Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

за спеціальністю 123 «Комп'ютерна інженерія»

Тема наукової роботи: МЕТОДИ ОПТИМІЗАЦІЇ МОНІТОРИНГУ ВИКОРИСТАННЯ РЕСУРСІВ ПК

Виконав _____ Д. К. Балик

Керівник роботи _____ А. О. Сенько

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2024

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти магістр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"____" _____ 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Балик Дмитро Костянтинович
(прізвище, ім'я, по батькові)

1. Тема роботи Методи оптимізації моніторингу використання ресурсів пк

керівник роботи Сенько Антон Олександрович, кандидат технічних наук,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "___"_____ 20__року №__

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи Аналіз існуючих методів моніторингу ресурсів ПК та їх оптимізації; створення програмного забезпечення для ефективного збору і візуалізації даних із мінімальним використанням системних ресурсів. Потреба збору даних швидкодії ПК у мережі

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вивчення існуючих методів та програмних рішень, Аналіз статистики та розробка алгоритмів, Розробка програмного рішення

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Презентація *Microsoft PowerPoint*;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел за темою	27.09.2024 – 30.09.2024	
2	Аналіз існуючих рекомендаційних систем для пошуку та організації заходів	01.10.2024 – 10.10.2024	
3	Аналіз методологій взаємодії	11.10.2024 – 14.10.2024	
4	Аналіз предметної області	15.10.2024 – 20.10.2024	
5	Підготовка матеріалів першого розділу роботи	21.10.2024 – 26.10.2024	
6	Розробка алгоритмів розроблюваного програмного забезпечення	27.10.2024 – 01.11.2024	
7	Підготовка матеріалів другого розділу роботи	02.11.2024 – 08.11.2024	
8	Підготовка матеріалів третього розділу роботи	09.11.2024 – 16.11.2024	
9	Реалізація програмного модуля	17.11.2024 – 20.11.2024	
10	Оформлення пояснювальної записки	20.11.2024 – 30.11.2024	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 82 сторінок, 31 рисунків, 8 таблиць, 1 додаток, 20 використаних джерел.

Об’єкт дослідження – процеси моніторингу обчислювальних ресурсів ПК та управління з метою оптимального розподілу навантаження

Кваліфікаційна робота включає три частини.

Перший розділ присвячено аналізу існуючих рішень та збору деяких досліджень з теми. Обґрунтовано необхідність розробки оптимізованого рішення.

Другий розділ присвячений дослідженню ефективності алгоритмів, точністю отриманих даних та видобуванню корисних даних з статистики.

У третьому розділі реалізовано визначені алгоритми у клієнтській програмі з сервером та інструментами для аналізу статистики.

Ключові слова: КОМП’ЮТЕРНА МЕРЕЖА, МОНІТОРИНГ,

СЕРВЕР, РОБОЧА СТАНЦІЯ, ПЕРСОНАЛЬНИЙ КОМП’ЮТЕР, СТАТИСТИКА, СЕНСОРИ, ЯДРО ОПЕРАЦІЙНОЇ СИСТЕМИ.

Master’s work: 82 pages, 31 figures, 8 tables, 1 additions, 20 used sources.

The object of research is the processes of monitoring PC computing resources and management for the purpose of optimal load distribution

The work consists of three sections.

The first section is devoted to the analysis of existing solutions and the collection of some research on the topic. The need to develop an optimized solution is justified.

The second section is devoted to experiments on the efficiency of algorithms, the accuracy of the obtained data and the extraction of useful data from statistics.

In the third section, the specified algorithms are implemented in a client program with a server and tools for statistical analysis.

Keywords: COMPUTER NETWORK, MONITORING,SERVER, WORKSTATION, PERSONAL COMPUTER, STATISTICS, SENSORS, KERNEL

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AES - (Advanced Encryption Standard) специфікація для шифрування електронних даних

API - (application programming interface) інтерфейс який дозволяє кільком програмам взаємодіяти між собою.

CPU - (Central processing unit) Центральний процесор.

IP - (Internet Protocol address) — унікальний числовий ідентифікатор мережевого рівня.

LAN - (Local area network) Локальна комп’ютерна мережа

RAM - (Random Access Memory) Оперативна пам’ять.

WMI - (Windows Management Instrumentation) інструментарій керування Windows.

WQL - (Windows Management Instrumentation Query Language) Мова запитів WMI це підмножина Американського національного інституту стандартів мови SQL з незначними семантичними змінами.

XML - (EXtensible Markup Language) Розширювана мова розмітки.

ПК - Персональний комп’ютер.

ВСТУП

Актуальність теми

Сучасний розвиток інформаційних технологій значно підвищив вимоги до ефективного використання ресурсів персональних комп’ютерів. Це особливо важливо в умовах високих обчислювальних навантажень, що виникають у різних сферах, таких як наукові дослідження, розробка алгоритмів штучного інтелекту, аналіз великих даних, моделювання складних систем, автоматизація процесів тощо. Зростаючі обсяги інформації та обчислень спричиняють необхідність постійного моніторингу ресурсів ПК для забезпечення їх стабільної та енергоефективної роботи.

Оптимізація моніторингу є ключовим завданням, яке дозволяє не лише уникнути збоїв у роботі систем, але й вчасно виявляти потенційні проблеми, наприклад, перегрів компонентів, надмірне використання пам’яті чи надмірні витрати енергії. У серверних середовищах це критично важливо для підтримки безперервності бізнесу, а для персональних комп’ютерів — для забезпечення продуктивної роботи користувачів. Особливої актуальності це питання набуває в умовах віддаленої роботи, коли комп’ютери використовуються як основний інструмент для виконання повсякденних завдань.

Ця проблема є актуальною як для серверних середовищ, так і для персональних систем, які працюють у режимі перевантаження. При цьому сама система моніторингу повинна мінімізувати власний вплив на продуктивність ПК.

Системи моніторингу виконують такі функції:

1. Супровід інвентаризації апаратного забезпечення.
2. Виявлення проблем, які можуть вплинути на безперервність роботи.
3. Локалізація "вузьких місць", що знижують продуктивність.
4. Оптимальний розподіл фінансових ресурсів.

Об'єкти дослідження

Системні ресурси ПК, операційна система Windows, центральний процесор, оперативна пам'ять, жорсткі та твердотільні диски, графічний процесор та мережеві інтерфейси. Особлива увага приділяється процесам моніторингу й управління ресурсами для мінімізації навантаження.

Предмети дослідження

Методи та алгоритми оптимізації моніторингу системних ресурсів ПК з метою підвищення їх продуктивності та енергоефективності в різних режимах роботи.

Мета дослідження

Розробка та вдосконалення методів моніторингу ресурсів ПК для забезпечення їх ефективної роботи з мінімальним впливом на систему. Метою також є пропозиція способів виявлення аномалій у роботі системи або мережі, а також отримання алгоритмів для обробки й аналізу даних, зібраних під час моніторингу.

Методи дослідження

Використовуються алгоритми оптимізації, статистичний аналіз для виявлення аномалій, методи часових рядів для аналізу змін параметрів у динаміці. Ефективність алгоритмів і програмних інструментів порівнюється експериментально.

Матеріали дослідження

Дослідження зосереджено на створенні інструменту, який витрачає мінімум системних ресурсів, одночасно здійснюючи моніторинг великої кількості підсистем і забезпечуючи розширені можливості аналізу даних. Завданням є пошук найкращих підходів до збору та обробки даних на основі наукового аналізу.

Наукова новизна

У роботі запропоновано нові підходи до оптимізації систем моніторингу ресурсів ПК, орієнтовані на зниження впливу на продуктивність та енергоефективність. Розроблено алгоритми, що дозволяють автоматично виявляти приховані аномалії в роботі системи, аналізувати взаємодію між компонентами та оптимізувати передачу даних. Додатково запропоновано методи інтегрованого аналізу, які враховують одночасну роботу кількох підсистем.

Практичне значення

Результати дослідження можуть бути використані для підвищення ефективності персональних ПК та серверів, забезпечення стабільності роботи системи, зменшення енергоспоживання та раннього виявлення критичних збоїв. Запропоновані методи можуть інтегруватися в існуючі системи моніторингу для оптимізації роботи в умовах високих навантажень та підвищення загальної продуктивності.

Апробація результатів

Результати дослідження були апробовані на конференції КІСМ-2023 Криворізького національного університету, що підтверджує їх наукову та практичну значущість.

1 ВИВЧЕННЯ ІСНУЮЧИХ МЕТОДІВ ТА ПРОГРАМНИХ РІШЕНЬ

1.1 Огляд проблеми

Задача моніторингу обчислювальних систем та їх стану існує давно, тому є велика кількість програмних рішень для різних спеціалізацій які можна розділити на такі категорії як: активність мережі, дискова пам’ять, обчислювальна потужність, активність користувачів, помилки програм та інші категорії які можуть бути створені відповідно до задачі промисловості.

У даному проекті буде розглядатись сценарій використання програми у середовищі моніторингу комп’ютерів та серверів у всій мережі на завантаженість та стан апаратного забезпечення. Такий моніторинг може допомогти з фінансовими рішеннями, підвищення комфорту роботи співробітників та запобігання простоїв у роботі.

Основна категорія такого моніторингу це звісно обчислювальна потужність пристроїв. Але оскільки основною операційною системою за якою буде проводитись моніторинг у такій ситуації зазвичай буде Windows то є можливість відстеження використання пропускної можливості мережі кожним користувачем. Це дає можливість статистично визначити чи вистачає встановлених мережевих пристроїв та їх пропускної можливості для обслуговування клієнтів.

Моніторинг апаратних ресурсів має на меті:

- Дати можливість налаштовувати події та сповіщення на підозрілу активність або високу температуру.
- Покращення планування обслуговування систем.
- Балансувати використання комп'ютерних систем на основі їх реальної завантаженості або умов використання.
- Контроль необхідності використання місця для зберігання даних, потрібність нового сервера чи розширення пам'яті.
- Аналіз зібраних даних моніторингу для планування розвитку та прийняття рішень по покращенню роботи.

Система моніторингу орієнтована на моніторинг великої мережі має відповідати таким вимогам:

- Масштабованість
- Централізоване керування
- Низьке навантаження на клієнтські пристрої
- Моніторинг у реальному часі і сповіщення
- Безпека даних
- Аналіз та звітність
- Автоматизація дій
- Гнучкість налаштувань
- Відмовостійкість та резервування

Треба зауважити що всі ці вимоги можуть відрізнятись від потреб замовника та цей список сформований з використанням декількох стандартів IT серед яких: ISO/IEC 20000, ISO/IEC 27001, ISO 22301, CoBiT (Control Objectives for Information and Related Technologies). Є і інші стандарти та поради, але усі вони повторюють принципи відносно моніторингу які перелічені вище: (Information Technology Infrastructure Library), RFC 3411-3418 (SNMPv3 Стандарт).

Додатково, система моніторингу має враховувати особливості інтеграції з існуючими корпоративними системами, включаючи можливість роботи з базами даних, платформами бізнес-аналітики та інструментами управління ресурсами. Це дозволяє не лише отримувати дані в реальному часі, а й аналізувати історичні тенденції, що сприяє довгостроковому плануванню та оптимізації.

Важливою складовою є також підтримка стандартів сумісності та інтероперабельності, що забезпечує легке впровадження у вже наявну IT-інфраструктуру. Наприклад, можливість роботи через API.

Особливу увагу слід приділити безпеці системи моніторингу. У контексті великої мережі критично важливо запобігати несанкціонованому доступу до даних або втручанню в роботу системи. Це досягається через використання шифрування даних, багатофакторної автентифікації, контролю доступу на основі ролей (RBAC) та забезпечення відповідності вимогам стандартів інформаційної безпеки.

Крім того, сучасні системи моніторингу мають враховувати можливість роботи з гібридними середовищами, де частина ресурсів розташована у локальній мережі, а частина — у хмарних інфраструктурах. Такі функції дозволяють об’єднати моніторинг локальних і хмарних компонентів у єдине середовище, забезпечуючи цілісну картину стану системи.

Ще одним важливим аспектом є підтримка адаптивності системи до змін у мережі. Зростання кількості пристроїв, збільшення обсягів даних чи зміни топології мережі не повинні суттєво впливати на продуктивність системи моніторингу. Це особливо актуально для великих організацій, де зміни можуть відбуватися часто, а їх відстеження потребує високої гнучкості.

Впровадження сучасної системи моніторингу має розглядатися не лише як технічний інструмент, але й як частина стратегічного підходу до управління IT-ресурсами, який спрямований на оптимізацію роботи, підвищення надійності та готовності до майбутніх викликів.

1.2 Існуючі програмні рішення

Оскільки проблема моніторингу у мережах існує давно то і різних існуючих рішень з’явилось доволі багато. Якщо розглядати тільки ті які орієнтовані на моніторингу у мережі можна виділити такі: Prometheus, Zabbix, Nagios, Splunk, Elastic Stack

Prometheus, збір завантаженості ПК не підтримується на пряму сама служба є лише колекціонером, тому для роботи потрібен агент Node Exporter, який збирає метрики про використання CPU, пам'яті, диску, мережі та інших системних показників.

Grafana є веб додатком для відображення отриманих даних, тому для збору можна використовувати все що завгодно, навіть власне рішення для дотримуватись протоколу передачі.

Zabbix, Nagios мають такий самий принцип роботи за виключенням, що агенти власні.

Splunk та Elastic Stack збирають дані з логів, що вже робить таку систему не надто сприятливою до створення оповіщень, бо запізнення буде завжди доволі великим. Але він може інтегруватися з іншими, що збільшує використання ресурсів на моніторинг, але перекривають недолік.

Більшість таких рішень мають високий рівень підлаштування під користувача, тобто можна встановити стільки сенсорів скільки потрібно, але у більшості презентаційних матеріалів, розробники самі попереджають про велике навантаження від додавання та налаштувань великої кількості сенсорів.

Детальніше дослідивши систему Nagios виявилось, що стандартний агент має не так багато детальних даних. А саме Моніторинг CPU включає в себе завантаження кожного ядра, середнє максимальне та мінімальне у реальному часі. Моніторинг диску повністю покладається на моніторинг зайнятого місця. Моніторинг інтерфейсу це статистика основного мережевого порту та його швидкість. Моніторинг пам’яті це збір використання віртуальної та оперативної пам’яті. Процеси та служби не представляють з себе перегляд запущених, а більше як перевірку чи запущений процес який уже завчасно відомий. Також дозволяє переглянути активних користувачів системи. Звісно головною перевагою тут називають підтримку плагінів, але вони сильно знижують простоту розгортання додатка.

З цієї системи хотілося б запозичити хіба, що перегляд користувачів, все ж таки реалізувати перегляд усіх процесів і збір даних про навантаження на кожне ядро окремо може бути корисним для виявлення не ефективно працюючого в один потік додатку. Тому замість збереження великої кількості даних можна зробити аналізатор даних який буде перевіряти чи не було перевантажене одне з ядер дуже довго неперервною задачею.

Якщо виділити з усіх чотири основних представника то серед них можна провести оцінку схематичний результат якої відображений на рисунку 1.1. Оцінка проводилась майже суб'єктивно, бо базується на відгуках з відкритих джерел, заявках видавців та форумах з рішеннями типових проблем, що до кожної з систем.

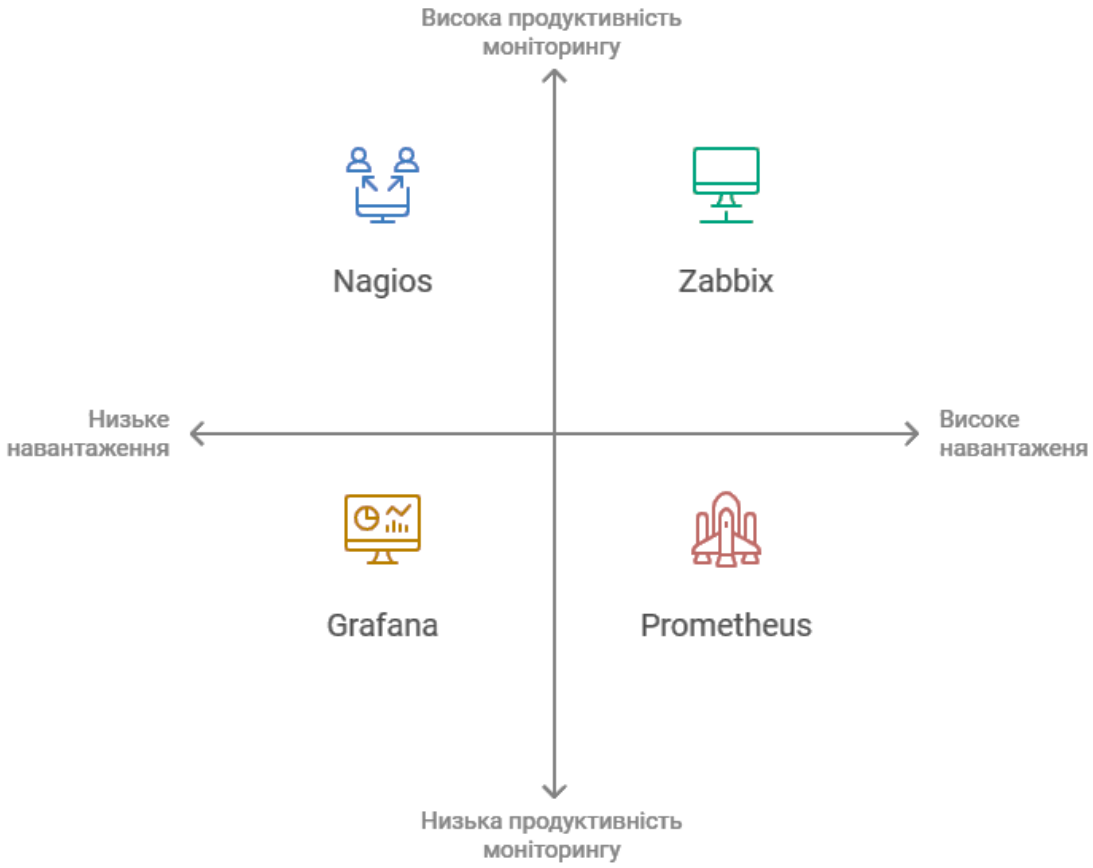

Рисунок 1.1 – Умовне порівняння моніторингових систем

Для більшої точності слід пояснити чому кожна система потрапила саме на своє місце. Найкраще місце тут віддано Nagios, через хороші відгуки про відносну простоту налаштування що підвищує продуктивність моніторингу та детальну документацію на випадок потреби давання додаткових сенсорів у систему. З навантаженням складніше, адже повноцінно оцінити систему без повного встановлення та тестування. Тому тут оцінка була визначена з заявок видавців та лаконічності базових наборів моніторингу.

Zabbix знаходиться на місці високого навантаження і одночасно високої якості моніторингу через програмні рішення які з самого початку потребують розміщення на потужному сервері. Компанія з самого початку пропонує повністю перевести моніторинг у хмару, що передбачає шифрування даних для мережі інтернет та складних систем адміністрування. Але одночасно має велику базу початкових параметрів та компанія запевняє у свої підтримці з боку розширення системи для особливих випадків. Але і інша сторона це користувачі які запевняють, що іноді затримка повідомлень першої

важливості надто запізнюються та моніторинг повноцінно можливо розмістити у локальній мережі. Моя особиста думка, що проблеми у обох користувачів виникли через спробу роботи активний моніторинг в умовах перевантаженості каналу.

Наступною на мій погляд вдалою системою треба оглянути Grafana. Сама названа натякає на основний аспект програми, це відображення даних у найзручнішій формі. Увесь сервіс оформлений у формі веб-додатку, що зараз актуально для бізнесу, але вона використовує інші системи як бек-енд, що на мій погляд повністю знижує простоту встановлення. Звісно відповідальність за рівень навантаження повністю лежить на іншій системі яка збирає дані, тому якість моніторингу може бути сумнівною.

Система Prometheus є більше двигуном моніторингу, а не готовим рішенням адже його треба налаштовувати з нуля. Перевага в тому що підключити до будь-якої іншої програми як розширення. З недоліків звісно заповнені форуми з вирішенням проблем, адже деякі моніторингові процеси доведеться майже програмувати самостійно.

Рішення для звичайних користувачів дуже часто не мають такого рівня налаштувань, але мають ширший діапазон вже облаштованих сенсорів моніторингу, але відсутність систем збору статистики та мережевої взаємодії для синхронізації статистики не дають повноцінної картини для оцінки результатів моніторингу.

Моніторинг мережі теж важливий, але він не підпадає під основний напрям роботи. У цій сфері моніторингу теж є існуючі служби, але вони зазвичай все ж використовують SNMP або функції які є тільки у пристроях конкретних виробників.

NetFlow, вперше реалізований на маршрутизаторах Cisco ще до 2000 року, дає змогу відстежувати IP-трафік, що проходить через інтерфейси пристроїв. Ця технологія дозволяє мережевим адміністраторам отримувати цінну інформацію про джерела й призначення даних, рівень обслуговування та причини виникнення перевантажень у мережі.

Моніторинг трафіку за допомогою NetFlow зазвичай базується на трьох етапах. Спочатку дані пакетів групуються у потоки й передаються до приймача. Далі вони зберігаються та обробляються, щоб підготувати їх до подальшого аналізу. На завершальному етапі спеціалізовані програми використовують ці дані, наприклад, для аналізу трафіку або виявлення потенційних загроз.

Paessler PRTG (Paessler Router Traffic Grapher) — ще одне програмне забезпечення для моніторингу мережі. Відстежує умови системи, такі як використання пропускної здатності або час безвідмовної роботи, і збирає статистику з різних хостів, таких як комутатори, маршрутизатори, сервери та інші пристрої та програми.

Підтримує багато способів комунікації таких як ICMP, SNMP, WMI, NetFlow, jFlow, sFlow, DCOM або RESTful API. та має власну систему виявлення [4]. Має веб інтерфейс для комунікації з людиною.

NTOPNG сучасний інструмент для моніторингу та аналізу мережевого трафіку, що забезпечує глибоке розуміння стану мережі в реальному часі. Побудований на старому ntop. Він створений як зручний і потужний засіб для адміністраторів мереж, розробників і аналітиків безпеки, надаючи широкий спектр можливостей для збору, обробки та відображення даних про активність у мережі.

Програма працює у вигляді мережевого зонда, який здатний аналізувати трафік як локальної мережі, так і зовнішніх підключень. Завдяки інтуїтивному веб-інтерфейсу, користувачі можуть отримувати вичерпну інформацію про використання пропускної здатності, виявляти проблеми з продуктивністю або безпекою, аналізувати взаємодію між пристроями та відслідковувати аномальну активність.

NTOPNG забезпечує підтримку сучасних протоколів, таких як IPv6, і здатний працювати з даними з різних джерел, включаючи NetFlow, sFlow і SNMP. Це дозволяє інтегрувати його з іншими системами моніторингу та управління. Система забезпечує детальний аналіз кожного потоку даних, визначаючи, які сервіси або програми використовують найбільше ресурсів.

Окрему увагу приділено аспектам безпеки. Інструмент допомагає виявляти загрози в мережі, такі як підозрілі підключення, сканування портів чи невідомі пристрої. Завдяки своїм алгоритмам обробки, NTOPNG може виявляти патерни поведінки, які вказують на потенційні ризики.

Завдяки відкритій архітектурі, програму можна розширювати та налаштовувати під конкретні потреби організації. Її використовують як для простого моніторингу домашніх мереж, так і для забезпечення стабільності та безпеки великих корпоративних систем. NTOPNG також інтегрується з базами даних і платформами аналізу, що дозволяє зберігати історичні дані для глибшого аналізу.

Таким чином, NTOPNG є універсальним інструментом, що поєднує зручність у використанні з потужним функціоналом для всебічного аналізу мережевої активності.

.

Інтерфейс як і всіх інших представлена веб сервером заснованого на HTML5 і Ajax веб-інтерфейсу.

Невеликий двигун програми, розумне споживання пам'яті та відмовостійкість. Можливість визначити протокол програми за допомогою nDP, DPI (Deep Packet Inspection) фреймворку ntop з відкритим вихідним кодом. Можливість користувачів використовувати скрипти, розширювати та модифікувати сторінки ntopng програмуючи за допомогою LuaJIT. Можливість збирати потоки - NetFlow використовуючи nProbe як зонд/проксі. Швидкий, дуже швидкий двигун, що масштабується до 10 Гбіт. Підтримка Unix, BSD, Mac OS та Windows.

