Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи

бакалавра

на тему: «Розробка демонстраційного програмного забезпечення, використання асинхронного введення в ОС Windows»

за освітньою програмою: «12 Інженерія програмного забезпечення»

зі спеціальності: «121 Інженерія програмного забезпечення»

Виконав: студент групи ПЗ2011: \_\_\_\_ /Cергій КУЛИК/

Керівник: \_\_\_\_ / Вадим АНДРЮЩЕНКО/

Нормоконтролер: \_\_\_\_ /Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів

без відповідних посилань.

Студент

Дніпро – 2024 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note

to Bachelor’s Thesis

on the topic: «Development of demonstration software, utilizing asynchronous input in the Windows operating system»

according to educational curriculum «12 Software engineering»

in the Speciality: «121 Software engineering»

Done by the student of the group PZ2011: /Kulyk Serhii/

Scientific Supervisor: \_\_\_\_ /Vadym ANDRYUSCHENKO/

Normative controller: \_\_\_\_ /Svitlana VOLKOVA/

Dnipro – 2024

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»

Кафедра: «Комп'ютерні інформаційні технології»

Рівень вищої освіти: бакалавр

Освітня програма: «12 Інженерія програмного забезпечення»

Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри КІТ

/Вадим ГОРЯЧКІН/

Дата

**ЗАВДАННЯ**

на кваліфікаційну роботу Бакалавр

студенту Кулику Сергію Вадимовичу

1. Тема роботи: «Розробка демонстраційного програмного забезпечення, використання асинхронного введення в ОС Windows»  
   Керівник роботи: Андрющенко Вадим Олександрович  
   затверджені наказом № 1773 ст від 27.12.2023
2. Строк подання студентом роботи: 24.06.2024
3. Вихідні дані до роботи: вихідний код програми, додаток, документація .
4. Зміст пояснювальної записки (перелік питань до розробки):
   1. Вступ;
   2. Збір та аналіз вимог до програмного забезпечення;
   3. Проектування системи;
   4. Тестування та налагодження;
   5. Висновки;
   6. Література;
5. Перелік демонстраційного матеріалу:
   1. Презентація;
   2. Демонстраційне відео.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів кваліфікаційної роботи | Строк виконання  етапів роботи | Примітка |
| 1 | Огляд предметної області та аналіз вимог | 25.04.2024 – 30.04.2024 |  |
| 2 | Проектування архітектури та вибір технологій | 01.05.2024 – 06.05.2023 |  |
| 3 | Розробка базової функціональності | 06.05.2024 – 12.05.2024 | 30% |
| 4 | Реалізація асинхронного збору даних | 12.05.2023 – 14.05.2024 |  |
| 5 | Розробка інтерфейсу користувача | 14.05.2024 – 19.05.2024 |  |
| 6 | Інтеграція та тестування компонентів | 20.05.2024 – 26.05.2024 | 60% |
| 7 | Доопрацювання та оптимізація | 27.05.2024 – 30.05.2024 |  |
| 8 | Підготовка документації та презентації | 01.06.2024 – 10.04.2023 |  |
| 9 | Подання кваліфікаційної роботи до кафедри | 10.06.2024 – 16.06.2024 | 100% |
| 10 | Захист кваліфікаційної роботи на засіданні  Екзаменаційної комісії | 24.06.2024 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | Сергій КУЛИК |

(підпис) (Ім’я ПРІЗВИЩЕ)

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник роботи |  |  | Вадим АНДРЮЩЕНКО |

(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавру: 83 сторінок, 8 рисунків, 2 додатки, 5 джерел

Тема: Опис розробки демонстраційного програмного забезпечення використання асинхронного введення-виведення в ОС Windows

Об'єкт розробки: демонстраційне програмне забезпечення для моніторингу системних показників комп'ютера в ОС Windows.

Мета роботи - розробка демонстраційного програмного забезпечення для моніторингу системних показників комп'ютера в ОС Windows з використанням асинхронного введення-виведення, що забезпечує ефективний збір даних, їх візуалізацію в режимі реального часу та надає додаткові функції, такі як ведення логів та налаштування оповіщень.

Методи дослідження: аналіз літератури та існуючих рішень, проектування архітектури програмного забезпечення, реалізація з використанням мови C# та .NET Framework, тестування та оцінка ефективності.

Одержані результати:

* Розроблено демонстраційне програмне забезпечення для моніторингу системних показників комп'ютера в ОС Windows.
* Реалізовано асинхронний збір даних про системні показники з використанням класу PerformanceCounter та Task Parallel Library.
* Розроблено інтерфейс користувача для відображення даних у режимі реального часу з використанням графіків, діаграм та числових значень.
* Реалізовано функції ведення логів та налаштування оповіщень для контролю стану системи та своєчасного реагування на проблеми.
* Проведено тестування та оцінку ефективності розробленого програмного забезпечення, що підтвердило його працездатність та відповідність поставленим вимогам.

Ключові слова: асинхронне введення-виведення, моніторинг системних показників, Windows, C#, .NET Framework, PerformanceCounter, Task Parallel Library.

**Зміст**

[ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ 9](#_Toc169256533)

[ТЕОРЕТИЧНІ ОСНОВИ АСИНХРОННОГО ВВЕДЕННЯ-ВИВЕДЕННЯ ТА СИСТЕМНОГО МОНІТОРИНГУ 12](#_Toc169256534)

[1.1 Принципи асинхронного введення-виведення в ОС Windows 12](#_Toc169256535)

[1.2 Переваги та недоліки використання асинхронного введення-виведення. 14](#_Toc169256536)

[1.3 Основні системні показники комп’ютера та їх значення 16](#_Toc169256537)

[1.4 Огляд існуючих рішень для моніторингу системних показників 18](#_Toc169256538)

[ПРОЕК ТУВАННЯ ДЕМОНСТРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 24](#_Toc169256539)

[2.1 Вибір технологій та інструментів розробки 24](#_Toc169256540)

[2.2 Архітектура програмного забезпечення 26](#_Toc169256541)

[2.3 Опис функціональних вимог 27](#_Toc169256542)

[2.4 Проектування інтерфейсу користувача 28](#_Toc169256543)

[РЕАЛІЗАЦІЯ ДЕМОНСТРРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 31](#_Toc169256544)

[3.1 Організація асинхронного збору данних про системні показники 31](#_Toc169256545)

[3.2 Візуалізація даних в режимі реального часу 32](#_Toc169256546)

[3.3 Реалізація додаткових функцій(наприклад, ведення логів, налаштування оповіщень) 34](#_Toc169256547)

[ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ 36](#_Toc169256548)

[4.1 Методи тестування програмного забезпечення 36](#_Toc169256549)

[ВИСНОВКИ 39](#_Toc169256550)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 40](#_Toc169256551)

[ДОДАТОК А 41](#_Toc169256552)

[ДОДАТОК Б 43](#_Toc169256553)

# ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

GUI (Graphical User Interface) – графічний інтерфейс користувача.

I/O (Input/Output) – введення/виведення.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

OS(ОС) ­– операційна система (Operating System).

RAM (Random Access Memory) – оперативна пам’ять

SSD (Solid State Drive) – твердотільний накопичувач.

UI (User Interface) – інтерфейс користувача.

WinAPI (Windows Application Programming Interface) – інтерфейс програмування додатків Windows.

WPF – Windows Presentation Foundation.

ВСТУП

Сучасні комп'ютерні системи характеризуються високою складністю та різноманітністю апаратних та програмних компонентів. Для забезпечення їх ефективної та стабільної роботи важливим є моніторинг системних показників, таких як завантаження процесора, використання пам'яті, активність диска, температура та інші. Це дозволяє виявляти потенційні проблеми, оптимізувати налаштування та забезпечувати високу продуктивність системи.

Одним з ключових аспектів розробки програмного забезпечення для моніторингу системних показників є організація ефективного збору та обробки даних. Враховуючи особливості операційної системи Windows, використання асинхронного введення-виведення (АВВ) є оптимальним рішенням для забезпечення чуйності інтерфейсу та ефективного використання ресурсів системи.

Актуальність теми дослідження зумовлена зростаючою потребою в надійних та ефективних інструментах моніторингу системних показників, які дозволяють забезпечити стабільну роботу комп'ютерних систем та своєчасно реагувати на можливі проблеми.

Мета роботи полягає в розробці демонстраційного програмного забезпечення для моніторингу системних показників в ОС Windows з використанням асинхронного введення-виведення.

Для досягнення поставленої мети необхідно вирішити такі завдання:

* проаналізувати теоретичні основи асинхронного введення-виведення в ОС Windows та особливості моніторингу системних показників;
* дослідити існуючі рішення для моніторингу системних показників та виявити їх переваги та недоліки;
* спроектувати архітектуру та функціональність демонстраційного програмного забезпечення;
* реалізувати програмне забезпечення з використанням обраних технологій та інструментів розробки;
* провести тестування та оцінку ефективності розробленого програмного забезпечення;

**Об'єкт дослідження:** процес розробки програмного забезпечення для моніторингу системних показників в ОС Windows.

**Предмет дослідження:** методи та технології організації асинхронного введення-виведення для збору даних про системні показники та їх візуалізації в режимі реального часу.

**Методи дослідження:** аналіз літератури, проектування, програмування, тестування, аналіз результатів.

**Практичне значення отриманих результатів:** розроблене демонстраційне програмне забезпечення може бути використане для моніторингу системних показників комп'ютера, виявлення потенційних проблем та оптимізації налаштувань системи.

# ТЕОРЕТИЧНІ ОСНОВИ АСИНХРОННОГО ВВЕДЕННЯ-ВИВЕДЕННЯ ТА СИСТЕМНОГО МОНІТОРИНГУ

**1.1 Принципи асинхронного введення-виведення в ОС Windows**

Асинхронне введення-виведення (I/O) є важливою концепцією в сучасних операційних системах, включаючи Windows. Це метод обробки вводу і виводу, який дозволяє програмі продовжувати виконання без блокування під час очікування завершення операцій з вводу-виводу. У Windows цей процес забезпечується різними механізмами та функціями, які допомагають досягти ефективного і безперебійного виконання програм.

**1.1.1 Основні принципи асинхронного введення-виведення**

**Неблокуючий режим**: асинхронні операції вводу-виводу виконуються в неблокуючому режимі, що означає, що виклик функції для виконання операції не блокує виконання програми. Наприклад, під час читання з файлу, програма може продовжувати виконання інших задач, поки операція читання не завершиться.

**Події та зворотні виклики (Callbacks)**: асинхронне введення-виведення в Windows часто використовує механізм подій та зворотних викликів. Коли операція завершується, викликається функція зворотного виклику або встановлюється відповідна подія, яка повідомляє програму про завершення операції.

**Оверлаппінг (Overlapped) I/O**: windows підтримує оверлаппінг I/O, де вводяться поняття оверлаппінг структур (OVERLAPPED structure). Ця структура містить інформацію про стан асинхронної операції, таку як позиція у файлі та ідентифікатори подій. Операції I/O, які використовують цю структуру, можуть бути виконані паралельно.

**I/O Completion Ports**: це потужний механізм для обробки асинхронного введення-виведення в високонавантажених серверах. Використовуючи порти завершення I/O, програма може обробляти велику кількість асинхронних операцій ефективно. Операційна система автоматично розподіляє завершені операції між робочими потоками.

**Task-Based Asynchronous Pattern (TAP)**: з появою .NET Framework, асинхронні операції стали ще більш зручними завдяки використанню задач (Tasks). TAP дозволяє використовувати ключові слова async та await для написання асинхронного коду, що робить його більш читабельним та підтримуваним.

**1.1.2 Реалізація асинхронного введення-виведення в Windows**

* **WinAPI**: використання низькорівневих функцій WinAPI, таких як ***ReadFileEx****,* ***WriteFileEx***, які забезпечують асинхронне виконання завдяки передачі оверлаппінг структур та зворотних викликів;
* **.NET Framework**: в .NET Framework асинхронне введення-виведення реалізується за допомогою методів, що повертають Task, наприклад, ***Stream.ReadAsync****,* ***Stream.WriteAsync****.* Це дозволяє використовувати сучасні підходи до написання асинхронного коду;
* **C++/WinRT**: В середовищі розробки C++/WinRT використання асинхронних операцій реалізується за допомогою функцій та бібліотек, що підтримують асинхронні операції на основі об'єктно-орієнтованих принципів.

**1.1.3 Переваги асинхронного введення-виведення**

* **підвищення продуктивності**: асинхронне введення-виведення дозволяє ефективно використовувати ресурси процесора, зменшуючи час простою і збільшуючи загальну продуктивність системи;
* **покращення відгуку користувача**: завдяки асинхронним операціям, користувацький інтерфейс залишається чутливим і реагує на дії користувача, навіть коли виконуються тривалі операції вводу-виводу;
* **масштабованість**: механізми, такі як I/O Completion Ports, дозволяють обробляти велику кількість одночасних з'єднань, що є критично важливим для серверних додатків.

В цілому, асинхронне введення-виведення в ОС Windows є потужним інструментом для розробників, який дозволяє створювати високопродуктивні і масштабовані програми, здатні ефективно обробляти великі обсяги даних та підтримувати високу чутливість користувацького інтерфейсу.

**1.2 Переваги та недоліки використання асинхронного введення-виведення.**

Асинхронне введення-виведення (I/O) є потужним інструментом для покращення продуктивності та відгуку програм. Проте, як і будь-яка технологія, воно має свої переваги і недоліки. Розглянемо їх детальніше.

**1.2.1 Переваги використання асинхронного введення-виведення**

**1.2.1.1 Підвищення продуктивності**:

* асинхронне I/O дозволяє програмам виконувати інші задачі, поки очікують завершення операцій вводу-виводу. Це означає, що процесор не простоює і може виконувати більше корисної роботи, що підвищує загальну продуктивність системи;
* завдяки неблокуючим операціям, програми можуть обробляти більше запитів одночасно, що є критично важливим для серверів і високонавантажених систем.

