе данных, полученных от аккумулятора устройства, если он присутствует в устройстве. Это позволит получать информацию о диапазоне времени, которое устройство может работать без подзарядки.

4 Интерфейс пользователя: создать удобный и информативный интерфейс для виджета, который будет понятен и доступен для широкого круга пользователей.

Таким образом курсовой проект направлен на повышене уровня информированности пользователя о состоянии своего компьютера, что позволит ему своевременно замечать возможные неисправности в устройстве и предотвращать проблемы и сбои в работе.

# **1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ**

## **1.1 Структура и архитектура вычислительной системы**

Вычислительная система (ВС) – результат интеграции аппаратных и программных средства, функционирующих в единой системе и предназначенных для решения задач определенного класса. Структура вычислительной системы состоит из пяти уровней. [1]

**1.1.1** Аппаратный уровень

Аппаратный уровень является первым уровнем вычислительной системы. Этот уровень определяется набором аппаратных компонентов и их характеристиками, используемых вышестоящими уровнями иерархии и определяющими воздействие на них. К физическим ресурсам этого уровня относятся: процессор, оперативная память, внешние устройства.

**1.1.2** Уровень управления физическими ресурсами

Уровень управления физическими ресурсами является вторым уровнем вычислительной системы и первым уровнем системного программного обеспечения, что подчеркивает то, что это первый слой программного обеспечения, взаимодействующий с аппаратным уровнем. Его основное назначение заключается в систематизации и стандартизации правил программного использования физических ресурсов. На этом уровне обеспечивается создание программ управления физическими ресурсами. Для обеспечения управления физическими ресурсами используются программы, которые называются драйверами физического ресурса (устройства).

Драйвер физического устройства представляет собой программное обеспечение, разработанное на основе команд управления конкретным физическим устройством и предназначенное для организации работы с данным устройством.

**1.1.3** Уровень управления логическими/виртуальными ресурсами

Логическое или виртуальное устройство представляет собой ресурс, характеристики которого реализованы программным образом. Драйвер логического или виртуального ресурса – это программное средство, обеспечивающее создание, управление и использование соответствующего ресурса. Эти компоненты ориентированы на конечного пользователя, и их команды не зависят от конкретных физических устройств, что облегчает взаимодействие с ресурсами для программных компонентов.

На этом уровне могут создаваться новые логические ресурсы, и для организации драйвера логического устройства могут использоваться как драйверы физических, так и драйверы логических или виртуальных устройств. Система поддерживает иерархию драйверов, что позволяет эффективно управлять различными уровнями ресурсов.

Ресурсы вычислительной системы включают в себя как физические, так и виртуальные ресурсы. Они характеризуются конечностью, что создает конкуренцию между программными потребителями за доступ к этим ресурсам. Операционная система, как комплекс программ, управляет этими ресурсами, обеспечивая их эффективное использование.

Существует разветвленная иерархия виртуальных и физических устройств, и драйверы могут быть классифицированы в три группы:

1 Драйверы физических устройств: обеспечивают взаимодействие с конкретными физическими компонентами вычислительной системы.

2 Драйверы виртуальных устройств, обобщающих характеристики физических устройств: позволяют создавать абстракции над физическими устройствами, обобщая их характеристики и предоставляя более удобный интерфейс программам.

3 Драйверы виртуальных устройств без аппаратной реализации: эти драйверы создают "полностью" виртуальные устройства, не имеющие физического эквивалента. Примером может служить драйвер файловой системы, который управляет доступом к файлам без конкретной аппаратной реализации.

Такая иерархия и классификация драйверов позволяют эффективно организовывать управление ресурсами на уровне операционной системы и предоставлять пользователю удобные интерфейсы для работы с вычислительной системой.

**1.1.4** Уровень систем программирования

Система программирования представляет собой комплекс программ, предназначенных для обеспечения эффективного управления жизненным циклом программного продукта в вычислительной системе. Жизненный цикл программы включает в себя пять ключевых этапов, к которым относятся проектирование программного продукта, кодирование или реализация, тестирование и отладка, а также внедрение программного продукта и его сопровождение.

Проектирование программного продукта представляет собой первый этап жизненного цикла программы. На этом этапе определяется структура, архитектура и основные характеристики будущего программного продукта. Кроме этого, на этапе проектирования определяются требования к программному продукту, его архитектура, проектирование интерфейса, выбор технологии и разработка алгоритмов для реализации.

Кодирование или реализация представляет собой продолжение этапа проектирования. На данном этапе программный продукт переходит из концепции в конкретный исполняемый код.

Тестирование представляет собой процесс проверки программы с использованием заранее определенных тестовых данных. Тесты, как заранее заданные входные данные, направлены на обеспечение корректного выполнения программы в различных сценариях использования. Тестирование также включает в себя оценку производительности и надежности программы.

Отладка – это процесс выявления, локализации и исправления ошибок, обнаруженных в процессе тестирования. Отладка направлена на обеспечение стабильной и надежной работы программы.

Этап внедрения включает в себя установку программного комплекса на объектную вычислительную систему и его первичную настройку. Внедрение также включает в себя проверку работоспособности программы в реальной среде.

В этап сопровождения входит исправление ошибок и недочетов, выявленных после внедрения программного комплекса. Сюда также включается выпуск патчей для улучшения функциональности или исправления обнаруженных проблем.

**1.1.5** Уровень прикладных систем

На уровне прикладных систем реализуются программные системы, ориентированные на решение или автоматизацию решения задач из конкретной предметной области. Этот уровень представляет собой высокоуровневую часть программного обеспечения, которая нацелена на конечного пользователя и предоставляет интерфейс для взаимодействия с компьютерной системой.

Уровень прикладных систем представляет собой ключевой элемент компьютерной архитектуры, где программы разрабатываются и оптимизируются для конечных пользователей, сосредотачиваясь на решении конкретных задач в различных областях человеческой деятельности.

**1.1.6** Характеристики вычислительная система для разработки проекта

Вычислительная система, на которой проводилась разработка данного курсового проекта, базируется на операционной системе *Windows 10* и аппаратном обеспечении ноутбука *Asus* из серии *TUF Gaming*. Эта система включает в себя следующие компоненты:

1 Процессор *Intel Core i5*: процессор (центральный процессор) – основной вычислительный движок компьютера. Данный процессор обеспечивает высокую производительность и является важной частью архитектуры системы.

2 Оперативная память (*RAM*): оперативная память служит для временного хранения данных, используемых программами во время их выполнения. Это важный аспект архитектуры, так как от объемы и скорости оперативной памяти зависит производительность системы.

3 Жестки диск: жесткий диск или *SSD* предназначен для хранения операционной системы, приложений и данных.

4 Видеокарты *NVIDIA GeForce GTX 1650 GDDR6*, в основе которой лежит графический процессор *TU117*,специализируется на обработке графики и выполнении сложных вычислений, связанных с отображением изображений на экране.

**1.1.7**Устройство центрального процессора

Процессор – мозг компьютера. Его задача – выполнять программы, находящиеся в основной памяти. Для этого он вызывает команды из памяти, определяет их тип, а затем выполняет одну за другой. Компоненты соединены шиной, представляющей собой набор параллельно связанных проводов для передачи адресов, данных и управляющих сигналов.

Процессор состоит из нескольких частей. Блок управления отвечает за вызов команд из памяти и определение их типа. Арифметика-логическое устройство выполняет арифметические и логические операции.

Тракт данных состоит из регистров, арифметико-логического устройства и нескольких соединительных шин. Содержимое регистров поступает во входные регистры арифметико-логического устройства. В них находятся входные данные, пока арифметико-логическое устройство производит вычисления. Тракт данных – важная составляющая часть всех компьютеров. [2]

В дополнении к процессору требуются по крайней мере некоторое оперативное запоминающее устройство, какой-то способ, позволяющий пользователю записать информацию в компьютер (устройство ввода), а также извлечь ее из него (устройство вывода).

Кроме того, необходимо еще несколько микросхем для объединения всех компонентов. Сам процессор мы можем представить в качестве «черного ящика», внутреннюю работу которого нам не обязательно досконально изучать, чтобы разобраться в его функциях. [3]

Центральный процессор выполняет каждую команду за несколько шагов. Для начала он вызывает следующую команду из памяти и переносит ее в регистр команд. Вторым шагом процессор меняет положение счетчика команд, которые после этого указывает на следующую команду. Затем процессор определяет тип вызванной команды и, если команда использует слово из памяти, определяет, где находится это слово. После процессор переносит слово в регистр центрального процессора, если это необходимо, и только после всех этих шагов процессор выполняет команду. После окончания выполнения команды, он переходит к первому шагу, чтобы начать выполнение следующей команды. Такая последовательность шагов является основой работы всех компьютеров.

**1.1.8** Основная (оперативная) память

Память – это тот компонент компьютера, в котором хранятся программы и данные. Также часто встречается термин «запоминающее устройство». Без памяти, откуда процессоры считывают и куда записывают информацию, не было бы современных цифровых компьютеров.

В общем случае основной памятью компьютера называют оперативную память (*Random Access Memory, RAM*). Оперативная память играет ключевую роль в функционировании компьютера и используется для временного хранения данных и кода, которые активно используются программами во время их выполнения.

Основной единицей измерения хранения данных в памяти является двоичный разряд, который называется битом. Бит может содержать 0 или 1. Эта самая маленькая единица памяти.

Применение двоичной системы счисления в компьютерах объясняется ее «эффективностью». При этом имеется в виду, что хранение цифровой информации может быть основано на отличиях между разными величинами какой-либо физической характеристики, например напряжения или тока. Чем больше величин нужно различать, тем меньше отличий между смежными величинами и тем менее надежна память. В двоичной системе счисления требуется различать всего две величины, следовательно, это – самый надежный метод кодирования цифровой информации.

Память состоит из ячеек, каждая из которых может хранить некоторую порцию информации. Ячейка – минимальная адресуемая единица памяти. В последние годы практически все производители выпускают компьютеры с 8-разрядными ячейками, которые называются байтами. Байты группируются в слова. Каждая ячейка имеет номер, который называется адресом. По адресу программы могут ссылаться на определенную ячейку.

В компьютерах, в которых используется двоичная система счисления (включая восьмеричной и шестнадцатеричное представление двоичных чисел), адреса памяти также выражаются в двоичных числах.

**1.1.9** Вспомогательная память

Каков бы ни был объем основной памяти, ее все равно будет мало. С развитием технологий людям приходят в голову такие вещи, которые раньше считались совершенно фантастическими. Огромный объем информации в настоящее время невозможно разместить в основной памяти, поэтому на данный момент все пользуются вспомогательной памятью.

Устройства на базе энергонезависимой флэш-памяти, часто называют твердотельными накопителями или *SSD*-дисками (*Solid State Disk*), которые являются высокоскоростной альтернативой традиционным технологиям магнитных дисков.

Так как *SSD*-диски по сути являются памятью, они обладают более высокой производительностью по сравнению с вращающимися магнитными дисками при нулевом времени поиска. Поскольку устройство не имеет подвижных частей, оно особенно хорошо подходит для ноутбуков (колебания в перемещении не влияют на его способность обращаться к данным).

**1.1.10** Видеокарта и графический процессор

Видеокарта основана на конкретной архитектуре, которая определяет организацию и характеристики вычислительных блоков. Видеокарта содержит основана на графическом процессоре *TU117*. В связи с этим далее подробно рассмотрено устройство графического процессора.

Конкретно к характеристикам видеокарты можно отнести объем видеопамяти (*VRAM*), используемый для хранения текстур, кадров и других графических данных.

Графические процессоры (*GPU*) представляют собой сложные вычислительные устройства, которые спроектированы для выполнения множества параллельных вычислений. Они играют важную роль в современных вычислениях, таких как компьютерная графика и научные исследования. Важно понимать, что графические процессоры имеют собственную архитектуру и инструкции, которые несколько отличаются от центральных процессоров.

Вот более подробное описание ключевых элементовграфического процессора:

1 Блоки *FP32* и *FP64*: это вычислительные блоки, предназначенные для операций с плавающей точкой различной точности. *FP32* (одинарная точность) обычно используется для большинства вычислений, в то время как *FP64* (двойная точность) используется для более точных вычислений, но с более низкой производительностью.

2 Блоки *INT32*: эти блоки предназначены для выполнения целочисленных операций. Они могут использоваться для разнообразных вычислений, таких как обработка изображений и кодирование данных.

3 Регистры: регистры являются небольшими хранилищами данных, которые используются для временного хранения результатов вычислений. Каждое вычислительное ядро имеет свои собственные регистры.

4 Встроенная память (кэши): графические процессоры обычно имеют несколько уровней кэша для ускорения доступа к данным. Эти уровни кэша могут быть использованы для хранения часто используемых данных, уменьшая время доступа.

5 Планировщик команд и диспетчер команд: эти компоненты отвечают за управление выполнением инструкций на графическом процессоре. Планировщик команд определяет, какие задачи будут выполнены, их порядок и распределение по вычислительным блокам.

6 *L0$*, *L1$*, и *L2$*: эти кэши представляют собой различные уровни кэширования памяти на графическому процессоре. *L0$* и *L1$* используются для временного хранения данных внутри вычислительных блоков, в то время как *L2$* представляет собой кэш на уровне всей графической карты.

7 Внешняя память (память внешнего устройства): это память, доступная для графического процессора, и она обычно используется для хранения данных и текстур.

8 *TMU* (*Texture Mapping Units*) и *ROP* (*Raster Operations Pipeline*): эти блоки отвечают за текстурирование и растеризацию графики. Они играют важную роль в обработке и отображении изображений.