1.3 Безпека у системі моніторингу.

Один з компонентів який вимагають стандарти ISO/IEC 20000, ISO/IEC 27001 це система безпеки.

Тобто парольний доступ, та шифрування даних по мережі. Але запланована у цій роботі програма не матиме ні яких елементів керування на стороні серверу, що виключає шкоду користувачам при проникненні. Шифрування по мережі збільшує навантаження і хоч у парі зі стисканням може дати вигоду, підвищує навантаження на сервер. При розкритті даних в мережі зловмисник може дізнатись імена ПК та можливо визначити топологію мережі по призначенню пакетів. Зазвичай ці знання не надто допомагають зловмиснику зі зламом, але у реальній програмі варто все ж додати шифрування хоча б і на частини пакетів. Такий підхід є нестандартним та як правило не виправданим.

Щодо додаткового захисту паролями, для серверної системи теж немає сенсу оскільки вона реалізована за допомогою внутрішнього інтерфейсу і пароль до серверу є паролем до моніторингової системи. Якщо потрібно буде зробити доступ з веб-сторінки тоді, додатковий рівень аутентифікації потрібен у будь якому разі. Що до захисту паролем агентів, то залежно від потреб адміністраторів можна не надавати доступ до інтерфейсу взагалі. Але якщо такої потреби немає і співробітникам дозволений доступ до перегляду власних моніторингових даних то пароль потрібен, бо треба якось заблокувати доступ до налаштування та скидання статистики. Таке блокування можна легко влаштувати дати можливість адміністратору додати пароль, а пароль зберегти на комп'ютері зашифрованим. Можна використовувати звичайний sha256 або шифрування AES. У реальній програмі плюсом також було б використовувати автентифікацію Windows на основі доменного адміністратора.

1.4 Напрямок розробки та обґрунтування актуальності проекту

Системи моніторингу комп'ютерних ресурсів є невід'ємною частиною сучасних інформаційних технологій, забезпечуючи стабільну роботу пристроїв та попередження можливих збоїв. З огляду на зростаючі обсяги даних, підвищені вимоги до продуктивності та енергоефективності, розвиток і вдосконалення таких систем стає актуальним. Інтеграція нових методів аналізу, штучного інтелекту та автоматизації дозволить підвищити рівень діагностики, ефективність управління ресурсами та забезпечити проактивний підхід до підтримки комп'ютерних систем.

Проаналізувавши існуючі аналоги та дослідження у сфері моніторингу. Були визначені запропоновані покращення та основні напрямки розробки нової системи.

Більшість аналогічних систем моніторингу мають високе навантаження на локальну мережу та на головний сервер. Основна причина щосекундне передавання даних. Якщо системний адміністратор хоче бачити більше метрик то він вимушений додавати їх, і вони також будуть оновлюватись з такою ж частотою. Системи збору статистики зберігають передані данні цілком, або ж не зберігають взагалі виводячи одразу глобальну

статистику. Ці два існуючі підходи теж направлені на оптимізацію, перший підхід розкриває повну свободу дій при аналізі даних, бо фактично зберігає сирі дані, таким чином моніторинг майже не використовує додаткових ресурсів окрім самого збору. Підхід з глобальної статистикою націлений на мінімальне використання місця, виконання такої агрегації може виконуватись як на сервері так і на пристроях які моніторяться, це призводить до повної втрати вільного аналізу даних бо ніякі часові періоди не зберігаються.

Мета розробки запропонувати таку агрегацію даних яка могла б зменшити кількість передаваної інформації та одночасно зберегти часові ряди даних та можливість подальшого аналізу. Концепція ідею це зниження навантаження на мережу яке досягти агрегацією даних ще на стороні відправника, витистакючи з кожного хвилинного набору даних проаналізовані статистичні дані які можуть знадобитись при подальшому аналізі. Такий підхід зумовлений тим що стиснуті таким чином дані будут нести максимум корисної інформації, проте це виключає моніторинг у реальному часі, який зазвичай не потрібний, бо дані за останню хвилину моніторингу мають надавати ту ж саму інформацію що отримані сирі дані на графіку.

Що до аналізу даних можна виділити, що мало які системи моніторингу пропонують якісь інструменти для аналізу, більшість з них є плагінами, а вбудовані дозволяють зробити тільки якийсь поверхневий аналіз серед пристроїв. Натомість аналіз “на місці” міг би покращити швидкість реагування на інциденти та зменшувати час очікування працівників. Серед таких інструментів які засновані на аналізі отриманих статистичних даних запропоновано, інструменти для:

- Визначення перевантаженого компонента
- Визначення орієнтовного часу обслуговування
- Виявлення аномальної роботи процесів
- Визначення присутності зависань або аномалій в роботі комп’ютера

Інструмент визначення перевантаженого компонента знадобиться при розподілені фінансів. У великій компанії є дуже корисним компонентом управління, економити бюджет покращуючи обладнання саме там де потрібно, а не за схемами пріоритету і можуть базуватись на програмах які використовують працівники.

Визначення орієнтовного часу обслуговування це теж інструмент націлений на економію робочого часу. Мета покращити планування та витрати на технічне обслуговування.

Підсистема виявлення аномальної роботи процесів це імітація інструментів у деяких антивірусах, якщо компанія не має коштів на придбання повноцінних систем захисту вона може скористатися такою не великою службою яка робить перепис процесів які поводяться всупереч роботи користувача або можуть заважати роботі. Переглянувши такий список можна побачити і явно зайві служби у автозапуску і можливо помітити роботу небезпечного софту.

Процес визначення присутності зависань або аномалій в роботі комп’ютера взагалі не має бути окремим інструментом. По перше тому що

виявивши аномалію її одразу можна задокументувати та агрегувати дані за період, а також можна використати як додаткову систему оповіщень якщо перевантаження відбуваються постійно.

Висновок до розділу 1

В даному розділі було розглянуто основні проблеми та недоліки існуючих систем моніторингу комп'ютерних ресурсів, а також запропоновані концептуальні вдосконалення для створення нової системи, яка б оптимально відповідала сучасним вимогам.

З виділених недоліків є високе навантаження на мережу та сервер через щосекундне оновлення даних. Визначено обмеження та недоліки традиційних підходів до зберігання даних, збереження сирих даних або лише глобальної статистики. Запропоновані рішення на ці проблеми це використання попередньої агрегації даних на стороні відправника для зменшення обсягу переданих даних. Це дозволить зберегти часові ряди та забезпечити можливість подальшого аналізу, водночас значно знизивши навантаження на мережу. Заміна реального часу на щохвилинний брифінг для отримання суттєвої інформації з мінімальними витратами ресурсів.

Також через нестачу готових рішень аналізу статистики були запропоновані легкі інструменти аналізу для визначення перевантажених компонентів для ефективного використання бюджету, прогнозування часу обслуговування для оптимізації планування технічних робіт, виявлення аномалій у роботі процесів, що дозволить покращити безпеку та зменшити вплив небажаного програмного забезпечення.

Актуальність запропонованих вдосконалень спрямовані на забезпечення енергоефективності, зниження витрат ресурсів та покращення якості аналізу, що робить систему більш зручною до сучасних умов. Розробка такої системи дозволить значно підвищити ефективність моніторингу та управління комп'ютерними ресурсами, зробивши її незамінним інструментом в умовах сучасних інформаційних технологій.

2 АНАЛІЗ СТАТИСТИКИ ТА РОЗРОБКА АЛГОРИТМІВ

2.1 Розробка алгоритмів збору

Серед методів оптимізації моніторингу націлених на зменшення навантаження, ефективний та структурований збір даних можна виділити такі:

- Регулювання частоти оновлення - підхід полягає у зменшенні моніторингових запитів якщо комп’ютер і так перевантажений.
- Агрегування та компресія даних - проведення попереднього аналізу з метою зберігання та передачі меншої кількості інформації без суттєвої втрати даних.
- Кешування даних - зберігання у пам’яті даних які не часто змінюються, але вони мають бути під рукою для читання. Це можуть бути дані про комплектацію комп’ютера або фізичний стан диску.
- Ієрархічний моніторинг - збирання даних з верхніх шарів автоматично отримуючи дані і про нижні. Наприклад збираючи інформацію про частоту процесора автоматичні отримувати і температури, що може допомогти виявити перегрів і пояснити зниження частот.
- Обмеження процесів моніторингу - другий фактор регулювання навантаження це якщо сама програма моніторингу на конкретному комп’ютері перевищує споживання ресурсів, то кількість запитів доведеться знизити або пропускати оновлення інтерфейсу якщо він є причиною.

Що до точності отриманих моніторингом даних сподіватись на велику точність не доводиться, бо система в першу чергу виконує завдання від користувачів або служб, а на моніторингу пріоритет завжди низький. Реальний час опитування та перерахунок можуть відрізнятись в залежності від навантаження. Інший фактор це зниження частот що використовується у сучасних системах для економії енергії. Відповідно показники навантаження та час розрахунків буде знижуватись разом з частотою.

У системі Windows є багато API які дозволяють отримати дані моніторингу, але вони є різного типу. Низькорівневі які взаємодіють з даними з ядра і більш високого рівня які працюють повільніше бо взаємодіють з ядром роблячи до нього додаткові запити. Для розробки ефективного рішення треба дослідити чи є взагалі сенс використання інтерфейсів високого рівня, та наскільки більше навантаження вони створюють та чи є різниця даних.

Оскільки запланована система має надсилати дані у локальній мережі на сервер, то не останнє місце має оцінка додаткового навантаження на мережу від роботи системи. У оптимізації цього процесу допоможе максимальна агрегація даних. а для впевненості у тому що агрегація допомогла можна розрахувати навантаження на основний сервер та на проміжні вузли і отримати наскільки вдалось знизити навантаження. Під агрегацією даних мається на увазі створення описової статистики для однакових відрізків часу. Таким чином ми тримаємо ті ж самі дані які захочемо отримати на сервері, але не будемо передавати весь ряд даних. Натомість треба зберегти максимальну кількість відомостей за проміжок часу для якісного аналізу.

Щоб впевнитись у швидкості роботи програми, будемо використовувати діагностичний клас Stopwatch він буде давати точні результати про час. Системний таймер windows має частоту 64 Гц і помітити вигоду у зміні швидкості деяких операцій через DataTime.Now майже не можливо, бо вони як правило виконуються швидше ніж 15 мс. і планувальник якщо може то організовує перемикання максимально правильно і виконуваний процесором код може перемикатись швидше за системний таймер. Щоб побачити зміну в часі ми можемо використати саме клас Stopwatch бо для вимірів він використовує не системний таймер, а High Precision Event Timer який можна ввімкнути у диспетчері пристроїв. Цей таймер згідно зі довідкою Microsoft має частоту 10 кГц, Що правда його програмна реалізація виводить результати з роздільною здатністю 10000000 тиків на секунду, що у 1000 разів більше.

2.1.1 Порівняння інструментів моніторингу у системі Windows

Як вже зазначалось система Windows має декілька інтерфейсів для взаємодії з ядром щодо взяття моніторингових даних. Деякі з них позиціонуються як інструменти вищого рівня, а деякі як більш низькі.

Для правильного проведення експерименту треба визначитись з таким набором параметрів:

- керованими змінними
- цільові функції
- обмеження на змінні
- множину пошуку рішень.

Керовані змінні, параметри, які можна змінювати для досягнення оптимального рішення.

У нашому випадку це метод збору даних, тобто API які надає Windows, а саме ManagementObjectSearcher (WMI), Performance Counter та прямі запити

до ядра через kernel32.dll. Та типи метрик, які збираються, а саме відсоток використання CPU, відсоток використання RAM, відсоток використання диску, інформація навантаження від кожного процесу.

Цільові функції визначають мету оптимізації. Для даного аналізу це мінімізувати час виконання запиту вимірюючи у тиках таймера, відповідно мінімізувавши споживання системних ресурсів, особливо це стосується процесорного часу. Гнучкість даних тобто отримання складних структур, таких як інформація про всі процеси. Отримувати актуальні данні без великої затримки.

Обмеження на змінні. Обмеження методів збору даних, використовуються виключно визначені інструменти, Performance Counter, WMI, запити до ядра. Типи даних для збереження не мають містити нічого зайвого. Частота збору висока близько 1 секунди для Performance Counter та Запитів до ядра. Для WMI частота запитів зменшена до 5 секунд.

Множина пошуку рішень або межі припустимих рішень це швидкість збору даних для реального часу та нормальної аналітики з агрегацією не повинна перевищувати затримки, яка унеможливить оперативний моніторинг, це не більше 1 секунди. Ресурсне обмеження тут виражається у використанні процесора звісно бажана максимальна мінімізація, навантаження на процесор не повинно впливати на інші критичні процеси та сумарне навантаження якщо оцінювати у часі, не має перевищувати десятої частини секунди на усі потоки, інакше усі заходи мінімізації можна вважати невдалими та не потрібними.

Основним інструментом зазвичай рекомендується Performance Counter, але як довели мої порівняння він найважчий серед аналогів, але звісно найпростіший у реалізації.

Після нього йде ManagementObjectSearcher або іншими словами WMI інтерфейс він є інструментом вже більш низького рівня бо взаємодіє зі службою яка в системі працює фоновим процесом та є невеликим містом між ядром та користувачем. Перевага те що вона зберігає у віртуальну базу даних усі дані, недолік це її низька продуктивність, бо більшість даних шукаються у ній з алгоритмом O(n) складності. Запити робляться у форматі WQL, це фактично SQL, але трохи спрощений.

Функції kernel32.dll Це фактично найнижчий рівень взаємодії, через нього швидкість доступу буде найшвидшою і залежати буде тільки від реального відсотку простою процесору і як маршал потоків запланує перемикання, але тут підвищується складність реалізації коду, бо доведеться більшість параметрів обчислювати вручну з сирих не опрацьованих даних, але таким чином я отримую повний контроль над процесами які протікають під час обчислення наступного значення моніторингу.

Для порівняння інструментів моніторингу у Windows, таких як ManagementObjectSearcher і Performance Counter і пряме звернення до ядра, варто враховувати кілька ключових аспектів:

- Швидкість виконання - час виконання у тиках або мілісекундах.
- Гнучкість - які дані можна отримати за період даних.
- Споживання ресурсів (навантаження на систему).

Порівнювати кожен спосіб на основі типових завдань: відсоток використання оперативної пам’яті, CPU, диску та отримання списку процесів з деякими параметрами використання ресурсів ними.

Клас ManagementObjectSearcher отримує колекцію об’єктів на основі вказаного запиту. Його зручно використовувати для перерахування всіх дисків, мережевих адаптерів, процесів і багатьох інших об’єктів керування в системі. При створенні цей клас приймає як вхідні дані запит WMI та інші параметри для уточнення запиту або зменшення часу його виконання за рахунок зменшення області пошуку. Він також може приймати додаткові розширені параметри в EnumerationOptions. Коли для цього об’єкта викликається метод Get, ManagementObjectSearcher виконує заданий запит та повертає колекцію об’єктів керування, які відповідають запиту.

Переваги інтерфейсу: легко отримати дані через запити WQL, Широка підтримка практично всіх можливих параметрів системи.

Недоліки цього інтерфейсу: висока затримка на виконання запитів, відносно велике споживання ресурсів, Видає результат погуком серед інших параметрів та іноді дає результат з зайвими полями.

Провівши заміри часу виконання запитів протягом 10 хвилин та обчислено середні значення було отримано результати продемонстровані на рисунку 2.1. Величини у яких записувались виміри, це тики таймера. Під збором даних по процесам мається на увазі отримання всієї таблиці процесів з останніми для них даними.

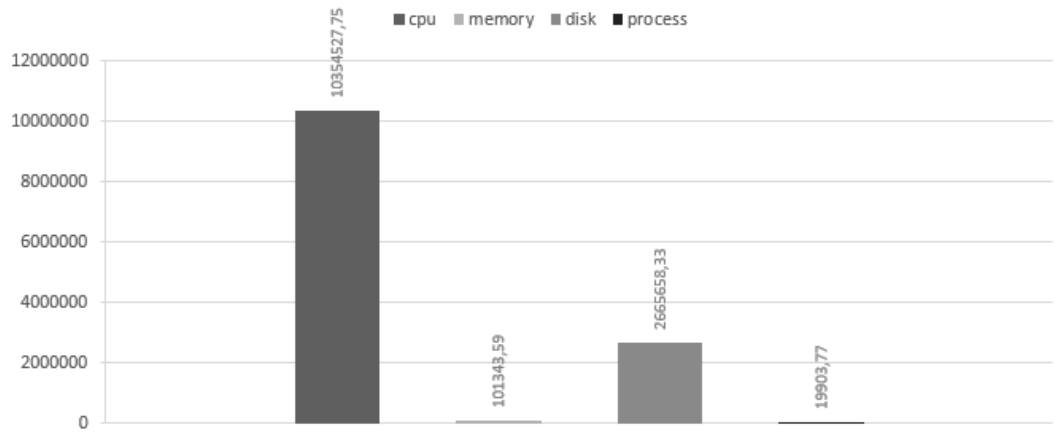

Рисунок 2.1 - Середній час виконання запиту ManagementObjectSearcher по розділам

З результатів очевидно що збирати відсоток навантаження на процесор за допомогою цього інструменту точно не варто, єдиний справді не поганий результат тут при зборі відомостей про процеси оскільки результат має вже всі обчислені поля які потрібні та має мінімум зайвих у відповіді.

Класс Performance Counter це лічильники продуктивності Windows які забезпечують високорівневий рівень абстракції, який забезпечує узгоджений інтерфейс для збору різноманітних системних даних у реальному часі.

Системні адміністратори зазвичай використовують лічильники продуктивності, щоб відстежувати проблеми продуктивності чи поведінки систем.

Їх переваги висока швидкість для високочастотного збору даних, підходить для моніторингу в реальному часі, у доменній системі можна запускати віддалено, без ніяких програм посередників.

Недоліки тут це менш зручний для роботи зі складними структурами, наприклад, список процесів, вимагає точного знання доступних лічильників.

Прямі звернення до ядра відбуваються через імпорт KERNEL32.DLL це надає додаткам більшість базових API Win32, таких як керування пам’яттю, операції введення/виведення (I/O), створення процесів і потоків, а також функції синхронізації. Більшість програм, які працюють у системі так чи інакше використовують цей файл. Переваги цього підходу максимально можлива швидкість, підходить для моніторингу в реальному часі, не використовує багато пам’яті. Головний недолік це те що він не зручний для використання, усі дельти треба обробляти вручну, більшість метрик навіть не існує їх треба обчислювати збираючи декілька.

Збір показників швидкості збору метрик за допомогою Performance Counter та звернення до ядра проводились протягом 10 хвилин та порівнювались середні значення тиків таймера. Детальні результати цього порівняння приведені на рисунку 2.2

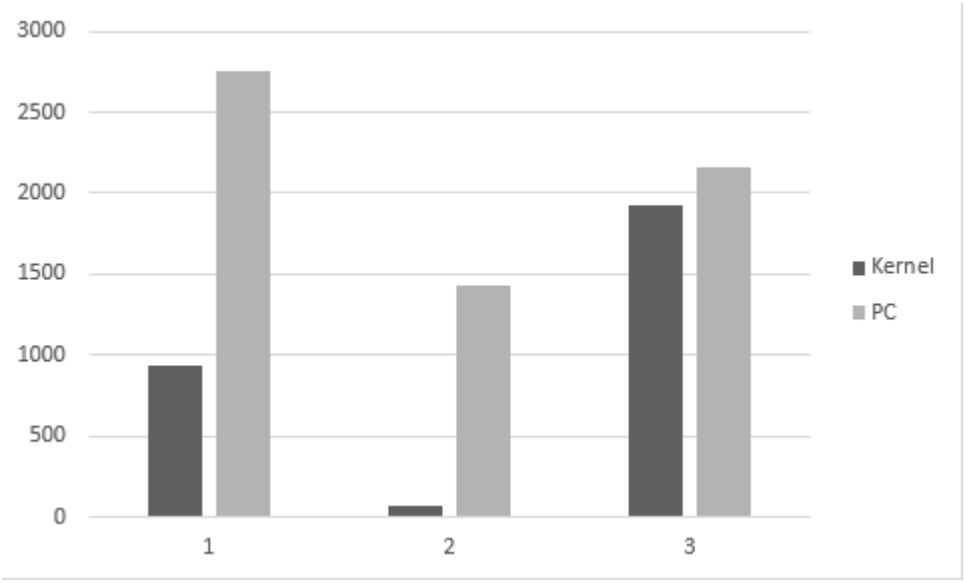

Рисунок 2.2 - Діаграма порівняння часу запитів до Kernel та PerformanceCounter

Для кращого порівняння усіх API, результати зібрано у таблицю 2.1.

Таблиця 2.1 - Час виконання запитів різними API моніторингу

Метод	CPU	RAM	Disk	Усі процеси
ManagementObjectSearcher	10354527	101343	2665658	19903
Performance Counter	2754	1432	2162	114311429
Запити до ядра	929	66	1930	-

Як видно з отриманих даних Performance Counter є оптимальним рішенням і достатньо швидким і водночас простим у реалізації, але його можна у деяких випадках замінити більш швидкі запити до ядра. Також цей інтерфейс має великий недолік це орієнтування на один параметр, це означає, що отримати одразу кілька значень одним запитом неможливо. А створення великої кількості каунтерів та повторні запити фактично до одного й того самого об’єкта роблять результат збору інформації по процесам непридатним для використання. На момент тесту у комп’ютері працювало 300 процесів. Число 114311429 у таблиці означає що для повного збору інформації по цій кількості процесів буде продовжуватись 11,5 секунд при повному навантаженні на один потік процесора. При цьому середній час який показав у цій задачі ManagementObjectSearcher набагато кращий, бо результат є об’єктом таблицею з усіма вже оновленими даними.

Висновки з отриманих спостережень можна зробити такі, що кожен з інструментів підходить під конкретну задачу.

Performance Counter Використовувати для реального часу де кількість запитів не висока або швидша альтернатива недоступна з причин сумісності. Найкращий варіант для простого періодичного моніторингу.

ManagementObjectSearcher(WMI) краще підходить для одноразових запитів або отримання складних структур даних. Можуть бути використані для налаштування автоматизації або діагностики, отримання даних про ПК або виконання не частих моніторингових запитів.

Запит до ядра системи є більш швидкою альтернативою Performance Counter та може мати проблеми з сумісністю на різних версіях систем.

2.1.2 Розрахунок впливу на мережу

Мета визначити скільки відсотків корисної роботи витрачається на передачу даних на головний сервер. Будемо прораховувати одразу для мережі з кількістю комп’ютерів 500 та порівнювати результат у найгіршому випадку для 100Мб мережі та 1 Гбіт. Під найгіршим випадком мається на увазі випадок коли абсолютно всі комп’ютери вирішили надіслати свої дані на сервер сбору статистики.

Для цього потрібно оцінити приблизний розмір кожного повідомлення яке буде надсилатися періодично. Якщо кожного разу затримок у синхронізації не буде то надсилання буде відбуватись по одному рядку щохвилини. Приблизні розміри даних які надсилаються наведені у таблиці 2.2.

Таблиця 2.2 – Приблизні розміри полів які пересилаються по мережі.

Компонент	Розмір
Мітка часу	8
ЦП	24
Оперативна	12
Дискова	32
Найважчі процеси	108
Останнє активне	64
Статистика	112
Статистика	24
Загалом для	384

Таким чином можна порахувати приблизно розмір пакета додавши 20%, щоб покрити довгі імена процесів у статистиці, та додаткові символи протоколу даних та TCP 461 байт.

Окрім даних статистики, ще можуть надсилатись раз на добу або після виявлення зміни параметрів відомості про пристрій вони включають усю інформацію про апаратні компоненти та стан диску. Ця передача може мати дуже великий вплив, але відмінити її не можна, бо програма має допомагати у інвентаризації та й з цими даними набагато простіше знайти комп’ютер фізично у великому офісі. Розміри цих даних наведені у таблиці 2.3.