**1.2.1.2 Покращення відгуку користувача**:

* у програмах з графічним інтерфейсом користувача (GUI) асинхронні операції дозволяють уникнути зависання інтерфейсу під час виконання тривалих операцій, таких як читання з мережі або диска. Це забезпечує кращий досвід для користувача;
* введення асинхронних операцій дозволяє реалізовувати фонові задачі без впливу на основний потік виконання програми, що робить інтерфейс більш чутливим і швидким.

**1.2.1.3 Масштабованість**:

* механізми, такі як порти завершення I/O (I/O Completion Ports), дозволяють ефективно обробляти величезну кількість одночасних підключень. Це робить асинхронне I/O незамінним для створення масштабованих мережевих серверів і служб;
* асинхронні операції забезпечують кращу масштабованість системи, оскільки дозволяють уникати створення великої кількості потоків, що знижує витрати на контекстне перемикання і споживання пам'яті.

**1.2.1.4 Ефективне використання ресурсів**:

* асинхронне I/O дозволяє краще використовувати ресурси комп'ютера, такі як процесорний час і пам'ять, оскільки блочні операції вводу-виводу можуть бути виконані паралельно з іншими завданнями.

**1.2.2. Недоліки використання асинхронного введення-виведення**

**1.2.2.1 Складність програмування**:

* написання і підтримка асинхронного коду може бути складнішою порівняно з синхронним кодом. Програмісту потрібно враховувати можливість виникнення гонок, станів гонок і інших проблем, пов'язаних з багатопоточністю;
* дебагінг асинхронних програм є більш складним процесом, оскільки помилки можуть бути важчими для відтворення і виявлення.

**1.2.2.2 Витрати на контекстне перемикання**:

* хоча асинхронне I/O зменшує кількість блокуючих операцій, це може збільшити кількість контекстних перемикань, особливо при неправильній реалізації. Часте переключення між задачами може знизити ефективність.

**1.2.2.3 Складність управління ресурсами**:

* асинхронні операції вимагають ретельного управління ресурсами, такими як пам'ять і дескриптори. Неправильне управління може призвести до витоків ресурсів або збоїв в роботі програми;
* зростання кількості асинхронних операцій може ускладнити відслідковування їх стану і управління ними.

**1.2.2.4 Необхідність використання спеціальних інструментів і бібліотек**:

* використання асинхронного введення-виведення може вимагати застосування додаткових інструментів і бібліотек, що збільшує складність розробки і залежність від сторонніх компонентів;
* вибір неправильних інструментів або бібліотек може призвести до поганої продуктивності або труднощів у підтримці коду.

Асинхронне введення-виведення в ОС Windows є потужним інструментом, що забезпечує значні переваги, такі як підвищення продуктивності, покращення відгуку користувача і масштабованість. Однак його використання також пов'язане з певними недоліками, включаючи складність програмування, необхідність ретельного управління ресурсами та можливі витрати на контекстне перемикання. Розуміння цих аспектів допомагає розробникам робити обґрунтований вибір щодо використання асинхронного введення-виведення в своїх програмах.

**1.3 Основні системні показники комп’ютера та їх значення**

Системні показники комп'ютера – це кількісні характеристики, що відображають поточний стан та продуктивність різних компонентів системи. Моніторинг цих показників дозволяє оцінити ефективність роботи комп'ютера, виявити потенційні проблеми та оптимізувати його налаштування.

**1.3.1 Основні системні показники:**

* **завантаження процесора (CPU Usage):** Показує, наскільки завантажений центральний процесор обчисленнями. Високе завантаження може свідчити про те, що процесор не справляється з поточним навантаженням або є програми, які споживають забагато ресурсів;
* **використання оперативної пам'яті (RAM Usage):** відображає, скільки оперативної пам'яті використовується програмами та операційною системою. Якщо пам'ять майже повністю заповнена, це може призвести до зниження продуктивності та "підгальмовування" системи;
* **завантаження диска (Disk Usage):** показує інтенсивність читання та запису даних на жорсткий диск або SSD. Високе завантаження диска може свідчити про активну роботу програм з файлами або фрагментацію диска;
* **мережева активність (Network Activity):** відображає обсяг даних, що передаються та приймаються через мережеві інтерфейси. Висока мережева активність може бути пов'язана з завантаженням файлів, онлайн-іграми або роботою програм у фоновому режимі;
* **температура компонентів (Temperature):** показує температуру процесора, відеокарти, жорсткого диска та інших компонентів. Перевищення допустимих температур може призвести до нестабільної роботи або пошкодження компонентів;
* **швидкість обертання вентиляторів (Fan Speed):** відображає швидкість обертання вентиляторів системи охолодження. Висока швидкість обертання може свідчити про перегрів компонентів або неправильну роботу системи охолодження;
* **напруга живлення (Voltage):** показує напругу, що подається на різні компоненти системи. Відхилення від норми можуть призвести до нестабільної роботи або пошкодження компонентів.

**1.3.2 Значення моніторингу системних показників:**

* **виявлення проблем:** моніторинг дозволяє виявити потенційні проблеми, такі як перегрів, недостатність ресурсів або неправильна робота компонентів, до того, як вони призведуть до серйозних наслідків;
* **оптимізація продуктивності:** аналіз системних показників допомагає виявити "вузькі місця" у продуктивності та вжити заходів для їх усунення, наприклад, збільшити обсяг оперативної пам'яті, замінити повільний жорсткий диск на швидший SSD або оптимізувати налаштування програм;
* **контроль стану системи:** регулярний моніторинг дозволяє відстежувати зміни у системних показниках та вчасно реагувати на них, що допомагає підтримувати стабільну та ефективну роботу комп'ютера.

Моніторинг системних показників є важливим інструментом для забезпечення стабільної та ефективної роботи комп'ютера. Він дозволяє виявляти проблеми, оптимізувати продуктивність та контролювати стан системи, що допомагає продовжити термін служби компонентів та забезпечити комфортну роботу користувача.

**1.4 Огляд існуючих рішень для моніторингу системних показників**

Існує безліч програмних рішень для моніторингу системних показників комп'ютера, як вбудованих в операційну систему, так і сторонніх. Вони відрізняються за функціональністю, зручністю використання, деталізацією інформації та іншими параметрами. Розглянемо деякі з найпопулярніших.

**1.4.1 Вбудовані засоби Windows:**

* **диспетчер завдань (Task Manager):** базовий інструмент для моніторингу завантаження процесора, пам'яті, диска та мережі. Надає інформацію про запущені процеси та їх споживання ресурсів;
* **монітор продуктивності (Performance Monitor):** більш просунутий інструмент, що дозволяє відстежувати широкий спектр показників у реальному часі та будувати графіки. Має можливість налаштування та збору логів;
* **монітор ресурсів (Resource Monitor):** надає детальну інформацію про використання ресурсів процесором, пам'яттю, диском та мережею окремими процесами та службами.

**1.4.2 Сторонні програми:**

* **HWMonitor:** проста та безкоштовна утиліта, що відображає основні показники температури, напруги та швидкості вентиляторів;

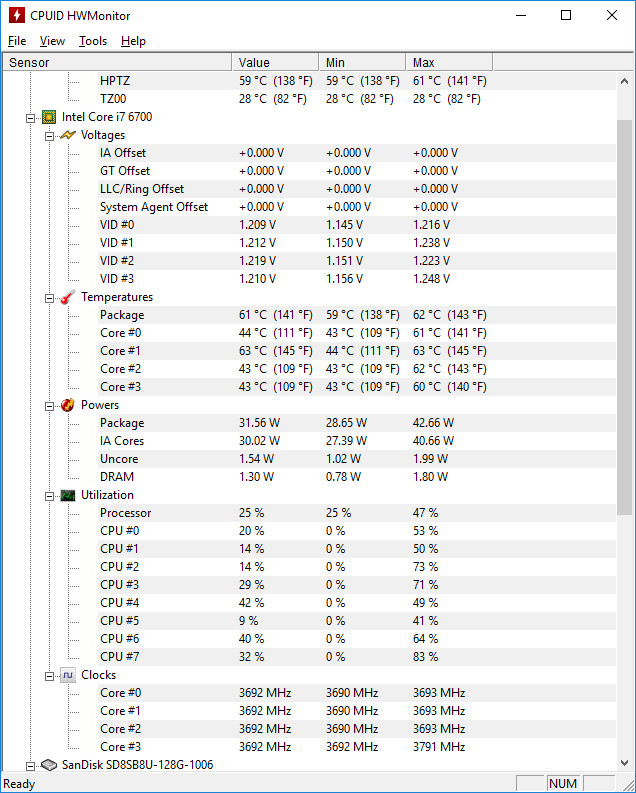


Рисунок 1.4.2.1 - Інтерфейс HWMonitor

* **Open Hardware Monitor:** безкоштовна програма з відкритим вихідним кодом, схожа на HWMonitor, але з більшою кількістю функцій та можливостей налаштування;

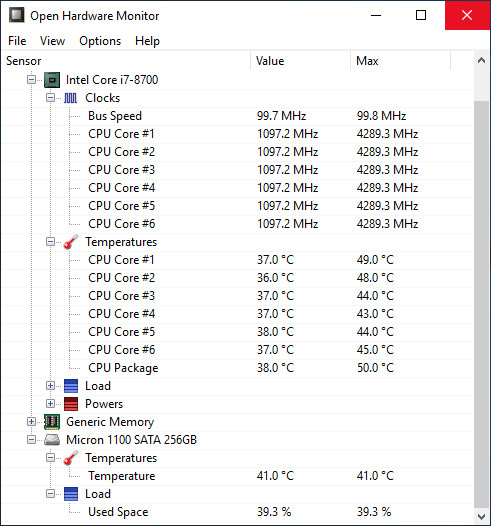


Рисунок 1.4.2.2 - Інтерфейс Open Hardware Monitor

* **HWiNFO:** потужний інструмент для детального аналізу апаратного забезпечення та моніторингу його стану. Надає величезну кількість інформації про всі компоненти системи;

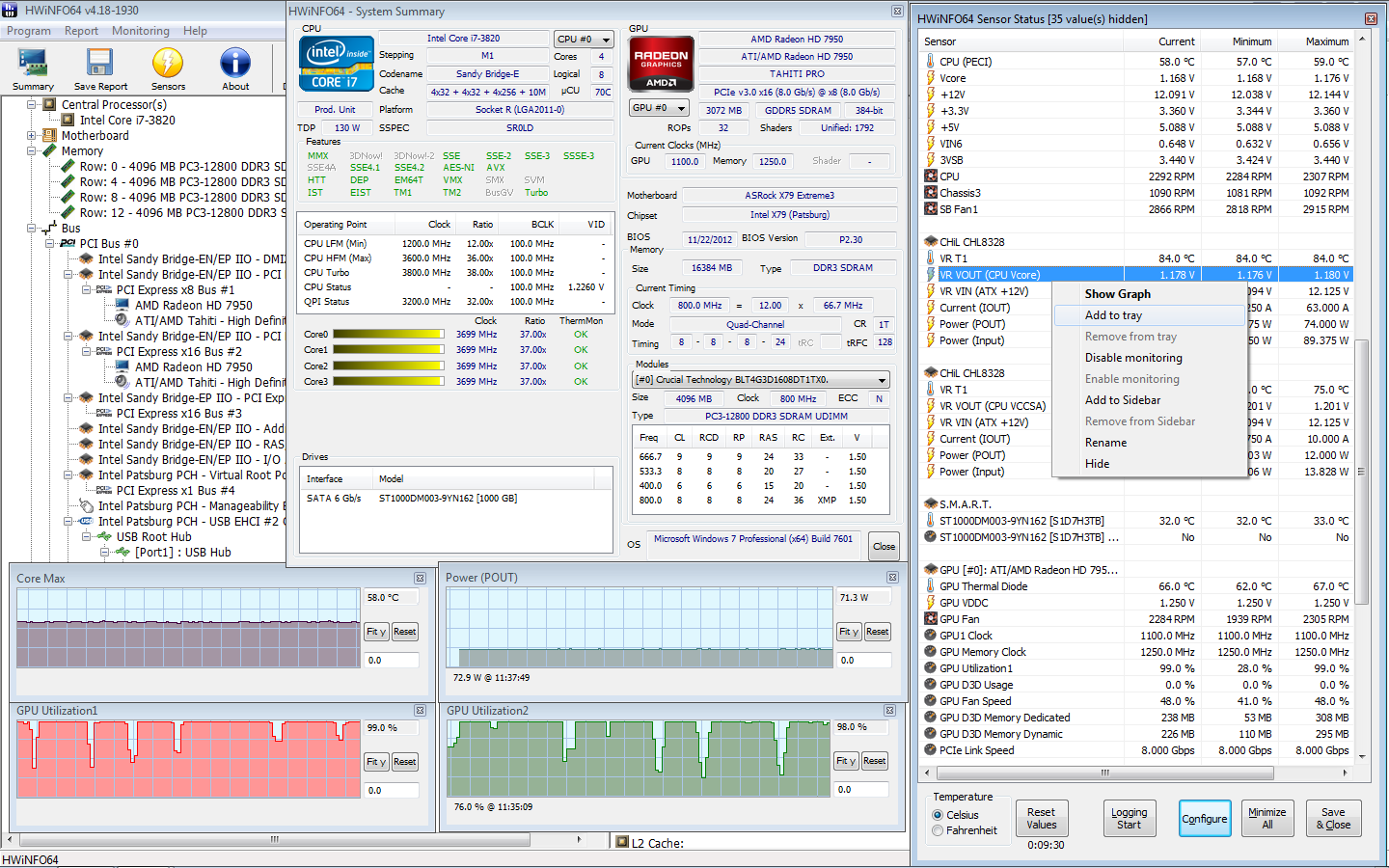


Рисунок 1.4.2.3 - Інтерфейс HWiNFO

* **MSI Afterburner:** популярна утиліта для розгону та моніторингу відеокарт. Містить вбудований оверлей для відображення показників під час гри;
* **MSI Afterburner:** Популярна утиліта для розгону та моніторингу відеокарт. Містить вбудований оверлей для відображення показників під час гри;