Графический процессор можно рассматривать как устройство асинхронного ожидания, где задачи выполняются параллельно, и управление заданиями и данными играет ключевую роль. Ядра *CUDA* в *GPU Nvidia* и ядра в *GPU AMD* представляют собой простые вычислительные блоки, способные выполнить параллельные вычисления с плавающей точкой. Важно понимать, что они не имеют некоторых возможностей, характерных для *CPU*, но обеспечивают высокую производительность при выполнении определенных действий. [7]

## **1.2 История, версии и достоинства**

**1.2.1** Архитектура x64

Архитектура *x86-64*, также известная как *Intel 64*, представляет собой расширение архитектуры x86, которая была разработана компаниями *AMD* и *Intel*. Ее история связана с потребностью в поддержке больших объемов памяти и более высокой производительности. Она была представлена в 2000 году.

С тех пор архитектура *x86-64* пережила несколько версий и улучшений. Наиболее значимыми версиями были:

1 *AMD 64*: исходная версия архитектуры, представленная *AMD* в 2000 году.

2 *Intel 64:* *Intel* также внедрила свою версию архитектуры *x86-64*, известную как *Intel* 64, также в начале 2000-х годов.

Архитектура *x86-64* имеет ряд значительных преимуществ:

1 Поддержка 64-битных приложений: позволяет запускать и обрабатывать 64-битные приложения, что повышает производительность и позволяет эффективно управлять большими объемами данных.

2 Поддержка больших объемов памяти: способность адресации более 4 ГБ оперативной памяти, что является критически важным для современных вычислительных задач.

3 Совместимость: обратная совместимость с 32-битными приложениями и операционными системами, что обеспечивает плавный переход к 64-разрядной архитектуре.

Архитектура *x86-64* внесла множество инноваций и технологических решений, включая:

1 Наборы инструкций: расширенные наборы инструкций, которые обеспечивают более высокую производительность и возможности оптимизации.

2 Виртуализация: встроенная поддержка технологий виртуализации для лучшей изоляции и управления виртуальными машинами.

3 Защита данных: механизмы защиты данных и аппаратная защита от вредоносных атак.

Архитектура *x86-64*, в том числе *Intel* 64, широко применяется в различных областях:

1 Серверы: в крупных серверных фермах для обработки данных и виртуализации.

2 Настольные и ноутбуки: в персональных компьютерах для выполнения разнообразных задач, включая игры, мультимедиа и офисные приложения.

3 Разработка и научные исследования: для высокопроизводительных вычислений и моделирования.

Архитектура *x86-64* соперничает с другими архитектурами, такими как *ARM64*, в различных областях, и каждая из них имеет свои сильные стороны и применение.

**1.2.2**Оперативная память (*RAM*) 8 ГБ

История развития оперативной памяти в компьютерах имеет долгий путь, начиная с ранних электромеханических устройств и электронных трубок в середине 20-го века. С появлением первых компьютеров, оперативная память представляла собой регистры и буферы, которые использовались для хранения ограниченного количества данных. Со временем разработчики стали осознавать необходимость увеличения объема и скорости оперативной памяти.

Первые интегральные схемы для оперативной памяти появились в 1970-х годах, что позволило значительно увеличить ее объем и скорость. Затем пришли стандарты памяти, такие как *DRAM* (динамическая оперативная память) и *SRAM* (статическая оперативная память), которые сделали оперативную память более доступной и эффективной.

С появлением компьютеров с графическими пользовательскими интерфейсами в 1980-х и 1990-х годах, объем оперативной памяти стал критически важным для обеспечения плавной работы графических приложений и многозадачности.

В последние десятилетия, с развитием многоядерных процессоров и 64-разрядных операционных систем, объем и скорость оперативной памяти продолжают расти. Современные стандарты памяти *DDR3*, *DDR4* и *DDR5* предлагают высокую производительность и эффективность.

На протяжении истории компьютерных технологий существует множество версий оперативной памяти. Однако основные различия заключаются в следующем:

1 *DDR* (*Double Data Rate*) – первые стандарты, которые существуют с 2000 года.

2 *DDR2* – следующее поколение памяти, улучшившее скорость передачи данных.

3 *DDR3* – стандарт, который сделал память еще быстрее и эффективнее.

4 *DDR4* – последнее поколение на момент написания, предлагающее высокую производительность и низкое энергопотребление.

5 *DDR5* – ожидаемый стандарт, который будет продолжать увеличивать производительность.

Оперативная память играет ключевую роль в производительности компьютерных систем. Ее основные преимущества включают в себя:

– быстрый доступ к данным, что ускоряет выполнение задач;

– поддержка многозадачности;

– улучшение производительности в графических и видео приложениях;

– снижение времени загрузки операционной системы и приложений;

– надежность и стабильность в работе системы.

Инновации в оперативной памяти включают в себя управление энергопотреблением для экономии электроэнергии, технологии исправления ошибок для обеспечения надежности данных и увеличение скорости передачи данных с каждым новым стандартом.

Оперативная память сравнивается с альтернативными формами хранения данных, такими как жесткие диски и *SSD* (твердотельные накопители). В сравнении с ними, оперативная память предлагает мгновенный доступ к данным, что делает ее идеальным выбором для выполнения задач, требующих высокой скорости чтения и записи.

Оперативная память широко используется во всех областях информационных технологий, включая домашние компьютеры, серверы, мобильные устройства и встраиваемые системы. Ее выдающиеся задачи включают в себя поддержку операционных систем, обработку данных в реальном времени и обеспечение высокой производительности в играх и профессиональных приложениях.

**1.2.3***SSD*-диск *Micron 2210*

Микрон (*Micron Technology*) – это американская компания, специализирующаяся на производстве полупроводников и твердотельных накопителей. Что касается *SSD* дисков *Micron*, их история связана с развитием технологии *NAND*-флэш памяти и ростом рынка твердотельных накопителей. Микрон начала производство *SSD* дисков в начале 2010-х годов. С течением времени *SSD* диски *Micron* стали обладать большей емкостью, более высокой скоростью чтения и записи, а также улучшенной надежностью благодаря развитию *NAND*-технологий и контроллеров. Рост объемов данных, требования к скорости и надежности, а также снижение стоимости *NAND*-флэш памяти оказали влияние на эволюцию *SSD* дисков *Micron*.

Серия *SSD* дисков *Micron* *2210*, как правило, имеет несколько версий с различными объемами памяти и характеристиками скорости. Улучшения могут включать в себя увеличение емкости, повышение скорости чтения/записи и снижение энергопотребления.

К достоинствам *SSD*-дисков *Micron* можно отнести:

– высокая скорость чтения и записи данных;

– надежность и долгий срок службы;

– отсутствие подвижных деталей;

– относительно низкое энергопотребление;

– быстрый доступ к данным.

*Micron* внедряет новые технологии *NAND*-флэш памяти, такие как *3D NAND*, и совершенствует контроллеры, чтобы улучшить производительность и надежность своих *SSD* дисков.

*SSD* диски *Micron* *2210* применяются в различных областях, включая настольные компьютеры, ноутбуки, серверы и встраиваемые системы. Они эффективно работают в задачах, где требуется быстрый доступ к данным, надежность и производительность, таких как загрузка операционной системы, работа с приложениями и обработка данных.

**1.2.4**Видеокарта *NVIDIA GeForce GTX 1650*

Архитектура графических процессоров развивалась на протяжении многих лет. Первые графические процессоры были разработаны в конце 20-го века и предназначались для ускорения графических вычислений на компьютерах. С развитием видеоигр и требований к графике, графические процессоры стали более мощными и функциональными.

Создание дискретных видеокарт, таких как *NVIDIA GeForce GTX 1650*, связано с ростом популярности компьютерных игр и профессиональных графических приложений. Даты выпуска различных версий видеокарт и ключевые изменения в их архитектуре также являются важными моментами в истории.

В архитектуре *GPU NVIDIA GeForce GTX* есть множество версий и поколений. Каждое новое поколение обычно вносит улучшения в производительность, эффективность и функциональность. Примерами могут служить различные модели в серии *GeForce GTX*, такие как 900-я, 1000-я, и 1600-я серии. Важно рассмотреть, какие конкретные изменения и улучшения были внесены в каждую версию и почему они были важны для пользователей.

Архитектура *NVIDIA GeForce GTX* *1650* обладает рядом преимуществ, включая высокую графическую производительность, поддержку современных графических технологий, таких как *Ray Tracing*, эффективное охлаждение, и возможность использования в игровых и профессиональных приложениях. Эти достоинства могут существенно повысить производительность и качество визуальных задач на компьютере.

В архитектуре графического процессора всегда присутствуют инновации и технологии. Это могут быть новые алгоритмы обработки графики, улучшенные методы рендеринга, аппаратная поддержка искусственного интеллекта, и многое другое.

## **1.3 Обоснование выбора вычислительной системы**

При выборе вычислительной системы важным фактором было обеспечение комфортных условий для работы, создания виджета и взаимодействия с аппаратным обеспечением. В данном случае выбранная вычислительная система отвечает этим требованиям по нескольким причинам:

1 Производительность процессора *Intel Core i5*: процессор *Intel Core i5* предоставляет хорошую производительность для выполнения широкого спектра задач. Его многозадачные способности позволяют комфортно работать с различными приложениями, включая разработку программного обеспечения, обработку данных и другие вычислительно интенсивные задачи.

2 Объем и скорость оперативной памяти: выбор оперативной памяти является важным аспектом архитектуры системы. 8 ГБ оперативной памяти, предоставляемой этой системой, обеспечивают достаточное количество ресурсов для эффективной работы с приложениями, включая среды разработки и виртуальные машины.

3 Скорость и емкость твердотельного: присутствие твердотельного накопителя обеспечивает высокую скорость загрузки операционной системы и приложений, что улучшает общую производительность системы. Он также обеспечивает хорошую отзывчивость и быстрое открытие файлов.

4 Дискретная видеокарта *NVIDIA GeForce GTX 1650*: для выполнения графических задач и игр, а также для разработки и тестирования графических приложений, важна дискретная видеокарта. *NVIDIA GeForce GTX 1650* обеспечивает высокую производительность и качественное отображение графики.

5 Совместимость с программными средами: *Windows 10* является популярной операционной системой, обеспечивающей совместимость с большим количеством программ и сред разработки. Это важно для выполнения курсового проекта и обеспечения легкости в разработке и отладке программ. Поэтому при разработке виджета мониторинга состояния аппаратного обеспечения была выбрана именно операционная система *Windows 10*.

6 Портативность и мобильность: ноутбук *Asus* из серии *TUF Gaming* предоставляет портативность, что позволяет работать над проектом в различных местах и условиях, что может быть важно для эффективного выполнения задач.

**1.4 Анализ выбранной вычислительной системы**

**1.4.1** Анализ *Intel Core i5-10300H*

Процессор *Intel Core i5-10300H* оборудован 4 ядрами и 8 потоками для управления многозадачностью. Максимальная тактовая частота процессора достигает 4.50 *GHz*, что гарантирует быстродействие системы. Процессор имеет максимальную температуру работы 100*°C*, что говорит о том, что работа процессора сбалансирована между производительностью и теплорассеиванием. Размеры кэша процессора включают *L1* (256 *KB)*, далее *L2* (1 *MB*) и *L3* (8 *MB*). *Intel Core i5-10300H* поддерживает память типа *DDR4-2933*, обеспечивая эффективное управление памятью. Максимально поддерживаемый размер памяти процессора достигает 128 *GB*, что обеспечивает необходимый объем ресурсов для работы с устройством. В процессор также интегрирована графика *Intel UHD Graphics*. [8]

**1.4.2** Анализ *NVIDIA GeForce GTX 1650* и *TU117*

Графический процессор обладает архитектурой, оптимизированной для параллельной обработки данных. Ядра *CUDA* предоставляют возможность использовать вычислительные ресурсы графического процессора. *NVIDIA GeForce GTX 1650* имеет 896 потоковых процессоров. Большое количество потоковых процессоров ведет к более высокой производительности в параллельных задача, таких как обработки графики или симуляции физики. Тактовая частота видеокарты может быть около 1485-1560 *MHz*. Объем видеопамяти (*VRAM*) у видеокарты *NVIDIA GeForce GTX 1650* составляет 4 *GB*. Этот объем видеопамяти обеспечивает достаточные ресурсы для обработки текстур, кадров и других графических данных. [9]

**1.4.3** Анализ оперативной памяти

Оперативная память выбранной вычислительной системы относится к типу *DDR4* и представлена в объеме 8 *GB*. Тип оперативной памяти обеспечивает высокую скорость передачи данных и эффективность. Объем оперативной памяти обеспечивает достаточное количество ресурсов для эффективного выполнения обработки данных, программ и многозадачных операций.

**1.4.4** Анализ твердотельного накопителя

SSD Micron 2210 представляет собой твердотельный накопитель с объемом хранения данных в размере 512 ГБ. Этот накопитель использует флеш-память с технологией 3D QLC NAND, что означает использование многозвенной клеточной структуры для хранения четырех уровней данных (quad-level cell). Технология 3D QLC NAND обеспечивает высокую плотность данных, что важно для максимального использования доступного пространства на накопителе.

Скорость последовательной записи на SSD Micron 2210 достигает 1070 мегабайт в секунду (МБ/с). Эта характеристика означает способность накопителя последовательно записывать данные на свою память со значительной скоростью.