Таблиця 2.3 - розміри даних для пересилання, конфігурація ПК

Компонент	Приблизний розмір (байт)
Процесор (CPU)	100
Материнська плата	224
Пам'ять (RAM)	68
Диски (HDD/SSD)	336
Відеокарта (GPU)	80
Встановлені програми	800
Інші пристрої	256
Загалом	1864

Перерахувавши на кількість комп'ютерів та додавши 20% вийшло 1 118,4 Кб. Провівши прості розрахунки отримаємо результати у таблиці 2.4.

Таблиця 2.4 – Результати процентного співвідношення використання мережі

Тип данних	Обсяг даних Кб	Відсоток для 100Мб	1 Гбіт.
Синхронізація статистики	225,1	1.8448%	0.18448%
Дані про комплектацію ПК	1 118,4	9,16%	0,916%

Отже з попередньої оцінки можна зробити висновок що навантаження на мережу у найгіршому випадку буде становити близько 10% для 100 Мбіт мережі. Для 1 Гбіт вплив можна вважати незначним, бо зазвичай він буде до 1% пропускної можливості. Також треба врахувати, що це значення для пересилання без стиснення даних.

Якщо ж дані будуть стискатись, це підвищить навантаження на процесори обох пристроїв учасників, та і цілому цей процес пригальмує процедуру передачі. Але зменшення кількості даних може переважити цю проблему з наступних причин. Для передавання даних пакувати їх

заплановано у JSON, оскільки це текстовий формат, що містить багато повторюваних структур, таких як дужки, лапки, ключі. Алгоритми стиснення ZIP або 7z ефективно усуває повторення такого типу, особливо для великих файлів.

Повноцінно передбачити чи покладатись на отримані за кілька разів співвідношення стиснення не доводиться, бо данні завжди різні і рівень стиснення має дуже великі розбіжності. Тому краще прийняти до уваги час архівування та розархівування даних. Розархівування зазвичай швидше і однакове по часу, але на різних наборах даних однакового об'єму. А от стиснення теж має деяку випадковість тому. Результати поверхневих замірів показали, що 7z є на 12% у середньому повільнішим за zip алгоритм.

Для великих JSON-файлів і мереж з низькою пропускною здатністю 7z є кращим вибором, якщо час стиснення не є критичним.

2.1.3 Виведення формули для приведення показника навантаження процесора до базової частоти.

Що до моніторингу процесора є кілька проблем з отриманими показниками протягом часу. Якщо порівняти показники отримані з ядра чи з каунтера вони будуть завжди вище ніж у більшості інших моніторингових програм. Це викликано тим що сучасні процесори за замовчуванням підтримують функцію економії енергії на ходу. Тобто поки висока частота не потрібна вона зменшується. Відповідно збільшується час прорахунків що напряму впливає на показник навантаження.

Показником навантаження на процесор є відсоток часу потягом якого він був навантажений у визначеному відрізку часу. Відповідно при меншій частоті цей показник зросте. Відповідно якщо вважати, що у процесора завжди доступна найвища частота то можна сказати що отриманий при низькій частоті завищений показник не відповідає дійсності і буде створювати не правильні результати статистичного аналізу або наштовхне на хибні висновки що до продуктивності. Показник навантаження можна адаптувати під частоту як це роблять у інших, але як саме це роблять не відомо через комерційність коду у більшості таких програм. Для того щоб правильно зробити адаптацію показника треба спочатку зрозуміти за якою саме залежністю він змінюється відносно частоти. Для цього треба провести експеримент.

Методика проведення цього експерименту буде така: створити постійне навантаження яке буде мати діапазон для змін, записати показник середньої частоти та навантаження від системи, провести аналіз змін. Навантаження створювалось за допомогою скрипту який вираховує корені випадкових чисел у два потоки. Що на конкретному процесору створює навантаження без враховування системи близько 30 відсотків. Якщо врахувати навантаження від системи і інших програм це в середньому 37 відсотків. Але нажаль зафіксувати навантаження дуже важко тому воно весь час коливається у межах десяти відсотків що вимагає усереднення серед великої кількості значень.

Провівши такий експеримент при різних частотах вийшла залежність показана на рисунку 2.3.

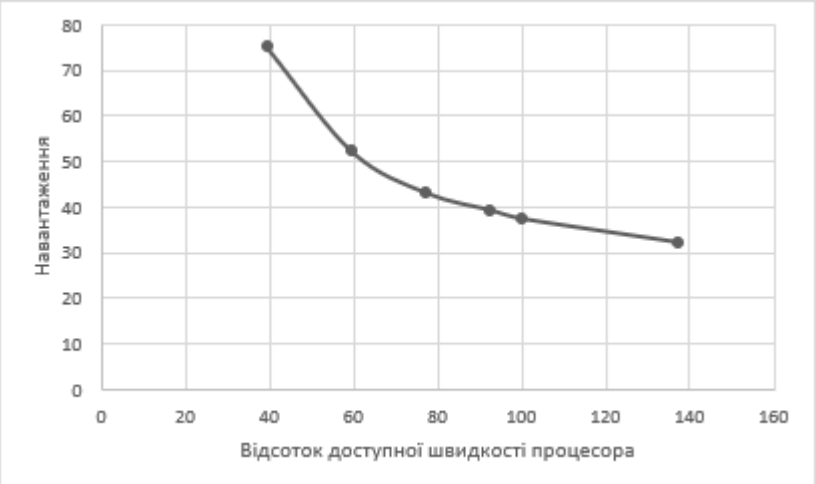

Рисунок 2.3 – Залежність зміни частоти процесора до показнику навантаження.

По наведеному графіку добре видно, що навіть при не великій кількості точок залежність точно не лінійна, що явно змінює ситуацію. Для такої адаптації показника треба зробити формулу з нерівномірними змінами. Для цього найкраще підходить експоненційного функція, але її треба додатково збалансувати до. Провівши моделювання та виведення функції у Microsoft Excel був отриманий методом перебору коефіцієнт зменшення змінення добутку примноження до отриманого сирого у цьому контексті показника. Він дорівнює 60.15 і примінений як константа у отриманій формулі (2.2). для спрощення запису відношення цільового відсотку частоти, який рівний 100 відсоткам та поточному відсотку був винесений у окрему змінну PerfomanceRatio.(2.1)

$$PerfomanceRatio = \frac{100}{currentPerformance} \tag{2.1}$$

де currentPerformance - потчний відсоток від базової частоти.

$$adaptedLoad = \frac{rawLoad}{PerfomanceRatio * 1 - \frac{\exp(PerfomanceRatio)}{60.15}} \tag{2.2}$$

де rawLoad – значення навантаження отримане у парі з currentPerformance;

Отримана функція наближено змінює значення на потрібні відносно базової частоти, точну функцію не можливо створити і через те що під час експерименту могли бути невеликі зміни навантаження. Прогнозовані зміни продемонстровані накладанням на оригінальний графік змін на рисунку 2.4. Прогнозовані значення показані сірим кольором.

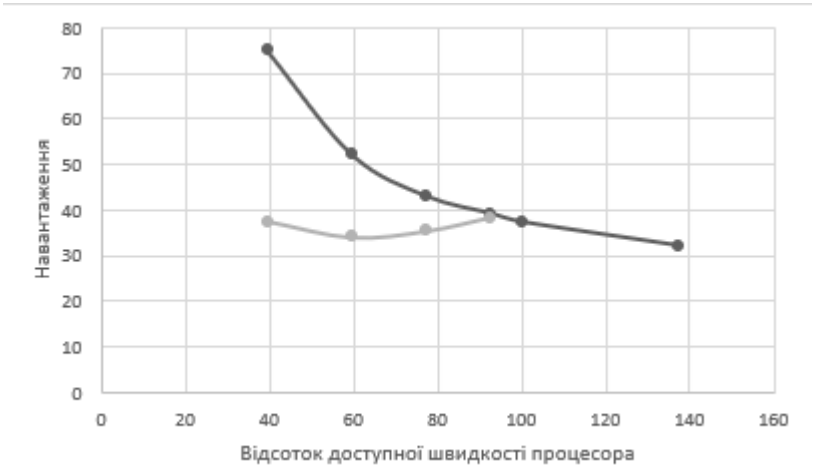

Рисунок 2.4 – Прогнозована робота функції

Наступний етап протестувати функцію у програмі та проаналізувати зміни. Провівши повторний експеримент можемо перевірити як спрацювала функція на отриманому графіку проілюстрованому на рисунку 2.5.

Рисунок 2.5 – Порівняльний графік значень без адаптивної функції та з нею.

Таким чином отримаємо працюючу функцію адаптування показників відносно поточної частоти. Звісно присутня неточність, але вона не помітна користувачу та не вносить сильних змін в статистику. Якщо конкретно то максимальна різниця помічена від цільового значення, яке теж дуже не стабільне, становить 10 відсотків. Це при умові, що без адаптування це значення мало б відхилення у 230 відсотків, що значно гірше для загальної картини статистики. Це порівняння стосується останнього діапазону значень частоти. На рисунку 2.4 це значення з 89 і до останнього. Важливе покращення для роботи це вимкнення цієї функції в межах роботи технології Turbo Boost, бо у такому випадку система показує позитивні відсотки відносно базової

частоти, що призводить до неправильної роботи функції оскільки вона і не планувалась для використання що до частот вищих за базові.

2.2 Розробка алгоритмів аналізу статистики

2.2.1 Виявлення аномалій навантаження

Аналіз аномалій спрямований на виявлення відхилень у використанні ресурсів, що можуть свідчити про потенційні проблеми, такі як технічні збої чи несанкціоноване втручання. Цей процес зосереджується на ідентифікації даних, які порушують загальні закономірності. Для ефективного виявлення аномалій зазвичай використовують історичні дані, хоча можливий і моніторинг у реальному часі за умови наявності наперед визначеного шаблону нормальної поведінки. Такі відхилення часто називають розбіжностями або виключеннями. А також серед них виділяють: точкові та колективні.

Точкові аномалії — це одиничні випадки, які не впливають на загальну картину. Колективні аномалії це група подій, корелюють між собою, хоч окремо і не є аномальними.

Методи виявлення аномалій поділяються на три основні категорії. Неконтрольовані алгоритми, перебирають дані без моделі, припускаючи, що більшість з них є нормальними, і шукають відхилення, базуючись на конкретному наборі. Контрольовані методи навчаються на попередньо позначених даних, де відомо, які з них є нормальними, а які — аномальними. Напівконтрольовані підходи створюють модель нормальної поведінки системи на основі заданого набору, а потім перевіряють нові дані на відповідність цій моделі.

Створення таких моделей передбачає використання математичних методів і технологій штучного інтелекту для опису звичайного стану системи та ідентифікації відхилень.

У системі моніторингу визначати аномалії може бути корисно з метою виявлення зависань які вже відбулись. Зазвичай вони можуть бути наслідком вузьких місць в системі або гірше вірусів.

За таким виявленням криється інша проблема пов’язана з природною роботою будь якої системи не реального часу тобто Windows чи то Linux або навіть Android. Сама по собі статистика роботи виглядає як правило коливання навантаження, але не можна вважати усі такі коливання аномаліями. Якщо у використанні ресурсів з’являється аномалія то зазвичай вона не видима на основних параметрах таких як завантаженість процесора чи оперативної пам’яті, але підвисання є. Попередньо експериментальним шляхом зберігаючи якомога більшу кількість даних було виявлено декілька показників на яких все ж таки підвисання через недостатності ресурсів пам’яті та апартні помилки помітні. Це «Faults» пам’яті які насправді можна сприймати не як помилки, бо вони скоріше описують простої та очікування процесором відповіді від оперативної пам’яті.

Помилки які реєструє ядро системи Windows є чотирьох видів.

Помилки транзиту виникають при спробі зчитати сторінку даних із оперативної пам'яті, коли система очікує, що дані мають бути в пам'яті (але їх там немає). Можуть свідчити про проблеми з пам'яттю, які можуть призвести до затримок у роботі системи через необхідність звертатися до диску або інших повільних носіїв для отримання даних.

Помилки сторінки виникають коли процесор намагається отримати доступ до сторінки пам'яті, яка не перебуває в оперативній пам'яті ,зазвичай такі сторінки знаходяться на диску, або ще не були виділені. Свідчать що системі не вистачає оперативної пам'яті, і їй доводиться використовувати файл підкачки, що впливає на продуктивність.

Помилки кеш-пам'яті виникають, коли процесор намагається зчитати дані з кешу, але ці дані не перебувають у ньому. Внаслідок цього необхідно отримати дані з більш повільної оперативної пам'яті або диска. Свідчить про недостатню ефективність кешування або надмірну роботу з великими обсягами даних, які не вміщуються в кеш.

Помилки запиту обнулення виникають, коли система повинна виділяти пам'ять і очищати її для безпеки перед наданням процесу. Висока кількість таких помилок може вказувати на активне виділення пам'яті процесами, що вимагає постійного обнулення великих ділянок пам'яті. Зазвичай не вказує на проблему з продуктивністю просто вказує на активний запуск програм, але тоді і інші помилки також мають трохи змінитись, якщо ж тільки помилки запитів обнулення високі, це може свідчити про неправильну роботу програми на ПК.

Як було згадано вище виявлення аномалій буває різне, без нагляду та контрольованого виявлення. У нашому випадку більше підходить виявлення без нагляду, бо таким чином не буде залежності від створеної завчасно моделі. Таким чином увесь пошук аномалій буде відбуватись після якогось періоду часу.

Таке рішення прийняте через статистичні спостереження які вказують на те що ці помилки мають тенденцію «фонового» шуму у даних, який змінюється в залежності від активності користувача. Наприклад запуск важкої програми спричиняє багато помилок пам`яті у будь якому разі, але потім навантаження врівноважується, але кількість помилок вже є трохи більш підвищеною, бо зазвичай сучасні графічні та офісні програми постійно редагують дані в пам’яті навіть якщо фактично користувач активно їх не використовує.

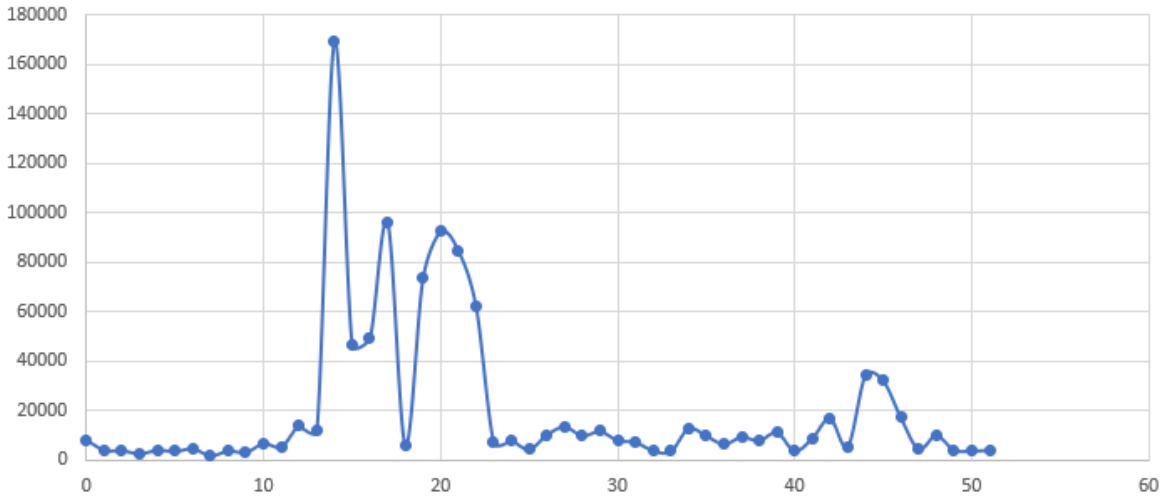

Рисунок 2.6 - Помилки обнулення до і після запуску програми

На рисунку 2.6 добре видно що до 12 заміру середнє значення помилок було доволі низьким, потім від 12 до 23 заміру відбувся запуск програми, що створив велике підвищення помилок, і після 23 заміру кількість помилок знову доволі стабільна, але в середньому підвищена.

Для більшої наочності на рисунку 2.7 Показані середні значення відповідних проміжків часу з попереднього графіка.

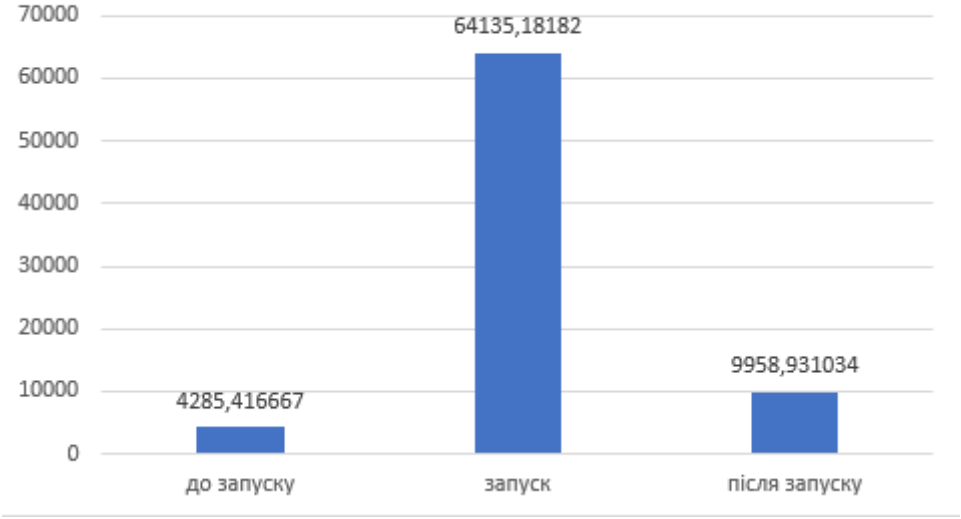

Рисунок 2.7 - Помилки обнулення до і після запуску програми

Тому контрольоване виявлення аномалій у такій ситуації буде дуже не точним і часто буде списувати нормальну зміну роботи на аномальні показники. Єдиний варіант це створити алгоритм який буде враховувати зміни середнього навантаження. Звісно використовувати для цього модель штучного інтелекту не вкладається у концепцію проекту, бо основне визначення аномалій буде відбуватись на стороні агентів клієнта, що означає штучний

інтелект буде зайвим навантаженням, а розміщення на сервері знизить корисну роботу мережі.

Тому треба використати алгоритм визначення аномально високого навантаження який буде швидко аналізувати отриманий масив даних за період часу.

Мета цих маніпуляцій з даними це отримати два коефіцієнти. Співвідношення між середнім значенням при сформованому нормалізованому ряді та оригінальними даними з усіма аномальними підвищеннями навантаження. Та співвідношення між діапазонами змін, щоб визначити на скільки динамічно відрізнявся пік навантаження. Такий коефіцієнт показав би наскільки часто та критично відбувається перевантаження.

На жаль повноцінно відділити зависання та звичайне завантаження важкої програми не можливо тільки по цим даним. Звісно можна реєструвати появу нових процесів, але такий підхід не забезпечить модульності програми. Тому для аналізу проявів перевантаження залишаються часові мітки. Тому таке виявлення найбільш корисним буде для користувачів у домашніх або персональних умовах використання.

Існує декілька методів які можна використати з метою відсіювання аномальних показників та створення нормалізованого ряду значень: метод медіанного фільтру, трикутний фільтр та простий метод заміни високих значень**.**

Метод медіанного фільтру працює шляхом заміни кожного значення в масиві на медіанне значення в його локальному вікні. Це дозволяє видаляти аномалії, які сильно відрізняються від сусідніх значень. Принцип фільтрації по елементний. Для кожного елемента визначається локальне вікно: обираються значення з масиву в межах заданого вікна навколо поточного елемента. Відсортовуються значення у вікні та обирається медіанне значення та замінюється поточний елемент.

Метод трикутним фільтром є зваженим варіантом згладжування. Він працює за принципом присвоєння більшої ваги значенням, ближчим до центру вікна, що має забезпечувати плавніше згладжування. Принцип передбачає таке ж визначення локального вікна. Встановлення ваг для елементів у вікні. Розподілення має вигляд: найбільша вага — для центрального елемента, менші — для віддалених. Для них обчислення зваженого середнього, помноживши кожне значення на його вагу, і поділивши суму на загальну суму ваг. І заміна поточного елемента на це зважене середнє.

Також я використав свій метод який націлений на максимальне спрощення обчислень, при наближеному до складних методів результатів.

Сенс цього методу не обробляти кожне значення як таке, а просто перевіряти чи не виходить воно за адекватні межі. Для цього я не можу визначити межі статично через причини “фонового” навантаження як було наведено раніше. Тому для цього я використовую підходящу для цього міру розсіювання міжквартильний розмах. А точніше сам принцип визначення діапазону для визначення верхньої межі передбачуваної нормальної кількості помилок. Для визначення межі потрібно спочатку визначити IQR по формулі.

Визначення значення IQR це знаходження різниці між третім квартилем Q3 і першим квартилем Q1:

$$IQR=Q3-Q1 \quad (2.3)$$

Межі нормальних значень визначаються як:

$$Q1-k\cdot IQR,\ Q3+k\cdot IQR \quad (2.4)$$

де k — коефіцієнт, який зазвичай приймається рівним 1.5.

Методом підбору на реальних даних було визначено що коефіцієнт k краще виставити більшим за стандартну рекомендацію, бо при стандартному значенні невеликі зміни вже створюють велику різницю наприкінці аналізу, тому в програмі він буде встановлений на 1.9. Це значення виставлене гіпотетично на основі візуальної оцінки результатів тому стверджувати, що воно може бути тільки таким не можна.

Оскільки мені в цій ситуації не потрібна нижня межа визначати її окремо не потрібно.

Для того щоб отримати відфільтрований набір даних цим методом треба з оригінального набору даних отримати кілька важливих значень:

- Мінімум
- Максимум
- Середнє
- Межа для “нормалізації”
- Відношення максимального до середнього

По відношенню максимального до середнього значення можна визначити чи стабільною була робота на цьому проміжку часу якщо значення близьке до 0,5 то це означає що середнє значення проходить ідеально посередині коливань що означає або стабільну роботу або настільки часті коливання, що їх вже не можливо відділити від не аномальних значень.