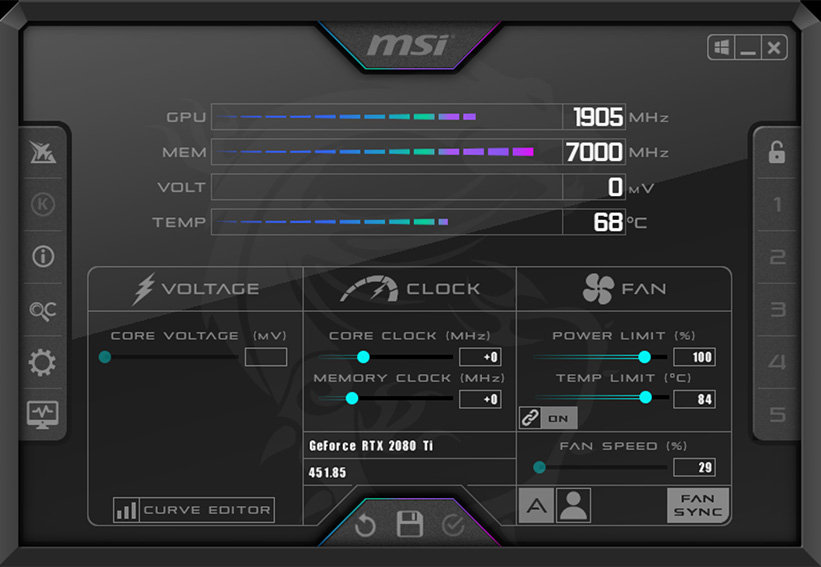


Рисунок 1.4.2.4 - Інтерфейс MSI Afterburner

* **NZXT CAM:** програмне забезпечення для управління компонентами NZXT та моніторингу системних показників. Має зручний інтерфейс та широкі можливості налаштування;

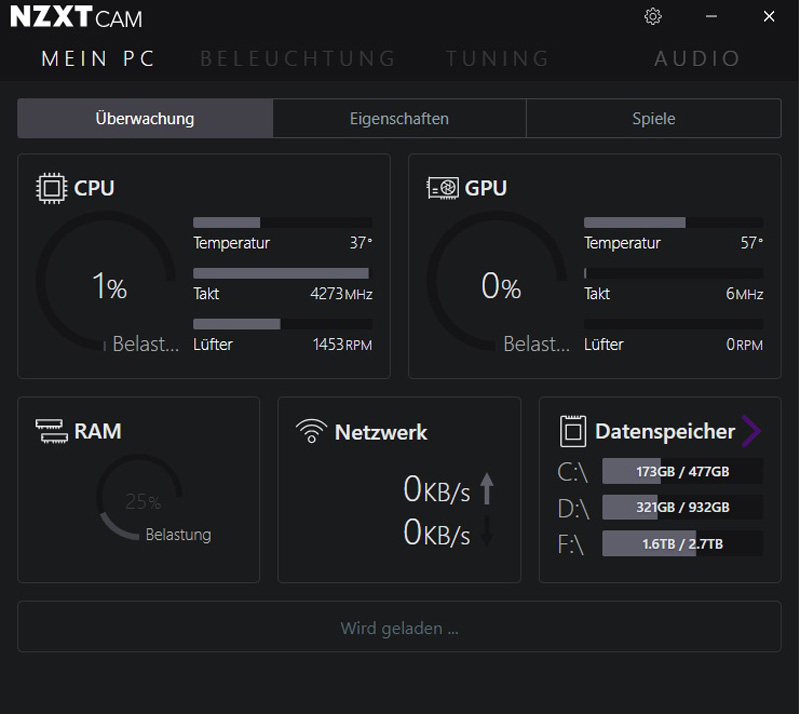


Рисунок 1.4.2.5 - Інтерфейс NZXT Cam

* **AIDA64:** комплексний інструмент для діагностики та тестування апаратного забезпечення. Містить модуль моніторингу з широким набором показників.

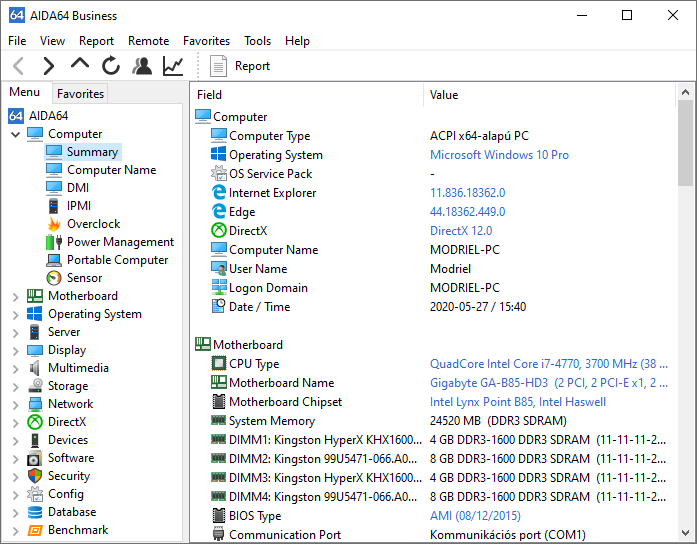


Рисунок 1.4.2.6 - Інтерфейс AIDA64

**1.4.3 Вибір рішення:**

При виборі рішення для моніторингу системних показників варто враховувати такі фактори:

* **потрібна функціональність:** визначити, які саме показники цікавлять та які функції потрібні (наприклад, ведення логів, оверлей, оповіщення);
* **зручність використання:** виберіть програму з зрозумілим та зручним інтерфейсом, який відповідає вашим потребам та рівню знань.
* **вартість:** багато програм для моніторингу є безкоштовними або мають безкоштовні версії з обмеженою функціональністю. Визначте свій бюджет та виберіть відповідне рішення.

ПРОЕК ТУВАННЯ ДЕМОНСТРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**2.1 Вибір технологій та інструментів розробки**

Вибір технологій та інструментів для розробки демонстраційного програмного забезпечення моніторингу системних показників є критичним етапом, що впливає на успіх проекту. Враховуючи специфіку завдання, доцільно обрати такі технології та інструменти:

**2.1.1 Мова програмування:**

* **C#:** об'єктно-орієнтована мова з широкими можливостями, що добре підходить для розробки додатків під Windows. Вона має потужну бібліотеку класів .NET Framework, яка надає засоби для роботи з асинхронним введенням-виведенням, системними показниками та графічним інтерфейсом користувача.

**2.1.2 Фреймворк:**

* **.NET Framework (або .NET Core):** надає широкий спектр бібліотек та інструментів для розробки додатків під Windows. Містить готові класи для роботи з асинхронним введенням-виведенням, доступу до системної інформації та створення графічного інтерфейсу користувача.

**2.1.3 Бібліотеки та компоненти:**

* **PerformanceCounter:** клас з .NET Framework, що дозволяє отримувати доступ до системних лічильників продуктивності Windows та збирати дані про різні показники системи;
* **Task Parallel Library (TPL):** бібліотека для спрощення роботи з асинхронним та паралельним програмуванням. Дозволяє запускати завдання у фоновому режимі та ефективно обробляти результати;
* **Windows Forms або WPF:** фреймворки для створення графічного інтерфейсу користувача. Windows Forms більш простий у використанні, але WPF надає більше можливостей для створення сучасних та гнучких інтерфейсів.

**2.1.4 Інструменти розробки:**

* **Visual Studio:** потужне інтегроване середовище розробки (IDE) від Microsoft, що надає широкий спектр інструментів для розробки, налагодження та тестування додатків на C# та .NET;
* **NuGet:** менеджер пакетів для .NET, що дозволяє легко підключати сторонні бібліотеки та компоненти до проекту.

**2.1.5 Обґрунтування вибору:**

* **C# та .NET:** ці технології є стандартними для розробки додатків під Windows, мають велику спільноту розробників та безліч доступних ресурсів;
* **PerformanceCounter:** надає простий та зручний спосіб отримання даних про системні показники;
* **TPL:** дозволяє легко реалізувати асинхронне збирання даних та оновлення інтерфейсу користувача;
* **Windows Forms або WPF:** обидва фреймворки добре підходять для створення інтерфейсу програми моніторингу. Вибір між ними залежить від особистих уподобань та вимог до дизайну інтерфейсу;
* **Visual Studio:** потужний та зручний інструмент, що значно спрощує процес розробки та налагодження;
* **NuGet:** дозволяє швидко підключати необхідні бібліотеки та компоненти, що економить час та зусилля розробника.

**2.2 Архітектура програмного забезпечення**

Архітектура демонстраційного програмного забезпечення для моніторингу системних показників буде базуватися на багатошаровій моделі з використанням компонентів, що забезпечують розділення відповідальності та гнучкість системи.

**2.2.1 Основні шари та компоненти:**

**2.2.1.1 Шар збору даних:**

* **модуль зчитування системних показників:** використовує клас PerformanceCounter для отримання даних з системних лічильників Windows;
* **модуль асинхронного збору даних:** забезпечує асинхронне зчитування показників у фоновому режимі, використовуючи можливості Task Parallel Library;
* **кеш даних:** зберігає останні отримані значення показників для швидкого доступу та відображення.

**2.2.1.2 Шар обробки даних:**

* **модуль аналізу даних:** виконує обробку отриманих даних, фільтрацію, агрегацію та розрахунок додаткових показників;
* **модуль оповіщень:** відповідає за генерацію оповіщень при перевищенні заданих порогів або виникненні інших подій.

**2.2.1.3 Шар представлення даних (інтерфейс користувача):**

* **головне вікно:** відображає поточні значення системних показників у зручному для користувача вигляді (графіки, діаграми, числові значення);
* **вікно налаштувань:** дозволяє користувачу налаштовувати параметри моніторингу, вибирати показники для відображення, встановлювати пороги для оповіщень;
* **вікно логів:** записує історію змін системних показників та події, що відбуваються.

**2.2.1.4 Шар взаємодії з користувачем:**

* **модуль обробки подій:** обробляє дії користувача, такі як зміна налаштувань, перемикання між вікнами, запит на оновлення даних.

## **2.3 Опис функціональних вимог**

Функціональні вимоги визначають, що саме має робити демонстраційне програмне забезпечення для моніторингу системних показників. Вони описують основні функції та можливості, які повинні бути реалізовані в програмі.

**2.3.1 Основні функціональні вимоги:**

**2.3.1.1 Збір даних:**

* збір даних про основні системні показники, такі як завантаження процесора, використання оперативної пам'яті, завантаження диска, мережева активність, температура компонентів, швидкість вентиляторів та напруга живлення;
* асинхронний збір даних у фоновому режимі для забезпечення чуйності інтерфейсу програми;
* можливість вибору користувачем показників для моніторингу.

**2.3.1.2 Відображення даних:**

* відображення поточних значень системних показників у зручному для користувача вигляді (цифри, графіки, діаграми);

**2.3.1.3 Оповіщення:**

* налаштування порогів для кожного показника, при перевищенні яких буде згенеровано оповіщення;
* різні типи оповіщень: візуальні (наприклад, спливаюче вікно), звукові;
* можливість налаштування параметрів оповіщень (звук, текст повідомлення).

**2.3.1.4 Логування:**

* ведення логів змін системних показників та подій (наприклад, перевищення порогів, помилки);
* можливість перегляду та фільтрації логів;
* можливість експорту логів у різні формати (наприклад, csv, txt).

**2.3.1.5 Налаштування:**

* можливість налаштування зовнішнього вигляду інтерфейсу програми (тема);
* можливість вибору одиниць вимірювання для показників.

**2.3.1.6 Інші функції:**

* можливість запуску програми разом із системою.

**2.3.2 Додаткові функціональні вимоги:**

* можливість відображення інформації про апаратне забезпечення (назва процесора, обсяг оперативної пам'яті, модель відеокарти тощо);
* можливість тестування продуктивності компонентів (наприклад, стрес-тест процесора або відеокарти).

## **2.4 Проектування інтерфейсу користувача**

Інтерфейс користувача (UI) демонстраційного програмного забезпечення для моніторингу системних показників повинен бути інтуїтивно зрозумілим, зручним та інформативним. Він повинен дозволяти користувачеві легко отримувати доступ до необхідної інформації, налаштовувати параметри моніторингу та взаємодіяти з програмою.

**2.4.1 Основні елементи інтерфейсу:**

**2.4.1.1 Головне вікно:**

область відображення даних:

* + графіки, діаграми або числові значення для відображення поточних значень системних показників.
  + можливість вибору типу відображення даних для кожного показника;
  + можливість масштабування та прокручування графіків;
  + відображення історії змін показників за певний період часу.

панель інструментів;

* + кнопки для керування програмою (старт/стоп моніторингу, оновлення даних, відкриття налаштувань, перегляд логів);

статусний рядок:

* + Відображення поточних значень деяких ключових показників (наприклад, завантаження процесора, температури);
  + Повідомлення про події та помилки.

**2.4.1.2 Вікно налаштувань:**

Вкладки для різних категорій налаштувань:

* + загальні налаштування (автозапуск, відображення в треї);
  + налаштування моніторингу (вибір показників);
  + налаштування оповіщень (пороги, типи оповіщень, параметри);
  + налаштування логування (шлях до файлу логів).