Максимальная производительность SSD Micron 2210 достигает 320,000 операций ввода/вывода в секунду (IOPS). Операции ввода/вывода представляют собой ключевой параметр для оценки способности накопителя обрабатывать запросы на чтение и запись данных. Высокая производительность IOPS обеспечивает отзывчивость системы при обработке множества мелких операций чтения/записи. [10]

В целом, выбранная вычислительная система обеспечивает баланс между производительностью, совместимостью, портативностью и доступностью, что делает ее максимально оптимальным решением для выполнения данного курсового проекта.

# **2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**2.1** **Структура и архитектура платформы**

**2.1.1** Выбранная платформа

Разработка проекта «Виджет мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы» ведется на языке *Python* с использованием среды разработки *PyCharm*.

Основной платформой разработки виджета является *Python*, так как именно этот язык программирования был выбран для проектирования проекта.

*PyCharm* – это специализированная интегрированная среда разработки *Python*, предоставляющая широкий спектр необходимых инструментов для разработчиков, тесно интегрированных для создания удобной среды для продуктивной разработки. Все, что происходит в *PyCharm*, происходит в контексте проекта. Он служит основой для помощи в кодировании, массового рефакторинга, согласованности стилей кодирования и прочего. [11]

Для работы с проектом была выбрана операционная система *Windows*. *Windows* – это операционная система, созданная корпорацией *Microsoft*. Операционная система (ОС) – главная программа, которая запускается при включении компьютера. Она позволяет пользователям компьютера работать с файлами, пользоваться нужными приложениями, выходить в Интернет. [12]

**2.1.2** Язык программирования

Язык программирования – формальный язык, предназначенный для записи компьютерных программ. Язык программирования определяет набор лексических, синтактических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под ее управлением.

Язык программирования предназначен для написания компьютерных программ, которые представляют собой набор правил, позволяющих компьютеру выполнить тот или иной вычислительный процесс, организовать управление различными объектами и прочее. Язык программирования отличается от естественных языков тем, что предназначен для управления ЭВМ, в то время как естественные языки используются, прежде всего, для общения людей между собой. Большинство языков программирования использует специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

**2.1.3** Интегрированная среда разработки

Интегрированная среда разработки (*IDE*) – комплекс программных средств, используемый программистами для разработки программного обеспечения.

Среда разработки включает в себя:

– текстовый редактор;

– транслятор (компилятор или интерпретатор);

– средства сборки и отладки.

Интегрированные среды разработки были созданы для того, чтобы максимизировать производительность программиста благодаря тесно связанным компонентам с простыми пользовательскими интерфейсами.

*IDE* обычно представляет собой единственную программу, в которой проводится вся разработка. Она, как правило, содержит много функций для создания, изменения, компилирования, развертывания и отладки программного обеспечения. Цель интегрированной среды заключается в том, чтобы объединить различные утилиты в одном модуле, который позволит абстрагироваться от выполнения вспомогательных задач, тем самым позволяя программисту сосредоточиться на решении собственно алгоритмической задачи и избежать потерь времени при выполнении типичных технических действий (например, вызове компилятора). Таким образом, повышается производительность труда разработчика. Также считается, что тесная интеграция задач разработки может далее повысить производительность за счёт возможности введения дополнительных функций на промежуточных этапах работы. Например, *IDE* позволяет проанализировать код и тем самым обеспечить мгновенную обратную связь и уведомить о синтаксических ошибках.

**2.1.4** Операционная система

Как уже отмечалось, современный компьютер организован в виде иерархии уровней, каждый из которых добавляет определенные функции к нижележащему уровню.

С точки зрения программиста, операционная система – это программа, добавляющая ряд команд и функций к командам и функциям, предлагаемым уровнем архитектуры команд. Обычно операционная система организуется программно, но нет никаких веских причин, по которым ее, как микропрограммы, нельзя было реализовать.

Все команды уровня операционной системы доступны для прикладных программистов. Это – практически все команды более низкого уровня, а также новые команды, добавленные операционной системой. Новые команды называются системными вызовами. Они вызывают предопределенную службу операционной системы, в частности одну из ее команд. Например, типичный системный вызов может читать данные из файла.

**2.2 История, версии и достоинства**

**2.2.1** Язык программирования *Python*

*Python* – это мощный и популярный язык программирования, который был создан Гвидо ван Россумом и впервые выпущен в конце 1980-х годов.

Работа над *Python* началась в конце 1980-х годов, и первая публичная версия, *Python 0.9.0*, была выпущена в 1991 году. Основными целями создания *Python* были читаемость кода, простота и ясность.

Версии *Python 2.x* стали очень популярными и использовались в различных проектах. Однако, с развитием языка возникли некоторые проблемы, связанные с совместимостью и дизайном.

*Python 3.x* (2008 – н.в.): *Python 3.x* был выпущен в 2008 году и представил множество улучшений и изменений, направленных на устранение проблем, обнаруженных в *Python 2.x*. Это привело к некоторым разрывам совместимости, но с течением времени многие проекты перешли на *Python 3.x*.

*Python* имеет множество версий, но две основные ветки – *Python 2.x* и *Python 3.x*. Каждая из этих веток имеет свои подверсии, и каждая версия вносит улучшения и изменения. На момент окончания моего знания (сентябрь 2021 года), актуальной версией *Python* была *Python 3.9*. Кроме того, *Python* активно развивается, и новые версии регулярно выпускаются с улучшениями и новыми возможностями.

*Python* пользуется широкой популярностью благодаря ряду выдающихся достоинств:

1 Простота и читаемость: *Python* известен своей простотой и легкочитаемым синтаксисом, который делает код более понятным и поддерживаемым.

2 Множество библиотек и фреймворков: *Python* имеет богатую экосистему библиотек и фреймворков для разработки, что упрощает создание разнообразных приложений.

3 Кроссплатформенность: *Python* поддерживает множество операционных систем, что делает его доступным на различных платформах.

4 Широкое применение: *Python* применяется в разных областях, включая веб-разработку, научные вычисления, искусственный интеллект, анализ данных и многое другое.

5 Активное сообщество: *Python* имеет активное сообщество разработчиков и пользователей, что обеспечивает поддержку и развитие языка.

**2.2.2** Среда разработки *PyCharm*

*PyCharm* – это одна из самых популярных сред разработки для языка программирования *Python*. Она была создана компанией *JetBrains* и впервые выпущена в 2010 году.

Работа над *PyCharm* началась в *JetBrains* как ответ на растущую популярность языка *Python*. Целью было создание мощной и интуитивно понятной среды разработки для *Python*-разработчиков.

Первая версия *PyCharm* была выпущена в феврале 2010 года. Она предоставила множество возможностей, таких как интегрированный отладчик, автодополнение кода, управление виртуальными окружениями и многое другое.

С течением времени *PyCharm* стала одной из самых популярных сред разработки *Python* и продолжает активно развиваться и совершенствоваться.

*PyCharm* имеет несколько версий, каждая из которых предоставляет разные уровни функциональности и целевые аудитории. На момент окончания моего знания (сентябрь 2021 года), существовали следующие версии *PyCharm*:

1 *PyCharm Community Edition*: бесплатная версия *PyCharm*, предназначенная для небольших проектов и индивидуальных разработчиков.

2 *PyCharm Professional Edition*: платная версия *PyCharm*, предоставляющая расширенные возможности для профессиональных разработчиков и команд.

*PyCharm* пользуется выдающимися достоинствами, которые делают ее одной из предпочтительных сред разработки *Python*:

1 Мощный редактор кода: *PyCharm* обладает мощным и интуитивно понятным редактором кода с функциями автодополнения, подсветки синтаксиса и другими инструментами для удобной работы с кодом.

2 Интегрированный отладчик: встроенный отладчик позволяет проще и быстрее находить и устранять ошибки в коде.

3 Управление виртуальными окружениями: *PyCharm* облегчает создание и управление виртуальными окружениями, что полезно при работе с разными проектами и библиотеками.

4 Поддержка множества фреймворков: *PyCharm* поддерживает различные фреймворки, включая *Django*, *Flask*, *PyQT* и другие, упрощая разработку веб-приложений и других приложений.

5 Анализ кода и рефакторинг: Среда предоставляет множество инструментов для анализа кода и проведения рефакторинга, что помогает улучшить качество кода.

**2.2.3** Операционная система *Windows*

Операционная система *Windows* разработана корпорацией *Microsoft* и является одной из самых распространенных операционных систем в мире. Ее история насчитывает несколько десятилетий, и вот ключевые моменты ее развития:

1 *Windows 1.0* (1985): первая версия *Windows* была выпущена в 1985 году. Она предоставляла графический интерфейс пользователя (*GUI*) для *MS-DOS* и включала приложения, такие как калькулятор и блокнот.

2 *Windows 3.0* (1990): *Windows 3.0* внесла множество улучшений, включая поддержку цветовых экранов и возможность мультитаскинга.

3 *Windows 95* (1995): *Windows 95* была революционным выпуском, введя значительные изменения в интерфейс пользователя и поддерживая 32-битные приложения.

4 *Windows XP* (2001): *Windows XP* была одной из самых долгожданных версий *Windows* и предоставила стабильность и производительность.

5 *Windows 7* (2009): *Windows 7* улучшила интерфейс пользователя и внесла улучшения в производительность.

6 *Windows 8* и *8.1* (2012): эти версии внесли радикальные изменения в интерфейс, с учетом сенсорных устройств.

7 *Windows 10* (2015): *Windows 10* была представлена как универсальная операционная система для ПК, планшетов и смартфонов.

*Windows* обладает рядом значительных преимуществ и достоинств:

1 Популярность и совместимость: *Windows* является самой популярной операционной системой, что обеспечивает широкую совместимость с программным обеспечением и аппаратным обеспечением.

2 Игровая платформа: *Windows* является ведущей платформой для игр и обеспечивает доступ к широкому спектру видеоигр.

3 Простота использования: интерфейс *Windows* обычно считается более интуитивным и легким для освоения, особенно для новых пользователей.

4 Множество функций: *Windows* предоставляет множество функций, включая встроенные инструменты для работы с офисными приложениями, веб-браузерами и мультимедиа

5 Обновления и поддержка: *Microsoft* регулярно выпускает обновления и обеспечивает техническую поддержку для *Windows*, обеспечивая безопасность и стабильность системы.

6 Широкий спектр аппаратных устройств: *Windows* поддерживает множество устройств, включая настольные ПК, ноутбуки, планшеты и смартфоны.

**2.3 Анализ программного обеспечения для написания программы и       обоснование выбора платформы**

Для разработки виджета мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы были рассмотрены несколько *IDE*:

– *PyCharm*;

– *Visual Studio*.

Был выбран именно *PyCharm* как основная среда разработки для языка *Python*. Эта мощная *IDE* предоставляет обширный набор инструментов для работы с *Python*, включая отладчик, поддержку виртуальной среды. Также *PyCharm* очень прост в использовании, не вызывает трудностей и имеет возможность «помогать» программисту в разработке кода, показывая специальные подсказки для упрощения работы.

Курсовой проект разрабатывается с использованием языка программирования *Python* и интегрированной среды разработки *PyCharm* также и последующим причинам. Во-первых, *Python* был выбран из-за существующего опыта работы с ним и предпочтения этого языка программирования. Во-вторых, *PyCharm* была выбрана как среда разработки из-за ее удобства и функциональности, а также из-за предпочтения этой среды разработки.

Этот выбор основывается на комфорте использования и знании инструментов, что способствует более эффективной разработке курсового проекта.

**3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ    ПРОГРАММНОГО ПРОДУКТА**

**3.1 Обоснование необходимости разработки**

В наше время использование компьютеров и других устройств на базе аппаратного обеспечения становится все более распространенным. Это приводит к необходимости надежного мониторинга и управления состоянием аппаратных ресурсов, а также предсказания времени работы устройств.

С развитием технологий и доступностью вычислительных устройств, больше людей и организаций используют компьютеры, ноутбуки, серверы и другие устройства. Важно иметь возможность наблюдать и анализировать работу аппаратного обеспечения, чтобы предотвращать сбои и оптимизировать его использование. Важно предсказывать перегрузки и устранять проблемы до их возникновения.

Многие процессы и бизнес-операции зависят от бесперебойной работы вычислительного оборудования. Отказы в работе могут вызвать значительные финансовые потери и ухудшить качество обслуживания клиентов. Мониторинг состояния аппаратного обеспечения и прогноз времени работы позволяет своевременно предпринимать меры для предотвращения простоев и увеличения надежности систем.

Существует множество различных конфигураций аппаратного обеспечения, начиная от персональных компьютеров и заканчивая большими серверными фермами. Каждая конфигурация может иметь свои особенности и требования к мониторингу. Необходимо иметь гибкий и настраиваемый инструмент для отслеживания состояния аппаратуры.

Мониторинг аппаратного обеспечения и прогноз времени работы позволяет оптимизировать использование ресурсов. Путем анализа данных о работе оборудования можно идентифицировать узкие места и распределить нагрузку равномерно. Это способствует увеличению производительности и снижению энергопотребления.

**3.2 Технологии программирования, используемые для решения       поставленных задач**

Для разработки виджета мониторинга состояния аппаратного обеспечения и прогноза времени работы, а также для обеспечения его эффективной и надежной работы, используются инструменты, описанные во второй г:

1 Python: язык программирования *Python* выбран в качестве основного языка разработки виджета. *Python* предоставляет удобный синтаксис, богатую стандартную библиотеку и обширное сообщество разработчиков, что облегчает создание и поддержку проекта. Этот язык также поддерживает множество библиотек для работы с аппаратным обеспечением и мониторинга системы.