Отримана межа за допомогою міжквартильного розмаху отримана уявна лінія буде знаходитись трохи вище середнього значення що і надасть свободу для коливань навантаження. Детальний алгоритм роботи показаний блок-схемою на рисунку 2.8

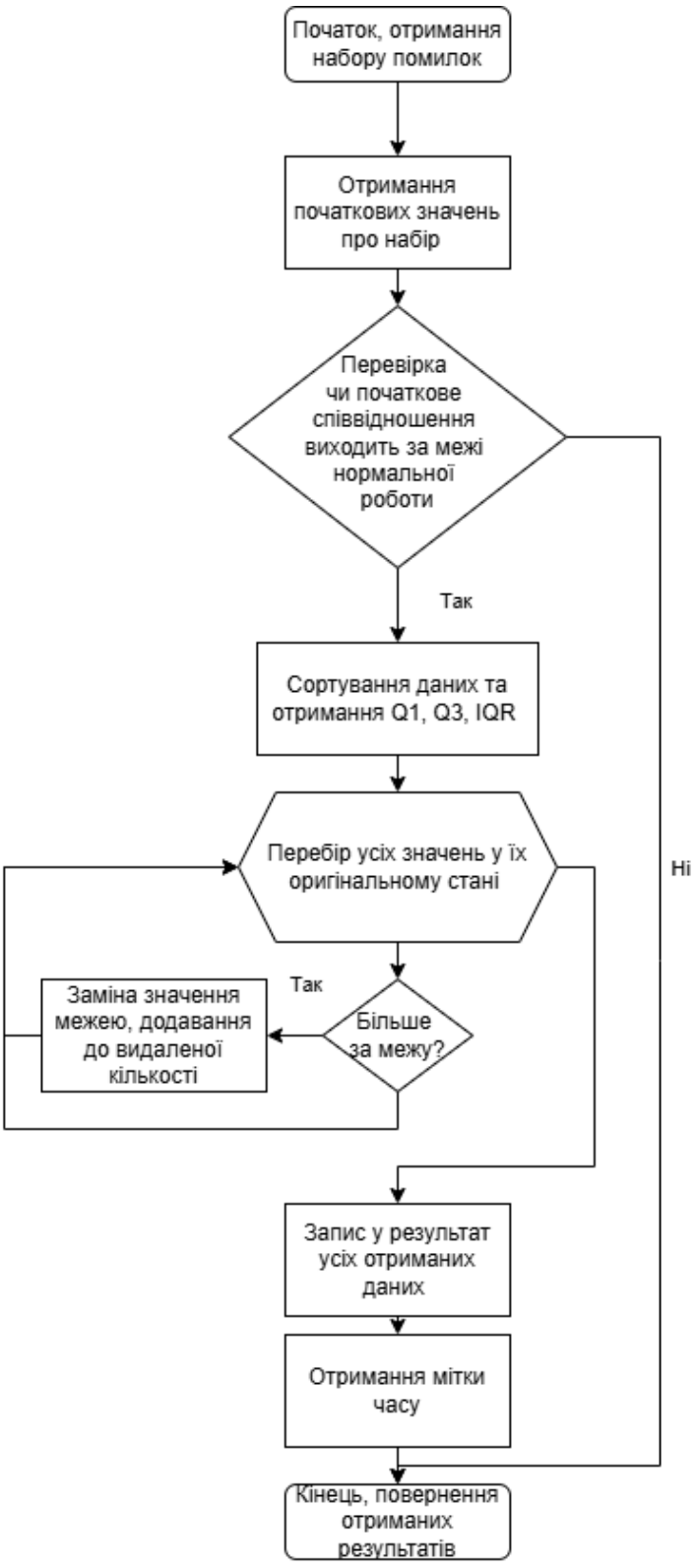

Рисунок 2.8 – Алгоритм роботи швидкого очищення від аномалій

Якщо порівняти результати очищення то добре видно, як себе поводить кожен алгоритм і з міркування порівняння значень стрибків помилок, таке згладжування які надаються алгоритм у цій задачі не потрібне. Результати продемонстровані на рисунку 2.9

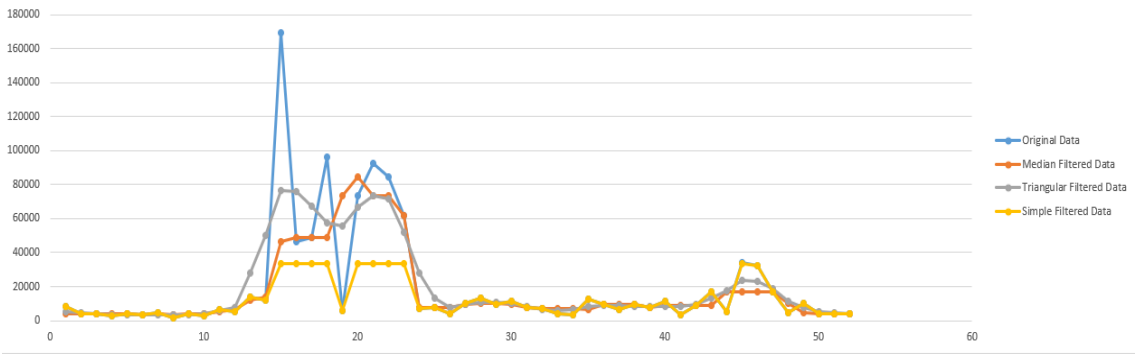

Рисунок 2.9 - Порівняння результатів очищення

Порівнюючи швидкість обчислень, швидкість спрощеного алгоритму безвідмовна, що добре видно на графіку середньої кількості тиків витрачених на фільтрацію продемонстровано на рисунку 2.10.

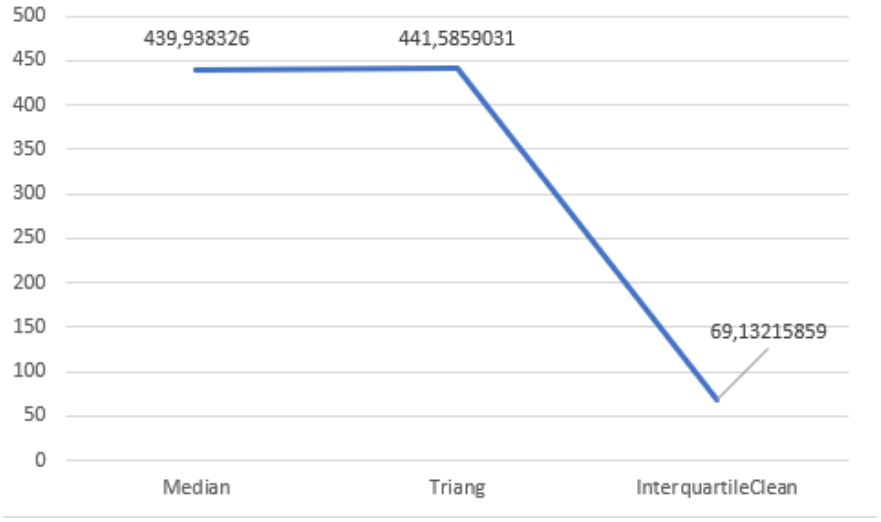

Рисунок 2.10 - Середня кількість тиків витрачених на фільтрацію аномалій.

Співвідношення між середнім значенням при сформованому нормалізованому ряді та оригінальними даними з усіма аномальними підвищеннями навантаження. Та співвідношення між діапазонами змін, щоб визначити на скільки динамічно відрізнявся пік навантаження. Такий коефіцієнт показав би наскільки часто та критично відбувається перевантаження. Для більшої інформативності про період часу у алгоритм введено перераховування кількості змінених алгоритмом точок для отримання відсотку аномальних зон.

Отримуючи такі коефіцієнти врешті решт можна буде визначити наскільки великі затримки довантаження та чи має комп’ютер короткочасну нестачу швидкодії.

Результат всіх проведених операцій добре видно на графіку на рисунку 2.11. на якому добре видно що о 1:18:02 був великий сплеск помилок передачі, це може свідчити про апаратні проблеми на пристрої.

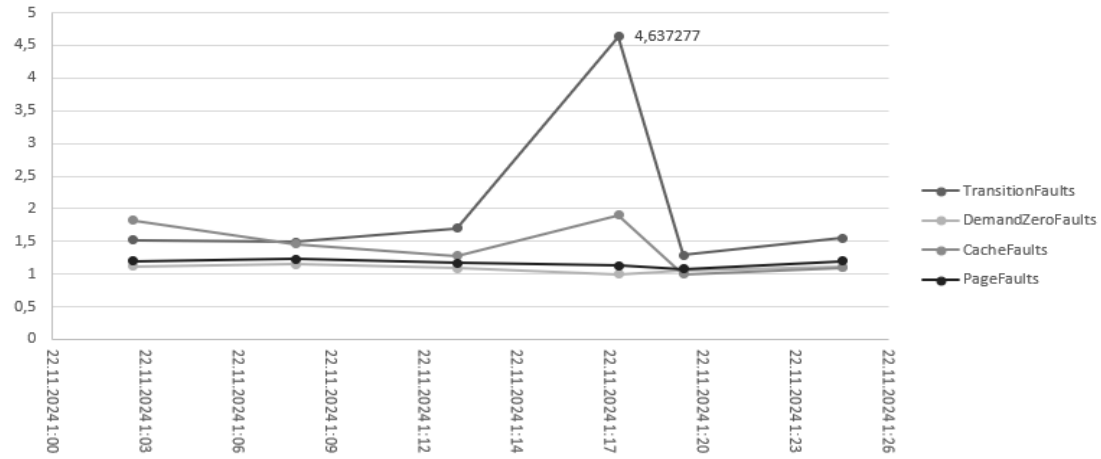

Рисунок 2.11 - Графік показників відношення середнього по нормалізованим даним та даним з аномальними навантаженнями.

2.2.2 Визначення недостатнього по потужності компонента.

Для аналізу статистики для визначення недостатнього по потужності компонента підходить метод багатофакторного аналізу. Використання багатофакторного аналізу передбачає набір даних, на щастя у цьому випадку є такі дані. Вони являють собою таблицю погодинного навантаження на компонент комп’ютера.

Формується такий запис з 60 хвилинних записів кожен з яких має записи: максимального значення, середнього, максимального і час в секундах під максимальним навантаженням. Потім раз в годину ці записи стискаються утворюючи запис з градацією навантажень за останню годину та сумарний час під максимальним навантаженням. Далі під час аналізу треба визначити що вважати високим середнім та низьким навантаженням.

Але виникає проблема що для цього потрібна змінна Y яка б вказувала на продуктивність або перевантаження в цілому системи. У реальній ситуації щоб заповнити такий стовпчик треба було б робити постійні опитування користувачів, але навряд чи це було б комфортним рішенням. Тому можна зробити припущення, що перевантаження виникає тоді коли довгий час компонент працює під високим або максимальним навантаженням і автоматично створити такий стовпчик, де 0 буде значенням за годину без довгих високих навантажень, а 1 буде значення де високе навантаження перейшло межу хоча б по одному компоненту. Таким чином можна проаналізувати усі стовпчики даних і порівняти коефіцієнти високих

навантажень. Той компонент який буде мати більші коефіцієнти, буде означати що користувачу частіше не вистачає саме його, бо його вплив на випадіння варіанту перевантаження за годину у останньому стовпчику більший.

Для наглядного спрощення прикладу використання регресійного аналізу у таблиці були залишені тільки два компоненти CPU та RAM. Їх показники під час активної роботи коливаються найчастіше, тому є сенс розглядати саме їх.

Приклад фінального датасету у таблиці 2.5. Останній стовпчик визначається автоматично в програмі тому його немає у таблиці.

X_n - у цьому випадку будуть стовпчики з часом CPU_low, CPU_medium, а Y останній стовпчик висновку перевантаження.

Написавши підпрограму для читання таблиці та проведення регресійного аналізу отримуємо список коефіцієнтів.

Таблиця 2.5 - Датасет погодинного навантаження

Hour	CPU_low	CPU_medium	CPU_high	CPU_max	RAM_low	RAM_medium	RAM_high	RAM_max
07.07.2024 13:00	15	31	8	6	27	18	4	11
07.07.2024 14:00	17	20	10	13	37	21	1	1
07.07.2024 15:00	43	0	6	11	6	4	43	7
07.07.2024 16:00	18	29	7	6	51	8	0	1
07.07.2024 17:00	55	0	3	2	30	2	4	24
07.07.2024 18:00	30	23	3	4	16	22	18	4

Коефіцієнти винесемо у діаграму для порівняння. З поточного набору даних вийшов результат, що більший вплив на перевантаження системи має RAM. Що добре видно на діаграмі на рисунку 2.9. І хоч самі значення коефіцієнтів не великі, але у масштабі різниця суттєва для визначення результату.

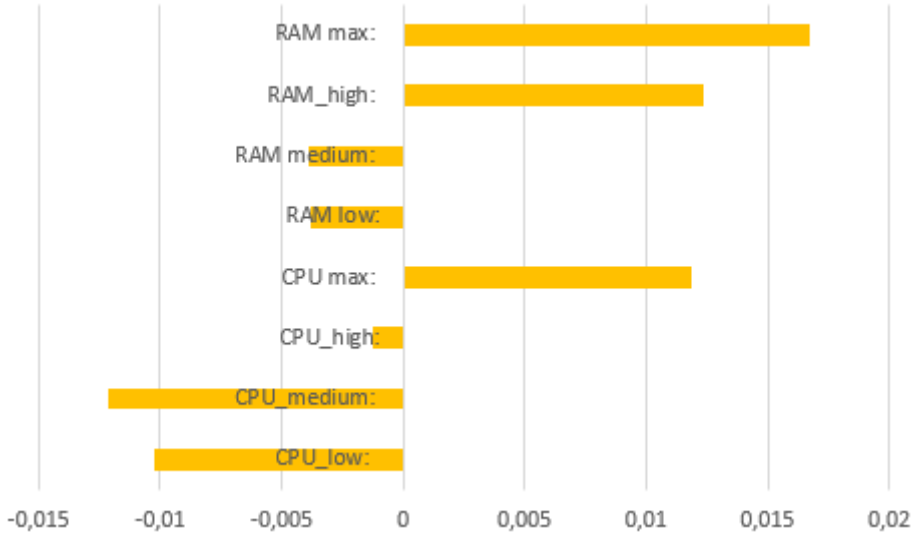

Рисунок 2.9 - Діапазони коефіцієнтів впливу діаграмою

2.2.3 Визначення часу обслуговування.

Обслуговування комп'ютерів потрібно робити час від часу, бо зазвичай пил та бруд потрапляють всередину та врешті решт заважають охолодженню. Ще і термопаста висихає і з часом експлуатації знижує ефективність. Що спричиняє поступове збільшення середньої температури. З великою кількістю записів можна знайти цей тренд поступового збільшення і задавши неприпустимий поріг температури вирахувати через скільки днів знадобиться обслуговування комп'ютера.

За допомогою лінії тренду, а саме нахилу лінійного тренду можна передбачити час коли обслуговування комп'ютера стає необхідним.

Записані температурні дані мають містити не тільки середнє, але мінімальне та максимальне за кожен період.

Максимальна температура може надати важливі дані щодо перегріву компоненту. Зазвичай сучасне обладнання не допускає вимкнень через перегрів, а корегує частоту та навантаження щоб тримати температуру у обмеженій зоні. Якщо графік максимальних температур часто показує фіксацію на конкретній температурі це можливо вже тротлінг і обслуговування потрібно негайно. Також багато сучасних процесорів мають показник температури Tjmax який показує запас до перегріву тому фіксація значення 0 може використовуватись як сигнал тривоги на клієнтських ПК.

Сезонність температур має відносно невеликий вплив. Звісно комп'ютер який працює у приміщенні не є під сильним впливом цього фактору через обігрівачі та кондиціонери, але навіть так середня температура в приміщенні все одно змінюється. На жаль зміною сезонної температури доведеться знехтувати.

Оскільки програма може збирати середню температуру за кожну годину то залежно від аппаратного компонента різниця може бути дуже значна. Тому за можливості треба охопити всі основні вузли системи

Для прикладу можна провести лінію тренду для температур CPU. Зазвичай його показник змінюється з доволі сильними коливаннями і розбіжностями. Тому потрібен набір даних за тривалий час. Вивівши лінію тренду на діаграму бачимо поступове збільшення показників. Y нахилу лінійної функції дорівнює 0,0025.

Проаналізувавши дані і побудувавши лінію тренду нам стає відомий кут нахилу лінійної функції і тепер визначити через скільки годин або днів знадобиться обслуговування не так вже і складно. На основі даних та визначення лінії тренду взявши за критичну середню температуру 75 градусів отримуємо що обслуговування знадобиться за 3191,28 годин що еквівалентно 132 дням. Лінія тренду температурних показників зображена на рисунку 2.10.

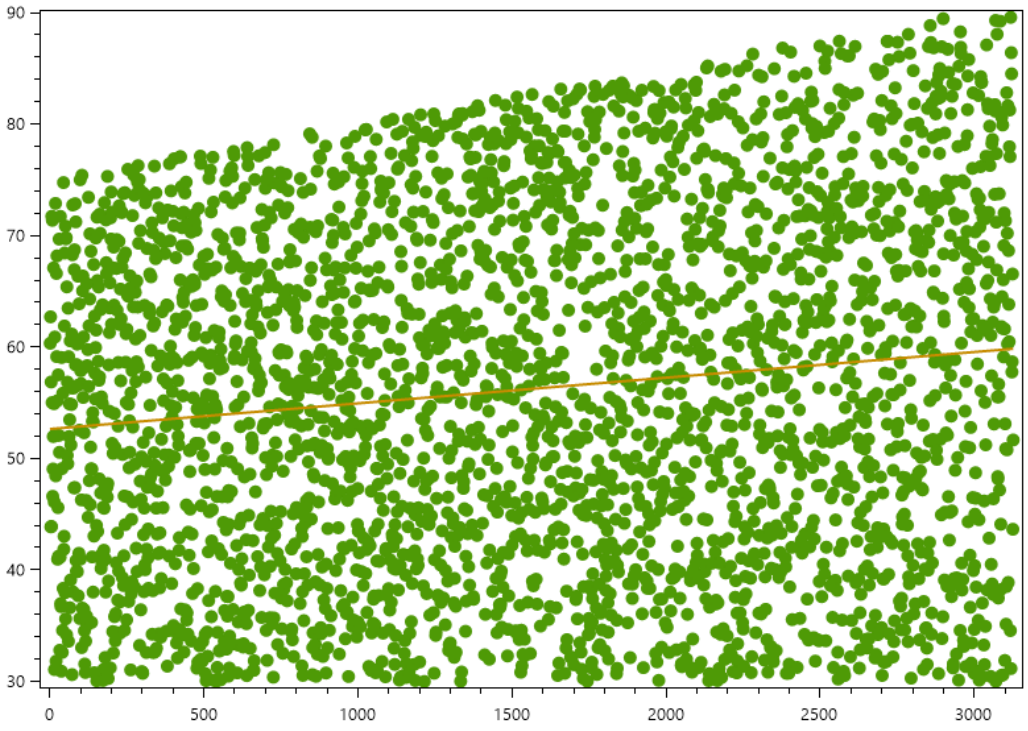

Рисунок 2.10 - Лінія тренду температурних показників.

Таким чином можна робити планові обслуговування тоді коли вони потрібні, і коригувати графік щоб він відповідав реальним потребам обслуговування. Результат аналізу температур від комп'ютера до комп'ютера теж може бути різним, через велику кількість зовнішніх факторів як кількість меблів у приміщеннях частота прибирання, місцеположення комп'ютера відносно килима, що ще раз підкреслює важливість достатньо великого набору даних.

2.2.4 Виявлення процесів які створюють аномальне навантаження

Оскільки програма може спостерігати за використанням ресурсів всіма процесами то на базі цих даних можна виявляти підозрілу активність фонових програм.

Звісно такий підхід для підвищення безпеки не зовсім новий і зазвичай не використовується, бо вимагає багато часу на аналіз та створення висновків, а користь від такого інструменту значно підвищується, якщо він працює у реальному часі. Тому я пропоную підхід який виключає збір великої кількості статистики, а за допомогою збереження ключових моментів активності користувача виконувати ту ж задачу аналізуючи менше даних.

Кореляція активного вікна користувача з навантаженням дозволяє виявляти незвичайну активність, що може свідчити про віруси-майнери, не бажані фонові процеси або інші загрози безпеці. Ця функція корисна як в корпоративному середовищі так і у домашніх умовах для виявлення потенційних загроз, перш ніж вони завдають багато шкоди. Для цього створюється підпрограма яка стежить за активним вікном і якщо активне вікно не створює високого навантаження, і це не дочірній процес активного вікна, це буде підозрілою активністю з боку іншого процесу. Але є програми як наприклад відео редактори у яких є етап візуалізації і навряд чи користувач дивитиметься на прогрес візуалізації відео, а натомість щось буде робити паралельно, наприклад читати пошту у браузері і це вже не підозріла поведінка, але може потрапити під першу умову.

Гіпотеза такого виявлення полягає в тому, що часової статистики по активному вікну має бути достатньо, щоб в реальному часі виявляти підвищену активність від тих процесів від яких її не очікували. А також за таким принципом можна покладатись на час відсутності активності від користувача, більшість зловмисного програмного забезпечення слідкує за активністю користувача, щоб їх дії були менш помітні.

Але тут постає дилема через неоднозначні висновки які можна отримати від звичайного алгоритму, через вище наведену нормальну поведінку користувача.

Для порівняння отримання кращих результатів є сенс спробувати використати алгоритми fuzzy logic чи навіть модель штучного інтелекту.

Що до використання fuzzy logic є складність інтеграції та постановки умов і фактично він буде повторювати роботу звичайного алгоритму, але замість результату так або ні, давати оцінку з плаваючим значенням. А за допомогою штучного інтелекту є проблема створення моделі адже набір правильних даних які не будуть максимально наближені до реальності важко сформувати та правильно інтерпретувати на етапі навчання. Ще й використання штучного інтелекту це у будь якому разі набагато більше споживаних ресурсів моніторинговою системою. Зважаючи що розмістити такий аналізатор можна буде тільки на стороні клієнта використання штучного інтелекту має бути мінімальним або модель примітивна.

Звісно не всі процеси мають активне вікно, і проблема такого виявлення це те що більшість таких процесів це служби Windows та віруси які як правило ховаються під мнемоніку цих самих служб. Звісно у такому

випадку міг би підійти словник і навіть так уникнути помилкового виявлення неможливо та й зберігання великої кількості даних не вписується у концепцію корпоративної моніторингової системи. Тож такий аналізатор аномальної активності процесів може тільки припускати, та далі давати на розгляд користувачу або адміністратору. Звісно все одно треба передбачити створення білого списку, щоб після впевненості що служба чи додаток не є небезпечним його можна було додати до цього списку.

Принцип збору даних: виявляється активне вікно, записуються середні показники ресурсів процесу цього вікна у окремий клас списком з часом та датою, дані накопичуються деякий час, потім переглядається список створений збереженням процесу активного вікна, з метою знайти патерн роботи користувача, періодично переглядається загальний список процесів з метою порівняння витрат ресурсів і виявлення процесів які у фоні витрачають багато ресурсів.

Процеси також мають враховувати динаміку використання оперативної пам’яті, що правда сутичка з початковою задумкою підпрограми, бо збирати дампи усього списку процесів це надто важка задача. Що правда система сама збирає максимальні показники пам’яті процесів які можна використати для порівняння, особливо визначивши патерн роботи користувача. Під патерном мається на увазі основний набір програм час та переключення між ними.

Для того щоб отримати усі показники знадобився набір функцій для взаємодії з системою на рисунку 2.11 наведений їх імпорт.

```
  [DllImport("user32.dll")]
        static extern bool GetLastInputInfo(ref LASTINPUTINFO plii);
        [StructLayout(LayoutKind.Sequential)]
        struct LASTINPUTINFO
        {
            public uint cbSize;
            public uint dwTime;
        }
        // Імпорт функції для отримання хендлу активного вікна
        [DllImport("user32.dll")]
        private static extern IntPtr GetForegroundWindow();
        // Імпорт функції для отримання заголовка вікна
        [DllImport("user32.dll", SetLastError = true, CharSet =
CharSet.Auto)]
        private static extern int GetWindowText(IntPtr hWnd, StringBuilder
lpString, int nMaxCount);
        [DllImport("user32.dll", SetLastError = true)]
        private static extern uint GetWindowThreadProcessId(IntPtr hWnd, out
uint lpdwProcessId);
        //Imports for parentfind
        [StructLayout(LayoutKind.Sequential)]
        public struct PROCESS_BASIC_INFORMATION
        {
            public int Reserved;
            public int PebBaseAddress;
            public int AffinityMask;
            public int BasePriority;
            public int UniqueProcessId;
            public int InheritedFromUniqueProcessId;  // Це ParentPID
        }
        // Тип функції NtQueryInformationProcess
        [DllImport("ntdll.dll", SetLastError = true)]
        public static extern int NtQueryInformationProcess(
            IntPtr hProcess,
            int processInformationClass,
            ref PROCESS_BASIC_INFORMATION processInformation,
            int processInformationLength,
            ref int returnLength);
        // Константи для ProcessInformationClass
        public const int ProcessBasicInformation = 0;
        // Декларація OpenProcess для отримання хендла до процесу
        [DllImport("kernel32.dll", SetLastError = true)]
        public static extern IntPtr OpenProcess(int dwDesiredAccess, bool
bInheritHandle, int dwProcessId);
        // Декларація CloseHandle для закриття хендлів
        [DllImport("kernel32.dll", SetLastError = true)]
        public static extern bool CloseHandle(IntPtr hObject);
        // Права доступу до процесу
        public const int PROCESS_QUERY_INFORMATION = 0x0400;
```

Рисунок 2.11 – Імпорти з user32.dll, ntdll.dll та kernel32.dll які потрібні для реалізації алгоритму.

Оскільки процеси перед виконанням дій будуть показуватися користувачу, то для допомоги з висновками треба показати наскільки процес є підозрілим. У такому випадку є декілька факторів на основі яких можна створити шкалу:

- Трохи підозрілий, мав високу активність, після періоду часу використання користувачем додатку, або є дочірнім з одним із додатків.
- Підозрілий, не пов'язаний з жодною з використовуваних програм.
- Дуже підозрілий, створював високе навантаження, проявляв активність коли користувач не був активний взагалі та не пов’язаний з жодною з нещодавно використовуваною програмою.