Елементи управління:

* + прапорці, перемикачі, списки, поля введення для зміни налаштувань;
  + кнопки для збереження та скасування змін.

**2.4.1.3 Вікно логів:**

Таблиця з записами логів:

* + колонки для дати/часу, типу події, опису події;
  + можливість сортування та фільтрації записів.

Панель інструментів:

* + кнопки для очищення логів, експорту логів, оновлення списку.

**2.4.2 Принципи проектування інтерфейсу:**

* **інтуїтивність:** інтерфейс повинен бути зрозумілим та легким у використанні навіть для недосвідчених користувачів;
* **зручність:** елементи управління повинні бути розташовані логічно та зручно для доступу;
* **інформативність:** інтерфейс повинен надавати користувачеві всю необхідну інформацію у зрозумілому вигляді;
* **естетичність:** інтерфейс повинен бути приємним для ока та відповідати загальному стилю операційної системи.

**2.4.2.1 Вибір технології:**

Для створення інтерфейсу можна використовувати Windows Forms або WPF. Windows Forms є більш простим варіантом, але WPF надає більше можливостей для створення сучасних та гнучких інтерфейсів.

РЕАЛІЗАЦІЯ ДЕМОНСТРРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**3.1 Організація асинхронного збору данних про системні показники**

Для забезпечення плавної роботи програми та уникнення блокування інтерфейсу користувача під час збору даних, необхідно організувати асинхронне зчитування системних показників. Це дозволить програмі продовжувати реагувати на дії користувача та оновлювати інтерфейс, поки дані збираються у фоновому режимі.

**3.1.1 Реалізація асинхронного збору даних**

**3.1.1.1 Створення фонового завдання:**

* За допомогою інструментів асинхронного програмування (наприклад, Task Parallel Library в C#) створюється окреме завдання, яке буде відповідальне за збір даних про системні показники.
* Це завдання буде виконуватися паралельно з основним потоком програми, не блокуючи його.

**3.1.1.2 Періодичне зчитування показників:**

* усередині фонового завдання організовується цикл, який буде періодично зчитувати значення системних показників;
* для зчитування показників використовуються відповідні системні api або бібліотеки (наприклад, performancecounter в .net).

**3.1.1.3 Передача даних в основний потік:**

* отримані значення показників необхідно передати в основний потік програми для оновлення інтерфейсу користувача;
* для цього можна використовувати механізми синхронізації потоків, такі як черги, події або делегати.

**3.1.1.4 Оновлення інтерфейсу користувача:**

* в основному потоці програми отримані дані використовуються для оновлення відповідних елементів інтерфейсу (графіків, діаграм, числових значень);
* оновлення інтерфейсу відбувається асинхронно, щоб не блокувати роботу програми.

**3.1.1.5 Обробка винятків:**

* у фоновому завданні необхідно передбачити обробку можливих винятків, таких як помилки доступу до системних даних або інші системні помилки;
* у разі виникнення винятку, його слід зареєструвати в логах та, за необхідності, повідомити користувача про проблему.

**3.1.2 Переваги асинхронного збору даних**

Асинхронний збір даних пропонує ряд переваг, які значно покращують користувацький досвід та ефективність роботи програми. По-перше, він забезпечує чуйність інтерфейсу, дозволяючи користувачеві взаємодіяти з програмою, навіть під час активного збору даних у фоновому режимі. Це усуває неприємні затримки та "зависання", роблячи роботу з програмою більш комфортною. По-друге, асинхронність сприяє ефективнішому використанню системних ресурсів. Оскільки збір даних відбувається в окремому потоці, основний потік програми не блокується, що дозволяє йому виконувати інші завдання та забезпечує оптимальне використання процесора та пам'яті. Крім того, асинхронний підхід забезпечує масштабованість програми, дозволяючи легко розширювати її функціональність та збільшувати обсяги оброблюваних даних без суттєвого впливу на продуктивність.

**3.2 Візуалізація даних в режимі реального часу**

Візуалізація даних у режимі реального часу є ключовим компонентом програми моніторингу системних показників. Вона дозволяє користувачеві отримувати миттєвий зворотний зв'язок про стан системи та відстежувати зміни показників у динаміці.

**3.2.1 Основні принципи візуалізації**

**3.2.1.1 Наочність та інформативність:**

* використання графічних елементів (графіки, діаграми, індикатори) для наочного представлення даних;
* вибір відповідного типу графіка для кожного показника (наприклад, лінійний графік для завантаження процесора, кругова діаграма для використання диска);
* чітке маркування осей та легенд для легкого розуміння даних;
* використання кольорів для виділення важливих значень або перевищення порогів.

**3.2.1.2 Динамічне оновлення:**

* оновлення графіків та інших елементів інтерфейсу в міру отримання нових даних від фонового завдання;
* плавне оновлення графіків для забезпечення візуальної безперервності;

**3.2.1.3 Ефективність:**

* оптимізація алгоритмів візуалізації для забезпечення плавної роботи програми навіть при великій кількості даних;
* використання апаратного прискорення графіки (якщо можливо) для підвищення продуктивності.

**3.2.2 Реалізація візуалізації**

Для візуалізації даних можна використовувати вбудовані компоненти графічного інтерфейсу (наприклад, Chart в Windows Forms або WPF) або сторонні бібліотеки (наприклад, OxyPlot, LiveCharts). Вибір конкретної технології залежить від вимог до функціональності та дизайну інтерфейсу.

**3.2.2.1 Приклад візуалізації:**

* **завантаження процесора:** лінійний графік, що відображає зміну завантаження процесора в часі. Вісь X - час, вісь Y - завантаження у відсотках;

**використання дискового простору:** кругова діаграма, що показує розподіл використання простору на накопичувачі;

* **температура компонентів:** виведення на екран цифрових показників температури компонентів.

Візуалізація даних в режимі реального часу є важливим компонентом програми моніторингу системних показників. Вона дозволяє користувачеві швидко та ефективно оцінювати стан системи та приймати рішення на основі отриманої інформації.

**3.3 Реалізація додаткових функцій(наприклад, ведення логів, налаштування оповіщень)**

Розробка додаткових функцій, таких як ведення логів та налаштування оповіщень, є важливим елементом для забезпечення повноцінного функціонування програмного забезпечення для моніторингу системних показників. Ці функції дозволяють зберігати історичні дані для подальшого аналізу та діагностики, а також забезпечувати своєчасне інформування користувачів про критичні зміни в стані системи

**3.3.1 Ведення логів**

Ведення логів є невід'ємною частиною моніторингового програмного забезпечення, оскільки дозволяє зберігати історичні дані про системні показники. Це важливо для проведення ретроспективного аналізу, діагностики проблем та підготовки звітів. Логи можуть бути збережені у різних форматах, таких як JSON або XML, що забезпечує зручність їхнього зчитування та обробки.

Використання бази даних SQLite для зберігання логів дозволяє ефективно обробляти великі обсяги даних. Цей підхід забезпечує структуроване зберігання та швидкий доступ до історичних даних. Для уникнення перевантаження системи, запис даних у лог-файли або базу даних повинен здійснюватися асинхронно. Це дозволяє мінімізувати вплив на продуктивність основних процесів програми.

Механізм ротації логів є критично важливим для запобігання переповненню дискового простору. Зазвичай, ротація логів реалізується шляхом зберігання логів лише за певний період (наприклад, за останні 30 днів) або обмеження максимального розміру лог-файлів.

Користувацький інтерфейс повинен включати можливість перегляду логів у зручному форматі, наприклад, у вигляді таблиць або графіків. Крім того, необхідно забезпечити можливість фільтрації та пошуку логів за різними параметрами, такими як дата, тип показника тощо.

**3.3.2 Налаштування оповіщень**

Налаштування оповіщень є важливою функцією, яка дозволяє інформувати користувачів про критичні зміни в системних показниках у режимі реального часу. Це дозволяє своєчасно реагувати на потенційні проблеми та забезпечувати стабільну роботу системи.

Користувачі повинні мати можливість встановлювати порогові значення для кожного системного показника. Наприклад, можна задати оповіщення про перевищення завантаження процесора понад 90% або підвищення температури процесора вище 80°C. При досягненні цих порогових значень програма повинна автоматично відправляти оповіщення.

Для запобігання надмірному надсиланню оповіщень важливо реалізувати механізм запобігання повторним оповіщенням протягом певного часу після першого спрацьовування. Наприклад, оповіщення може надсилатися один раз кожні 5 хвилин при збереженні критичних показників.

Інтерфейс користувача повинен включати розділ для налаштування порогових значень, типів оповіщень. Також необхідно забезпечити відображення історії оповіщень, щоб користувачі могли переглядати всі спрацьовані оповіщення та їхні деталі для подальшого аналізу та діагностики.

Додаткові функції, такі як ведення логів та налаштування оповіщень, значно підвищують корисність та функціональність програмного забезпечення для моніторингу системних показників. Вони забезпечують збереження історичних даних для ретроспективного аналізу та своєчасне інформування користувачів про критичні зміни в системі. Асинхронна реалізація цих функцій дозволяє мінімізувати вплив на продуктивність системи, забезпечуючи при цьому високу надійність та зручність використання.

ТЕСТУВАННЯ ПРОГРАМИ

**4.1 Методи тестування програмного забезпечення**

Для забезпечення якості та надійності демонстраційного програмного забезпечення для моніторингу системних показників необхідно провести ретельне тестування. Застосування різних методів тестування дозволить виявити та виправити потенційні помилки та недоліки на ранніх етапах розробки.

**4.1.1 Модульне тестування (Unit Testing):**

* Перевірка коректності роботи окремих модулів та компонентів програми ізольовано один від одного.
* Використання спеціальних фреймворків для модульного тестування (наприклад, NUnit, xUnit).
* Створення тестів, що перевіряють різні сценарії роботи модуля та обробку винятків.

**4.1.2 Інтеграційне тестування (Integration Testing):**

* Перевірка взаємодії між різними модулями та компонентами програми.
* Виявлення проблем, пов'язаних з передачею даних, синхронізацією та сумісністю інтерфейсів.
* Тестування роботи програми з різними конфігураціями апаратного забезпечення та операційної системи.

**4.1.3 Системне тестування (System Testing):**

* Перевірка роботи програми як єдиного цілого в умовах, максимально наближених до реального використання.
* Тестування всіх функцій та можливостей програми, включаючи збір даних, візуалізацію, оповіщення та логування.
* Виявлення проблем з продуктивністю, стабільністю та зручністю використання.

**4.1.4 Тестування продуктивності (Performance Testing):**

* Оцінка швидкодії та ефективності програми під різним навантаженням.
* Вимірювання часу відгуку, споживання ресурсів (процесор, пам'ять, диск) та інших показників продуктивності.
* Виявлення "вузьких місць" та оптимізація роботи програми.

**4.1.5 Стрес-тестування (Stress Testing):**

* Перевірка роботи програми в екстремальних умовах (наприклад, при максимальному завантаженні системи або великій кількості одночасних запитів).
* Виявлення проблем зі стабільністю та відмовостійкістю.
* Оцінка максимальної продуктивності програми.

**4.1.6. Тестування зручності використання (Usability Testing):**

* Оцінка зручності та інтуїтивності інтерфейсу програми.
* Залучення потенційних користувачів для тестування та збору зворотного зв'язку.
* Виявлення проблем з дизайном інтерфейсу та внесення необхідних змін.

**4.2 Додаткові методи тестування:**

* **Регресійне тестування:** Перевірка, чи не призвели внесені зміни до появи нових помилок або погіршення роботи існуючих функцій.
* **A/B тестування:** Порівняння різних варіантів інтерфейсу або функціональності для визначення найбільш ефективного рішення.
* **Тестування безпеки:** Перевірка на наявність вразливостей та захисту від несанкціонованого доступу.

Застосування комплексного підходу до тестування програмного забезпечення дозволяє забезпечити його високу якість, надійність та зручність використання. Це особливо важливо для програм моніторингу системних показників, оскільки вони використовуються для контролю критичних параметрів роботи комп'ютера.

**4.3 Результати тестування та аналіз ефективності**

Під час розробки програмного забезпечення для моніторингу системних ресурсів виникли труднощі з доступом до деяких показників, таких як частота процесора та показники відеокарти, через обмеження стандартної бібліотеки PerformanceCounter. Для вирішення цієї проблеми було залучено Windows Management Instrumentation (WMI) та сторонні бібліотеки, зокрема OpenHardwareMonitor, що дозволило отримати доступ до необхідних даних.

**Тестування та виявлені проблеми:**

* **інтеграційне тестування:** під час інтеграції модулів збору даних, що використовують різні джерела (PerformanceCounter, WMI, OpenHardwareMonitor), виникли проблеми з узгодженням форматів даних та їх синхронізацією. Для вирішення цих проблем було розроблено уніфікований інтерфейс для збору даних та реалізовано механізм кешування, що забезпечує консистентність даних у часі.
* **системне тестування:** програму було протестовано на різних конфігураціях апаратного забезпечення (різні моделі процесорів, відеокарт, обсяги оперативної пам'яті) та версіях ОС Windows. В ході тестування було виявлено та виправлено декілька незначних помилок, пов'язаних з відображенням даних та обробкою винятків.
* **порівняльне тестування:** для перевірки коректності роботи програми було проведено порівняння її показників з даними, отриманими з Диспетчера завдань Windows. Для створення навантаження на систему запускалися ресурсоємні програми (ігри, бенчмарки), що дозволило переконатися у відповідності показників та коректній роботі програми в умовах високого навантаження.