2 *PyCharm*: интегрированная среда разработки (*IDE*) *PyCharm* используется для разработки, отладки и тестирования виджета. *PyCharm* предоставляет мощные инструменты для написания кода, рефакторинга, автодополнения и управления проектом. Его функциональность позволяет значительно упростить и ускорить процесс разработки.

3 *Windows 10*: проект будет разрабатываться и тестироваться на операционной системе *Windows 10*, что обеспечит совместимость с широким кругом пользователей.

Кроме того, при разработке курсового проекта были рассмотрены три модуля языка программирования *Python* для работы с аппаратным обеспечением компьютера. В их числе:

– *psutil*;

– *GPUtil*;

– *wmi*.

**3.2.1** Технологии языка *Python*

В результате разработки курсового проекта были рассмотрены три модуля языка *Python* для работы с аппаратным обеспечением компьютера:

Библиотека *WMI* предоставляет доступ к информации об аппаратном и программном обеспечении в системе. *WMI* предоставляет доступ к множеству счетчиков производительности операционной системы и других компонентов. Эти счетчики включают данные о загрузке процессора, использовании памяти, сетевом трафике и других параметрах, позволяя мониторить производительность системы в реальном времени. [16]

Получение мощности и температуры каждого процессора было выполнено при помощи модуля *wmi*, так как другие библиотеки не предоставляют возможности получить необходимые данные.

*Psutil* предоставляет простой и понятный *API* для доступа к информации о ресурсах компьютера и процессах. Эта библиотека обеспечивает полную совместимость с различными операционными системами, что делает ее универсальным инструментом для мониторинга аппаратного обеспечения. Она также предоставляет широкий спектр функций, включая мониторинг использования ЦП, оперативной памяти, дискового пространства и других аспектов системы.

Библиотека *psutil* предоставляет разработчикам многочисленные возможности для мониторинга аппаратного обеспечения компьютера и прогноза времени его работы. [17]

Возможности библиотеки *psutil* позволяют получить информацию о различных аспектах компьютерной системы, что критически важно для разработки виджета мониторинга и обеспечения надежной работы программы.

Возможности библиотеки *psutil*:

– мониторинг процессора;

– мониторинг оперативной памяти;

– мониторинг процессов;

– системная информация;

– прогноз времени работы батареи.

Эти возможности *psutil* обеспечивают разработчикам важные данные для создания виджета мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы. Разработчики могут анализировать эти данные, предоставляя пользователям информацию о текущем состоянии и производительности и оптимизируя работу компьютера для повышения эффективности и продолжительности его работы.

Для мониторинга *GPU* был использован модуль *GPUtil*, так как с помощью него можно легко получать информацию о графических процессорах в системе. Модуль *GPUtil* предоставляет методы для:

– получения информации о загрузке процессора;

– получения информации о использовании памяти процессора;

– получения общей информации о процессоре.

*GPUtil* упрощает доступ к информации о *GPU* и предоставляет удобный интерфейс для интеграции с приложениями, которые требуют мониторинга и управления графическими ресурсами. [18]

**3.3 Связь архитектуры вычислительной системы с       разрабатываемым программным обеспечением**

Архитектурные характеристики аппаратного обеспечения, такие как процессор, оперативная память, накопитель (*SSD* или *HDD*), и видеокарта, играют важную роль в разработке виджета мониторинга. Процессор, его архитектура и скорость определяют способность виджета анализировать и обрабатывать данные о состоянии системы. Объем оперативной памяти и ее скорость напрямую влияют на производительность и отзывчивость виджета. Накопитель, в данном случае *SSD*, несет ответственность за быстрый доступ к операционной системе и данным, что важно для быстрого запуска виджета и отклика на запросы пользователей. Видеокарта, в свою очередь, может использоваться для визуализации информации, что повышает интерактивность виджета.

Операционная система, в данном случае *Windows*, играет ключевую роль в архитектуре вычислительной системы, на которой будет функционировать виджет. Это предполагает не только учет архитектурных особенностей ОС, но и интеграцию виджета с операционной системой. Это интеграцию позволяет пользователям получать доступ к информации о состоянии аппаратного обеспечения и прогнозу времени его работы через удобные интерфейсные решения.

Гармоничное взаимодействие виджета с аппаратным обеспечением и операционной системой обеспечивает оптимальную производительность. Это критически важно, так как мониторинг и прогнозирование состояния требуют вычислительных ресурсов. Эффективная оптимизация кода и использование ресурсов компьютера позволяют достичь выдающейся производительности виджета, что является основополагающим аспектом при разработке подобных программных решений.

**4 ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ    ПРОГРАММЫ**

**4.1 Обоснование и описание функций программного обеспечения**

**4.1.1** Цели и задачи программного обеспечения

Целью разработанного программного обеспечения "Виджет мониторинга состояния аппаратного обеспечения и прогноз времени его работы" является предоставление пользователям актуальной и полной информации о состоянии аппаратного обеспечения и времени работы компьютера с целью оптимизации его использования, предотвращения потери данных и повышения общей производительности.

Программное обеспечение предоставляет пользовательский интерфейс, который позволяет пользователям легко и интуитивно просматривать использование ресурсов, температуру компонентов и производительность работы компьютера. Оно также предоставляет информацию о времени работы от аккумулятора на основе текущего использования и оставшегося заряда.

Основные характеристики программного обеспечения включают в себя:

1 Мониторинг аппаратного обеспечения: пользователям доступна информация о состоянии процессора, оперативной памяти, жесткого диска, видеокарты и других компонентов компьютера. Это позволяет оперативно реагировать на изменения в использовании ресурсов и предотвращать возможные проблемы.

2 Прогноз времени работы: для ноутбуков и портативных устройств, программное обеспечение предоставляет информацию о времени работы от аккумулятора на основе текущего использования и оставшегося заряда. Это помогает пользователям эффективно управлять энергопотреблением.

3 Настраиваемые параметры: пользователи имеют возможность настройки виджета в соответствии с их потребностями. Они могут выбрать, какие компоненты мониторить, настроить уровни оповещений и выбрать предпочтительный язык интерфейса.

4 Документация и поддержка: для удобства использования программного обеспечения предоставляется подробная документация и возможность получения поддержки в случае возникновения вопросов или проблем.

Таким образом, разработанное программное обеспечение обеспечивает надежный и удобный инструмент для мониторинга состояния аппаратного обеспечения компьютера и оптимизации его использования.

**4.1.2** Полное описание функций

Мониторинг процессора: позволяет пользователю отслеживать загрузку процессора, текущую частоту и температуру. Это помогает идентифицировать процессы, которые могут потреблять большое количество ресурсов и вызывать проблемы с производительностью. Пользователи могут просматривать информацию о *CPU* и процессах в виджете и принимать меры, если обнаружат проблемы. Результат работы мониторинга процессора с помощью виджета мониторинга аппаратного обеспечения компьютера представлен в приложении Д на рисунке 1.

Для подтверждения правильности полученных результатов к каждому изображению работы программы для центрального и графического процессора будет представлено изображение результатов, отображенных в специальной утилите *Open Hardware Monitor* и запечатленных в тот же момент, в который запечатлен результат разработанного виджета.

Утилита *Open Hardware Monitor* – это программа мониторинга аппаратных ресурсов компьютера, предназначенная для отслеживания и отображения данных о температуре, напряжении, скорости вентиляторов и других характеристиках железа. Она обеспечивает пользователей и системных администраторов информацией о текущем состоянии аппаратных компонентов, что позволяет им контролировать и оптимизировать работу системы. Программа может отслеживать температуру различных компонентов, напряжение на различных шинах и элементах питания, скорость загрузки процессора, оперативной памяти, графической карты и других компонентов.

Результат мониторинга процессора при помощи *Open Hardware Monitor* предоставлен на рисунке 4.1.

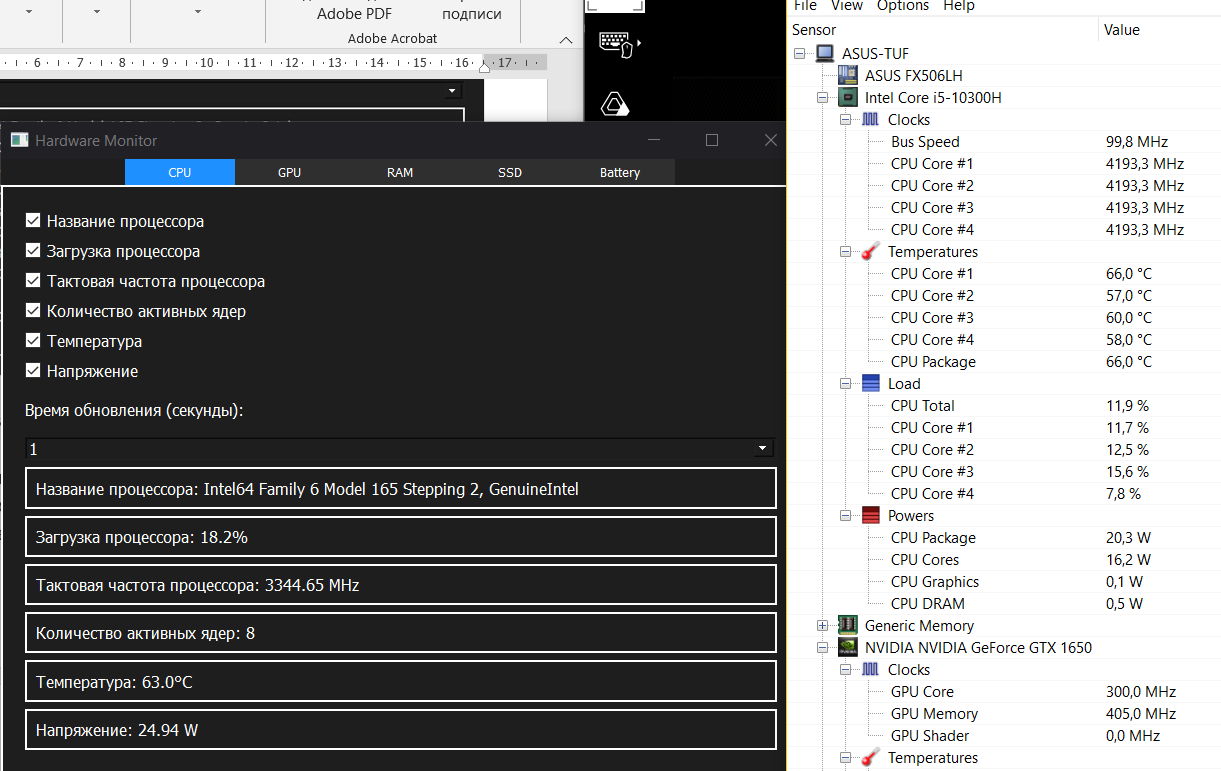


Рисунок 4.1 – Мониторинг *CPU* при помощи *Open Hardware Monitor*

Мониторинг графического процессора: позволяет пользователю отслеживать использование графического процессора, его температуру и производительность. Информация о графическом процессоре отображается в виджете, и пользователи могут следить за ее состоянием. Информация о мониторинге графического процессора предоставлена в приложении Д на рисунке 2.

На рисунке 4.2 предоставлена информация мониторинга графического процессора при помощи утилиты *Open Hardware Monitor*.

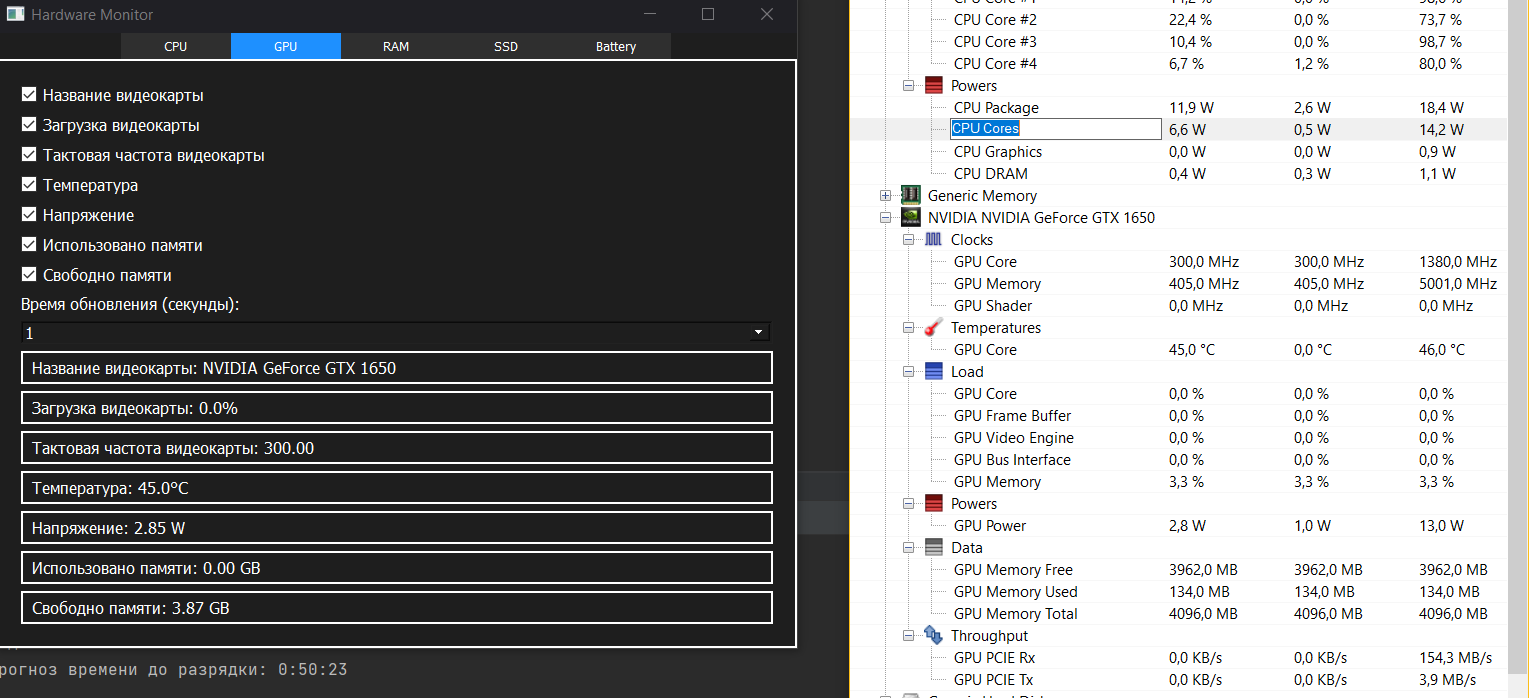


Рисунок 4.2 – Мониторинг *GPU* при помощи *Open Hardware Monitor*

Мониторинг оперативной памяти: предоставляет информацию о использовании оперативной памяти, включая общий объем и доступную память. Информация о *RAM* отображается в виджете, и пользователи могут принимать меры по оптимизации использования памяти. Результат мониторинга информации о *RAM* представлен в приложении Д на рисунке 3.