Шкала застосована для того, щоб не заплутувати зайвий раз користувача та не зберігати багато інформації з описами подій які є причиною додавання процесу у список підозрілих. Для адміністраторів як правило достатньо буде і імені підозрілого процесу для прийняття рішення про внесення у білий список чи видалення процесу з систем. Алгоритм виявлення описаний блок-схемою зображений на рисунку 2.12.

Важливі ремарки що до блок схеми. На ній показані деякі складні внутрішні процеси. UpdateUserOfflineTime це метод який ховає логіку звернення до ядра через user.dll. GetActiveProcessName це основний метод через який власне і вдається взагалі розуміти з якою програмою взаємодіє користувач. SearchParentProcesesCoincidences має доволі складну внутрішню логіку тому не показаний, шукає до 5 рівнів вниз по дереву процесів, шукаючи серед них хоч один з яким взаємодіяв користувач.

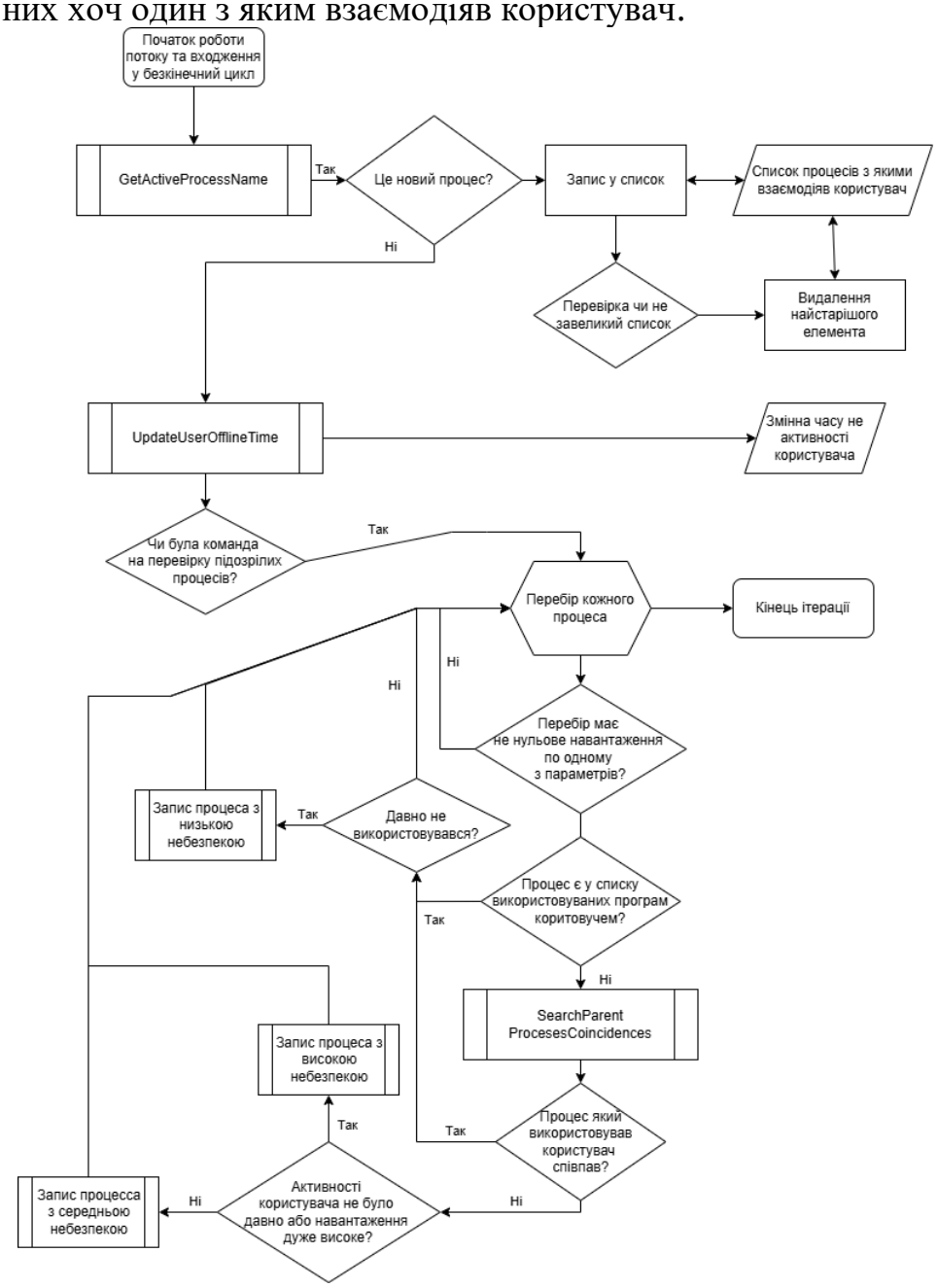

Рисунок 2.12 - Алгоритм виявлення підозрілих процесів.

Висновок до розділу 2

У цьому розділі було проведено комплексний аналіз статистичних даних системи та розроблено алгоритми, які забезпечують ефективний моніторинг та обробку інформації про стан системи. Основні результати можна підсумувати так:

- Проведено порівняння існуючих інструментів моніторингу для системи Windows, зокрема науковим методом досліджено їх функціональні можливості, вплив на продуктивність системи та зручність інтеграції у проект. Це дозволило обрати оптимальне рішення для збору даних.
- Розроблено підхід до оцінки впливу процесів моніторингу на мережеву активність, що дозволяє мінімізувати негативний вплив на продуктивність системи.
- Розроблено алгоритми для виявлення аномалій у навантаженні системи. Це дає змогу оперативно виявляти та усувати потенційні проблеми, такі як надмірне споживання ресурсів.
- Визначено методологію для ідентифікації компонентів системи, потужності яких недостатньо для виконання завдань. Це дозволяє оптимізувати ресурси та планувати модернізацію.
- Встановлено алгоритм визначення часу, необхідного для обслуговування системи, з використанням аналізу зібраних даних про температуру, продуктивність та активність компонентів.
- Розроблено підхід до ідентифікації процесів, які створюють аномальне навантаження. Це дозволяє швидко реагувати на критичні ситуації та забезпечувати стабільну роботу системи.

Таким чином, результати цього розділу забезпечують фундамент для створення ефективної системи моніторингу та управління ресурсами. Запропоновані алгоритми дозволяють покращити продуктивність, зменшити ризики перевантаження системи та оптимізувати процеси технічного обслуговування.

3 РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ

3.1 Обрані інструменти реалізації та загальні структурні рішення

Як було зазначено раніше у звіті науково-дослідної практики фінальна програма моніторингової системи може бути розрахована на декілька груп користувачів. Перша група це корпоративні користувачі, яким потрібен збір з великої мережі безпека та в основному серверний інтерфейс та програми агенти на клієнтах.

Друга група, це користувачі яким не потрібен масовий збір статистики, а навпаки потрібен зручний інтерфейс та доступ до моніторингових показників.

У повному своєму обсязі програма має два обов’язкових компоненти: агент та сервер додаткового моніторингу та збору даних статистики.

3.1.1 Інструменти розробки

Оскільки на даний момент основною операційною системою на більшості персональних комп'ютерів є Windows. Тому для написання коду моніторингової системи було обрано інтегроване середовище розробки Visual Studio. Забезпечує повний набір інструментів для зручної та ефективної розробки додатків на C# .NET та інших мовах програмування.

Середовище розробки Visual Studio має багато функцій, які спрощують процес програмування. Однією з ключових є підтримка автоматичного завершення коду за допомогою IntelliSense, що допомагає писати код швидше та зменшує кількість помилок. Visual Studio також включає потужний аналізатор коду, який виявляє синтаксичні помилки, попереджає про потенційні проблеми та надає рекомендації щодо їх усунення. Visual Studio інтегрується з популярними системами контролю версій, такими як Git, Azure DevOps, та інші.

Visual Studio має розширену підтримку популярних .NET-фреймворків, таких як ASP.NET, Entity Framework, WPF та інші. IDE включає вбудовані інструменти для роботи з цими фреймворками: потужний дебагер, редактор XAML, інструменти для тестування та аналізу продуктивності, а також автоматичні генератори коду. Це дозволяє зосередитися на розробці функціональності без витрати часу на налаштування середовища.

Для реалізації програми було обрано мову програмування C# через зручність, великий вибір бібліотек підтримку Windows API за замовчуванням.

Також, дана мова програмування відома у сфері високошвидкісних програм і надає велику кількість інструментів у системній області розробки.

У проекті використовуються здебільшого системні бібліотеки які у C# називаються просторами імен.

Простори імен System.Threading.Tasks та System.Threading які забезпечують інструменти для ефективного управління багатопотоковістю та асинхронними операціями. Функції включають створення, синхронізацію та управління потоками, що дозволяє вирішувати складні завдання одночасного виконання коду та уникати нерівномірного навантаження на сучасних процесорах. Це простір імен є ключовим у створенні додатків, які мають максимально ефективно використовувати апаратні ресурси, залишаючись при цьому зрозумілими та керованими.

Простір імен у .NET System.Linq надає інструменти для виконання запитів до колекцій об'єктів у декларативному стилі, використовуючи мову запитів LINQ. Це універсальний підхід до роботи з даними, що забезпечує зручність, читабельність і можливість спрощення складних операцій над колекціями.

Ключові особливості підтримка операцій фільтрації, проекції, сортування, групування, об'єднання, агрегації та іншого з уже максимально ефективними алгоритмами в середині. LINQ-операції виконуються лише під час реального доступу до даних, що дозволяє оптимізувати використання ресурсів. Linq допомагає уникати складного імперативного коду, забезпечуючи інтуїтивний і виразний підхід до роботи з даними будь-якої складності.

System.Security.Principal простір імен у .NET, який забезпечує основну функціональність для управління та перевірки ідентичності користувачів та їхніх ролей у системі безпеки. У проекті він потрібен для визначення рівня доступу додатку до даних системи. Система Windows доволі сильно розмежовує простір користувачів та адміністраторів і доступ до сенсорів апаратних компонентів та деякої інформації про систему просто не доступний для користувачів.

Наступні простори імен використовуються для доступу до моніторингових метрик тим чи іншим способом.

System.Diagnostics забезпечує інструментами для моніторингу, налагодження, вимірювання продуктивності та діагностики додатків. Відкриває доступ до багатьох інструментів з яких в контексті цієї роботи були корисними PerformanceCounter та Stopwatch. Цей простір імен невід'ємна частина інструментарію розробників, яка спрощує виявлення вузьких місць у коді та моніторинг параметрів комп’ютерів.

System.Management забезпечує доступ до інструментів управління Windows через WMI (Windows Management Instrumentation). Цей простір імен дозволяє розробникам отримувати інформацію та виконувати дії, пов'язані з апаратним забезпеченням, операційною системою, мережами та іншими компонентами системи. У контексті роботи використовується для отримання даних про апаратну частину комп’ютера та оновлення списку процесів.

Для того щоб все працювало коректно треба щоб в системі працював Інструментарій WMI. Встановлюється він за замовчуванням на всіх платформах windows desktop та server. Однак, деякі постачальники WMI можуть бути встановлені або не встановлені в залежності від випуску та конфігурації ОС. Наприклад, постачальник SNMP не ввімкнений за промовчанням, а постачальник інсталятора Windows, не встановлений за промовчанням у 64-розрядних операційних системах.[3]

У складі цього простору імен є багато класів для керування та отримання даних. Середних найкориснішим для моніторингу є ManagementObjectSearcher, також є такі класи як ManagementEventWatcher який дозволяє слухати й реагувати на події в системі, наприклад, підключення нового пристрою. Загалом цей простір імен надає набагато ширший спектр функцій ніж використовує розроблена програма. Він є великий помічником в задачах: системного адміністрування для керування службами, зміна конфігурацій, автоматизація завдань методами написання скриптів, додатків для автоматизації управління ПК і серверів інвентаризація: збирання інформації про апаратне та програмне забезпечення для побудови бази даних інвентаризації.

System.Runtime.InteropServices теж використовуваний для реалізації моніторингу простір імен .NET, який забезпечує інструменти для взаємодії з кодом, написаним на мовах, що використовують платформу Win32, COM або інші нативні API. Він реалізує імпорт DLL для моніторингу ПК цей простір імен дозволяє використовувати функції з нативних бібліотек Windows для отримання низькорівневих даних про систему. Завдяки атрибуту DllImport, можна викликати функції з бібліотек, таких як kernel32.dll, user32.dll або psapi.dll, які можна використовуються для моніторингу ресурсів ПК або активності користувача. Простір імен дозволяє працювати з нативними структурами, які часто потрібні для викликів WinAPI, через атрибути StructLayout і FieldOffset. Клас Marshal дозволяє працювати з дескрипторами і COM-об'єктами, таким чином можна реалізувати керування деякими Windows програмами які підтримують взаємодію через COM, наприклад Microsoft Excel.

LibreHardwareMonitor.Hardware це простір імен з сторонньої бібліотеки яка використовується для моніторингу апаратного забезпечення комп’ютера. Цей компонент дозволяє отримувати дані про стан різних апаратних компонентів, таких як процесор, оперативна пам’ять, накопичувачі, материнська плата, відеокарта, блок живлення тощо. З переваг використання підтримка широкого спектра центральних процесорів та графічних процесорів NVIDIA, AMD та Intel, датчики, вбудовані в материнські плати різних виробників. Аналогічною цій бібліотеці є бібліотека OpenHardwareMonitor, на базі якої навіть розроблений однойменний додаток. Але під час розробки виявилась не сумість з деякими відеокартами NVIDIA через що і відбувся перехід на LibreHardwareMonitor.

Для взаємодії з файловою системою використовуються простори імен System.Xml та System.Data.SQLite.

System.Xml забезпечує підтримку для роботи з XML-документами. Його можна використовувати для читання, запису, перевірки та обробки XML-файлів, що є основою файлу параметрів у конкретній доробці.

System.Data.SQLite — це ADO.NET-провайдер для роботи з базами даних SQLite у .NET. SQLite це вбудована база даних, яка не потребує сервера й зберігає дані в одному файлі. Фактично підключення до бази даних це відкриття потоку файлу. Дані у цьому зберігаються у не читабельному для людини форматі та дані частково стискаються.

ADO.NET має багатий набір компонентів для створення розподілених додатків, які спільно використовують дані. Це невід'ємна частина платформи .NET Framework, яка надає доступ до реляційних даних.

Для спрощення деяких мережевих все ж таки будемо використовувати Json. Системний простір System.Text.Json для цього підходить, бо високопродуктивний і зручний для використання API у .NET для роботи з JSON-даними. Цей простір імен забезпечує серіалізацію та десеріалізацію об'єктів, а також дозволяє ефективно працювати з JSON-структурами безпосередньо. Він є легковаговою альтернативою Newtonsoft.Json і інтегрується у .NET Core та .NET Framework.

System.Net.Sockets надає інтерфейси для мережевого програмування. Він забезпечити доступ до протоколів TCP, UDP і управління сокетами для побудови клієнт-серверних додатків. Оскільки у додатку не заплановано веб-інтерфейс мережева взаємодія по цим протоколам більш ніж достатня.

Серед бібліотек для інтерфейсу є сторонній фреймворк MetroFramework.

MetroFramework це бібліотека для створення сучасних і стильних користувацьких інтерфейсів у Windows Forms. Вона забезпечує дизайн у стилі “Modern UI”, який нагадує вигляд програм Windows 8/10, з акцентом на мінімалізм.

З основних переваг є сучасний дизайн, швидка зміна тем та кольорів, розширені варіанти кнопок полів вводу та інших елементів керування, широка кастомізація вже у скомпільованій програмі.

Щоб забезпечити адаптивність інтерфейсу для роботи в режимі розділеного екрана, я об'єднав елементи MetroFramework із класичними Windows Forms. Це рішення дало змогу досягти гнучкого компонування і масштабування, але призвело до втрати швидкого перемикання тем, яке підтримує MetroFramework.

Тепер кожен елемент Windows Forms вимагає окремого налаштування стилю, оскільки стандартні компоненти Windows Forms не інтегруються з системою тем MetroFramework.

У якості правильного рішення на цю проблему було б використання класів-обгорток для додавання підтримки тем у стандартні елементи. Але тоді я втрачаю ще одну зручність розробки, елементи з наслідуванням можна створювати лише з коду програми, вбудований редактор Visual Studio такий елемент не додасть.

Незважаючи на зростання складності, такий підхід забезпечив правильне масштабування та адаптивність для багатозадачності якої не було у просторі імен MetroFramework.Form.

Accord.Statistics підпростір імен у бібліотеці Accord.NET, який надає широкий набір інструментів для статистичного аналізу, обробки даних та ймовірнісного моделювання. Він використовується для виконання складних математичних обчислень, пов'язаних із вивченням розподілів даних, оцінкою параметрів і побудовою статистичних моделей.

Він містить реалізації основних розподілів, таких як нормальний, пуассонівський, експоненціальний, біноміальний тощо. Можливість обчислення функцій щільності, кумулятивних функцій, генерації випадкових значень. Засоби для розрахунку середнього, дисперсії, коваріації, кореляції та інших характеристик даних. Підтримує тестування гіпотез, статистичні тести, включаючи t-тест, тест ANOVA, тест хі-квадрат, тест Манна-Уітні та інші. Можливість перевірки нормальності розподілу, моделювання часових рядів і прогнозування. Accord.Statistics Чудово підходить для аналізу моніторингової статистики тому використовується у інструменті визначення найперевантаженішого компонента системи.

Microsoft.Win32.RegistryKey це клас у .NET, що дозволяє працювати з реєстром Windows. Він надає методи для створення, читання, запису та видалення ключів і значень реєстру, а також для управління дозволами та структурою реєстру. Знадобиться отримуючи корисну інформацію для адміністратора.

3.1.2 Структура програми агента

Програма призначена в першу чергу для роботи у фоні. Основний підхід, це зменшення використання ресурсів моніторингом, але утримання повноцінної роботи функцій. Увага приділена об’єму обов’язкової пам’яті який виділяється для роботи та кількістю потоків які працюють одночасно.

Дана програма мала б мати дуже малу кількість потоків та використовуваної пам’яті, але з самого початку обрана платформа на Windows .NET У будь якому разі має власні комунікаційні канали всередині системи. Але це не змінює ситуації на стільки кардинально, щоб повністю відмовлятись від .NET. Робота з оперативною пам’яттю у Windows працює таким чином, що процес який довго не використовує якісь функції програмного коду, то ця частина оперативної пам’яті кешується та вивантажується у файл підкачки. Через це чітко визначити скільки витрачає оперативної пам’яті програма доволі важко, але можна підрахувати скільки витрачає купа та стек програми. Згідно з профайлером Visual Studio програма використовує без відкритого вікна 2,2 мегабайта пам’яті. Що до потоків у програмі їх 4 основних та декілька тимчасових які працюють тільки до того моменту поки треба записати у базу

даних або надіслати статистику або повідомлення про попередження на сервер.

Потоки реалізовані для того, щоб додаток не створював високого навантаження на одне ядро процесору, а також при відкритті інтерфейсу не гальмував його. Основний потік програми включає в себе збір та обробку легко отриманих або основних для будь якого комп'ютера метрик моніторингу. Основні це зазвичай навантаження на процесор, використання оперативної пам'яті та завантаження диску. У деяких випадках основним ще вважають навантаження на мережевий інтерфейс, але в реальності серйозно ніхто не враховує цей параметр на звичайних ПК. В окремі потоки виділені такі процедури як моніторинг процесів, температурні сенсори, та підпрограма виявлення підозрілих процесів. Оновлення інтерфейсу реалізоване через систему подій. Таким чином інтерфейс не виконує ніяких дій, щодо слідкування за оновленнями всі оновлення викликаються після всіх розрахунків з класу моніторингу. Оновлення інтерфейсу хоч і відбувається з головного потоку, але багато часу не займає через алгоритми, які мінімізують кількість викликів оновлення. Схематично показані потоки на рисунку 3.1.

Структура класів програми агента в першу чергу орієнтована на модульність та незалежність від інтерфейсу.

З іншої сторони поява незалежного інтерфейсу викликає потребу зв'язку між даними та об'єктами на екрані. На допомогу у такому випадку приходить статичний клас який зберігає вказівники на екземпляри класів. Перевага такого підходу у тому, що доступ до публічних даних нічим не обмежений, не потребує передавання параметрів та реалізація будь яких нових елементів не займає багато часу. У програмі цей клас отримав назву DataBridge. Крім інстанції на основні вузли як основний клас програми та систему подій також має кілька публічних прапорців та зберігає поточні параметри.

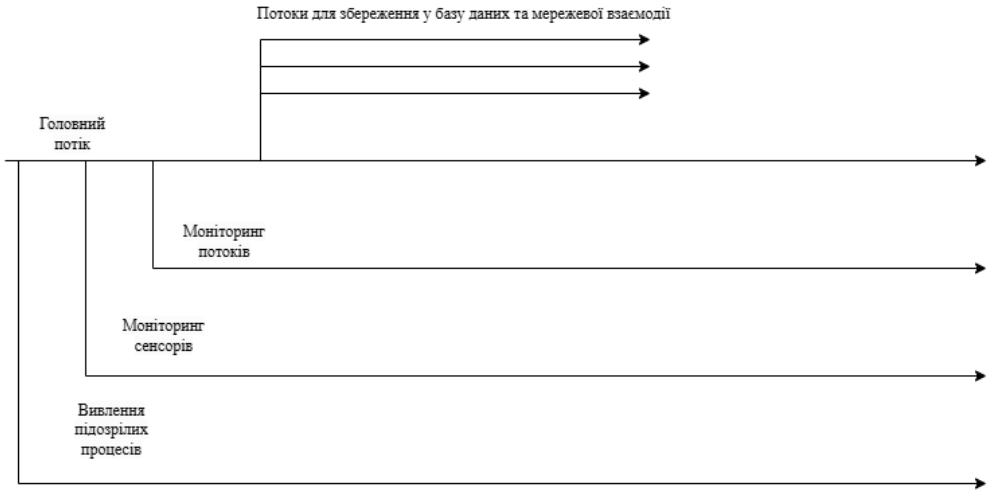

Рисунок 3.1 - Схема потоків програми.

Програма агенту розподілена на класи для забезпечення модульності та підвищення надійності при неможливості ініціалізації компонентів програми. Головним та початковим класом програми агента є клас MonitoringSystem. Він містить таймери оновлень, посилання на інші компоненти моніторингу, прапорці для керування потоками. Разом з цим у клас вшитий моніторинг за основними метриками системи. Тобто моніторинг за загальним навантаженням на процесор, оперативну пам’ять, диски, мережу.

Іншим також обов’язковим стартовим класом система подій яка отримала назву MonitoringSystemEvents. У цьому класі зібрані всі глобальні події які можуть виникнути у програмі та які потребують синхронізації та оповіщення динамічно ініціалізованих класів. Основний сценарій використання подій це оновлення інтерфейсу. Наприклад фоновий потік виявлення підозрілих процесів працює постійно у фоні, але якщо відкрити вікно для перегляду знайдених процесів і знайдеться ще один, то треба якось оновити список. Тут як раз і підходить використання подій. Відкрившись вікно на інтерфейсі підписуються на подію і якщо вона виникає то спрацьовує метод приписаний на виклик події і таким чином список оновлюється. Після виклику закриття, інтерфейс відписується від події.