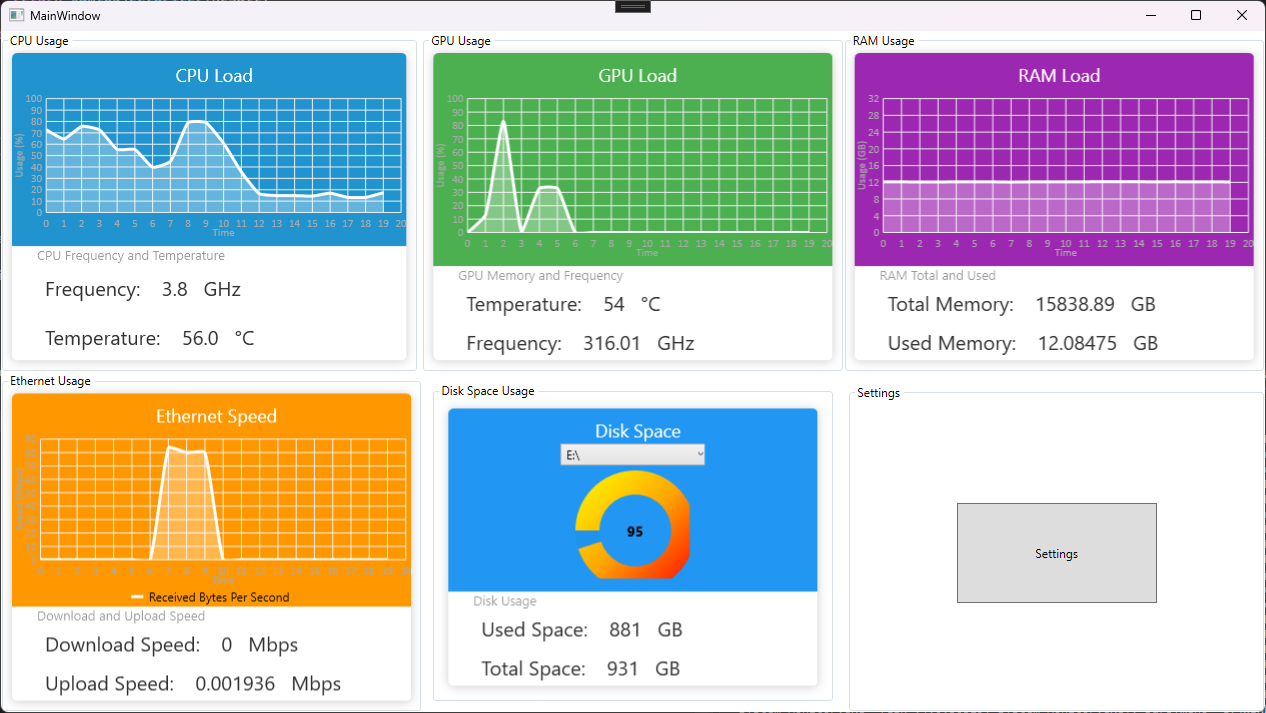


Рисунок 4.2 – Скріншот з додатку

Програма успішно виконує всі поставлені завдання, надаючи користувачеві детальну інформацію про стан системи у зручному та зрозумілому вигляді. Зокрема, реалізовано моніторинг наступних показників:

* завантаження процесора (загальне);
* частота процесора;
* температура процесора;
* завантаження відеокарти;
* температура відеокарти;
* використання оперативної пам'яті;
* завантаження дискового простору;
* мережева активність (вхідний/вихідний трафік).

# ВИСНОВКИ

У даній дипломній роботі було успішно розроблено демонстраційне програмне забезпечення для моніторингу системних показників комп'ютера в ОС Windows з використанням асинхронного введення-виведення.

В ході роботи було проведено огляд теоретичних основ асинхронного введення-виведення та особливостей моніторингу системних показників. Було проаналізовано існуючі рішення та виявлено їх переваги та недоліки, що дозволило сформулювати вимоги до розроблюваного програмного забезпечення.

В результаті проектування було обрано оптимальну архітектуру програми, що забезпечує модульність, гнучкість та масштабованість. Для реалізації було використано мову програмування C# та .NET Framework, що забезпечило високу продуктивність та надійність роботи програми.

Розроблене програмне забезпечення дозволяє здійснювати асинхронний збір даних про різноманітні системні показники, такі як завантаження процесора, використання пам'яті, активність диска, мережеву активність, температуру компонентів та інші. Завдяки асинхронному підходу, збір даних відбувається у фоновому режимі, не впливаючи на чуйність інтерфейсу користувача.

Отримані дані візуалізуються в режимі реального часу за допомогою графіків, діаграм та числових значень, що дозволяє користувачеві легко оцінювати поточний стан системи та відстежувати зміни показників у динаміці.

Програмне забезпечення також надає додаткові функції, такі як ведення логів та налаштування оповіщень. Ведення логів дозволяє відстежувати історію змін показників та виявляти потенційні проблеми, а оповіщення допомагають вчасно реагувати на критичні ситуації, такі як перевищення допустимих температур або завантаження ресурсів.

Проведене тестування підтвердило працездатність та ефективність розробленого програмного забезпечення. Воно успішно виконує свої функції, забезпечуючи моніторинг системних показників в режимі реального часу та надаючи користувачеві зручний інструмент для контролю стану комп'ютера.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Perfomance Counters. [Virtual Resource] / Microsoft Docs. – 2024 – Access Mode: URL:[https://learn.microsoft.com/ru-ru/dotnet/api/system.diagnostics.performancecounter](https://learn.microsoft.com/ru-ru/dotnet/api/system.diagnostics.performancecounter?view=net-8.0&viewFallbackFrom=dotnet-plat-ext-8.0)
2. Task Parallel Library. [Virtual Resource] / Microsoft Docs. – 2024 – Access Mode: URL:<https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl>
3. Asynchronus I/O. [Virtual Resource] / Wikipedia. – 2024 – 03 June. – Access Mode: URL: <https://en.wikipedia.org/wiki/Asynchronous_I/O>
4. Input/output completion port. [Virtual Resource] / Wikipedia. – 2021 – 29 June. – Access Mode: URL: <https://en.wikipedia.org/wiki/Input/output_completion_port>
5. Overlapper I/O. [Virtual Resource] / Wikipedia. – 2021 – 9 Nov. – Access Mode: URL: <https://en.wikipedia.org/wiki/Overlapped_I/O>

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Проректор Українського

державного університету

науки і технологій

\_\_\_\_\_\_\_Анатолій РАДКЕВИЧ

05.04.2024

РОЗРОБКА ДЕМОНСТРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВИКОРИСТАННЯ АСИНХРОННОГО ВВЕДЕННЯ В ОС WINDOWS

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.1392-01-ЛЗ

Представники

підприємства-розробника

Завідувач кафедри КІТ

\_\_\_\_\_\_\_Вадим ГОРЯЧКІН

10.06.2024

Керівник розробки

\_\_Вадим АНДРЮЩЕНКО

10.06.2024

Виконавець  
 \_\_\_\_\_\_\_Сергій КУЛИК

10.06.2024

Нормоконтролер

\_\_\_\_\_\_\_ Світлана ВОЛКОВА

10.06.2024

2024

ЗАТВЕРДЖЕНО

44165850.1392-01-ЛЗ

РОЗРОБКА ДЕМОНСТРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВИКОРИСТАННЯ АСИНХРОННОГО ВВЕДЕННЯ В ОС WINDOWS

Технічне завдання

44165850.1392-01

Листів 12

2024

ВВЕДЕННЯ

Програмний засіб призначений для моніторингу системних показників комп'ютера в режимі реального часу. Головною метою цього проекту є розробка демонстраційного програмного забезпечення, яке надає користувачеві детальну та актуальну інформацію про стан системи, використовуючи асинхронний підхід для ефективного збору та обробки даних.

У реалізації цієї системи передбачено використання класу PerformanceCounter та Windows Management Instrumentation (WMI) для отримання різноманітних показників, таких як завантаження процесора, використання пам'яті, активність диска та мережі, температура компонентів тощо. Асинхронний підхід забезпечує плавну роботу програми та чуйність інтерфейсу, навіть під час інтенсивного збору даних.

При впровадженні програми надається можливість візуалізації даних у режимі реального часу за допомогою графіків, діаграм та числових значень, що дозволяє користувачеві легко оцінювати стан системи та виявляти потенційні проблеми. Додаткові функції, такі як ведення логів та налаштування оповіщень, розширюють можливості моніторингу та допомагають вчасно реагувати на критичні ситуації.

Цей продукт призначений для використання системними адміністраторами, тестувальниками програмного забезпечення та звичайними користувачами, які хочуть мати детальну інформацію про роботу свого комп'ютера. Він сприяє кращому розумінню процесів, що відбуваються в системі, та допомагає оптимізувати її налаштування для досягнення максимальної продуктивності.

ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 27.12.23 №1173 ст ректора Українського державного університету науки і технологій “Про призначення наукових керівників та затвердження тем бакалаврських робіт” за спеціальністю 121 “Інженерія програмного забезпечення» факультету “Комп’ютерних технологій і систем” по кафедрі “Комп’ютерні інформаційні технології”.

ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональним призначенням програми є моніторинг системних показників комп'ютера в режимі реального часу. Ключові функції програми включають:

* **збір даних:** отримання інформації про завантаження процесора, використання пам'яті, активність диска та мережі, температуру компонентів тощо;
* **візуалізація:** наочне відображення зібраних даних у вигляді графіків, діаграм та числових значень;
* **оповіщення:** інформування користувача про перевищення встановлених порогів або виникнення інших подій, що потребують уваги;
* **логування:** збереження історії змін показників для подальшого аналізу та виявлення тенденцій.

Експлуатаційне призначення програми включає:

* **оцінка продуктивності:** дозволяє користувачеві оцінити поточну продуктивність комп'ютера та виявити "вузькі місця", які можуть обмежувати його швидкість та ефективність;
* **діагностика проблем:** допомагає виявити проблеми з апаратним забезпеченням (наприклад, перегрів) або програмним забезпеченням (наприклад, витік пам'яті), що можуть призводити до нестабільної роботи системи;
* **оптимізація налаштувань:** надає інформацію, необхідну для налаштування параметрів системи та програм для досягнення оптимальної продуктивності;
* **контроль стану системи:** дозволяє відстежувати зміни у системних показниках в режимі реального часу та вчасно реагувати на них.

Цей програмний засіб призначений для широкого кола користувачів:

* **системні адміністратори:** використовують програму для моніторингу серверів та робочих станцій, виявлення проблем та оптимізації продуктивності;
* **розробники програмного забезпечення:** використовують програму для аналізу споживання ресурсів своїми додатками та виявлення потенційних проблем з продуктивністю;
* **геймери:** використовують програму для моніторингу температури та завантаження компонентів під час гри, щоб уникнути перегріву та забезпечити максимальну продуктивність;
* **звичайні користувачі:** використовують програму для отримання загальної інформації про стан свого комп'ютера та виявлення можливих проблем.

ВИМОГИ ДО ПРОГРАМИ

**Вимоги до функціональних характеристик**

Програма повинна надавати можливість:

* **Налаштування інтервалу оновлення:** Користувач може змінювати частоту оновлення даних (наприклад, кожну секунду, кожні 5 секунд тощо).
* **Налаштування оповіщень:** Користувач може встановлювати пороги для кожного показника, при перевищенні яких буде згенеровано оповіщення (візуальне або звукове).
* **Перегляд логів:** Користувач може переглядати історію змін системних показників у вигляді логів.

**Вимоги до надійності**

Програма має забезпечити:

* **Стійку роботу:** Програма повинна працювати стабільно та без збоїв протягом тривалого часу.
* **Коректність даних:** Програма повинна відображати коректні дані про системні показники.
* **Обробку помилок:** Програма повинна коректно обробляти помилки, які можуть виникнути під час збору або відображення даних (наприклад, відсутність доступу до лічильників продуктивності).

**Умови експлуатації**

* **Апаратне забезпечення:** Програма повинна працювати на комп'ютерах з процесором Intel або AMD, оперативною пам'яттю не менше 512 МБ та вільним місцем на диску не менше 100 МБ.
* **Програмне забезпечення:** Програма повинна працювати під управлінням ОС Windows 10 або новішої версії.
* **Кваліфікація користувача:** Для використання програми достатньо базових знань роботи з комп'ютером.

**Вимоги до складу і параметрів технічних засобів**

**Апаратне забезпечення:**

* + Процесор: Intel Core i3 або аналогічний.
  + Оперативна пам'ять: 4 ГБ або більше.
  + Вільне місце на диску: 200 МБ.

**Програмне забезпечення:**

* + Операційна система: Windows 10 (64-bit) або новіша.
  + .NET Framework 4.7.2 або новіша версія.

**Вимоги до інформаційної і програмної сумісності**

Програма повинна бути сумісна з іншими програмами, що працюють в ОС Windows, та не повинна впливати на їх роботу.

**Вимоги до маркування і упаковки**

Програма повинна мати інсталятор, який забезпечує її коректну установку та налаштування на комп'ютері користувача. Інсталятор повинен містити всю необхідну інформацію про програму (назву, версію, автора, ліцензію тощо).

**Вимоги до транспортування і зберігання**

Програма може бути розповсюджена у вигляді інсталяційного файлу через Інтернет або на фізичному носії (наприклад, USB-флешці). Для зберігання програми не потрібні спеціальні умови.

ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації повинні входити:

* специфікація;
* текст програми;
* опис програми;
* керівництво користувача.