Для сравнения предоставлен результат мониторинга информации о *RAM* при помощи специальной утилиты *ARMOURY CRATE*. Информация о мониторинге *RAM* при помощи данной утилиты предоставлена на рисунке 4.3.

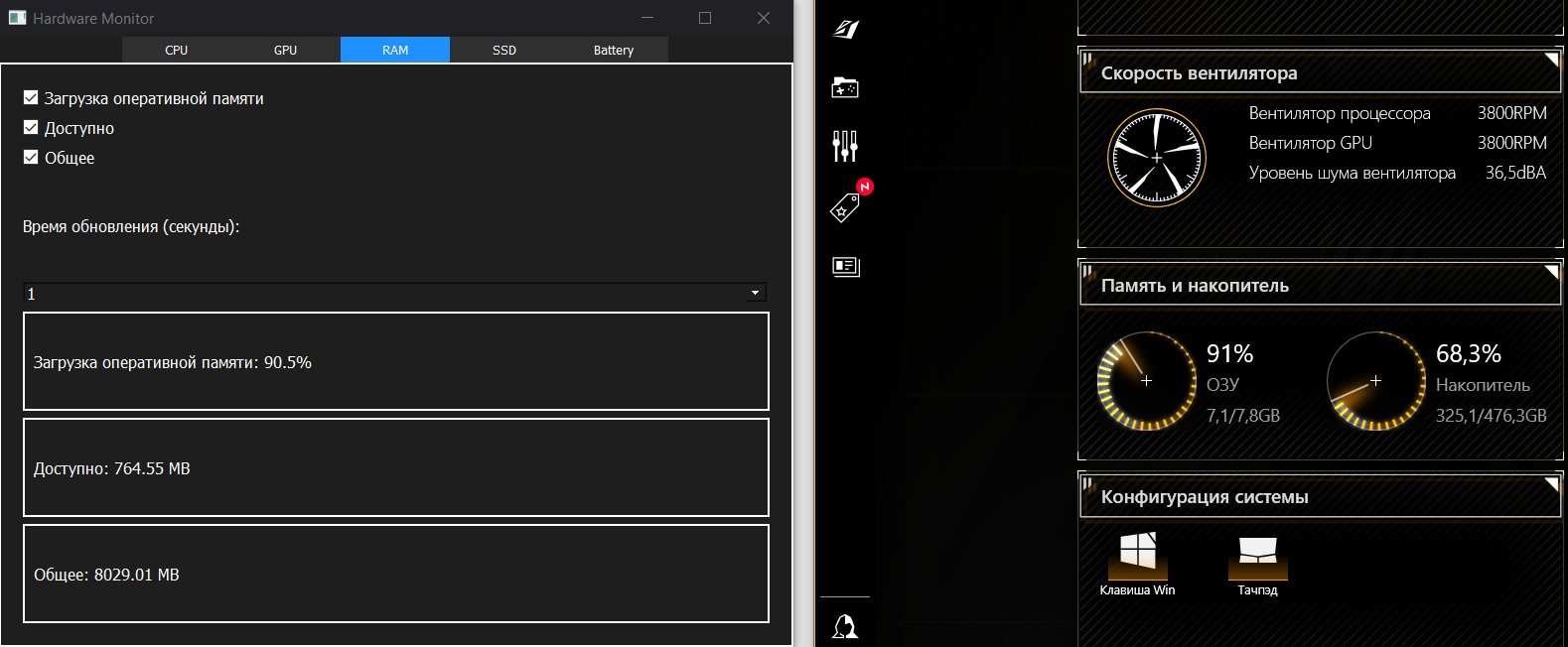


Рисунок 4.3 – Мониторинг *RAM* при помощи утилиты *ARMOURY CRATE*

В разделе мониторинга жестких дисков и SSD осуществляется отслеживание важных параметров, таких как доступное пространство на диске, текущее использование и общее состояние каждого диска. Эта функциональность крайне важна, поскольку позволяет предотвращать возможные проблемы, такие как переполнение диска, что может привести к потере данных.

Виджет мониторинга обращается к системе и получает информацию о количестве дисков, установленных на компьютере. Далее для каждого диска собирается информация о доступном пространстве, что позволяет пользователю в реальном времени видеть, сколько места осталось на каждом диске.

Важной частью мониторинга является отслеживание текущего использования дисков и их общего состояния. Это включает в себя информацию о скорости передачи данных, температуре и других характеристиках, которые могут указывать на потенциальные проблемы с дисками.

Для подтверждения точности предоставляемой информации виджета, производится сравнение данных, отображаемых виджетом, с информацией о дисках, полученной из системы через свойства компьютера. Это гарантирует, что виджет отображает достоверную и актуальную информацию.

Визуализация через информацию о компьютере: Результаты мониторинга для каждого диска предоставляются в виде удобной визуализации через информацию о компьютере. Пользователь может сравнивать данные, предоставленные виджетом, с официальной информацией о своих дисках. Информация о мониторинге диска *C* представлена в приложении Д на рисунке 4.

Информация через вкладку «Свойства» о диске *С* предоставлена на рисунке 4.4.

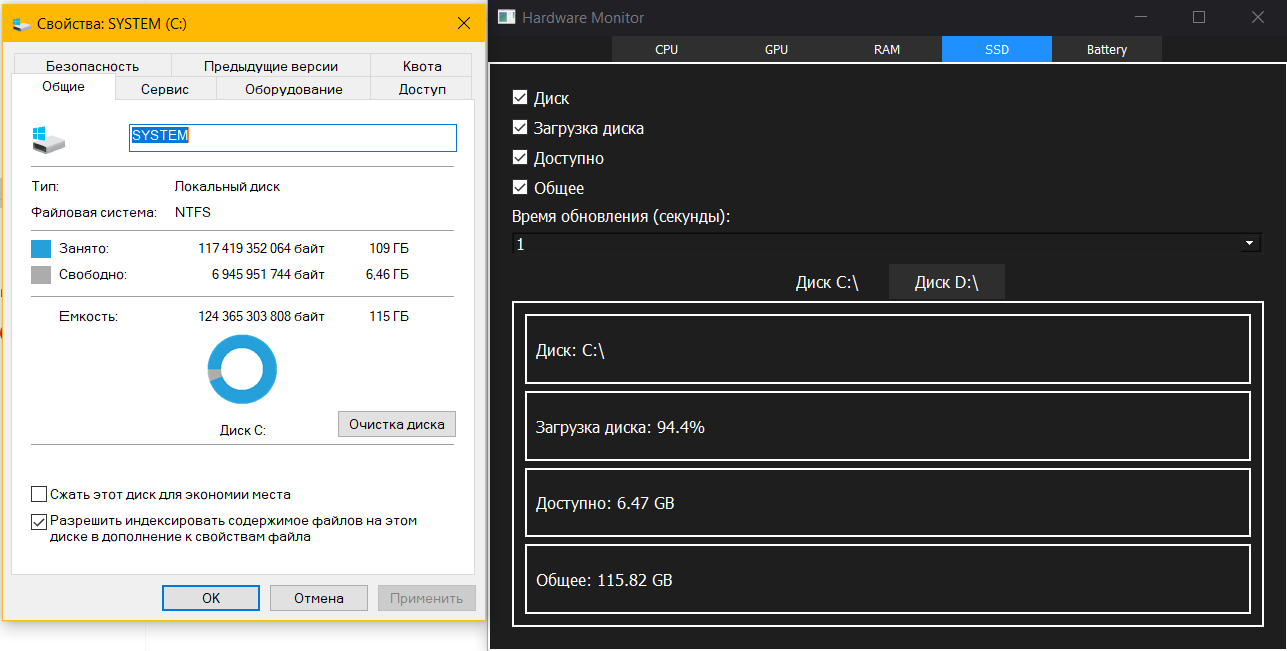


Рисунок 4.4 – Мониторинг диска *С* при помощи отображения «Свойств» компьютера

Информация о мониторинге диска *D* при помощи разработанного виджета предоставлена в приложении Д на рисунке 5.

Информация через вкладку «Свойства» о диске *D* предоставлена на рисунке 4.5.

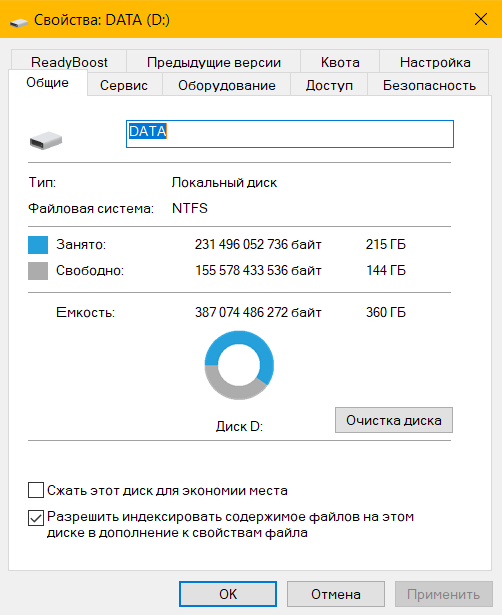


Рисунок 4.5 – Мониторинг диска *D* при помощи отображения «Свойств» компьютера

Прогноз времени работы от аккумулятора (для ноутбуков): предоставляет информацию о времени работы от аккумулятора на основе текущего использования и оставшегося заряда. Это важно для ноутбуков, чтобы предотвратить потерю данных при разряде батареи. Пользователи могут контролировать текущий заряд и наблюдать прогноз времени работы. Информация о времени работы от аккумулятора отображается в виджете и предоставлена в приложении Д на рисунке 6.

Для сравнения приведена информация, отображающаяся на виджете устройства. Данная информация предоставлена на рисунке 4.6.

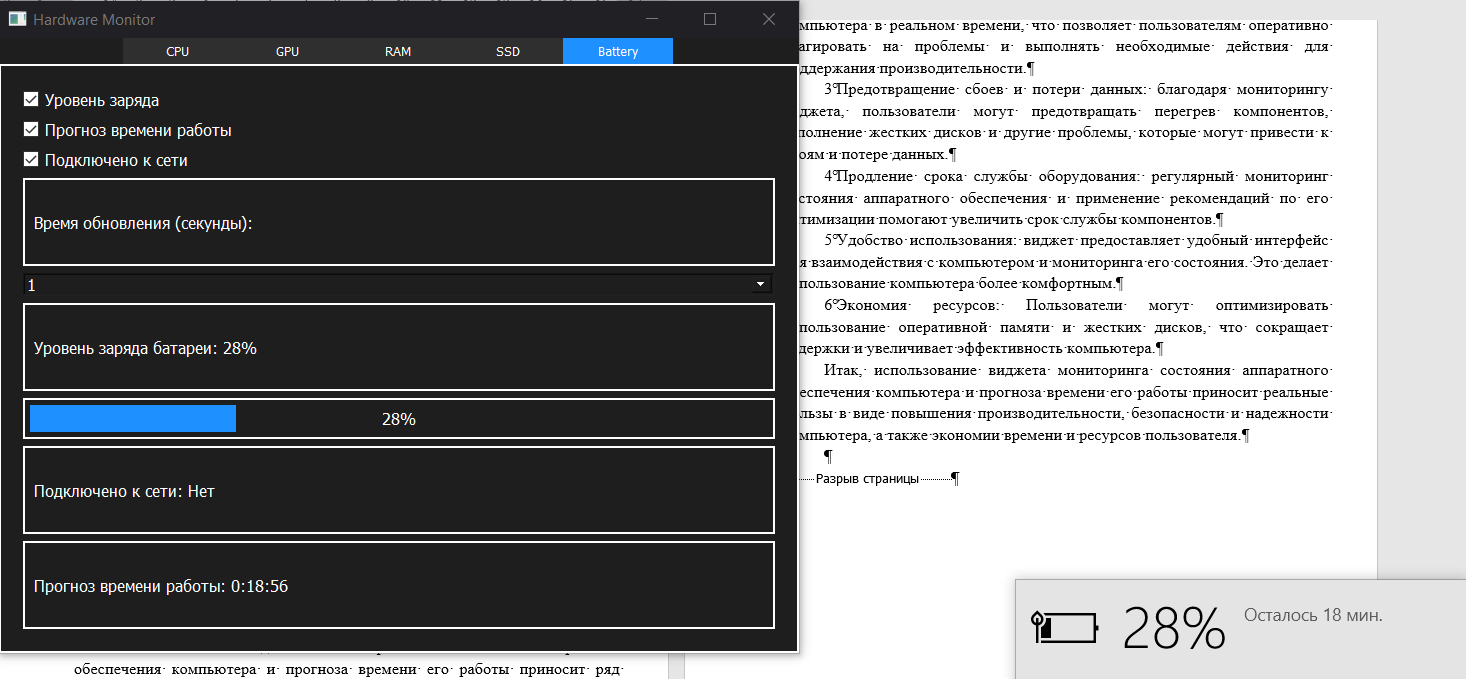


Рисунок 4.6 – Мониторинг батареи при помощи виджета устройства

Если аккумулятор не доступен, разработанный виджет состояния аппаратного обеспечения и прогноза времени его работы обработает данное условие и выведет пользователю информацию о том, что у виджета нет доступа к данным аккумулятора.