Наступний клас без якого не будуть повноцінно працювати багато інших функцій це ProcessesManager. Фактично це просто збирач даних по процесам за виключенням можливості понизити пріоритет чи примусово завершити процес. Дані зібрані цим класом завжди публічні тому у інших класів немає проблеми для отримання оновлення. У своєму складі окрім списку моніторингових результатів по процесам клас також має записи про найважчий на даний момент процес та його показники та звісно потік виконання та прапори керування ним. Рутина оновлення моніторингових даних процесів для кращого представлення зображена на рисунку 3.2

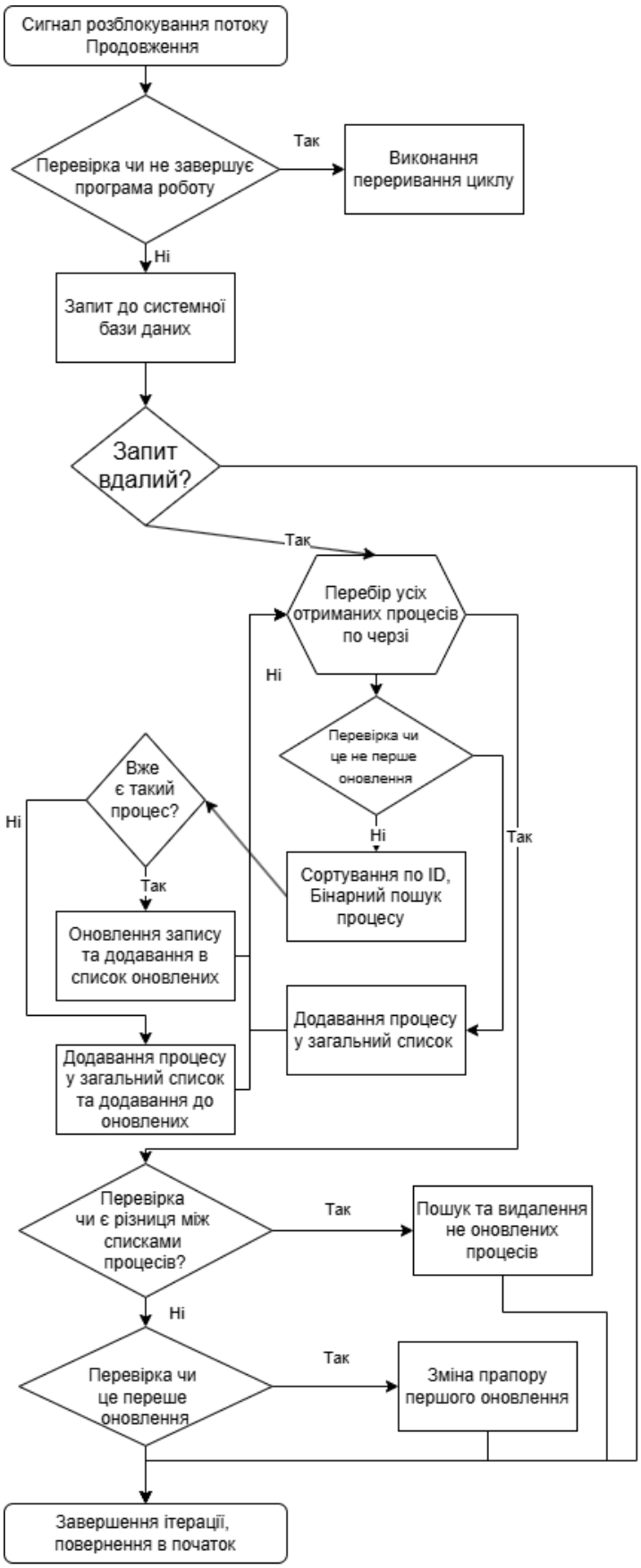

Рисунок 3.2 - Алгоритм циклічного оновлення моніторингових даних процесів.

Клас SensorsMonitor виконує моніторинг за сенсорами на кшталт температурних, частотних або сенсорами потужності. В основі класу лежить використання бібліотеки LibreHardwareMonitor. Усі процеси реалізовані у окрему потоці. Алгоритм трохи спрощений, бо розробник бібліотеки вже заклав збір елементарної статистики та й набір даних менший у кожному елементі списку. Рутина оновлення моніторингових даних сенсорів для кращого представлення зображена на рисунку 3.3

Рисунок 3.3 - Алгоритм циклічного оновлення сенсорів

Моніторинг диск на більш детальному рівні реалізований набором класів DiskDetailedMonitor. Кожен інстанс класу моніторить конкретний фізичний диск. Клас MonitoringSystem реалізує стартову ініціалізацію масиву та періодичну перевірку на нові пристрої у ПК. Під час ініціалізації масиву

класів, програма шукає усі доступні диски у системі, створює потрібну кількість елементів та запускає оновлення даних моніторингу для кожного диску. У процесі періодичних оновлень при виявленні втрати зв'язку з пристроєм він буде видалений з масиву моніторингу та буде вважатись відключеним. Цей клас не є обов'язковим. Детальна статистика не збирається та сам клас може бути вимкнений з програми для додаткового зменшення навантаження.

NetIntfPanelDetailedMonitor клас який повторює усі характеристики DiskDetailedMonitor, але він звісно призначений для детального моніторингу кожного з мережевих інтерфейсів, і його повністю вимкнути не можна, бо він є обов'язковим для збору статистики.

Клас бази даних отримав ім'я DBCommunicationi включає в себе методи для запису та читання статистики з пам'яті ПК. Усі операції виконуються асинхронно для запобігання зависання програми у випадку довгого очікування диску, або очікування завершення читання, іншою частиною програми. Потоки використовують додаткові мютекси для запобігання накладання і таким чином при накладанні задач стають у чергу для запитів до файлу бази даних.

Клас який реалізує формування та відправку всіх мережевих пакетів для синхронізації статистики або оповіщення адміністратора, отримав назву NETSending. Особливістю є те, що клас використовує Json для пакування статистики. Це зумовлено, що за раз краще відправляти дані з кількох таблиць, а у json замість масиву можна серіалізувати null, що дає можливість надсилати статистику навіть тоді коли у деяких таблиць не було оновлень.

ProcessMonitoring це клас програми який зберігає та виконує оновлення даних статистики конкретного процесу заданого користувачем за допомогою інтерфейсу. Якщо інтерфейс не використовується цй клас навіть ніколи не ініціалізується. З особливостей моніторингу цим класом: подовжений час збереження статистики реального часу. Підрахунок кількості записаних та прочитаних з дискуданих. Моніторинг реалізований за допомогою PerformanceCounter тому результати будуть дещо точнішими ніж у загальному списку процесів.

Наступна низка класів присвячена низькорівневій обробці статистики.

Перший це звісно AnalyzerOfMemoryAnomalies який займається очищенням аномалій у статистиці помилок оперативної пам'яті. Він використовує максимально швидкий метод та порівнює псевдо нормальний результат згладжування аномалій та оригінальні показники на часовому відрізку. За замовчуванням такий аналіз відбувається раз на 5 хвилин. Запуск відбувається за командою головного класу програми, тому за непотрібністю цей аналіз легко відключити.

Пошуком підозрілих процесів займається SuspicionProcessSearcher який має свій власний потік, працює у звичайному режимі майже як SensorsMonitor або ProcessesManager. Але один раз на 12 оновлень робить аналіз всього списку процесів на аномальну поведінку. відносно зібраної активності користувача. кількість оновлень обрано не випадково. А

підлаштоване під оновлення вторинного таймера на 5 секунд, щоб аналіз списку відбувався щохвилини.

Клас StatisticsResolver створений для створення параметризації статистики. Саме через чей клас проходять запити з визначенням того, що саме користувач хоче вважати високим низьким чи середнім навантаженням. Клас сам робить запит до бази даних та аналізує отримані дані на основі параметрів, які йогому були видані. Класи форм та модульних візуальних елементів займаються в основному виводом інформації, хоча на деяких з них закладені важливі функції програми.

Клас FormMain є головною формою програми з усіма вкладками, доступом до всіх інструментів та є основним способом перегляду статистики користувачем на локальному пристрої. Саме цей клас охоплює основну більшість подій у MonitoringSystemEvents, бо в залежності від обраної на даний момент вкладки може оновлювати інтерфейс на подію кожного таймера.

Дуже важливий клас візуальної складової це FormSettings який можна ініціалізувати з форми яку відображає FormMain. Відповідно до назви вікно цього класу форми дозволяє змінювати налаштування. Особливо це стосується вимкнення компонентів програми, змінення зовнішнього вигляду та має можливість очищення статистики.

Група трьох класів: PCComponentsPerformensAnalyserForm, PCMaintenancePredictionForm та SuspicionProcesesForm Представляють з себе просто невеликі вікна для перегляду результатів аналізу статистики. У випадку SuspicionProcesesForm перегляду підозрілих процесів, вирізняється серед усіх тим, що оновлення відбуваються з фонового потоку. Всі ці вікна напряму роблять запити у базу даних тому асинхронний підхід реалізований на етапі проектування класу комунікації з базою даних тут підходить, бо так навіть читання повної таблиці статистики навряд чи спричинить збій у роботі програми.

Невеликий клас LoadForm призначений для візуалізації процесу запуску основного інтерфейсу клієнтської програми. Призначений просто щоб у випадку більш тривалого завантаження у користувача не було хибних підозр що вікно не почало відкриватись взагалі.

Класи DiskPanel, NetIntfPanel, CPUChartElement це класи візуальних компонетів які наслідують UserControl, Таким чином створення однотипного набору елементів з коду форми спрощується в рази.

DiskPanel, NetIntfPanel це класи які візуалізують зібрані дані класами DiskDetailedMonitor та NetIntfPanelDetailedMonitor. Виклик оновлення відбувається з головного класу по завершенню збору нових даних.

Як можна помітити з структури програми тут немає ніяких наслідувань а розділення на класи спричинені тільки з питання відділення даних які оновлюються окремим потоком або ці данні дуже мають багато методів націлених тільки на них. Я давно помітив тенденцію що надмірна інкапсуляція наслідування та поліморфізм знижують продуктивність цільової програми. На це є дуже логічне пояснення. Робота з кешем та пам’яттю кардинально

змінюється при розділенні класу на менші класи. Оскільки процесор робить набагато більше маленьких запитів у пам’ять замість одного великого. [7]

Клієнтська програма зважаючи на цільове розповсюдження має бути як і фоновою службою так і зручним інтерфейсом. Тому логіка програми має повністю вимикати інтерфейс коли працює у режимі агенту. Звісно в залежності від компанії та підходів адміністраторів до роботи може і не бути заборони для персоналу у перегляду моніторингових даних тому користувацький інтерфейс може бути дозволений, але зміна параметрів тільки з дозволу адміністратора.

Тому стосовно розповсюдження програма має мати пристосований набір функцій. Розподіл версій клієнтської програми буде виглядадти наступним чином:

Основна версія - включає в себе всі компоненти та можливості відповідно до всіх версій які у версіях мають відрізнятись.

- Сервер статистики та аналізу даних.
- Можливість вказати адресу серверу збору статистики для надсилання.
- Встановлення паролю для захисту від змін звичайними користувачами.
- Інтерфейс для перегляду показників в реальному часі.
- Інтерфейс для перегляду статистики.
- Набір інструментів для аналізу у програмі клієнті.

Версія не корпоративного характеру не буде мати набору інструментів для аналізу на стороні програми агента, бо аналіз статистики може бути важким та довгим процесором і зважаючи на звичайне розподілення обчислювальних потужностей увесь аналіз переноситься на серверну програму.

Версія для домашнього використання не матиме більшого набору функцій, тобто встановлення паролю, серверу статистики, але набір інструментів присутній.

3.1.3 Структура бази даних

База даних не має займати багато місця тому таблиці побудовані таким чином, щоб при стисненні з щохвилинних записів або п'яти хвилинних, як у випадку виявлення аномалій, щоб втрата корисної інформації була мінімальною.

База даних представлена двома файлами через рішення використання SQLLite складається з таких таблиць статистики:

- Основна статистика
- Статистика виявлення аномалій
- Статистика сенсорів температури, потужності, напруги.

Перший файл це статистика детальна, тобто не стиснена погодинно. Вона має зберігатись якийсь час щоб можна було проводити якісний аналіз даних.

Другий файл це стиснена статистика і оскільки мета отримати максимум інформації навіть із такої статистика кількість полів може відрізнятись. Додані поля таблиці будуть націлені на більшу деталізацію початкової статистики.

Основна статистика це таблиця яка представляє з себе одночасно як і основні показники навантажень так і активність користувача так і інші дані що до процесів. Вона оновлюється щохвилинно по події таймеру в головному класі програми. Поля основної щохвилинної таблиці наведені у таблиці 3.1.

Таблиця 3.1 – Поля щохвилинної основної таблиці статистики

Ім’я поля	Тип даних
Мітка часу	DataTime(8 байт)
Середеній відсоток ЦП	float(4 байт)
Максимальний відсоток ЦП	float(4 байт)
Мінімальний відсоток ЦП	float(4 байт)
Час під максимальним навантаженням ЦП	float(4 байт)
Середеній відсоток ОЗП	float(4 байт)
Максимальний відсоток ОЗП	float(4 байт)
Мінімальний відсоток ОЗП	float(4 байт)
Середній відсоток Дискової активності	float(4 байт)
Максимальний відсоток	float(4 байт)
Мінімальний відсоток	float(4 байт)
Середня швидкість	float(4 байт)
Максимальна швидкість	float(4 байт)
Мінімальна швидкість	float(4 байт)

Продовження таблиці 3.1

Ім’я поля	Тип даних
Час під максимальним навантаженням	float(4 байт)
Найважчий процес по CPU	string(16 – 96 байт)

Найважчий процес по RAM	string(16 – 96 байт)
Найважчий процес по швидкості Disk	string(16 – 96 байт)
Процес останнє активне вікно	string(16 – 96 байт)

Таблиця 3.2 – Поля п’ятихвилинної таблиці статистики виявлення аномалій

Ім’я поля	Тип даних
Мітка часу	DataTime(8 байт)
Тип помилок	string(32 - 48 байт)
Початкове середнє	float(4 байт)
Початкове максимальне	float(4 байт)
Початкове мінімальне	float(4 байт)
Рівень відсікання	float(4 байт)
Нормалізоване Середнє	float(4 байт)
Відсоток видалених даних після нормалізації	float(4 байт)

Таблиця 3.3 – Поля щохвилинної таблиці статистики температур

Ім’я поля	Тип даних
Мітка часу	DataTime(8 байт)
Максимальна температура процесора	string(32 - 48 байт)
Мінімальна температура процесора	float(4 байт)
Середня температура процесора	float(4 байт)
Максимальна Температура відео ядра	float(4 байт)
Мінімальна Температура відео ядра	float(4 байт)
Середня температура відео ядра	float(4 байт)

Окрім основної статистики звісно зберігаються також статистика виявлення аномалій в роботі пам’яті та температур, їх поля перераховані у таблиці 3.2 та 3.3.

У погодинній таблиці статистики кількість полів у записі не зменшується, а навіть збільшується, оскільки дані, представлені середнім значенням у щохвилинних таблицях, розбиваються на додаткові поля. Це

дозволяє аналізувати середні значення майже без втрат бо взятий інтервал становить 20%.

Наприклад, для параметра середній відсоток CPU додаються п'ять полів замість одного:

- Середній відсоток ЦП (0-20%)
- Середній відсоток ЦП (20-40%)
- Середній відсоток ЦП (40-60%)
- Середній відсоток ЦП (60-80%)
- Середній відсоток ЦП (80-100%)

Аналогічний підхід застосовується до інших середніх значень, таких як середній відсоток оперативної пам’яті, середній відсоток дискової активності, середня швидкість, середня температура тощо. Таким чином, у погодинній таблиці зберігається деталізація статистичних даних, що дозволяє проводити більш аналіз системних параметрів у межах годин приблизно так само як і з щохвилинними.

Крім того, у записі залишаються всі інші поля з попередніх таблиць, наприклад, максимальні, мінімальні значення, час під максимальним навантаженням, найважчі процеси за кожним параметром.

Рівні агрегації моніторингових даних у системі.

а) Сирі дані зібрані протягом хвилини або п’яти хвилин.
б) Дані після аналізу однієї хвилини моніторингу.
в) Проаналізовані дані виявлення аномалій та підозрілих процесів.
г) Стиснені дані погодинно в попереднім аналізом кожної години.

Треба зауважити що показники навантаження на диск це відносна завантаженість диска, тобто скільки часу диск був зайнятий обробкою запитів протягом певного інтервалу часу, але насправді цей показник сильно корелює з чергою виконання і фактично 100% навантаження означає лише те що за попередній проміжок часу диск не виконав всю поставлену чергу.

3.1.4 Структура програми серверу

Програма серверу, має чітко розділений інтерфейс як і програма агента.

Окрім збору статистики даних з агентів, програма серверу звісно підтримує дещо простіші способи перевірки роботи. Наразі прийнято до уваги найпростіший метод, це командою ping перевіряти чи ввімкнений в мережу пристрій. Звісно для того щоб не перевищити нормальний рівень запитів і не створити мережевих штормів рекомендована частота оновлення максимум раз на 5 секунд хоча у реальній ситуації нормальною частотою могли б бути значення 1-2 хвилини.

Звісно для моніторингу через пінг для програми треба додати IP адреси та імена пристроїв. Також корисна функція яку важко додати у інші системи моніторингу це моніторинг тільки у робочий час. Якщо мета слідкувати за збоями робочих станцій, або інших не працюючих цілодобово пристроїв то в

цілому такий підхід потрібен, бо зі стандартним підходом який розрахований на сервери адміністратор буде бачити багато непотрібних сповіщень.

Як було зазначено вище користь моніторингу робочих станцій за допомогою ping може проявитись у виявленні несправності пристроїв фізичного та канального рівнів. Особливо це буде помітно якщо постійно пропадає зв'язок, або має високу для локальної мережі затримку якась група користувачів, що може вказувати на несправний або працюючий в неправильних умовах комутатор.

Коли сервер запускається то одразу вмикає доступ для надсилання щохвилинних оновлень з агентів та запускає потік пінгування пристроїв на присутність онлайн. Звісно надсилаючи щохвилинно статистику, пристрої вже заявляють про те що працюють, тому пінг для перевірки онлайн вже фактично не потрібне. Його можна лишити якщо важливим є час відповіді з хостів. Тому у програмі реалізований список який синхронізований з локальною базою даних у якій зберігаються IP адреси імена хостів та часові періоди моніторингу за ними.

Під час роботи програми у цьому списку створюються відмітки про останній час зв’язку з простоєм і якщо відмітка не набрала достатнього часу для пінгу, то ping не відбувається. За виключенням встановлення параметру пінгувати у будь якому разі.

Оскільки сервер зберігає копії всіх даних статистики з клієнтів це означає, що ми можемо реалізувати всі дані з комп’ютерів як і передбачалось. Для цього у інтерфейсі сервера передбачені вкладка аналогічними інструментами як і у клієнті от тільки аналіз можна проводити одразу на всі комп’ютери.

Що до реалізації отримання посилок передбачається час від часу доволі високе навантаження, адже декілька комп’ютерів можуть майже одночасно синхронізувати статистику. Щоб не затримувати мережеві пакети допоможе розпаралелювання процесу прийняття та записування у базу. Але це можна зробити двома способами.

Системний простір імен у C# дозволяє дуже лаконічний інтерфейс для створення потоків, що дозволяє створювати велику кількість потоків які можуть не жорстко синхронізовано працювати та мають завершуватись самостійно. Такий підхід уже використаний у програмі агента для запису у базу даних, щоб не допускати затримок роботи та не отримувати помилки через зайнятість файлу. Але на відміну від програми агента де максимальна кількість одночасних запитів може становити 6, то у випадку сервера така буде дорівнювати кількості пристроїв. Як відомо при великій кількості потоків навіть сучасні процесори переходять у стан постійного переключення між потоками або трашингу. Звісно для цього треба багато буде багато програмних потоків, але зважаючи що сервери і так дуже перевантажені краще не збільшувати вірогідність такої ситуації.

Трашинг процесора це стан, коли процесор витрачає більше часу на обробку операцій, пов’язаних з управлінням пам’яттю, та кадрами обчислень ніж на виконання корисної роботи. На рисунку 3.4 відображена тенденція

появи цього явища. Це зазвичай відбувається через те, що програма або система активно використовує віртуальну пам’ять, і через брак фізичної пам’яті постійно відбувається підвантаження і вивантаження пам’яті потрібної з диска. Звісно при надто великій кількості потоків навіть великий об’єм оперативної пам’яті не допоможе, бо планувальник буде намагатись охопити все одно всі процеси і пам’ять не зможе так швидко відповісти. Тому точка показана на графіку рисунка 3.4 є абсолютно умовною та не може бути прив’язаною до конкретних значень.

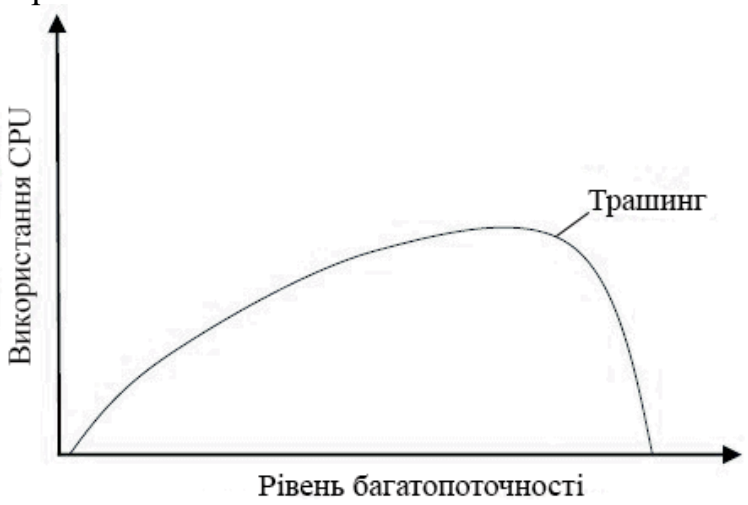

Рисунок 3.4 – Умовна залежність появи трашингу

Трашинг можна помітити через моніторинг Page Faults. Якщо програма генерує багато page faults, це може бути ознакою недостатньої кількості доступної фізичної пам’яті. Якщо помилок мало це вказує на те що скоріш за все дані просто завеликі і тому діляться між оперативною пам’яттю та файлом підвантаження. Прикладом як правило є відтворення відео.

Також трашинг може бути помітним по фактору високого використання диска. Часу на підвантаження потрібно багато тому висока активність диска дуже помітна серед загальної статистики. Система постійно переміщує сторінки між диском і фізичною пам’яттю. Часто зрозуміти яка програма є причиною виникнення дуже важко, бо висока активність диска є наслідком і високий показник відображається на процесі системи.[6]

Тому, щоб програма сама не стала причиною такого неприємного явища варто використати менш зручний спосіб, це створення черги задач і потоку виконувача. У такій ситуації доречно адже диск для запису даних один, а клієнтів які їх хочуть записати багато тому комусь доведеться чекати. Більш детально порівняння двух методів розпаралелювання методів зображено на рисунку 3.5.

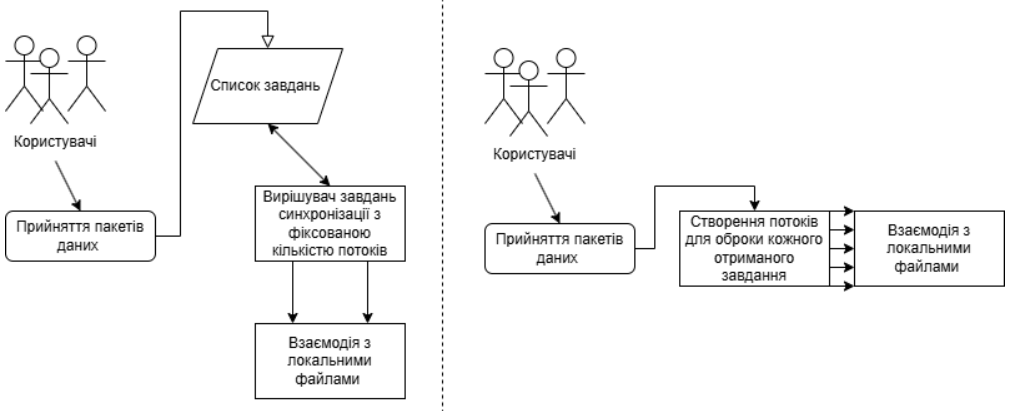

Рисунок 3.5 – Принципова робота алгоритмів розпаралелювання на сервері.

Програма серверу надає три шляхи взаємодії агентам з сервером, один це просто надсилання статистики без ніяких інших команд, а інший це отримання старих даних статистики які були завантажені на сервер, але видаленні з локального пристрою. Третій це надсилання клієнтами сповіщень про небезпеку або якусь загрозу. Туди входять сповіщення про знайдені підозрілі процеси, брак місця на логічних дисках, високі температури, та стан S.M.A.R.T. диску, на випадок якщо він погіршився.