Уся документація до програми повинна задовольняти вимогам державного стандарту до оформлення програмних документів

СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця A.1 – Стадії та етапи розробки

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів кваліфікаційної роботи | Строк виконання  етапів роботи | Примітка |
| 1 | Огляд предметної області та аналіз вимог | 25.04.24 – 30.04.24 |  |
| 2 | Проектування архітектури та вибір технологій | 01.05.24 – 06.05.24 |  |
| 3 | Розробка базової функціональності | 06.05.24 – 12.05.24 | 30% |
| 4 | Реалізація асинхронного збору даних | 12.05.24 – 14.05.24 |  |
| 5 | Розробка інтерфейсу користувача | 14.05.24 – 19.05.24 |  |
| 6 | Інтеграція та тестування компонентів | 20.05.24 – 26.05.24 | 60% |
| 7 | Доопрацювання та оптимізація | 27.05.24 – 30.05.24 |  |
| 8 | Підготовка документації та презентації | 01.06.24 – 10.04.23 |  |
| 9 | Подання кваліфікаційної роботи до кафедри | 10.06.24 – 16.06.24 | 100% |
| 10 | Захист кваліфікаційної роботи на засіданні  Екзаменаційної комісії | 24.06.2024 |  |
|  |  |  |  |

ПОРЯДОК КОНТРОЛЮ ТА ПРИЙОМУ

Контроль здійснює керівник розробки – Іванов О.П.

Прийом програми здійснює уповноважена комісія.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.:  
Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн.  
трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ

Проректор Українського

державного університету

науки і технологій

\_\_\_\_\_\_\_Анатолій РАДКЕВИЧ

24.06.2024

РОЗРОБКА ДЕМОНСТРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВИКОРИСТАННЯ АСИНХРОННОГО ВВЕДЕННЯ В ОС WINDOWS

Код програми

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.1392-01-ЛЗ

Представники

підприємства-розробника

Завідувач кафедри КІТ

\_\_\_\_\_\_\_Вадим ГОРЯЧКІН

10.06.2024

Керівник розробки

\_\_Вадим АНДРЮЩЕНКО

10.06.2024

Виконавець  
 \_\_\_\_\_\_\_Сергій КУЛИК

10.06.2024

Нормоконтролер

\_\_\_\_\_\_\_ Світлана ВОЛКОВА

10.06.2024

2024

ЗАТВЕРДЖЕНО

44165850.1392-01-ЛЗ

РОЗРОБКА ДЕМОНСТРАЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВИКОРИСТАННЯ АСИНХРОННОГО ВВЕДЕННЯ В ОС WINDOWS

Технічне завдання

44165850.1392-01

Листів 43

2024

ТЕКСТ ПРОГРАМИ

Файл AsyncDataCollector.cs

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

internal class AsyncDataCollector : IDisposable

{

private readonly SystemMetricsCollector \_metricsCollector;

private readonly MetricsCache \_metricsCache;

private CancellationTokenSource \_cancellationTokenSource;

private bool \_disposed = false;

public AsyncDataCollector()

{

\_metricsCollector = new SystemMetricsCollector();

\_metricsCache = new MetricsCache();

}

public async Task StartCollectingAsync(CancellationToken cancellationToken = default) // Додаємо CancellationToken

{

\_cancellationTokenSource = new CancellationTokenSource();

var linkedTokenSource = CancellationTokenSource.CreateLinkedTokenSource(cancellationToken, \_cancellationTokenSource.Token);

var token = linkedTokenSource.Token;

try

{

await Task.Run(async () =>

{

while (!token.IsCancellationRequested)

{

try // Додаємо try-catch для обробки винятків

{

CollectMetrics();

}

catch (Exception ex)

{

Console.WriteLine($"Error collecting metrics: {ex.Message}");

// Додаткова обробка помилки (наприклад, логування)

}

await Task.Delay(1000, token);

}

}, token);

}

catch (OperationCanceledException)

{

Console.WriteLine("Data collection canceled.");

}

}

public void StopCollecting()

{

\_cancellationTokenSource?.Cancel();

}

private void CollectMetrics()

{

float cpuUsage = \_metricsCollector.GetCpuUsage();

float cpuFrequency = \_metricsCollector.GetCpuFrequency();

float cpuTemperature = \_metricsCollector.GetCpuTemperature();

float gpuLoad = \_metricsCollector.GetGpuLoad();

float gpuMemory = \_metricsCollector.GetGpuMemory();

float gpuFrequency = \_metricsCollector.GetGpuFrequancy();

float gpuTemperature = \_metricsCollector.GetGpuTemperature();

float ramUsage = \_metricsCollector.GetRamUsage();

float totalMemory = \_metricsCollector.GetTotalMemory();

float usedMemory = \_metricsCollector.GetUsedMemory();

float sentBytesPerSecond = \_metricsCollector.GetEthernetUploadSpeed();

float receivedBytesPerSecond = \_metricsCollector.GetEthernetDownloadSpeed();

\_metricsCache.UpdateMetrics(new Dictionary<string, float>

{

{ "CPU Usage", cpuUsage },

{ "CPU Frequency", cpuFrequency },

{ "CPU Temperature", cpuTemperature },

{ "GPU Load", gpuLoad },

{ "GPU Memory", gpuMemory },

{ "GPU Frequency", gpuFrequency },

{ "GPU Temperature", gpuTemperature },

{ "RAM Usage", ramUsage },

{ "Total Memory", totalMemory },

{ "Used Memory", usedMemory },

{ "Sent Bytes Per Second", sentBytesPerSecond },

{ "Received Bytes Per Second", receivedBytesPerSecond }

});

}

public MetricsCache GetMetricsCache() => \_metricsCache;

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (!\_disposed)

{

if (disposing)

{

\_metricsCollector?.Dispose();

\_cancellationTokenSource?.Cancel();

\_cancellationTokenSource?.Dispose();

}

\_disposed = true;

}

}

}

public class MetricsCache

{

private readonly Dictionary<string, float> \_metrics = new Dictionary<string, float>();

public void UpdateMetrics(Dictionary<string, float> metrics)

{

foreach (var metric in metrics)

{

\_metrics[metric.Key] = metric.Value;

}

}

public float GetMetric(string metricName)

{

return \_metrics.TryGetValue(metricName, out float value) ? value : -1;

}

public Dictionary<string, float> GetAllMetrics()

{

return new Dictionary<string, float>(\_metrics);

}

}

Файл DiskInfo.cs  
using System;

public class DiskInfo

{

public string Name { get; set; }

public double TotalSpace { get; set; }

public double UsedSpace { get; set; }

public int UsagePercentage => Convert.ToInt32((UsedSpace / TotalSpace) \* 100);

}

Файл DiskViewModel.cs

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace System\_Monitor.CodeBase

{

internal class DiskViewModel : INotifyPropertyChanged

{

public ObservableCollection<DiskInfo> DiskDrives { get; set; }

private DiskInfo \_selectedDisk;

public DiskInfo SelectedDisk

{

get => \_selectedDisk;

set

{

\_selectedDisk = value;

OnPropertyChanged(nameof(SelectedDisk));

}

}

public DiskViewModel()

{

DiskDrives = new ObservableCollection<DiskInfo>(GetAllDisks());

}

private ObservableCollection<DiskInfo> GetAllDisks()

{

var disks = new ObservableCollection<DiskInfo>();

foreach (var drive in DriveInfo.GetDrives().Where(d => d.IsReady))

{

disks.Add(new DiskInfo

{

Name = drive.Name,

TotalSpace = drive.TotalSize / (1024 \* 1024 \* 1024),

UsedSpace = (drive.TotalSize - drive.AvailableFreeSpace) / (1024 \* 1024 \* 1024)

});

}

return disks;

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

}

Файл MetricsViewModel.cs

using LiveCharts;

using LiveCharts.Wpf;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Threading.Tasks;

using System.Windows.Threading;

using System\_Monitor.CodeBase;

using System.IO;

using System.Linq;

public class MetricsViewModel : INotifyPropertyChanged

{

private readonly AsyncDataCollector \_asyncDataCollector;

private readonly DispatcherTimer \_timer;

private float \_cpuUsage;

private float \_cpuFrequency;

private float \_cpuTemperature;

private float \_ramUsage;

private float \_usedMemory;

private float \_totalMemory;

private float \_gpuLoad;

private float \_gpuMemory;

private float \_gpuFrequancy;

private float \_uploadSpeed;

private float \_downloadSpeed;

public ObservableCollection<DiskInfo> DiskDrives { get; set; }

private DiskInfo \_selectedDisk;

public DiskInfo SelectedDisk

{

get => \_selectedDisk;

set

{

\_selectedDisk = value;

OnPropertyChanged(nameof(SelectedDisk));

}

}

// Series collections for different metrics

public SeriesCollection CpuLoadSeries { get; set; }

public SeriesCollection RamLoadSeries { get; set; }

public SeriesCollection CpuFrequencySeries { get; set; }

public SeriesCollection CpuTemperatureSeries { get; set; }

public SeriesCollection GpuTemperatureSeries { get; set; }

public SeriesCollection GpuLoadSeries { get; set; }

public SeriesCollection GpuTotalMemorySeries { get; set; }

public SeriesCollection GpuFrequencySeries { get; set; }

public SeriesCollection UploadSpeedSeries { get; set; }

public SeriesCollection DownloadSpeedSeries { get; set; }

public Func<double, string> Formatter { get; set; }

// Properties for metrics

public float CpuUsage

{

get => \_cpuUsage;

set

{

\_cpuUsage = value;

OnPropertyChanged();

}

}

public float CpuFrequency

{

get => \_cpuFrequency;

set

{

\_cpuFrequency = value;

OnPropertyChanged();

}

}

public float CpuTemperature

{

get => \_cpuTemperature;

set

{

\_cpuTemperature = value;

OnPropertyChanged();

}

}

public float RamUsage

{

get => \_ramUsage;

set

{

\_ramUsage = value;

OnPropertyChanged();

}

}

public float UsedMemory

{

get => \_usedMemory;

set

{

\_usedMemory = value;

OnPropertyChanged();

}

}

public float TotalMemory

{

get => \_totalMemory;

set

{

\_totalMemory = value;

OnPropertyChanged();

}

}

public float GpuLoad

{

get => \_gpuLoad;

set

{

\_gpuLoad = value;

OnPropertyChanged();

}

}

public float GpuTotalMemory

{

get => \_gpuMemory;

set

{

\_gpuMemory = value;

OnPropertyChanged();

}

}

public float GpuFrequency

{

get => \_gpuFrequancy;

set

{

\_gpuFrequancy = value;

OnPropertyChanged();

}

}

public float GpuTemperature

{

get => \_gpuFrequancy;

set

{

\_gpuFrequancy = value;

OnPropertyChanged();

}

}

public float UploadSpeed

{

get => \_uploadSpeed;

set

{

\_uploadSpeed = value;

OnPropertyChanged();

}

}

public float DownloadSpeed

{

get => \_downloadSpeed;

set

{

\_downloadSpeed = value;

OnPropertyChanged();

}

}

// Constructor

public MetricsViewModel()

{

\_asyncDataCollector = new AsyncDataCollector();

DiskDrives = new ObservableCollection<DiskInfo>(GetAllDisks());

InitializeSeriesCollections();

Formatter = value => value.ToString("F");

\_timer = new DispatcherTimer { Interval = TimeSpan.FromSeconds(1) };

\_timer.Tick += UpdateMetrics;

\_timer.Start();

}

private void InitializeSeriesCollections()

{

CpuLoadSeries = new SeriesCollection

{

new LineSeries

{

Title = "CPU Usage",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

RamLoadSeries = new SeriesCollection

{

new LineSeries

{

Title = "RAM Usage",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

CpuFrequencySeries = new SeriesCollection

{

new LineSeries

{

Title = "CPU Frequency",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

CpuTemperatureSeries = new SeriesCollection

{

new LineSeries

{

Title = "CPU Temperature",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

GpuLoadSeries = new SeriesCollection

{

new LineSeries

{

Title = "GPU Load",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

GpuTemperatureSeries = new SeriesCollection

{

new LineSeries

{

Title = "GPU Temperature",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

GpuTotalMemorySeries = new SeriesCollection

{

new LineSeries

{

Title = "GPU Total Memory",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

GpuFrequencySeries = new SeriesCollection

{

new LineSeries

{

Title = "GPU Frequency",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

UploadSpeedSeries = new SeriesCollection

{

new LineSeries

{

Title = "Sent Bytes Per Second",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

DownloadSpeedSeries = new SeriesCollection

{

new LineSeries

{

Title = "Received Bytes Per Second",

Values = new ChartValues<float>(),

PointGeometry = null

}

};

}

private ObservableCollection<DiskInfo> GetAllDisks()

{

var disks = new ObservableCollection<DiskInfo>();

foreach (var drive in DriveInfo.GetDrives().Where(d => d.IsReady))

{

disks.Add(new DiskInfo

{

Name = drive.Name,

TotalSpace = drive.TotalSize / (1024 \* 1024 \* 1024),

UsedSpace = (drive.TotalSize - drive.AvailableFreeSpace) / (1024 \* 1024 \* 1024)

});

}

return disks;

}

private void UpdateMetrics(object sender, EventArgs e)

{

var metricsCache = \_asyncDataCollector.GetMetricsCache().GetAllMetrics();

// Update CPU Usage

CpuUsage = metricsCache.TryGetValue("CPU Usage", out float cpuUsage) ? cpuUsage : -1;

AddValueToSeries(CpuLoadSeries, CpuUsage);

// Update RAM Usage

RamUsage = metricsCache.TryGetValue("RAM Usage", out float ramUsage) ? ramUsage : -1;

AddValueToSeries(RamLoadSeries, RamUsage);

// Update Used Memory and Total Memory

UsedMemory = metricsCache.TryGetValue("Used Memory", out float usedMemory) ? usedMemory : -1;

TotalMemory = metricsCache.TryGetValue("Total Memory", out float totalMemory) ? totalMemory : -1;

// Update CPU Frequency

CpuFrequency = metricsCache.TryGetValue("CPU Frequency", out float cpuFrequency) ? cpuFrequency : -1;