Интерфейс пользователя: пользователю предоставлен удобный и интуитивно понятный интерфейс, который позволяет пользователям взаимодействовать с виджетом, просматривать данные и выполнять необходимые операции. Визуальное отображение интерфейса пользователя можно увидеть на рисунках, предоставленных выше.

**4.1.3** Ожидаемые результаты и выгоды для пользователя

Использование виджета мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы приносит ряд конкретных выгод и результатов для пользователей:

1 Улучшение производительности: пользователи могут оптимизировать работу своих компьютеров, прослеживая использование *CPU*, оперативной памяти и жестких дисков. Это помогает предотвратить замедление системы и снижение производительности.

2 Экономия времени: виджет предоставляет информацию о состоянии компьютера в реальном времени, что позволяет пользователям оперативно реагировать на проблемы и выполнять необходимые действия для поддержания производительности.

3 Предотвращение сбоев и потери данных: благодаря мониторингу виджета, пользователи могут предотвращать перегрев компонентов, заполнение жестких дисков и другие проблемы, которые могут привести к сбоям и потере данных.

4 Продление срока службы оборудования: регулярный мониторинг состояния аппаратного обеспечения и применение рекомендаций по его оптимизации помогают увеличить срок службы компонентов.

5 Удобство использования: виджет предоставляет удобный интерфейс для взаимодействия с компьютером и мониторинга его состояния. Это делает использование компьютера более комфортным.

6 Экономия ресурсов: Пользователи могут оптимизировать использование оперативной памяти и жестких дисков, что сокращает издержки и увеличивает эффективность компьютера.

Итак, использование виджета мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы приносит реальные пользы в виде повышения производительности, безопасности и надежности компьютера, а также экономии времени и ресурсов пользователя.

**5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ**

**5.1 Общая структура программы**

Программа представлена в виде графического виджета, который пользователь сможет установить и запустить на своем компьютере. Графический интерфейс предоставляет информацию о текущем состоянии аппаратного обеспечения и прогнозе времени его работы.

Основные компоненты виджета включают в себя:

1 Панель мониторинга: это главная часть интерфейса, на которой отображаются данные о состоянии аппаратного обеспечения. Здесь можно увидеть информацию о процессоре, оперативной памяти, дисковом пространстве, видеокарте и других аппаратных компонентах.

2 Информация о времени работы: виджет также будет предоставлять информацию о прогнозе времени работы компьютера от аккумулятора (в случае ноутбуков). Это важно для пользователей, которые хотят знать, сколько времени осталось до разряда батареи.

3 Настройки и управление: пользователь может настраивать виджет, выбирая, какие аппаратные параметры отображать и как часто обновлять информацию.

Структура программы следует принципам объектно-ориентированного программирования. Код программы разделен на модули и классы, отвечающие за сбор информации, отображение данных, управление интерфейсом и другие задачи.

Таким образом, общая структура программы ориентирована на предоставление пользователю информации о состоянии аппаратного обеспечения и прогнозе времени его работы в удобном и интуитивно понятном виде.

**5.2 Описание функциональной схемы программы**

Описание функциональной схемы программы "Виджет мониторинга состояния аппаратного обеспечения компьютера и прогноз времени его работы", представленная в приложении Б, включает следующие ключевые компоненты и функции:

– сбор информации о состоянии аппаратного обеспечения;

– обработка и анализ данных;

– отображение информации;

– настройка пользовательского интерфейса;

– прогноз времени работы (в случае при работе с ноутбуком).

Программа собирает информацию о состоянии аппаратного обеспечения компьютера, после чего обрабатывает и анализирует данные, для последующего вывода информации на экран пользователя. Пользователь в свою очередь может просмотреть всю необходимую информацию с помощью пользовательского интерфейса. Так же, если виджет используется на ноутбуке, пользователю будет доступна возможность просмотреть оставшееся время работы ноутбука в зависимости от заряда его батареи.

**5.3 Описание блок схемы алгоритма программы**

Блок схема алгоритма программы "Виджет мониторинга состояния аппаратного обеспечения компьютера и прогноз времени его работы" представляет основные шаги и этапы алгоритма программы для виджета мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы. Блок схема алгоритма представлена в приложении В.

Программа начинается с инициализации и настройки, затем переходит в цикл мониторинга, где собирает информацию о компьютере, анализирует и обрабатывает данные о обрабатываемых данных, то есть о *CPU*, *GPU*, *RAM*, *SSD* и батарее, отображает их на виджете. Если компьютер является ноутбуком, программа также рассчитывает прогноз времени работы от батареи.

Взаимодействие с пользователем, настройки, интерфейс и взаимодействие с операционной системой дополняют основной алгоритм. Так же в блок схеме алгоритма есть цикл обновления данных, где программа проверяет, обновлены данные или нет. Если данные были обновлены, происходит выход из цикла и отображение данных на экран. Цикл по умолчанию применяется ежесекундно или в зависимости от выбранного времени обновления.

В итоге, разработанный виджет представляет собой интуитивно понятный, функциональный и настраиваемый инструмент, который удовлетворяет потребности разнообразной аудитории пользователей. Сочетание простоты использования с богатой функциональностью делает этот виджет неотъемлемой частью вычислительного опыта пользователей.

# **ЗАКЛЮЧЕНИЕ**

В рамках данного курсового проекта был разработан и реализован виджет мониторинга состояния аппаратного обеспечения компьютера, а также прогноза времени его работы на основе данных о заряде батареи при ее наличии. В процессе разработки использовались современные технологии, такие как язык программирования *Python*, среда разработки *PyCharm*, библиотека *PyQt5* для создания графического интерфейса пользователя и библиотеки *psutil*, *wmi*, *GPUtil* для сбора информации о состоянии аппаратного обеспечения, которое включает в себя *CPU*, *GPU*, *RAM*, *SSD* и аккумулятор при его наличии.

Интерфейс приложения включает в себя информацию о центральном процессоре, графическом процессоре, загрузке оперативной памяти и жесткого диска, а также о состоянии батареи, предоставляя пользователю полную картину о текущем состоянии компьютера. Также добавлены элементы визуализации, такие как полоса прогресса для более наглядного отображения уровня заряда батареи.

Основной функционал приложения реализован с использованием объектно-ориентированного подхода, что обеспечивает удобство расширения и поддержки кода в будущем. Таймер обновления данных позволяет мгновенно получать актуальную информацию о состоянии системы каждую секунду, обеспечивая актуальность данных в интерфейсе.

Таким образом, созданный виджет предоставляет пользователю удобный и информативный инструмент для мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы.

# **CПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ**

[1] Архитектура вычислительных систем [Электронный ресурс]: учебное пособие – Эл. изд. – Электрон. текстовые дан. (1 файл pdf: 77 с.). – Грейбо С.В., Новосёлова Т.Е., Пронькин Н.Н., Семёнычева И.Ф. 2019. – Режим доступа: <http://scipro.ru/conf/computerarchitecture.pdf> – Дата доступа 30.09.2023

[2] Таненбаум, Э. Архитектура компьютера – 6-е изд. / Э. Таненбаум. СПб. : Питер, 2013. – 816 с.

[3] Петцольд, Ч. Код. Тайный язык информатики / Ч. Петцольд. – М. : Манн, Иванов и Фербер, 2021. – 448 с.

[4] Харрис, Д. М. Цифровая схемотехника и архитектура компьютера – 2-е изд. / Д. М. Харрис, С. Л. Харрис. – NY : Elsevier Inc, 2013. – 1662 c.

[5] Шитов, В. Н. Windows 10 : самый простой и понятный самоучитель / В. Шитов. – М. : Эксмо, 2023. – 464 с.

[6] Русинович, М. Внутреннее устройство Windows – 7-е изд. / М. Русинович [и др.]. – СПб. : Питер, 2018. – 944 с.

[7] [Электронный ресурс]. – Режим доступа:  [https://coremission.net/gamedev/kak-rabotaet-gpu – Дата доступа: 30.09.2023](https://learn.microsoft.com/ru-ru/windows/win32/api%20–%20Дата%20доступа%2020.09.2023)

[8] Обзор процессора Intel Core i5-10300H [Электронный ресурс] –https://askgeek.io/ru/cpus/Intel/Core-i5-10300H#google\_vignette – Дата доступа: 09.11.2023

[9] Обзор видеокарты NVIDIA GeForce GTX 1650 [Электронный ресурс] – https://3dnews.ru/987707/obzor-nvidia-geforce-gtx-1650 – Дата доступа: 09.11.2023

[10] SSD Micron 2210 [Электронный ресурс] – https://3dnews.ru/987707/obzor-nvidia-geforce-gtx-1650 – Дата доступа: 09.11.2023

[11] Попов, А. Администрирование Windows с помощью WMI и WMIC. / А. В. Попов, Е. А. Шикин. СПб. : БХВ-Петербург, 2004. – 752 с.

[12] [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/help/pycharm/quick-start-guide –Дата доступа: 02.09.2023

[13] [Электронный ресурс]. – Режим доступа: https://internet-lab.ru/windows\_os – Дата доступа: 02.09.2023

[14] [Электронный ресурс] – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/wmisdk/wmi-start-page> – Дата доступа: 14.11.2023

[15] [Электронный ресурс] – Режим доступа: <https://script-coding.com/WMI.html> – Дата доступа: 14.10.2023

[16] [Электронный ресурс] – Режим доступа: <https://openhardwaremonitor.org/documentation/> – Дата доступа: 14.10.2023

[17] [Электронный ресурс] – Режим доступа: <https://pypi.org/project/psutil/> – Дата доступа: 10.10.2023

[18] [Электронный ресурс] – Режим доступа: <https://github.com/williamyang1991/GP-UNIT> – Дата доступа: 10.10.2023

[19] [Электронный ресурс] – Режим доступа: <https://wiki.python.org/moin/PyQt> – Дата доступа: 15.10.2023

[20] [Электронный ресурс] – Режим доступа: <https://codeby.net/threads/poluchaem-informaciju-o-sisteme-s-pomoschju-python-chast-1.79797/> – Дата доступа: 15.10.2023

[21] [Электронный ресурс] – Режим доступа: <https://www.nix.ru/computer_hardware_news/hardware_news_viewer.html?id=211599> – Дата доступа: 22.11.2023

[22] [Электронный ресурс] – Режим доступа: <https://ru.stackoverflow.com/questions/88615/python> – Дата доступа: 10.10.2023

[23] [Электронный ресурс] – Режим доступа: https://psk68.ru/files/metod/uchebnik\_Informatika/arhit.html – Дата доступа: 10.10.2023

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг программного кода**

**CPU.py**

class CPU:

def \_\_init\_\_(self):

self.cpu\_name = 'N/A'

self.cpu\_percent = 0

self.cpu\_frequencies = []

self.num\_active\_cores = 0

self.temperature = 0

self.power = 0

def get\_name(self):

self.cpu\_name = platform.processor()

return self.cpu\_name

def get\_percent(self):

self.cpu\_percent = psutil.cpu\_percent(interval=1)

return self.cpu\_percent

def get\_cpu\_frequency(self):

try:

w = wmi.WMI(namespace="root/OpenHardwareMonitor")

sensors = w.Sensor()

self.cpu\_frequencies = []

for core\_num in range(1, self.get\_num\_active\_cores() + 1):

core\_name = f'CPU Core #{core\_num}'

for sensor in sensors:

if sensor.SensorType == 'Clock' and sensor.Name == core\_name:

self.cpu\_frequencies.append(sensor.Value)

except Exception as error:

print(f'Error: {error}')

def get\_frequency(self):

self.get\_cpu\_frequency()

if self.cpu\_frequencies:

total\_frequency = sum(self.cpu\_frequencies)

average\_frequency = total\_frequency / len(self.cpu\_frequencies)

return f'{average\_frequency:.2f} MHz'

else:

return 'Информация не доступна'

def get\_num\_active\_cores(self):

self.num\_active\_cores = multiprocessing.cpu\_count()

if self.num\_active\_cores:

return self.num\_active\_cores

else:

return 'Информация не доступна'

@staticmethod

def get\_cpu\_temperature():

try:

w = wmi.WMI(namespace="root/OpenHardwareMonitor")

sensors = w.Sensor()

for sensor in sensors:

if sensor.SensorType == 'Temperature' and sensor.Name == 'CPU Package':

return sensor.Value

except Exception as error:

print(f'Error: {error}')

def get\_temperature(self):

self.temperature = self.get\_cpu\_temperature()

if self.temperature is not None:

return f'{self.temperature}°C'

else:

return 'Информация не доступна'