Зазвичай коли організовують між машинне спілкування тим більше з використанням JSON, використовують повноцінне спілкування між комп'ютерами. Але оскільки у нашому випадку гарантом доставки є протокол TCP та у будь якому випадку на сервері дані відобразяться навіть при неуспішному записі у базу даних, немає сенсу надсилати зайвих повідомлень. Щоб не організовувати повноцінний Rest API, який у цій задачі ніколи не буле реалізований повністю з усіма його перевагами, було прийняте рішення зробити елементарну систему синхронізації. У випадку якщо приходить пакет з даними статистики то він має спеціально сформовану структуру для правильного розпізнавання сервером хто саме та що саме надіслав клієнт. Приклад такого класу для прийняття нерозпізнаних даних на рисунку 3.6

```
public class JsonWrapper {

      public string Type { get; set; }

      public JsonElement Data { get; set; }
}
```

Рисунок 3.6 – Клас для отримання даних з попереднім розпізнаванням призначення.

Є інша проблема синхронізації яка вирішена не найбільш кращим чином. Оскільки агент зберігає коли останній раз він надсилав статистику по кожній категорії у файлі параметрів, то звісно після довгої відсутності зв'язку він має надіслати усі що не надіслав до цього. Максимальна кількість визначена у програмі це 300 записів що дорівнює п'яти годинам простою. Якшо цієї кількості не вистачило то надсилається ще один пакет і так поки все не синхронізується. Проблема тут полягає у тому що у випадку виникнення такої ситуації на серверу уже будуть більшість записів то сервер має перевірити чи справді з нього немає такого запису перед внесенням, а у випадку синхронізації статистики помилок доведеться навіть перевірити ще одне поле, тип помилки. Проте якщо дата присланого більша ніж остання доступна у базі даних, то не треба буде перевіряти усі інші. Тому на жаль процес оновлення бази даних на сервері це важка процедура яка включає перевірку на те чи немає у списку вже записів з таким же часом. Саме тому для зменшення переривань на диску кожен пристрій пишеться у свою окрему базу даних.

Інтерфейс для серверу надає доступ до перегляду списку пристроїв які коли небудь надсилали статистику. Робить він це переглядаючи імена всіх баз даних які сформувались внаслідок отримання статистики, та відображає останні надіслані дані які збираються теж в окрему базу даних.

3.2 Оптимізація критичних вузлів програми

Під критичними вузлами маються на увазі частини коду які часто викликаються або можуть створити велике навантаження.

У програмі є 3 основних таймерів які працюють весь час. Щоб уникнути надлишковості та не створювати динамічну систему з різними часовими періодами було встановлено таймери на одну та п'ять секунд як первинний та вторинний таймінг. Третій таймер спрацьовує раз на хвилину, призначений для попереднього аналізу та збереження статистики. По кожному тику таймера виконується код який має збирати дані з показників системи.

Звісно найбільш важливою є оптимізація коду який виконується кожної секунди. Під час розробки були обрані API для кожної метрики. Для що секундного оновлення треба у будь якому разі обирати найшвидші інструменти збору. Відповідно на цьому етапі перевага віддається методам які використовують прямий запит до ядра. Щоправда у випадку отримання інформації з про диски отримання даних напряму від ядра не показав переваг по часу виконання, а по пам'яті використовував навіть трохи більше за PerformanceCounter. Щодо моніторингу мережі така ситуація як і з дисковою підсистемою. Ситуація дещо ускладнена тим, що оцінити перевантаженість мережевої підсистеми при наявності декількох адаптерів неможливо, тому фактично треба виводити навантаження з того адаптера який навантажений найбільше. Тому є сенс використовувати прямий запит тільки для CPU та

RAM. Приклад коду з використанням імпорту Kernel32.dll зображено на рисунку 3.7.

```
[DllImport("kernel32.dll")]
        public static extern bool GetSystemTimes(out SYSTEMTIME
lpIdleTime, out SYSTEMTIME lpKernelTime, out SYSTEMTIME lpUserTime);

        [StructLayout(LayoutKind.Sequential)]
        public struct SYSTEMTIME
        {
            public uint dwLowDateTime;
            public uint dwHighDateTime;
        }

        static SYSTEMTIME idleTimeBefore, kernelTimeBefore,
userTimeBefore;
        static void CPULoadInitKernel()
        {
            GetSystemTimes(out idleTimeBefore, out kernelTimeBefore,
out userTimeBefore);
        }
        static float GetCPULoadKernel()
        {
            SYSTEMTIME idleTimeAfter, kernelTimeAfter, userTimeAfter;
            GetSystemTimes(out idleTimeAfter, out kernelTimeAfter,
out userTimeAfter);

            long idleDiff = (long)(idleTimeAfter.dwLowDateTime -
idleTimeBefore.dwLowDateTime);
            long kernelDiff = (long)(kernelTimeAfter.dwLowDateTime -
kernelTimeBefore.dwLowDateTime);
            long userDiff = (long)(userTimeAfter.dwLowDateTime -
userTimeBefore.dwLowDateTime);

            idleTimeBefore = idleTimeAfter;
            kernelTimeBefore = kernelTimeAfter;
            userTimeBefore = userTimeAfter;
            long totalDiff = kernelDiff + userDiff;
            float cpuUsage = (float)(totalDiff - idleDiff) /
totalDiff * 100;
            stopwatch.Stop();

            return cpuUsage;
        }
```

Рисунок 3.7 – Реалізація отримання відсотку навантаження на процесор за допомогою kernel32.dll

Вторинний таймер бере на себе трохи важчі оновлення, це оновлення ресурсів всього списку процесів та температурні сенсори, а також виклики оновлень для виявлення підозрілих процесів. Більшість вторинних оновлень, це сигналізування потокам які виконуються у фоні, а єдиний не відділений током процес це отримання кількості помилок оперативної пам’яті. У свою чергу разом з цим потоком запускається і аналіз аномалій тому раз на п’ять хвилин навантаження може значно зрости на короткий відрізок часу.

API який використовується для оновлення списку процесів це ManagementObjectSearcher, для сенсорів між Windows API є ще бібліотека LibreHardwareMonitor. Як було доведено експериментально, під час розробки,

час запиту збільшується від кількості опитуваних компонентів системи не залежно від використовуваної бібліотеки.

Третій же таймер щохвилинний він читає велику кількість даних тому у будь якому разі виконує найдовші по доступу операції початкового аналізу статистики. По цьому таймеру відбувається перебір усіх даних на даний момент та формування основного списку статистики для збереження у базу даних та надсилає завдання для формування та надсилання статистики на сервер. Окрім основних показників також переписуються показники температури і до того ж не важливо яка саме частота оновлення температури чи процесів.

3.3 Використання користувачем та візуальні рішення

3.3.1 Програма агент

Інструкція конкретного програмного рішення може мати багато підрозділів, але сам індивідуальний інтерфейс не так важливий, бо з самого початку він з’явився з причини потрібності зручної відладки коду. Але в решті решт став достатньо зручним для використання рядовим користувачем. Тому наступний огляд функцій та інструкція до використання деяких елементів буде орієнтована одразу на два сценарії використання. І у випадку якщо користувач використовує агент як програму у індивідуальному порядку та у корпоративній мережі.

Як вже зазначалось запустившись додаток себе показує тільки іконкою у треї панелі завдань. Подвійне натискання зробить спробу відкриття. Адміністратор може параметром встановити пароль на відкриття інтерфейсу. Як показали деякі практичні тести. Через високе загальне навантаження відкриття інтерфейсу може займати багато часу. Тому було додане невелике віконце яке сигналізує, що завантаження принаймні почалось, звісно воно закривається автоматично коли завершується завантаження основного вікна.

Якщо розглядати цей інтерфейс то він має стандартну для більшості Windows додатків архітектуру інтерфейсу. Інтерфейс із вкладками, у якому представлений набір усіх можливих потрібних вкладок. Деякі з вкладок можна закрити зовсім, якщо вимкнути їх у налаштуваннях та перезавантажити програму. Звісно таке вимкнення насправді вимикає не вкладку и компонент моніторингу. У іншому випадку не потрібно прибирати вкладки, бо якщо на вкладку користувач не дивиться то вона не оновлюється та у деяких випадках елементи повністю видаляються. Інтерфейс для виконаний у темній темі тільки тому, що зараз такий підхід є актуальним. Перед повним огляд треба виділити графічний елемент програмування Windows додатків який дуже допомагає при розробці UserControl. В Windows Forms він є розширенням, яке дозволяє створювати власні перевикористовувані елементи керування controls. Він дозволяє збирати елементи керування та відображення, в один батьківський елемент. По своїй суті усі інші елементи у Windows Forms також

мають таку структуру. Тобто набір параметрів відображення та список controls який по своїй суті є вказівниками на інші візуальні класи.

Переваги використання таких елементів включають створення власних елементів керування з унікальною функціональністю та зовнішнім виглядом. Повторне використання UserControl дозволяє створювати елементи керування, які можна повторно використовувати в різних частинах програми або навіть в інших проектах. Це дозволяє ефективно використовувати код та забезпечує єдино образний зовнішній вигляд та функціональність. Кожен такий елемент є своїм класом тому можна створити власні події та методи, які дозволяють комунікувати з ним іншими елементами програми та визначити події, які викликаються при певних діях користувача або змінах стану елемента, і реагувати на них в інших частинах програми.[12]

3.3.1.1 Головна вкладка

Огляд інтерфейсу є сенс почати з того що користувач побачить у першу чергу, головну сторонку. Стандартний її вигляд зображений на рисунку 3.8. Але є ще редагований варіант який дозволяє розміщувати елементи у довільному порядку. У налаштуваннях до нього йде додатковий редактор який дозволяє розмістити доступні елементи у зручній для користувача формі.

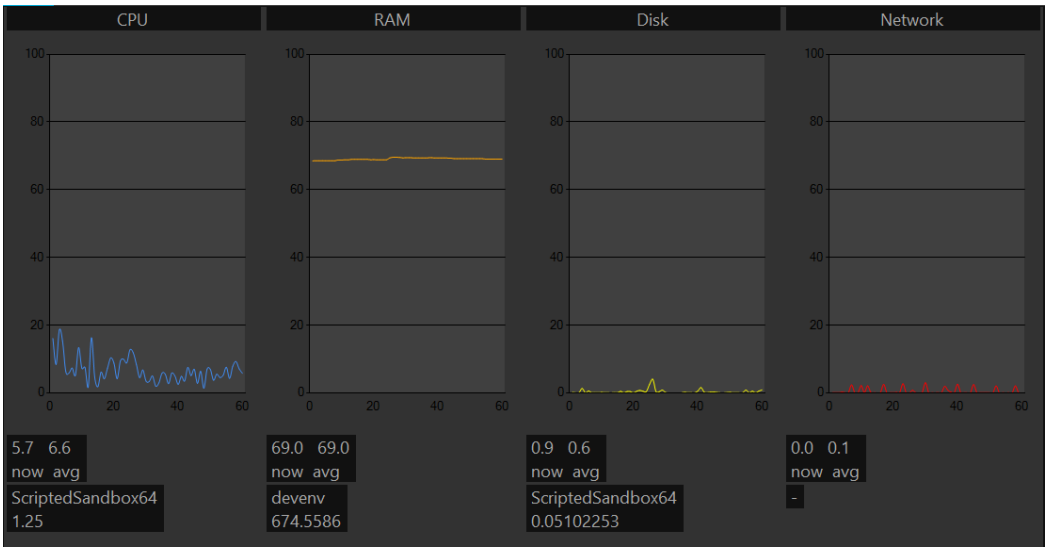

Рисунок 3.8 - Головна сторінка програми агенту

У налаштуваннях програми можна переключитись на додатковий дашборд який замінює стандартну головну сторінку, для цього налаштування є спеціальний редактор інтерфейсу, який дозволяє додавати та змінювати розташування доступних елементів, таких як, графіки, інформативні лейбли, таблиці або деякі інші компоненти інтерфейсу. Користувач може легко переміщувати елементи та встановлювати їх розміри або навіть приховувати

непотрібні елементи, що дає можливість створити найбільш ефективний та персоналізований дашборд.

Звісно за замовчуванням головна сторінка має інтуїтивно зрозуміле та максимальне просте розташування елементів, щоб користувачі могли швидко знайти важливу інформацію. Кожен компонент, такий як графіки, таблиці, або панелі статистики, має чітке та логічне місце, що мінімізує час на пошук необхідної функції.

Можливість налаштування та персоналізації цієї вкладки дозволяє користувачам адаптувати інтерфейс до своїх індивідуальних потреб, покращуючи ефективність та зручність роботи. Гнучкість у розміщенні елементів, зручність в налаштуваннях і динамічні елементи сприяють створенню комфортного та ефективного робочого середовища для кожного користувача.

3.3.1.2 Вкладка моніторингу пам’яті

Наступна сторінка це сторінка детальної інформації про оперативну пам’ять тут можна більш детально побачити скільки використано пам’яті файлом підкачки, скільки виділено віртуальної. Найголовніше на цій сторінці це відображення кількості помилок оперативної пам’яті у реальному часі. Перегляд цих показників допоможе зрозуміти причини повільної роботи, або впевнитись у правильних налаштуваннях. Перегляд графіків помилок у реальному часі може допомогти з оптимізацією програмного забезпечення. Моніторинг цих помилок допомагає ідентифікувати ділянки коду, які надмірно завантажують пам'ять. Спостереження може вказати, коли потрібно оптимізувати алгоритми, структури даних або розмір робочого набору програми. Як було зазначено раніше по Page Faults можна навіть виявити перехід процесора у стан трашингу. Дуже часто таких помилок припускаються при багато поточному завантаженні програми, коли потоків все більшає і розмір даних для передачі більшає і виходить, що процесор не завершуючи очікування переходить далі. Переходи у такий стан звісно поворотні навіть під час роботи системи і іноді навіть не помітні для користувача, але не допускаючи такого переходу продуктивність може бути набагато більшою. Діагностика допомагає визначити, чи потрібне розширення оперативної пам’яті, налаштування пріоритетів процесів або оптимізація апаратної конфігурації. Виявлення незвичайної кількості помилок, наприклад, після оновлення програми чи ОС може вказати на потенційні витоки пам'яті або інші помилки в роботі системного програмного забезпечення. Регулярний моніторинг допомагає запобігти раптовому збою через вичерпання пам'яті. Аналіз помилок Demand Zero Faults дозволяє зрозуміти, скільки пам’яті як розподілити пам’ять між процесами, виявити чи є необхідність оптимізації процесу ініціалізації об'єктів.

Також на сторінці розміщена кнопка поруч з графіком помилок для відкриття локальної бази даних з статистикою помилок. Після її натискання відкриється окреме вікно з таблицею.

3.3.1.3 Вкладка детального моніторингу CPU

Після пам’яті розміщена вкладка присвячена центральному процесору. На ній відображається повне найменування кількість потоків та ядер. І графіки по ядерного навантаження. Спостереження за цією вкладкою може вказати на неочевидні проблеми з потоками процесора. Наприклад можна виявити що якась програма працює дуже не оптимізовано та використовує лише один потік для важких задач, що призводить до низької продуктивності самої програми та повне перевантаження одного з логічних потоків.

Перевантаження одного потоку процесора може бути небезпечним з точки зору нерівномірного теплового навантаження. Це не часто у сучасних комп’ютерах викликає проблему, бо на процесорі не просто так розміщуються тепло розсіювальну кришку, але у дуже не частих випадках постійне таке навантаження може привести до швидкої деградації чипу.

Що до реалізації цієї сторінки був трохи не стандартний підхід. Було використано UserControl у який була поміщена одна діаграма, це витрачає трохи більше пам’яті, проте дозволяє робити деякі більш важливі маніпуляції з графіками. Створений UserControl при ініціалізації отримує вказівку за яке саме ядро він відповідає. Та має декілька методів щоб самооптимізовуватись на дії користувача що до переведення фокусу з сторінки. При розміщені на сторінку графічні об’єкти кріпляться до об’єкту FlowLayoutPanel, який автоматично розміщує об’єкти, щоб вони не накладались один на одного та розміщувались як букви у текстовому полі, тобто по черзі. Залишається тільки підлаштовувати розміри діаграм з навантаженням до вікна і буде досягнута нормальна адаптивність. На рисунку 3.9 зображений вкладки з детальними даними ЦП.

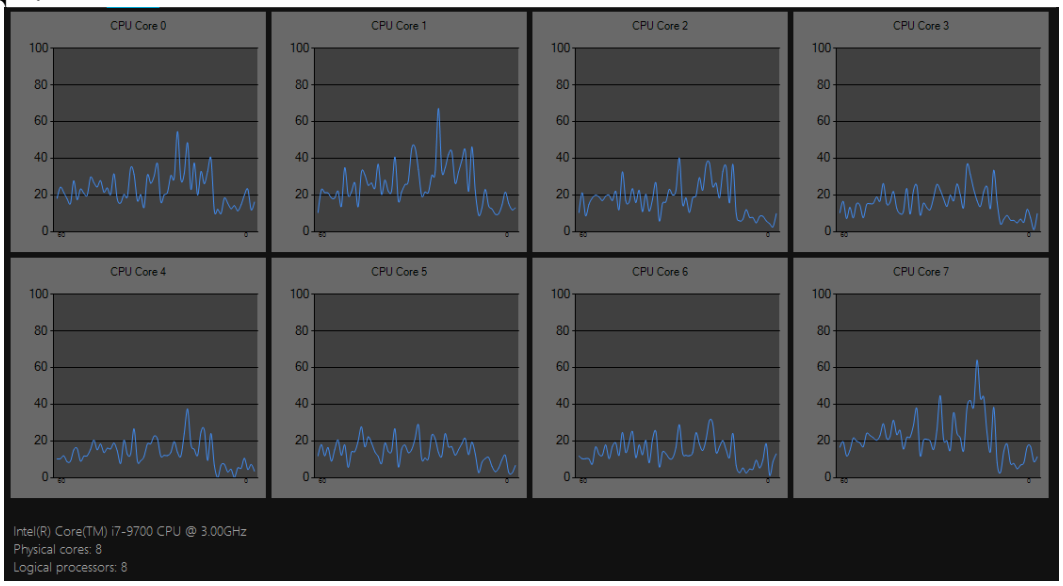

Рисунок 3.9 – Вкладка ЦП

3.3.1.4 Вкладка зібраної статистики

Вкладка статистики тут представлена у вигляді кругових діаграм по основним характеристикам метрикам ПК. Та дозволяє переключитись у режим перегляду на графіку середніх показників що хвилинної зібраної статистики.

Щоб не створити зайвого перевантаження диску при читанні статистики під це виділена окрема кнопка на сторінці статистики. Це зручно тим що перед зчитуванням користувач може встановити який рівень завантаженості він вважає низьким середнім та високим. Натиснувши на оновлення після встановлення параметрів діаграми оновлюються показуючи відповідне співвідношення.

При подвійному натисканні на кругову діаграму відображається більш детальний опис зібраних даних уже не у форматі кругової діаграми, а графіку. Інструменти для аналізу статистики тут не розміщені лише елементи для експорту та подальшого самостійного аналізу. Автоматизований аналіз статистики розміщений на іншій вкладці з назвою “Tools”. Вигляд вкладки статистика переставлений на рисунку 3.10.

Рисунок 3.10 – Вкладка статистика

3.3.1.5 Вкладка детального моніторингу дисків та мережі

Розділи дисків та мережевих інтерфейсів мають схожу структуру та принцип дії.

На кладці Disk відображаються фізичні диски встановлені у системі. Під час роботи програми вкладка виглядає як на рисунку 3.10

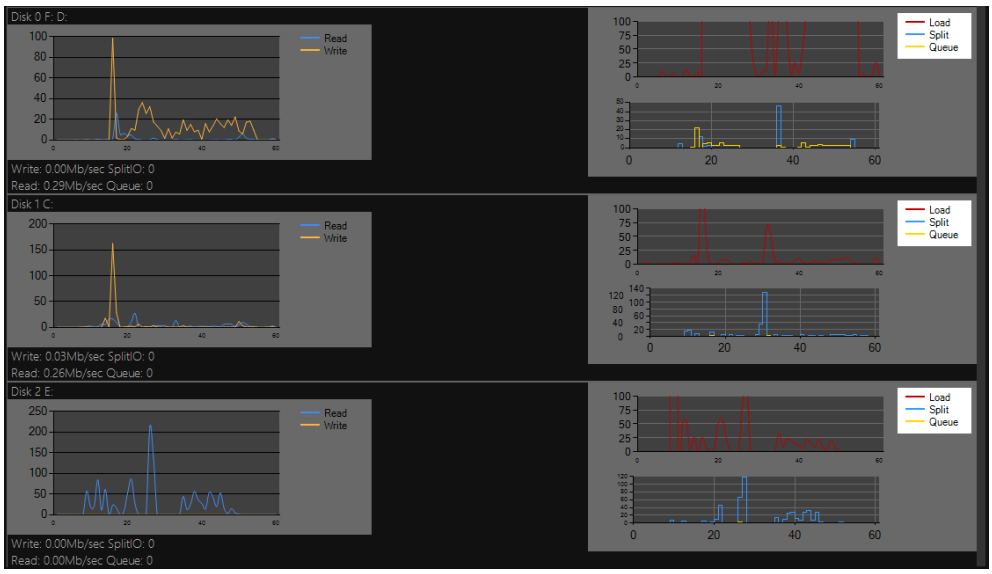

Рисунок 3.10 – Вміст вкладки Disk

Для реалізації знову використовується UserControl

Тож створюємо елемент з унікальним зовнішнім виглядом використовуючи стандартні елементи. Важливо одразу визначити розміщення, розміри та прив'язку елементів. У моєму випадку треба розмістити 3 графіки з навантаженнями перериваннями, чергою та швидкістю та декілька елементів *label* для виводу показників швидкості, номеру диску та логічних розділів які на ньому розміщені.

Оскільки моніторинг уже реалізований у головному класі програми все що робить форма це встановлює посилання на данні в головному класі форми та зберігає посилання на створені контроли, які створюються повністю на основі вже існуючих у головному класі масиви даних.

Елемент для кожного диску має особливу структуру та метод оновлення, бо має суміщавати декілька наборів даних на одному чарті. Тому потрібен особливий метод для вказування саме конкретного набору графіку, бо у випадку дисків на одному полі діаграми відображаються 2 графіки читання та запису, а також розриви операцій та чергу, результат можна побачити на рисунку 3.11.

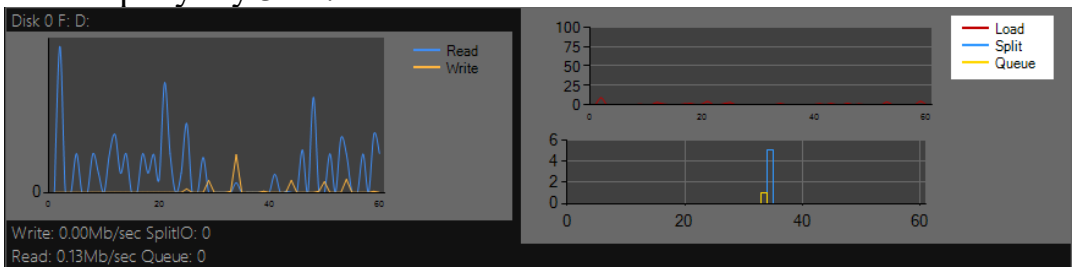

Рисунок 3.11 – Елемент для одного диску на вкладці

Вкладка мережі не може бути видалена бо фактично вона відображає все те що не показує графік на головній панелі, через динамічну кількість

інтерфейсів у комп'ютерах. Тут варто зазначити що це стосується не тільки фізичних інтерфейсів та Teredo Tunneling Pseudo-Interface, У ПК також можуть бути віртуальні інтерфейси моніторинг за ними теж є, бо фактично для системи не має різниці.