AddValueToSeries(CpuFrequencySeries, cpuFrequency);

// Update CPU Temperature

CpuTemperature = metricsCache.TryGetValue("CPU Temperature", out float cpuTemperature) ? cpuTemperature : -1;

AddValueToSeries(CpuTemperatureSeries, cpuTemperature);

// Update GPU Load

GpuLoad = metricsCache.TryGetValue("GPU Load", out float gpuLoad) ? gpuLoad : -1;

AddValueToSeries(GpuLoadSeries, gpuLoad);

// Update GPU Total Memory

GpuTotalMemory = metricsCache.TryGetValue("GPU Total Memory", out float gpuTotalMemory) ? gpuTotalMemory : -1;

AddValueToSeries(GpuTotalMemorySeries, gpuTotalMemory);

// Update GPU Frequency

GpuFrequency = metricsCache.TryGetValue("GPU Frequency", out float gpuFrequency) ? gpuFrequency : -1;

AddValueToSeries(GpuFrequencySeries, gpuFrequency);

GpuTemperature = metricsCache.TryGetValue("GPU Temperature", out float gpuTemperature) ? gpuTemperature : -1;

AddValueToSeries(GpuTemperatureSeries, gpuTemperature);

// Update Upload Speed (converted to Mbps)

UploadSpeed = metricsCache.TryGetValue("Sent Bytes Per Second", out float uploadSpeed) ? uploadSpeed / 125000 : -1;

AddValueToSeries(UploadSpeedSeries, uploadSpeed / 125000);

// Update Download Speed (converted to Mbps)

DownloadSpeed = metricsCache.TryGetValue("Received Bytes Per Second", out float downloadSpeed) ? downloadSpeed / 125000 : -1;

AddValueToSeries(DownloadSpeedSeries, downloadSpeed / 125000);

// Trim data in series if it exceeds a certain number of points

TrimSeriesData(CpuLoadSeries[0].Values);

TrimSeriesData(RamLoadSeries[0].Values);

TrimSeriesData(CpuFrequencySeries[0].Values);

TrimSeriesData(CpuTemperatureSeries[0].Values);

TrimSeriesData(GpuLoadSeries[0].Values);

TrimSeriesData(GpuTotalMemorySeries[0].Values);

TrimSeriesData(GpuFrequencySeries[0].Values);

TrimSeriesData(UploadSpeedSeries[0].Values);

TrimSeriesData(DownloadSpeedSeries[0].Values);

}

private void AddValueToSeries(SeriesCollection series, float value)

{

if (series != null && series.Count > 0)

{

var lineSeries = (LineSeries)series[0];

lineSeries.Values.Add(value);

}

}

private void TrimSeriesData(IChartValues values)

{

if (values is ChartValues<float> floatValues)

{

if (floatValues.Count > 20)

floatValues.RemoveAt(0);

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

public async Task StartAsync()

{

await \_asyncDataCollector.StartCollectingAsync();

}

public void Stop()

{

\_asyncDataCollector.StopCollecting();

}

}

Файл Settings.cs  
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Text.Json;

using System.IO;

using System.ComponentModel;

namespace System\_Monitor.CodeBase

{

public partial class Settings : INotifyPropertyChanged

{

private string \_selectedMemoryUnit;

private string \_selectedSpeedUnit;

public string[] MemoryUnits { get; } = { "GB", "MB" };

public string[] SpeedUnits { get; } = { "Mbps", "MB/s" };

public bool AutoStart { get; set; }

public bool ShowInSystemTray { get; set; }

public string SelectedMetric { get; set; }

public string SelectedFrequency { get; set; }

public double NotificationThreshold { get; set; }

public string SelectedNotificationType { get; set; }

public string LogFilePath { get; set; }

public string SelectedMemoryUnit

{

get { return \_selectedMemoryUnit; }

set

{

\_selectedMemoryUnit = value;

OnPropertyChanged(nameof(SelectedMemoryUnit));

}

}

public string SelectedSpeedUnit

{

get { return \_selectedSpeedUnit; }

set

{

\_selectedSpeedUnit = value;

OnPropertyChanged(nameof(SelectedSpeedUnit));

}

}

// Сохранение настроек в JSON файл

public void Save(string filePath)

{

try

{

string json = JsonSerializer.Serialize(this);

File.WriteAllText(filePath, json);

}

catch (Exception ex)

{

// Обработка ошибок сохранения

Console.WriteLine($"Ошибка сохранения настроек: {ex.Message}");

}

}

// Загрузка настроек из JSON файла

public static Settings Load(string filePath)

{

try

{

if (File.Exists(filePath))

{

string json = File.ReadAllText(filePath);

return JsonSerializer.Deserialize<Settings>(json);

}

else

{

return new Settings(); // Возвращаем новый экземпляр настроек, если файл не найден

}

}

catch (Exception ex)

{

// Обработка ошибок загрузки

Console.WriteLine($"Ошибка загрузки настроек: {ex.Message}");

return new Settings(); // Возвращаем новый экземпляр настроек в случае ошибки

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

}

Файл SystemMetricsCollector.cs

using System;

using System.Diagnostics;

using System.Linq;

using System.Management;

using LibreHardwareMonitor.Hardware;

internal class SystemMetricsCollector : IDisposable

{

private bool \_disposed = false;

private readonly Computer \_computer;

public SystemMetricsCollector()

{

\_computer = new Computer

{

IsCpuEnabled = true, // Enable CPU monitoring

IsGpuEnabled = true, // Enable GPU monitoring

IsMemoryEnabled = true, // Enable RAM monitoring

IsNetworkEnabled = true // Enable network monitoring

};

\_computer.Open();

}

public float GetCpuUsage() => GetHardwareSensorValue(HardwareType.Cpu, SensorType.Load);

public float GetCpuFrequency()

{

try

{

using (var searcher = new ManagementObjectSearcher("SELECT CurrentClockSpeed FROM Win32\_Processor"))

{

foreach (var obj in searcher.Get())

{

return Convert.ToSingle(obj["CurrentClockSpeed"]) / 1000; // Convert from MHz to GHz

}

}

}

catch (Exception ex)

{

Console.WriteLine($"Error getting CPU frequency: {ex.Message}");

}

return -1; // Or handle error differently

}

public float GetCpuTemperature() => GetHardwareSensorValue(HardwareType.Cpu, SensorType.Temperature);

public float GetGpuLoad()

{

var computer = new OpenHardwareMonitor.Hardware.Computer();

computer.Open();

computer.GPUEnabled = true;

foreach (var hardware in computer.Hardware)

{

if (hardware.HardwareType == OpenHardwareMonitor.Hardware.HardwareType.GpuNvidia || hardware.HardwareType == OpenHardwareMonitor.Hardware.HardwareType.GpuAti)

{

hardware.Update();

foreach (var sensor in hardware.Sensors)

{

if (sensor.SensorType == OpenHardwareMonitor.Hardware.SensorType.Load && sensor.Name.Contains("Core"))

{

return sensor.Value.GetValueOrDefault();

}

}

}

}

computer.Close();

return -1;

}

public float GetGpuMemory()

{

var totalMemory = GetHardwareSensorValue(HardwareType.GpuAmd, SensorType.SmallData); // Replace with GpuAmd if you have an AMD GPU

if (totalMemory == -1)

{

return -1; // Handle sensor not found or unavailable

}

return totalMemory / (1024 \* 1024); // Convert to MB

}

public float GetGpuFrequancy()

{

var computer = new OpenHardwareMonitor.Hardware.Computer();

computer.Open();

computer.GPUEnabled = true;

foreach (var hardware in computer.Hardware)

{

if (hardware.HardwareType == OpenHardwareMonitor.Hardware.HardwareType.GpuNvidia || hardware.HardwareType == OpenHardwareMonitor.Hardware.HardwareType.GpuAti)

{

hardware.Update();

foreach (var sensor in hardware.Sensors)

{

// Шукаємо сенсор з назвою "GPU Core" або подібним, який має тип Clock

if (sensor.SensorType == OpenHardwareMonitor.Hardware.SensorType.Clock && sensor.Name.Contains("Core"))

{

return sensor.Value.GetValueOrDefault(); // Повертаємо значення частоти

}

}

}

}

computer.Close();

return -1; // Повертаємо -1, якщо сенсор не знайдено

}

public float GetGpuTemperature() => GetHardwareSensorValue(HardwareType.GpuAmd, SensorType.Temperature);

public float GetRamUsage() => GetHardwareSensorValue(HardwareType.Memory, SensorType.Data);

public float GetTotalMemory()

{

try

{

using (var searcher = new ManagementObjectSearcher("SELECT TotalVisibleMemorySize FROM Win32\_OperatingSystem"))

{

foreach (var obj in searcher.Get())

{

return Convert.ToSingle(obj["TotalVisibleMemorySize"]) / 1024; // Convert from KB to MB

}

}

}

catch (Exception ex)

{

Console.WriteLine($"Error getting total memory: {ex.Message}");

}

return -1; // Or handle error differently

}

public float GetUsedMemory()=> GetHardwareSensorValue(HardwareType.Memory, SensorType.Data);

private float GetNetworkSpeedValue(string adapterName, string propertyName)

{

try

{

using (var searcher = new ManagementObjectSearcher(

$"SELECT \* FROM Win32\_PerfFormattedData\_Tcpip\_NetworkInterface WHERE Name LIKE '%{adapterName}%'"))

{

foreach (var obj in searcher.Get())

{

return Convert.ToSingle(obj[propertyName]);

}

}

}

catch (Exception ex)

{

Console.WriteLine($"Error getting network speed: {ex.Message}");

}

return -1;

}

public float GetEthernetDownloadSpeed()

{

return GetNetworkSpeedValue("Realtek PCIe GbE Family Controller", "BytesReceivedPerSec");

}

public float GetEthernetUploadSpeed()

{

return GetNetworkSpeedValue("Realtek PCIe GbE Family Controller", "BytesSentPerSec");

}

private float GetHardwareSensorValue(HardwareType hardwareType, SensorType sensorType)

{

foreach (var hardware in \_computer.Hardware)

{

if (hardware.HardwareType != hardwareType) continue;

hardware.Update();

return hardware

.Sensors

.Where(s => s.SensorType == sensorType && s.Value.HasValue)

.Select(s => s.Value.Value)

.FirstOrDefault();

}

return -1; // Or handle the case where no sensor is found differently

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (!\_disposed)

{

if (disposing)

{

\_computer.Close();

}

\_disposed = true;

}

}

}

Файл SystemMetricsLogger.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

public class SystemMetricsLogger

{

private AsyncDataCollector \_dataCollector;

private string \_logFilePath;

private const int MaxLogFileSize = 1024 \* 1024;

public SystemMetricsLogger(AsyncDataCollector dataCollector, string logFilePath = "system\_metrics\_log.txt")

{

\_dataCollector = dataCollector;

\_logFilePath = logFilePath;

}

public async Task StartLoggingAsync()

{

while (!\_dataCollector.IsDisposed)

{

var metrics = \_dataCollector.GetMetricsCache().GetAllMetrics();

string logEntry = $"{DateTime.Now:dd.MM.yyyy HH:mm}";

foreach (var metric in metrics)

{

if (!float.IsNaN(metric.Value))

{

logEntry += $" {metric.Key}: {metric.Value}";

}

}

if (new FileInfo(\_logFilePath).Length > MaxLogFileSize)

{

RotateLogFile();

}

File.AppendAllText(\_logFilePath, logEntry + Environment.NewLine);

await Task.Delay(5000);

}

}

private void RotateLogFile()

{

string newLogFileName = $"{Path.GetFileNameWithoutExtension(\_logFilePath)}\_{DateTime.Now:yyyyMMddHHmmss}.log";

string newLogFilePath = Path.Combine(Path.GetDirectoryName(\_logFilePath), newLogFileName);

File.Move(\_logFilePath, newLogFilePath);

}

}

Файл MainWindow.xaml

<Window x:Class="System\_Monitor.MainWindow"

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

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

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

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

xmlns:lvc="clr-namespace:LiveCharts.Wpf;assembly=LiveCharts.Wpf"

mc:Ignorable="d"

Title="MainWindow" Height="720" Width="1280" Background="White">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0\*" />

<ColumnDefinition/>

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<GroupBox Header="CPU Usage" Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="3" Margin="0,0,426,0">

<Grid Margin="5">

<Grid.Effect>

<DropShadowEffect BlurRadius="15" Direction="-90" RenderingBias="Quality" Opacity=".2" ShadowDepth="1"/>

</Grid.Effect>

<Grid.OpacityMask>

<VisualBrush Visual="{Binding ElementName=Border1}" />

</Grid.OpacityMask>

<Grid.Resources>

<Style TargetType="lvc:LineSeries">

<Setter Property="StrokeThickness" Value="3" />

<Setter Property="Stroke" Value="White" />

<Setter Property="Fill" Value="#4EFFFFFF" />

<Setter Property="PointGeometrySize" Value="0" />

<Setter Property="LineSmoothness" Value="0.3" />

</Style>

<Style TargetType="lvc:Axis">

<Setter Property="ShowLabels" Value="True" />

<Setter Property="IsEnabled" Value="True" />

</Style>

</Grid.Resources>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Border x:Name="Border1" Grid.Row="0" Grid.RowSpan="4" CornerRadius="5" Background="White" />

<Border Grid.Row="0" Grid.RowSpan="3" Background="#FF2193CE" />

<TextBlock Grid.Row="0" TextAlignment="Center" Padding="10, 10, 0, 5" Foreground="White" FontSize="18">

CPU Load

</TextBlock>

<lvc:CartesianChart Grid.Row="2" Margin="0,0,0,0" Series="{Binding CpuLoadSeries}" Hoverable="True" DataTooltip="{x:Null}" DisableAnimations="True">