@staticmethod

def get\_cpu\_power():

try:

w = wmi.WMI(namespace="root/OpenHardwareMonitor")

sensors = w.Sensor()

for sensor in sensors:

if sensor.SensorType == 'Power' and sensor.Name == 'CPU Package':

return sensor.Value

except Exception as error:

print(f'Error: {error}')

def get\_power(self):

self.power = self.get\_cpu\_power()

if self.power is not None:

return f'{self.power:.2f} W'

else:

return 'Информация не доступна'

**CPU\_QT.py**

class CPUMonitorApp(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

# Инициализируем CPU

self.cpu = CPU()

# Инициализируем таймер для обновления данных

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_data)

# Инициализируем интерфейс пользователя

self.init\_ui()

def init\_ui(self):

layout = QVBoxLayout()

# Стилизуем виджеты

self.setStyleSheet("""

background-color: #1E1E1E; /\* Темный фон \*/

color: white; /\* Белый цвет текста \*/

font-size: 16px; /\* Размер шрифта \*/

""")

# Инициализируем метки

self.labels = {

'Название процессора': self.create\_label(''),

'Загрузка процессора': self.create\_label(''),

'Тактовая частота процессора': self.create\_label(''),

'Количество активных ядер': self.create\_label(''),

'Температура': self.create\_label(''),

'Напряжение': self.create\_label('')

}

# Инициализируем флажки для выбора отображаемых меток

self.checkbox\_labels = {

'Название процессора': QCheckBox('Название процессора', checked=True),

'Загрузка процессора': QCheckBox('Загрузка процессора', checked=True),

'Тактовая частота процессора': QCheckBox('Тактовая частота процессора', checked=True),

'Количество активных ядер': QCheckBox('Количество активных ядер', checked=True),

'Температура': QCheckBox('Температура', checked=True),

'Напряжение': QCheckBox('Напряжение', checked=True)

}

for label\_text, checkbox in self.checkbox\_labels.items():

checkbox.stateChanged.connect(self.update\_visibility)

layout.addWidget(checkbox)

# Инициализируем выпадающий список для выбора времени обновления

self.refresh\_time\_label = QLabel('Время обновления (секунды):')

self.refresh\_time\_combobox = QComboBox()

self.refresh\_time\_combobox.addItems(['1', '2', '5', '10', '30'])

self.refresh\_time\_combobox.setCurrentIndex(0)

self.refresh\_time\_combobox.currentIndexChanged.connect(self.update\_refresh\_time)

layout.addWidget(self.refresh\_time\_label)

layout.addWidget(self.refresh\_time\_combobox)

# Добавляем метки в макет

for label\_text, label\_widget in self.labels.items():

layout.addWidget(label\_widget)

label\_widget.setStyleSheet("""

border: 2px solid white; /\* Белая рамка \*/

padding: 5px;

""")

self.setLayout(layout)

self.setWindowTitle('CPUMonitorApp')

self.setGeometry(100, 100, 400, 400)

# Запускаем таймер с интервалом в 1 секунду

self.timer.start(1000)

def create\_label(self, text):

label = QLabel(text)

label.setStyleSheet('font-size: 14px;')

return label

def update\_data(self):

# Обновляем данные каждую секунду

for label\_text, checkbox in self.checkbox\_labels.items():

if checkbox.isChecked():

if label\_text == 'Название процессора':

self.labels[label\_text].setText(f'{label\_text}: {self.cpu.get\_name()}')

elif label\_text == 'Загрузка процессора':

self.labels[label\_text].setText(f'{label\_text}: {self.cpu.get\_percent()}%')

elif label\_text == 'Тактовая частота процессора':

self.labels[label\_text].setText(f'{label\_text}: {self.cpu.get\_frequency()}')

elif label\_text == 'Количество активных ядер':

self.labels[label\_text].setText(f'{label\_text}: {self.cpu.get\_num\_active\_cores()}')

elif label\_text == 'Температура':

self.labels[label\_text].setText(f'{label\_text}: {self.cpu.get\_temperature()}')

elif label\_text == 'Напряжение':

self.labels[label\_text].setText(f'{label\_text}: {self.cpu.get\_power()}')

else:

self.labels[label\_text].clear() # Очищаем текст лейбла

self.labels[label\_text].show() # Показываем лейбл, если флаг выбран

def update\_visibility(self):

# Обновляем видимость меток в соответствии с состоянием флажков

self.update\_data()

def update\_refresh\_time(self):

# Обновляем время обновления

refresh\_time = int(self.refresh\_time\_combobox.currentText())

self.timer.setInterval(refresh\_time \* 1000) # Переводим секунды в миллисекунды

self.timer.start()

**GPU.py**

class GPU:

def \_\_init\_\_(self):

self.gpu\_name = 'N/A'

self.gpu\_load = 'N/A'

self.gpu\_memory\_total = 'N/A'

self.gpu\_memory\_used = 'N/A'

self.gpu\_memory\_free = 'N/A'

self.clock = 'N/A'

self.temperature = 0

self.power = 0

self.gpu = None

self.get\_gpu()

def get\_gpu(self):

self.gpu = GPUtil.getGPUs()[0]

return self.gpu

def get\_name(self):

self.gpu\_name = self.gpu.name

return self.gpu\_name

def get\_load(self):

self.gpu\_load = self.gpu.load \* 100

return self.gpu\_load

def get\_memory\_total(self):

self.gpu\_memory\_total = f'{self.gpu.memoryTotal / 1024:.2f} GB'

return self.gpu\_memory\_total

def get\_memory\_used(self):

self.gpu\_memory\_used = f'{self.gpu.memoryUsed / 1024:.2f} GB'

return self.gpu\_memory\_used

def get\_memory\_free(self):

self.gpu\_memory\_free = f'{self.gpu.memoryFree / 1024:.2f} GB'

return self.gpu\_memory\_free

@staticmethod

def get\_clock\_info():

try:

w = wmi.WMI(namespace="root/OpenHardwareMonitor")

sensors = w.Sensor()

for sensor in sensors:

if sensor.SensorType == 'Clock' and sensor.Name == 'GPU Core':

return sensor.Value

except Exception as error:

print(f'Error: {error}')

def get\_clock(self):

self.clock = self.get\_clock\_info()

if self.clock is not None:

return f'{self.clock:.2f}'

else:

return 'Информация не доступна'

@staticmethod

def get\_gpu\_temperature():

try:

w = wmi.WMI(namespace="root/OpenHardwareMonitor")

sensors = w.Sensor()

for sensor in sensors:

if sensor.SensorType == 'Temperature' and sensor.Name == 'GPU Core':

return sensor.Value

except Exception as error:

print(f'Error: {error}')

def get\_temperature(self):

self.temperature = self.get\_gpu\_temperature()

if self.temperature is not None:

return f'{self.temperature}°C'

else:

return 'Информация не доступна'

@staticmethod

def get\_gpu\_power():

try:

w = wmi.WMI(namespace="root/OpenHardwareMonitor")

sensors = w.Sensor()

for sensor in sensors:

if sensor.SensorType == 'Power' and sensor.Name == 'GPU Power':

return sensor.Value

except Exception as error:

print(f'Error: {error}')

def get\_power(self):

self.power = self.get\_gpu\_power()

if self.power is not None:

return f'{self.power:.2f} W'

else:

return 'Информация не доступна'

**GPU\_QT.py**

class GPUMonitorApp(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.gpu = GPU()

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_data)

self.init\_ui()

def init\_ui(self):

layout = QVBoxLayout()

self.setStyleSheet("""

background-color: #1E1E1E;

color: white;

font-size: 16px;

""")

self.labels = {

'Название видеокарты': self.create\_label(''),

'Загрузка видеокарты': self.create\_label(''),

'Тактовая частота видеокарты': self.create\_label(''),

'Температура': self.create\_label(''),

'Напряжение': self.create\_label(''),

'Использовано памяти': self.create\_label(''),

'Свободно памяти': self.create\_label(''),

}

self.checkbox\_labels = {

'Название видеокарты': QCheckBox('Название видеокарты', checked=True),

'Загрузка видеокарты': QCheckBox('Загрузка видеокарты', checked=True),

'Тактовая частота видеокарты': QCheckBox('Тактовая частота видеокарты', checked=True),

'Температура': QCheckBox('Температура', checked=True),

'Напряжение': QCheckBox('Напряжение', checked=True),

'Использовано памяти': QCheckBox('Использовано памяти', checked=True),

'Свободно памяти': QCheckBox('Свободно памяти', checked=True),

}

for label\_text, checkbox in self.checkbox\_labels.items():

checkbox.stateChanged.connect(self.update\_visibility)

layout.addWidget(checkbox)

self.refresh\_time\_label = QLabel('Время обновления (секунды):')

self.refresh\_time\_combobox = QComboBox()

self.refresh\_time\_combobox.addItems(['1', '2', '5', '10', '30'])

self.refresh\_time\_combobox.setCurrentIndex(0)

self.refresh\_time\_combobox.currentIndexChanged.connect(self.update\_refresh\_time)

layout.addWidget(self.refresh\_time\_label)

layout.addWidget(self.refresh\_time\_combobox)

for label\_text, label\_widget in self.labels.items():

layout.addWidget(label\_widget)

label\_widget.setStyleSheet("""

border: 2px solid white;

padding: 5px;

""")

self.setLayout(layout)

self.setWindowTitle('GPUMonitorApp')

self.setGeometry(100, 100, 400, 300)

self.timer.start(1000)

def create\_label(self, text):

label = QLabel(text)

label.setStyleSheet('font-size: 14px;')

return label

def update\_data(self):

for label\_text, checkbox in self.checkbox\_labels.items():

if checkbox.isChecked():

if label\_text == 'Название видеокарты':

self.labels[label\_text].setText(f'{label\_text}: {self.gpu.get\_name()}')

elif label\_text == 'Загрузка видеокарты':

self.labels[label\_text].setText(f'{label\_text}: {self.gpu.get\_load()}%')

elif label\_text == 'Тактовая частота видеокарты':

self.labels[label\_text].setText(f'{label\_text}: {self.gpu.get\_clock()}')

elif label\_text == 'Температура':

self.labels[label\_text].setText(f'{label\_text}: {self.gpu.get\_temperature()}')

elif label\_text == 'Напряжение':

self.labels[label\_text].setText(f'{label\_text}: {self.gpu.get\_power()}')

elif label\_text == 'Использовано памяти':

self.labels[label\_text].setText(f'{label\_text}: {self.gpu.get\_memory\_used()}')

elif label\_text == 'Свободно памяти':

self.labels[label\_text].setText(f'{label\_text}: {self.gpu.get\_memory\_free()}')

else:

self.labels[label\_text].clear()

self.labels[label\_text].show()

def update\_visibility(self):

self.update\_data()

def update\_refresh\_time(self):

refresh\_time = int(self.refresh\_time\_combobox.currentText())

self.timer.setInterval(refresh\_time \* 1000)

self.timer.start()

**RAM.py**

class RAM:

def \_\_init\_\_(self):

self.memory = None

self.memory\_percent = 0

self.available\_memory = 0

self.total\_memory = 0

self.get\_memory()

def update\_data(self):

self.get\_memory()

def get\_memory(self):

self.memory = psutil.virtual\_memory()

return self.memory

def get\_memory\_percent(self):

self.memory\_percent = self.memory.percent

return self.memory\_percent

def get\_available\_memory(self):

self.available\_memory = f'{self.memory.available / (1024 \*\* 2):.2f} MB'

return self.available\_memory

def get\_total\_memory(self):

self.total\_memory = f'{self.memory.total / (1024 \*\* 2):.2f} MB'

return self.total\_memory

**RAM\_QT.py**

class RAMMonitorApp(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.ram = RAM()

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_data)

self.init\_ui()

def init\_ui(self):

layout = QVBoxLayout()

self.setStyleSheet("""

background-color: #1E1E1E;

color: white;

font-size: 16px;

""")

self.labels = {

'Загрузка оперативной памяти': self.create\_label(''),

'Доступно': self.create\_label(''),

'Общее': self.create\_label(''),

}

self.checkbox\_labels = {

'Загрузка оперативной памяти': QCheckBox('Загрузка оперативной памяти', checked=True),

'Доступно': QCheckBox('Доступно', checked=True),

'Общее': QCheckBox('Общее', checked=True),

}

for label\_text, checkbox in self.checkbox\_labels.items():

checkbox.stateChanged.connect(self.update\_visibility)

layout.addWidget(checkbox)

self.refresh\_time\_label = QLabel('Время обновления (секунды):')

self.refresh\_time\_combobox = QComboBox()

self.refresh\_time\_combobox.addItems(['1', '2', '5', '10', '30'])

self.refresh\_time\_combobox.setCurrentIndex(0)

self.refresh\_time\_combobox.currentIndexChanged.connect(self.update\_refresh\_time)

layout.addWidget(self.refresh\_time\_label)

layout.addWidget(self.refresh\_time\_combobox)

for label\_text, label\_widget in self.labels.items():

layout.addWidget(label\_widget)

label\_widget.setStyleSheet("""

border: 2px solid white;

padding: 5px;