Адаптер Microsoft Teredo (Teredo Tunneling Pseudo-Interface) є компонентом, який використовується в операційній системі Windows та деяких інших ОС для організації мережевого протоколу. Він призначений для прибирання обмежень мережевих адрес IPv4 шляхом передачі даних між IPv6 та IPv4 адресами. Адаптер Teredo також підтримує роботу з технологією NAT, що спрощує передачу даних через мережу Інтернет.

У мережевих операційних системах, таких як macOS X, BSD і Linux, функціональність, подібна до Microsoft Teredo, надається через Miredo - аналогічний протокол зі зміненими деталями реалізації. Miredo також використовується для передачі даних по мережі, але працює з IPv6 адресами.

Microsoft ISATA також виконує функцію тунелювання між IPv6 та IPv4. Ці псевдо інтерфейси допомагають забезпечити сумісність і передачу даних між мережевими адресами різних поколінь.[12]

Драйвер Teredo лежить в директорії: C\Windows\System32\drivers\tunnel.sys

Якщо адаптер Microsoft Teredo відсутній треба переконатися чи не замінений він іншим протокол, чи не вимкнений він у цілях забезпечення додаткової безпеки.

Відсутність адаптера Microsoft Teredo необов'язково впливає на загальну роботу мережі, особливо якщо використовується інший протокол або технологія для перетворення мережевих адрес. Проте, в окремих випадках деякі додатки або послуги можуть вимагати присутності адаптера Microsoft Teredo для правильної роботи.

Важливо відзначити, що робота адаптерів Teredo та Microsoft ISATAP відбувається на нижньому рівні операційної системи, і їх деталі реалізації не повинні бути доступними або відомими для звичайних користувачів.

Моніторинг навантаження на мережу є важливим і корисним інструментом для аналізу роботи мережевої інфраструктури. Наприклад можна виявити проблему перевантаження, заторів, втрати пакетів або недостатню пропускну здатність. Це дозволяє оперативно реагувати на проблеми та швидко вирішувати їх, зменшуючи вплив на користувачів.

Елемент релізується по тим самим принципам і є візуалізатором даних з фонового моніторингу. Результат відображення продемонстровано на рисунку 3.12.

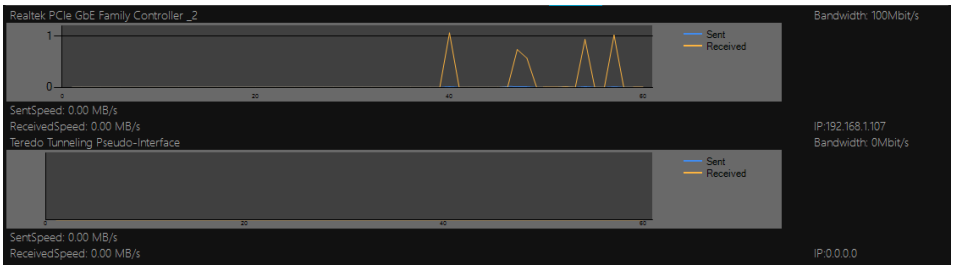

Рисунок 3.12 – Вкладка мережа

3.3.1.6 Вкладка інформації про пристрій

Вкладка INFO у додатку має на меті відображати всі ключові характеристики пристрою, а також системні дані, необхідні для аналізу, діагностики чи просто впізнання пристрою. Її функціонал забезпечує надання структурованої, доступної та актуальної інформації.

Основні елементи вкладки це випадаючий список та нижнє поле відображення. Обравши розділ у списку у нижньому полі відобразить інформація по розділу.

Таким чином можна отримати інформацію про такий компонент як CPU, але повну інформацію з повним вказування і родини і частоти, цей запит займає найбільше часу серед інших і видає дуже велику кількість параметрів а також час отримання всіх параметрів дорівнює часу отримання навантаження на ЦП. З цього випливає, що ManagementObjectSearcher читає всі параметри навіть не потрібні, щоб знайти конкретний і це підтверджує правильність його використання у контексті оновлення інформації про процеси.

Компонент пам’ять, це детальна інформація про фізичні банки пам’яті. Їх частоти і іноді в залежності від моделі материнської плати тут ще можна отримати інформацію про таймінги та напругу.

Дискова підсистема тут представлена двома різними розділами один з яких це фізичні диски а друга логічні диски. З логічних дисків можна дізнатись скільки вільного місця які розділи які саме за що відповідають. А у випадку фізичної можна дізнатись де саме розміщені розділи які характеристики має диск та куди він підключений у деяких випадках.

Наступний пункт це відео підсистема, звісно що інформації багато як і випадку процесора, але зазвичай найкорисніша інформація це модель та кількість відеопам’яті та версію драйверів.

Розділ батарея актуальний тільки для ноутбуків на Комп’ютерах цей розділ буде порожній зазвичай цей розділ корисний там що показує конкретну напругу, температуру та стан акумулятора якщо звісно ця інформація взагалі збирається апаратно.

Підрозділ Мережеві адаптери звісно містить абсолютно усі специфікації мережевих адаптерів системи як віртуальних так і фізичних. Параметрів дуже багато але корисних серед них не багато, фактично тільки MAC-адреса та IP.

Серед категорій заліза можна отримати список встановлених програм. Для отримання даних використовується простір імен Microsoft.Win32.RegistryKey, тобто продивляється реєстр ключів системи, використовуваний шлях ключів у цьому випадку "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall". Хоч і каталог називається Uninstall, тобто видалення, він фактично зберігає усі дані про встановлені програми.

Загалом ця сторінка дозволяє користувачам отримувати повну картину про пристрій в одному місці, з акцентом на зручність і швидкий доступ до ключової інформації.

3.3.1.7 Вкладка моніторингу окремого процесу

Вкладка для моніторингу окремого процесу надає користувачеві інструменти для глибокого аналізу конкретного процесу системи, що дозволяє відстежувати його параметри в реальному часі, таких як використання ресурсів, активні потоки та відкриті файли.

Переваги моніторингу окремого процесу у деталізації даних про конкретний процес, включаючи використання CPU, пам'яті, диска, мережевих ресурсів, що дає змогу точніше оцінити навантаження на систему. Можливість спостерігати за різними метриками процесу в реальному часі тут як раз головна перевага, тому час збереження даних на графіках збільшений до п’яти хвилин. Зручне відстеження аномальних піків навантаження чи підвищеного використання ресурсів. Подовження графіків моніторингу, це дозволяє аналізувати зміни в поведінці процесів за певний період, порівнювати дані в різні моменти часу та виявляти тенденції чи патерни. Вкладка для моніторингу окремого процесу є надзвичайно корисним інструментом для адміністраторів, розробників і користувачів, що дозволяє проводити детальну діагностику та моніторинг окремих процесів. Вона забезпечує швидкий доступ до критичних даних, допомагаючи знижувати час на усунення неполадок і підвищуючи загальну ефективність роботи системи. Єдина можлива проблема підвищене споживання пам’яті та процесорного часу у цьому режимі. Через обробку збільшеної кількості даних. Зовнішній вигляд кладки зображений на рисунку 3.10

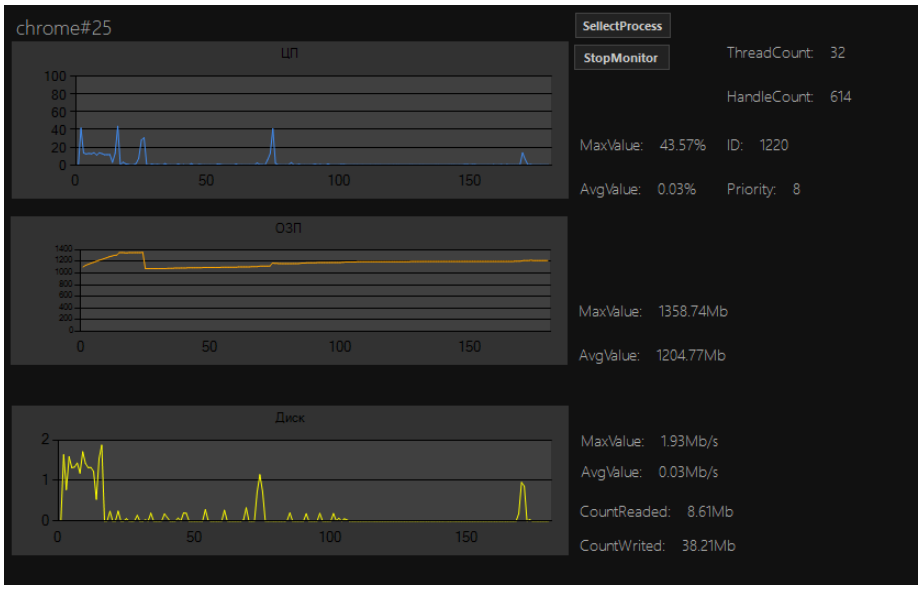

Рисунок 3.13 – Вкладка моніторингу конкретного процесу

3.3.1.8 Інструмент виявлення підозрілих процесів.

Інструмент виявлення підозрілих процесів представляє з себе вікно для перегляду знайдених підозрілих процесів. Усі процеси знайдені тут будуть

копіюватись у попередження та збережені навіть до наступного перезавантаження. Звісно окрім перегляду було додавання та видалення їх з білого списку. Для цього справа від таблиці є дві кнопки і власне таблиця яка показує білий список. Також з практичних міркувань було додано знизу кнопку для видалення підозрілих процесів, на випадок не впевненості користувача. Ця кнопка просто видалить запис з таблиці для того щоб не вносити у білий список процеси які були усунуті з системи. Зовнішній вигляд вікна на рисунку 3.11

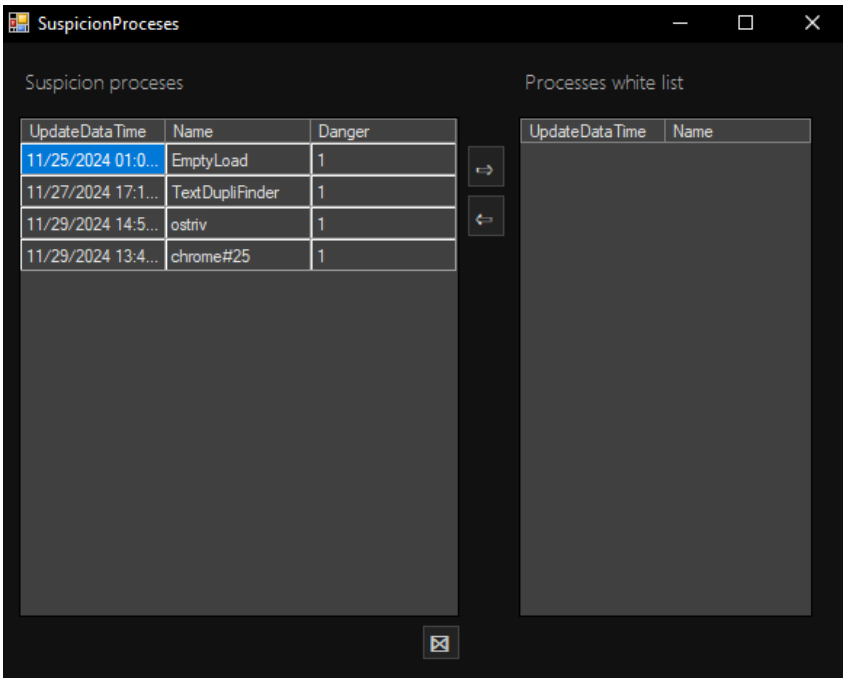

Рисунок 3.14 – Вікно перегляду підозрілих процесів.

Невеликий практичний тест показав, як працює алгоритм, що правда результат був дещо не точним що до визначення півня небезпеки. Оскільки програма була запущена вже після того як був запущені інші процеси і користувач не взаємодіяв з вікнами цих процесів, то у список потрапили не цільові для виявлення процеси, які могли б бути рівня небезпеки 0. Наприклад процес браузера chrome.

3.3.1.10 Віджети на панелі задач

Для покращення розуміння поточної ситуації в системі дуже часто доречно використовувати віджети. Повноцінний напівпрозорий віджет з інформацією про велику кількість показників Підходить не всім користувачам бо часто займає місце. Частіше за все на екрані залишається панель задач тому розмістити на неї наважливі показники, як на мене є дуже доречним рішенням.

Іконка з показниками на панелі задач іншими словами в System Tray є одним з найбільш ефективних рішень у плані економії простору на екрані. Багато системних моніторів і програм для моніторингу використовують подібний підхід для постійного відображення актуальних показників.

Аналоги такого рішення є у програмах Rainmeter, MSI Afterburner, Networx. Хоча Rainmeter в основному відомий як інструмент для створення та налаштування десктопних віджетів, він також підтримує виведення деяких параметрів на панель задач.

У MSI Afterburner хоч він і призначений в першу чергу для моніторингу продуктивності відеокарти також підтримує відображення різних параметрів, таких як температура, швидкість вентиляторів і споживана потужність, через невеликі іконки на панелі задач.

Networx це інструмент для моніторингу мережевого трафіку, який може відображати поточну швидкість передавання даних прямо в панелі задач.

Task Manager Windows не має можливості налаштування, але виведення іконки з завантаженням процесора на панель задач теж робить за замовчуванням. Хоча дуже часто це не той показник за яким хотілось би спостерігати.

Так реалізація виводу інформації ідеально поєднує простоту і ефективність, надаючи користувачам можливість швидко переглядати важливі показники без зайвого захаращення екрана.

3.3.1.11 Перегляд попереджень

Якщо є попередження про роботу системи то кнопка вгорі вікна змінює колір на жовтий або навіть червоний, залежно від серйозності ситуації. Це візуальне позначення допомагає швидко привернути увагу користувача до важливих подій.

Жовтий колір сигналізує про менш серйозні попередження, що вимагають уваги, але не є критичними для роботи системи. Це може бути, наприклад, попередження визначений підозрілий процес низького рівня небезпеки або логічний диск чи системний диск дуже близький до переповнення.

Червоний колір означає серйозні проблеми або критичні помилки, які потребують негайного повідомлення користувача, наприклад, поява битих секторів на диску або сильне підвищення температури.

По натисненню кнопки відкриється окреме вікно з детальним списком усіх активних попереджень. Користувач може переглянути список повідомлень, що з'явились, з детальним описом кожного попередження.

Якщо проблема, що спричинила попередження, була усунена то, система автоматично оновить статус попередження. Звісно окрім зміни кольору кнопки, система посилає повідомлення у панель сповіщень, якщо система Windows 10, але якщо система старіша, то сповіщення з’явиться з іконки агента на панелі. Цей спосіб сповіщень зазвичай підтримується на старіших системах.

Функціонал перегляду попереджень дозволяє користувачам оперативно реагувати на проблеми в системі, зберігаючи контроль над ситуацією. Зміна кольору кнопки дає чітке візуальне сповіщення, а вікно з

детальним переглядом попереджень дозволяє детальніше ознайомитись з проблемою та зробити якісь дії.

Висновок до розділу 3

У цьому розділі був детально розглянутий процес розробки та продемонстрована моніторингової системи, яка складається з двох основних компонентів: програми агента та програми серверу. Розробка здійснювалася із застосуванням сучасних інструментів і технологій, які забезпечують ефективність, гнучкість і масштабованість рішення.

На основі опису обраних інструментів розробки визначено, що використання відповідного програмного забезпечення та бібліотек дозволило створити стабільну та продуктивну систему, яка створює мінімум навантаження. Структурні рішення для програми агента та серверу були спрямовані на забезпечення швидкого збору даних, їх передачі й аналізу в реальному часі. При цьому структура бази даних була розроблена з урахуванням вимог до надійного зберігання великих обсягів інформації. Розроблені алгоритми швидкого аналізу даних для їх агрегації у базу даних

Особлива увага була приділена оптимізації критичних вузлів програми. Правильний підбір програмних інструментів. Застосування ефективних алгоритмів і оптимізація ресурсоємних операцій дозволили мінімізувати навантаження на систему та забезпечити її стабільну роботу навіть за значної кількості підключень і великого обсягу оброблюваних даних.

Методика використання системи була спроектована з урахуванням зручності користувача. Інтерфейс програми агента та серверу забезпечує інтуїтивно зрозуміле керування функціями системи та надає можливість візуалізувати дані моніторингу. Гнучкість налаштувань дозволяє адаптувати систему під різні сценарії використання, що робить її універсальною та зручною для широкого кола задач.

Таким чином, реалізоване рішення моніторингової системи є сучасним, надійним та функціональним інструментом, здатним ефективно виконувати поставлені завдання та відповідати вимогам користувачів.

ВИСНОВКИ

У ході виконання кваліфікаційної магістерської роботи було проведено дослідження методів та інструментів Windows щодо моніторингу використання ресурсів ПК та розробку прототипу системи моніторингу ПК у локальній мережі.

Метою роботи було розробка та вдосконалення методів моніторингу системних ресурсів ПК та створення програмного рішення для спрощення інвентаризації, виявлення проблем з апаратною частиною ПК, виявити вузькі місця, які можуть нашкодити продуктивності співробітників та допомоги з розподілом фінансових ресурсів. Оцінити ефективність запропонованих методів на практичних прикладах.

Досліджено існуючі аналоги моніторингових систем ресурсів комп'ютерів. проведено аналіз існуючих програмних рішень та вивчені можливості викладені у дослідженнях пов’язаних з моніторингом, для врахування всіх аспектів моніторингу навантаження на процесор, пам'ять, диск та інші компоненти комп'ютера, а також вплив активності користувача. Обґрунтовано необхідність розробки та оптимізації програми на основі удосконалення існуючих підходів до розробки таких систем.

Розроблено програму з використанням сучасних API системи Windows, робота якої удосконалена шляхом застосування методів оптимізації моніторингу. Значно знижено рівень навантаження на мережу методом агрегації моніторингових даних.

Було розроблено алгоритм для виявлення підозрілої активності процесів, який забезпечує ефективність навіть за умов обмеженого споживання системних ресурсів. Основна мета алгоритму полягає у моніторингу процесів та кількісній оцінці помилок пам’яті для виявлення аномальної роботи системи. Цей підхід дозволяє ідентифікувати незвичайну поведінку окремих компонентів та виявляти потенційні загрози, що можуть впливати на стабільність роботи системи.

Крім того, розроблено методологію визначення найбільш перевантаженого компонента у системі, що дозволяє точніше оцінювати стан апаратної частини. На основі зібраних температурних даних програма здатна прогнозувати дату необхідного технічного обслуговування, наприклад, очищення комп’ютера від пилу та інших елементів обслуговування, що значно покращує планування та знижує ризик несподіваних збоїв у роботі.

Новизна полягає у впровадженні оптимізованих алгоритмів для збору та обробки великих обсягів даних із мінімальним використанням ресурсів мережі та пристроїв на яких працює. Продуктивність алгоритмів агента зведена до стану коли додаток мінімально використовує ресурси у системи.

Отриманні дані тестування показали максимум 3,6% навантаження на процесорі з частотою 1.5 Ггц., при середньому показнику у 0,6%. На процесорі 4 Ггц яких зараз більшість, максимальний відсоток сягнув лише до 1.5% при середньому у 0,1%.

Розроблена система демонструє стабільну роботу при одночасному підключенні до 50 клієнтів без втрат даних. Обсяг даних, які обробляються і зберігаються в базі даних, оптимізований шляхом впровадження нових підходів до агрегації та структурування інформації.

У процесі було створено уніфікований протокол передачі даних між агентом і сервером, реалізовано оптимізовану структуру бази даних для обробки великих обсягів інформації, впроваджено зручний інтерфейс користувача для налаштування і моніторингу параметрів системи.

Таким чином, робота виконана в повному обсязі, відповідає поставленим завданням і має практичне та наукове значення.

Подальший розвиток системи передбачає поширення на моніторинг мережі. Оцінка стабільності підключення до мережі та моніторинг вузьких місць, що можуть впливати на продуктивність системи. Покращення алгоритмів для більш точних моделей для аналізу поведінки процесів, з урахуванням історичних даних та сучасних методів машинного навчання. Це дозволить визначати підозрілу активність навіть у складних сценаріях. Впровадження алгоритмів раннього попередження про можливі проблеми, які ґрунтуються на аналізі трендів та інших методів прогнозування. Додавання додаткових інструментів для аналізу статистики, що дозволить користувачам більш детально досліджувати навантаження на ресурси та поведінку процесів у різних часових інтервалах. Поліпшення інтерфейсу користувача для полегшення роботи із моніторинговою системою, покращення візуалізації та інтерактивних елементів. Інтеграція з існуючими системами моніторингу або IT-інфраструктурою для масштабного застосування.

Ці покращення перетворять моніторингову систему на універсальний інструмент для підтримки продуктивності ПК, забезпечення його стабільності та безпеки, а також раннього виявлення потенційних проблем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) IEEE Communications Surveys & Tutorials : Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX / Hofstede, Rick, Čeleda, Pavel, Trammell, Brian, Drago, Idilio, Sadre, Ramin, Sperotto, Anna, Pras, Aiko ; за ред. Aiko Pras, 2014. 2064 с.
2) Міхав В. В. Модель та методи збору і обробки даних для рекомендаційних систем у peer-to-peer комп'ютерних мережах : дис. ... канд. техн. наук: 05.13.06 / Черкаський державний технологічний університет. Черкаси : ЧДТУ, 2023. 179 с.
3) Operating System Availability of WMI Components https://learn.microsoft.com/ru-ru/windows/win32/wmisdk/operating-system-availability-of-wmi-components
4) List of Available Sensor Types - PRTG Network Monitor User Manual. URL: https://www.paessler.com/manuals/prtg/available_sensor_types (дата звернення: 14.11.2024).
5) IEEE A real-time equipment monitoring and fault detection system. / R.S. Guo; A. Chen; C.L. Tseng; I.K. Fong; A. Yang; C.L. Lee та ін. ; Hsinchu, Taiwan : IEEE, 2002. 312 с.
6) What Is Thrashing? Written by:baeldung Reviewed by:Michal Aibin URL: https://www.baeldung.com/cs/virtual-memory-thrashing (дата звернення: 16.11.2024).
7) Casey Muratori "Clean" Code, Horrible Performance URL: https://www.computerenhance.com/p/clean-code-horrible-performance (дата звернення: 8.11.2024)
8) Скалозуб В. В., Мурашов О. В. Моделювання даних процесів моніторингу при нерівномірних і нечітких інтервалах спостережень : стаття. – Харків, 2021.
9) International Research Journal of Engineering and Technology: Predictive Maintenance of Industrial Machines using Machine Learning, 2013. 524 с.
10) Priscilla Moraes Time-Series Analysis for Performance Monitoring and Anomaly Detection in Computer Networks, São Paulo: UD College of Engineering, 2013. 358 с.
11) Видмиш А. А. к.т.н., доцент Возняк О. М. к.т.н., доцент Купчук І. М. Дослідження медіанної фільтрації одновимірних сигналів, Вінницький національний аграрний університет: Вінницький : Вінницький національний аграрний університет 2020. 88 с.

12) SERIES Windows 10 App Development for Absolute Beginners. Microsof: веб-сайт. URL:https://learn.microsoft.com/en-us/shows/windows-10-development-for-absolute-beginners/ (дата звернення: 10.11.2024)
13) David Makofske, Michael J. Donahoo, and Kenneth L. Calvert TCP/IP Sockets in C#: Practical Guide for Programmers : Бостон, 2004. 192с.
14) Дяків Р. І. Метод моніторингу системних ресурсів операційних систем Тернопіль, 2021. 67 с.
15) Stephen Cleary. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming. O'Reilly Media. 2019. 254 с.
16) Lee Holmes. PowerShell Cookbook: Your Complete Guide to Scripting the Ubiquitous Object-Based Shell. O'Reilly Media. 2021. 1002 с.
17) Yule G. An Introduction to the Theory of Statistics. 1911. 236с.
18) Brett D. McLaughlin, Gary Pollice, Dave West. Head First Object-Oriented Analysis and Design. O'Reilly Media, 2006. 636 с.
19) Глибовець М. М. Розробка системи управління персоналом з застосуванням баз даних та знань Київ: НУ«КМА», 2020. 65 с.
20) «Розвиток освіти, науки та бізнесу: результати 2021» ISBN 978-617-95218-2-9. URL: http://www.wayscience.com/wp-content/uploads/2021/12/Materials-of-conference-6-7.12.2021-2.pdf (дата звернення: 16.11.2024)

Додаток А.

Код програми