<lvc:CartesianChart.AxisX>

<lvc:Axis MinValue="0" MaxValue="20" Title="Time">

<lvc:Axis.Separator>

<lvc:Separator Step="1"/>

</lvc:Axis.Separator>

</lvc:Axis>

</lvc:CartesianChart.AxisX>

<lvc:CartesianChart.AxisY>

<lvc:Axis MinValue="0" MaxValue="100" Title="Usage (%)">

<lvc:Axis.Separator>

<lvc:Separator Step="10"/>

</lvc:Axis.Separator>

</lvc:Axis>

</lvc:CartesianChart.AxisY>

</lvc:CartesianChart>

<StackPanel Grid.Row="3" VerticalAlignment="Center" Margin="25,0">

<TextBlock Opacity=".4" FontSize="13">CPU Frequency and Temperature</TextBlock>

<StackPanel Orientation="Horizontal" Margin="0,5">

<TextBlock Foreground="#303030" FontSize="20" Text="Frequency: " Margin="8, 6" />

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding CpuFrequency, StringFormat={}{0:N1}}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">GHz</TextBlock>

</StackPanel>

<StackPanel Orientation="Horizontal" Margin="0,5">

<TextBlock Foreground="#303030" FontSize="20" Text="Temperature: " Margin="8, 6"/>

<!-- Corrected binding to CpuTemperature -->

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding CpuTemperature, StringFormat={}{0:N1}}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">°C</TextBlock>

</StackPanel>

</StackPanel>

</Grid>

</GroupBox>

<!--GPU -->

<GroupBox Header="GPU Usage" Grid.Row="0" Grid.Column="2">

<Grid Margin="5">

<Grid.Effect>

<DropShadowEffect BlurRadius="15" Direction="-90" RenderingBias="Quality" Opacity=".2" ShadowDepth="1"/>

</Grid.Effect>

<Grid.OpacityMask>

<VisualBrush Visual="{Binding ElementName=Border1}" />

</Grid.OpacityMask>

<Grid.Resources>

<Style TargetType="lvc:LineSeries">

<Setter Property="StrokeThickness" Value="3" />

<Setter Property="Stroke" Value="white" />

<Setter Property="Fill" Value="#4EFFFFFF" />

<Setter Property="PointGeometrySize" Value="0" />

<Setter Property="LineSmoothness" Value="0.3" />

</Style>

<Style TargetType="lvc:Axis">

<Setter Property="ShowLabels" Value="True" />

<Setter Property="IsEnabled" Value="True" />

</Style>

</Grid.Resources>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Border x:Name="Border3" Grid.Row="0" Grid.RowSpan="4" CornerRadius="5" Background="White" />

<Border Grid.Row="0" Grid.RowSpan="3" Background="#4CAF50" />

<TextBlock Grid.Row="0" TextAlignment="Center" Padding="10, 10, 0, 5" Foreground="White" FontSize="18">

GPU Load

</TextBlock>

<!-- Replace with your GPU-specific chart component -->

<lvc:CartesianChart Grid.Row="2" Margin="0, 0, 0, 0" Series="{Binding GpuLoadSeries}" Hoverable="True" DataTooltip="{x:Null}" DisableAnimations="True">

<lvc:CartesianChart.AxisX>

<lvc:Axis MinValue="0" MaxValue="20" Title="Time">

<lvc:Axis.Separator>

<lvc:Separator Step="1"/>

</lvc:Axis.Separator>

</lvc:Axis>

</lvc:CartesianChart.AxisX>

<lvc:CartesianChart.AxisY>

<lvc:Axis MinValue="0" MaxValue="100" Title="Usage (%)" >

<lvc:Axis.Separator>

<lvc:Separator Step="10"/>

</lvc:Axis.Separator>

</lvc:Axis>

</lvc:CartesianChart.AxisY>

</lvc:CartesianChart>

<StackPanel Grid.Row="3" VerticalAlignment="Center" Margin="25, 0">

<TextBlock Opacity=".4" FontSize="13">GPU Memory and Frequency</TextBlock>

<StackPanel Orientation="Horizontal">

<TextBlock Foreground="#303030" FontSize="20" Text="Temperature: " Margin="8, 6" />

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding GpuTemperature}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">°C</TextBlock>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBlock Foreground="#303030" FontSize="20" Text="Frequency: " Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding GpuFrequency}" Margin="8, 6" />

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">GHz</TextBlock>

</StackPanel>

</StackPanel>

</Grid>

</GroupBox>

<!-- Блок для загрузки RAM -->

<GroupBox Header="RAM Usage" Grid.Row="0" Grid.Column="3">

<Grid Margin="5">

<Grid.Effect>

<DropShadowEffect BlurRadius="15" Direction="-90" RenderingBias="Quality" Opacity=".2" ShadowDepth="1"/>

</Grid.Effect>

<Grid.OpacityMask>

<VisualBrush Visual="{Binding ElementName=Border1}" />

</Grid.OpacityMask>

<Grid.Resources>

<Style TargetType="lvc:LineSeries">

<Setter Property="StrokeThickness" Value="3" />

<Setter Property="Stroke" Value="White" />

<Setter Property="Fill" Value="#4EFFFFFF" />

<Setter Property="PointGeometrySize" Value="0" />

<Setter Property="LineSmoothness" Value="0.3" />

</Style>

<Style TargetType="lvc:Axis">

<Setter Property="ShowLabels" Value="True" />

<Setter Property="IsEnabled" Value="True" />

</Style>

</Grid.Resources>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Border x:Name="Border6" Grid.Row="0" Grid.RowSpan="4" CornerRadius="5" Background="White" />

<Border Grid.Row="0" Grid.RowSpan="3" Background="#9C27B0 " />

<TextBlock Grid.Row="0" TextAlignment="Center" Padding="10, 10, 0, 5" Foreground="White" FontSize="18">

RAM Load

</TextBlock>

<lvc:CartesianChart Grid.Row="2" Margin="0, 0, 0, 0" Series="{Binding RamLoadSeries}" Hoverable="True" DataTooltip="{x:Null}" DisableAnimations="True">

<lvc:CartesianChart.AxisX>

<lvc:Axis MinValue="0" MaxValue="20" Title="Time">

<lvc:Axis.Separator>

<lvc:Separator Step="1"/>

</lvc:Axis.Separator>

</lvc:Axis>

</lvc:CartesianChart.AxisX>

<lvc:CartesianChart.AxisY>

<lvc:Axis MinValue="0" MaxValue="32" Title="Usage (GB)" >

<lvc:Axis.Separator>

<lvc:Separator Step="4"/>

</lvc:Axis.Separator>

</lvc:Axis>

</lvc:CartesianChart.AxisY>

</lvc:CartesianChart>

<StackPanel Grid.Row="3" VerticalAlignment="Center" Margin="25, 0">

<TextBlock Opacity=".4" FontSize="13">RAM Total and Used</TextBlock>

<StackPanel Orientation="Horizontal">

<TextBlock Foreground="#303030" FontSize="20" Text="Total Memory: " Margin="8, 6" />

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding TotalMemory}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">GB</TextBlock>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBlock Foreground="#303030" FontSize="20" Text="Used Memory: " Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding UsedMemory}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">GB</TextBlock>

</StackPanel>

</StackPanel>

</Grid>

</GroupBox>

<!-- Нижний ряд -->

<GroupBox Header="Ethernet Usage" Grid.Row="1" Grid.Column="1">

<Grid Margin="5">

<Grid.Effect>

<DropShadowEffect BlurRadius="15" Direction="-90" RenderingBias="Quality" Opacity=".2" ShadowDepth="1"/>

</Grid.Effect>

<Grid.OpacityMask>

<VisualBrush Visual="{Binding ElementName=Border1}" />

</Grid.OpacityMask>

<Grid.Resources>

<Style TargetType="lvc:LineSeries">

<Setter Property="StrokeThickness" Value="3" />

<Setter Property="Stroke" Value="White" />

<Setter Property="Fill" Value="#4EFFFFFF" />

<Setter Property="PointGeometrySize" Value="0" />

<Setter Property="LineSmoothness" Value="0.3" />

</Style>

<Style TargetType="lvc:Axis">

<Setter Property="ShowLabels" Value="True" />

<Setter Property="IsEnabled" Value="True" />

</Style>

</Grid.Resources>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Border x:Name="Border7" Grid.Row="0" Grid.RowSpan="4" CornerRadius="5" Background="White" />

<Border Grid.Row="0" Grid.RowSpan="3" Background="#FF9800 " />

<TextBlock Grid.Row="0" TextAlignment="Center" Padding="10, 10, 0, 5" Foreground="White" FontSize="18">

Ethernet Speed

</TextBlock>

<lvc:CartesianChart Grid.Row="2" Margin="0, 0, 0, 0" Series = "{Binding DownloadSpeedSeries}" LegendLocation="Bottom" Hoverable="True" DataTooltip="{x:Null}" DisableAnimations="True" Zoom="Y" >

<lvc:CartesianChart.AxisX>

<lvc:Axis MinValue="0" MaxValue="20" Title="Time">

<lvc:Axis.Separator>

<lvc:Separator Step="1"/>

</lvc:Axis.Separator>

</lvc:Axis>

</lvc:CartesianChart.AxisX>

<lvc:CartesianChart.AxisY >

<lvc:Axis MinValue="0" Title="Speed (Mbps)" >

<lvc:Axis.Separator >

<lvc:Separator Step="10"/>

</lvc:Axis.Separator>

</lvc:Axis>

</lvc:CartesianChart.AxisY>

</lvc:CartesianChart>

<StackPanel Grid.Row="3" VerticalAlignment="Center" Margin="25, 0">

<TextBlock Opacity=".4" FontSize="13">Download and Upload Speed</TextBlock>

<StackPanel Orientation="Horizontal">

<TextBlock Foreground="#303030" FontSize="20" Text="Download Speed: " Margin="8, 6" />

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding DownloadSpeed}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">Mbps</TextBlock>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBlock Foreground="#303030" FontSize="20" Text="Upload Speed: " Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding UploadSpeed}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">Mbps</TextBlock>

</StackPanel>

</StackPanel>

</Grid>

</GroupBox>

<GroupBox Header="Disk Space Usage" Grid.Row="1" Grid.Column="2" Margin="10">

<Grid Margin="10">

<Grid.Effect>

<DropShadowEffect BlurRadius="15" Direction="-90" RenderingBias="Quality" Opacity=".2" ShadowDepth="1"/>

</Grid.Effect>

<Grid.OpacityMask>

<VisualBrush Visual="{Binding ElementName=Border5}" />

</Grid.OpacityMask>

<Grid.Resources>

<Style TargetType="lvc:Gauge">

<Setter Property="Width" Value="120" />

<Setter Property="Height" Value="120" />

<Setter Property="From" Value="0" />

<Setter Property="To" Value="100" />

<Setter Property="Value" Value="30" />

<Setter Property="LabelsVisibility" Value="Collapsed" />

<Setter Property="GaugeActiveFill">

<Setter.Value>

<LinearGradientBrush>

<GradientStop Color="Yellow" Offset="0.0" />

<GradientStop Color="Orange" Offset="0.5" />

<GradientStop Color="Red" Offset="1.0" />

</LinearGradientBrush>

</Setter.Value>

</Setter>

</Style>

</Grid.Resources>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Border x:Name="Border5" Grid.Row="0" Grid.RowSpan="4" CornerRadius="5" Background="White" />

<Border Grid.Row="0" Grid.RowSpan="3" Background="#2196F3" />

<TextBlock Grid.Row="0" TextAlignment="Center" Padding="10, 10, 0, 5" Foreground="White" FontSize="18">

Disk Space

</TextBlock>

<ComboBox Margin="112,35,113,3" Width="150"

ItemsSource="{Binding DiskDrives}" SelectedItem="{Binding SelectedDisk}">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding Name}" />

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<lvc:Gauge Grid.Row="1" Margin="127,2,128,13" Uses360Mode="True" FromColor="#FF1A37D4" Grid.RowSpan="2">

<lvc:Gauge.GaugeActiveFill>

<LinearGradientBrush>

<GradientStop Color="Yellow" Offset="0.0" />

<GradientStop Color="Orange" Offset="0.5" />

<GradientStop Color="Red" Offset="1.0" />

</LinearGradientBrush>

</lvc:Gauge.GaugeActiveFill>

<lvc:Gauge.Value>

<Binding Path="SelectedDisk.UsagePercentage" StringFormat="{}{0:F1}"/>

</lvc:Gauge.Value>

</lvc:Gauge>

<StackPanel Grid.Row="3" VerticalAlignment="Center" Margin="25, 0">

<TextBlock Opacity=".4" FontSize="13">Disk Usage</TextBlock>

<StackPanel Orientation="Horizontal">

<TextBlock Foreground="#303030" FontSize="20" Text="Used Space: " Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding SelectedDisk.UsedSpace}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">GB</TextBlock>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBlock Foreground="#303030" FontSize="20" Text="Total Space: " Margin="8, 6" />

<TextBlock Foreground="#303030" FontSize="20" Text="{Binding SelectedDisk.TotalSpace}" Margin="8, 6"/>

<TextBlock Foreground="#303030" FontSize="20" VerticalAlignment="Bottom" Margin="8, 6">GB</TextBlock>

</StackPanel>

</StackPanel>

</Grid>

</GroupBox>

<GroupBox Header="Settings" Grid.Column="2" Grid.ColumnSpan="2" Margin="426,352,0,0" Grid.RowSpan="2">

<Grid>

<Button Content="Settings" HorizontalAlignment="Center" VerticalAlignment="Center" Width="200" Height="100" Click="SettingsButton\_Click"/>

</Grid>

</GroupBox>

</Grid>

</Window>