""")

self.setLayout(layout)

self.setWindowTitle('RAMMonitorApp')

self.setGeometry(100, 100, 400, 300)

self.timer.start(1000)

def create\_label(self, text):

label = QLabel(text)

label.setStyleSheet('font-size: 14px;')

return label

def update\_data(self):

self.ram.update\_data()

for label\_text, checkbox in self.checkbox\_labels.items():

if checkbox.isChecked():

if label\_text == 'Загрузка оперативной памяти':

self.labels[label\_text].setText(f'Загрузка оперативной памяти: {self.ram.get\_memory\_percent()}%')

elif label\_text == 'Доступно':

self.labels[label\_text].setText(f'Доступно: {self.ram.get\_available\_memory()}')

elif label\_text == 'Общее':

self.labels[label\_text].setText(f'Общее: {self.ram.get\_total\_memory()}')

else:

self.labels[label\_text].clear()

self.labels[label\_text].show()

def update\_visibility(self):

self.update\_data()

def update\_refresh\_time(self):

refresh\_time = int(self.refresh\_time\_combobox.currentText())

self.timer.setInterval(refresh\_time \* 1000)

self.timer.start()

**SSD.py**

class SSD:

def \_\_init\_\_(self, partition):

self.partition = partition

self.disk\_usage = None

self.disk\_percent = 0

self.available\_space = 'N/A'

self.total\_space = 'N/A'

self.get\_disk()

def get\_disk(self):

if 'Windows' in self.partition.fstype:

return

self.disk\_usage = psutil.disk\_usage(self.partition.mountpoint)

def get\_name(self):

return self.partition.device

def get\_percent(self):

self.disk\_percent = self.disk\_usage.percent

return self.disk\_percent

def get\_available\_space(self):

self.available\_space = f'{self.disk\_usage.free / (1024 \*\* 3):.2f} GB'

return self.available\_space

def get\_total\_space(self):

self.total\_space = f'{self.disk\_usage.total / (1024 \*\* 3):.2f} GB'

return self.total\_space

**SSD\_QT.py**

class SSDMonitorApp(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.tab\_widget = QTabWidget(self)

self.tabs = []

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_data)

self.init\_ui()

def init\_ui(self):

self.setStyleSheet("""

background-color: #1E1E1E;

color: white;

font-size: 16px;

""")

self.tab\_widget.setStyleSheet("""

QTabWidget::pane {

border: 2px solid white;

background-color: #1E1E1E;

}

QTabBar::tab {

background-color: #2E2E2E;

color: white;

border: 2px solid #1E1E1E;

padding: 8px;

}

QTabBar::tab:selected {

background-color: #1E1E1E;

}

""")

main\_layout = QVBoxLayout(self)

# Добавляем флажки для выбора отображаемых меток

self.checkbox\_labels = {

'Диск': QCheckBox('Диск', checked=True),

'Загрузка диска': QCheckBox('Загрузка диска', checked=True),

'Доступно': QCheckBox('Доступно', checked=True),

'Общее': QCheckBox('Общее', checked=True),

}

for checkbox in self.checkbox\_labels.values():

checkbox.stateChanged.connect(self.update\_visibility)

main\_layout.addWidget(checkbox)

# Добавляем выпадающий список для выбора времени обновления

self.refresh\_time\_label = QLabel('Время обновления (секунды):')

self.refresh\_time\_combobox = QComboBox()

self.refresh\_time\_combobox.addItems(['1', '2', '5', '10', '30'])

self.refresh\_time\_combobox.setCurrentIndex(0)

self.refresh\_time\_combobox.currentIndexChanged.connect(self.update\_refresh\_time)

main\_layout.addWidget(self.refresh\_time\_label)

main\_layout.addWidget(self.refresh\_time\_combobox)

self.setLayout(main\_layout)

partitions = psutil.disk\_partitions()

for partition in partitions:

disk\_info = SSD(partition)

tab = QWidget()

layout = QVBoxLayout(tab)

labels = {

'Диск': self.create\_label(''),

'Загрузка диска': self.create\_label(''),

'Доступно': self.create\_label(''),

'Общее': self.create\_label(''),

}

for label\_text, label\_widget in labels.items():

layout.addWidget(label\_widget)

label\_widget.setStyleSheet("""

border: 2px solid white;

padding: 5px;

""")

tab.setLayout(layout)

self.tabs.append((tab, labels))

self.tab\_widget.addTab(tab, f'Диск {disk\_info.get\_name()}')

main\_layout.addWidget(self.tab\_widget)

self.setWindowTitle('SSDMonitorApp')

self.setGeometry(100, 100, 400, 300)

self.timer.start(1000)

def create\_label(self, text):

label = QLabel(text)

label.setStyleSheet('font-size: 14px;')

return label

def update\_data(self):

for tab, labels in self.tabs:

partition = psutil.disk\_partitions()[self.tabs.index((tab, labels))]

disk\_info = SSD(partition)

for label\_text, checkbox in self.checkbox\_labels.items():

if checkbox.isChecked():

if label\_text == 'Диск':

labels[label\_text].setText(f'Диск: {disk\_info.get\_name()}')

elif label\_text == 'Загрузка диска':

labels[label\_text].setText(f'Загрузка диска: {disk\_info.get\_percent()}%')

elif label\_text == 'Доступно':

labels[label\_text].setText(f'Доступно: {disk\_info.get\_available\_space()}')

elif label\_text == 'Общее':

labels[label\_text].setText(f'Общее: {disk\_info.get\_total\_space()}')

else:

labels[label\_text].clear()

labels[label\_text].show()

def update\_visibility(self):

self.update\_data()

def update\_refresh\_time(self):

refresh\_time = int(self.refresh\_time\_combobox.currentText())

self.timer.setInterval(refresh\_time \* 1000)

self.timer.start()

**battery.py**

class Battery:

def \_\_init\_\_(self):

self.battery = None

self.percent = 0

self.power\_plugged = ''

self.time\_left = ''

self.get\_battery()

def update\_data(self):

self.get\_battery()

self.get\_time\_left()

def get\_battery(self):

try:

self.battery = psutil.sensors\_battery()

return self.battery

except Exception as error:

print(f'Отсутствует батарея: {error}')

def get\_percent(self):

if self.battery is not None:

self.percent = self.battery.percent

return self.percent

else:

return 'Нет информации о батарее'

def get\_power\_plugged(self):

if self.battery is not None:

self.power\_plugged = self.battery.power\_plugged

return self.power\_plugged

else:

return 'Нет информации о батарее'

def get\_time\_left(self):

power = self.get\_power\_plugged()

if self.battery is not None:

if not power:

time\_left = timedelta(seconds=self.battery.secsleft) if self.battery.secsleft is not None else None

if time\_left is not None:

hours, remainder = divmod(time\_left.seconds, 3600)

minutes, seconds = divmod(remainder, 60)

self.time\_left = f'{int(hours)}:{int(minutes)}:{int(seconds)}'

return self.time\_left

else:

return 'Недоступно'

else:

return 'Устройство подключено к сети'

else:

return 'Нет информации о батарее'

**battery\_QT.py**

class BatteryMonitorApp(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.battery = Battery()

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_data)

self.init\_ui()

def init\_ui(self):

self.setStyleSheet('''

QWidget { background-color: #1E1E1E; color: white; font-size: 16px; }

QLabel { border: 2px solid white; padding: 5px; }

QProgressBar {

border: 2px solid white;

padding: 5px;

text-align: center;

}

QProgressBar::chunk {

background-color: #1E90FF;

}

''')

main\_layout = QVBoxLayout(self)

# Добавляем флажки для выбора отображаемых меток

self.checkbox\_labels = {

'Уровень заряда': QCheckBox('Уровень заряда', checked=True),

'Время до разрядки': QCheckBox('Время до разрядки', checked=True),

'Подключено к сети': QCheckBox('Подключено к сети', checked=True),

}

for checkbox in self.checkbox\_labels.values():

checkbox.stateChanged.connect(self.update\_visibility)

main\_layout.addWidget(checkbox)

# Добавляем выпадающий список для выбора времени обновления

self.refresh\_time\_label = QLabel('Время обновления (секунды):')

self.refresh\_time\_combobox = QComboBox()

self.refresh\_time\_combobox.addItems(['1', '2', '5', '10', '30'])

self.refresh\_time\_combobox.setCurrentIndex(0)

self.refresh\_time\_combobox.currentIndexChanged.connect(self.update\_refresh\_time)

main\_layout.addWidget(self.refresh\_time\_label)

main\_layout.addWidget(self.refresh\_time\_combobox)

# Добавляем виджеты монитора батареи

self.label\_percent = QLabel(f'Уровень заряда батареи: {self.battery.get\_percent()}%')

main\_layout.addWidget(self.label\_percent)

self.progress\_bar = QProgressBar(self)

self.progress\_bar.setValue(self.battery.get\_percent())

self.progress\_bar.setAlignment(Qt.AlignCenter)

main\_layout.addWidget(self.progress\_bar)

self.label\_power\_plugged = QLabel(f'Подключено к сети: {"Да" if self.battery.get\_power\_plugged() else "Нет"}')

main\_layout.addWidget(self.label\_power\_plugged)

self.label\_time\_left = QLabel(f'Прогноз времени до разрядки: {self.battery.get\_time\_left()}')

main\_layout.addWidget(self.label\_time\_left)

self.setLayout(main\_layout)

self.setWindowTitle('BatteryMonitorApp')

self.setGeometry(100, 100, 400, 300)

self.timer.start(1000)

def update\_data(self):

self.battery.update\_data()

for label\_text, checkbox in self.checkbox\_labels.items():

if checkbox.isChecked():

if label\_text == 'Уровень заряда':

self.label\_percent.setText(f'Уровень заряда батареи: {self.battery.get\_percent()}%')

self.progress\_bar.setValue(self.battery.get\_percent())

elif label\_text == 'Время до разрядки':

self.label\_time\_left.setText(f'Прогноз времени до разрядки: {self.battery.get\_time\_left()}')

elif label\_text == 'Подключено к сети':

if self.battery.get\_power\_plugged():

self.label\_power\_plugged.setText('Подключено к сети: Да')

else:

self.label\_power\_plugged.setText('Подключено к сети: Нет')

self.label\_power\_plugged.show()

else:

if label\_text == 'Уровень заряда':

self.label\_percent.clear()

self.label\_percent.show()

self.progress\_bar.setValue(0)

elif label\_text == 'Время до разрядки':

self.label\_time\_left.clear()

self.label\_time\_left.show()

elif label\_text == 'Подключено к сети':

self.label\_power\_plugged.clear()

def update\_visibility(self):

self.update\_data()

def update\_refresh\_time(self):

refresh\_time = int(self.refresh\_time\_combobox.currentText())

self.timer.setInterval(refresh\_time \* 1000)

self.timer.start()

**MN\_QT.py**

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.init\_ui()

def init\_ui(self):

# Создаем QTabWidget для управления вкладками

tab\_widget = QTabWidget(self)

# Создаем вкладки для каждого из окон

cpu\_tab = QWidget()

gpu\_tab = QWidget()

ram\_tab = QWidget()

ssd\_tab = QWidget()

battery\_tab = QWidget()

# Добавляем окна мониторинга на соответствующие вкладки

cpu\_layout = QVBoxLayout(cpu\_tab)

cpu\_layout.addWidget(CPUMonitorApp())

cpu\_tab.setLayout(cpu\_layout)

gpu\_layout = QVBoxLayout(gpu\_tab)

gpu\_layout.addWidget(GPUMonitorApp())

gpu\_tab.setLayout(gpu\_layout)

ram\_layout = QVBoxLayout(ram\_tab)

ram\_layout.addWidget(RAMMonitorApp())

ram\_tab.setLayout(ram\_layout)

ssd\_layout = QVBoxLayout(ssd\_tab)

ssd\_layout.addWidget(SSDMonitorApp())

ssd\_tab.setLayout(ssd\_layout)

battery\_layout = QVBoxLayout(battery\_tab)

battery\_layout.addWidget(BatteryMonitorApp())

battery\_tab.setLayout(battery\_layout)

# Добавляем вкладки в QTabWidget

tab\_widget.addTab(cpu\_tab, 'CPU')

tab\_widget.addTab(gpu\_tab, 'GPU')

tab\_widget.addTab(ram\_tab, 'RAM')

tab\_widget.addTab(ssd\_tab, 'SSD')

tab\_widget.addTab(battery\_tab, 'Battery')

self.setCentralWidget(tab\_widget)

# Устанавливаем стили для MainWindow

self.setStyleSheet('''

QMainWindow {

background-color: #1E1E1E; /\* Темный фон \*/

color: white; /\* Белый цвет текста \*/

}

QTabWidget::pane {

border: 2px solid white; /\* Белая рамка для вкладок \*/

}

QTabWidget::tab-bar {

alignment: center; /\* Выравнивание вкладок по центру \*/

}

QTabBar::tab {

background-color: #2F2F2F; /\* Цвет фона вкладок \*/

color: white; /\* Белый цвет текста вкладок \*/

min-width: 100px; /\* Минимальная ширина вкладок \*/

padding: 5px;

}

QTabBar::tab:selected {

background-color: #1E90FF; /\* Цвет выделенной вкладки \*/

}

''')

self.setWindowTitle('Hardware Monitor')

self.setGeometry(100, 100, 800, 600)

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

main\_window = MainWindow()

main\_window.show()

sys.exit(app.exec\_())

# **ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Функциональная схема алгоритма, реализующего программное средство**

# **ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Блок схема алгоритма, реализующего программное средство**

# **ПРИЛОЖЕНИЕ Г**

**(обязательное)**

**Графический интерфейс пользователя**

# **ПРИЛОЖЕНИЕ Д**

**(обязательное)**

**Ведомость курсового проекта**