Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи моніторингу та оптимізації функціонування операційних систем

(тема)

Виконав:
здобувач 2 року навчання,
групи СПм-23-5
Ілля ШЕВЧЕНКО
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо–професійна або освітньо–наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: доц. Віталій ТКАЧОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо–професійна або освітньо–наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Шевченку Іллі Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи моніторингу та оптимізації функціонування операційних систем

затверджена наказом по університету від "21" квітня 2025 р. № 296Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи системи моніторингу:вбудовані утіліти, Zabbix, Grafana, Prometheus; 2) компоненти системи моніторингу: модуль збора даних, модуль сповіщень, модуль взаємодії з користувачем, модуль реагування, модуль налаштування політик; 3) методи оптимізації основних метрик операційних систем: центрального процесора, оперативної пам'яті, дискового простору, мережних драйверів; 4) експериментальне тестування; 5) методи оцінки функціональності та ефективності

4. Перелік питань, що потрібно опрацювати у роботі
1)огляд вбудованих та сторонніх утиліт для моніторингу операційних систем;
2) вибір та обгрунтування методики та засобів дослідження;
3) програмна реалізація моделі вебзастосунку для моніторингу операційної системи;
4) проведення експериментальних досліджень;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 16 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
	Огляд систем моніторингу	22.04.25-29.04.25	
	корпоративних комп'ютерних мереж		
	Вибір та обгрунтування методики дослідження	30.04.25-05.05.25	
	Вибір інструментальних засобів	06.05.25-09.05.25	
	Розробка моделі системи моніторингу	10.05.25-21.05.25	
	Проведення експериментів	22.05.25-02.06.25	
	Оформлення матеріалів кваліфікаційної	03.06.25-07.06.25	
	роботи		
	Подання кваліфікаційної роботи керівникові та	08.06.25-10.06.25	
	її попередній захист		
	Подання кваліфікаційної роботи на рецензування	12.06.25-14.06.25	

Дата видачі завдання " 21 " квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Віталій ТКАЧОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 84 с., 12 рис., 8 табл., 4 дод., 44 джерела.

СИСТЕМА, РЕСУРСИ, МОНІТОРИНГ, МЕТОД, ОПТИМІЗАЦІЯ, UNIX, LINUX.

Метою кваліфікаційної роботи є підвищення ефективності функціонування операційних систем за рахунок впровадження методів моніторингу та оптимізації використання системних ресурсів. Основна увага приділяється розробці програмної системи, яка забезпечує збір діагностичних даних, аналіз продуктивності та реалізацію адаптивних рішень для покращення стабільності та швидкодії операційного оточення.

В ході виконання кваліфікаційної роботи досліджено сучасні підходи до моніторингу та оптимізації операційних систем, а також проаналізовано вплив різних методів управління ресурсами на загальну продуктивність та надійність систем. Було розроблено метод підвищення продуктивності функціонування Unix-подібних систем, який забезпечує стабільну та надійну роботу серверної інфраструктури корпоративної мережі в умовах високого навантаження та обмежених апаратних ресурсів.

Запропонований метод може бути застосований для підвищення стабільності, продуктивності та надійності роботи серверної інфраструктури корпоративних комп’ютерних мереж в умовах динамічно змінного навантаження та обмежених апаратних ресурсів.

ABSTRACT

Master`s thesis: 84 pages, 12 figures, 8 tables, 4 appendices, 44 sources.

SYSTEM, RESOURCES, MONITORING, METHOD, OPTIMIZATION, UNIX, LINUX.

The major goal of this thesis is to increase the efficiency of operation of operating systems by introducing methods of monitoring and optimizing the use of system resources. The focus is on the development of a software system that provides diagnostic data, productivity analysis and adaptive solutions to improve the stability and performance of the operating environment.

In order to analize the approaches of the modern monitoring and optimization of operating systems were investigated, as well as the impact of different methods of resource management on the overall productivity and reliability of systems. A method of increasing the functioning of UNIX-shaped systems has been developed, which provides stable and reliable operation of the server infrastructure of the corporate network in high load conditions and limited hardware resources.

The proposed method can be applied to increase the stability, productivity and reliability of server infrastructure of corporate computer networks in conditions of dynamically variable load and limited hardware resources.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОЗП – оперативна пам’ять

ОС – операційна система

ПЗ – програмне забезпечення

ЦП – центральний процесор

API – інтерфейс прикладного програмування (англ., Application Programming Interface)

CDN – мережа доставки контенту (англ., Content Delivery Network)

CPU – центральний процесор (англ., Central Processing Unit)

ETM – модуль розширеного телемоніторингу (англ., Enhanced Telemetry Module)

FCFS – першим прийшов – першим обслуговується (англ., First Come, First Served)

HDD – жорсткий диск (англ., Hard Disk Drive)

IDS – система виявлення вторгнень (англ., Intrusion Detection System)

IPC – взаємодія між процесами (англ., Inter-Process Communication)

NTFS – нова файлова система технологій (англ., New Technology File System)

QoS – якість обслуговування (англ., Quality of Service)

RAID – надлишковий масив незалежних дисків (англ., Redundant Array of Independent Disks)

RAM – оперативна пам’ять (англ., Random Access Memory)

ROS – операційна система реального часу (англ., Real-time Operating System)

RR – кругове планування (англ., Round Robin)

SIEM – система управління подіями та інформацією безпеки (англ., Security Information and Event Management)

SJF – найкоротше робоче завдання першим (англ., Shortest Job First)

SNMP – простий протокол керування мережею (англ., Simple Network Management Protocol)

SSD – твердотільний накопичувач (англ., Solid–State Drive)

TCP/IP – набір мережних протоколів передачі даних (англ., Transmission Control Protocol / Internet Protocol)

WMI – інструментарій керування Windows (англ., Windows Management Instrumentation)

ВСТУП

В умовах стрімкого розвитку цифрових технологій операційні системи (ОС) становлять ключовий компонент інформаційної інфраструктури підприємств, установ та організацій. Вони забезпечують узгоджену роботу програмного та апаратного забезпечення в будь-яких сферах діяльності людини, від державного управління та бізнесу до критичної та мережної інфраструктури. Зростання вимог до продуктивності, надійності та інформаційної безпеки супроводжується ускладненням архітектури систем, підвищеним навантаженням на ресурси, збільшенням кількості фонових процесів і зростанням ризиків втрати даних.

В умовах інтенсивної експлуатації актуальним постає завдання моніторингу та оптимізації використання ресурсів операційної системи, зокрема процесора, оперативної пам’яті, мережних інтерфейсів і підсистеми зберігання даних. Накопичення тимчасових та застарілих файлів та недостатній контроль за ресурсами можуть спричиняти зниження продуктивності, збої в роботі програмного забезпечення та втрату інформації.

Оптимізаційні підходи до ефективного функціонування ОС включають як апаратні, так і програмні засоби. зокрема застосування SSD-накопичувачів, балансування навантаження, керування пріоритетами процесів і фоновими службами. Інтеграція таких підходів в єдину систему адміністрування ОС підвищує ефективність IT-інфраструктури.

Метою кваліфікаційної роботи є аналіз, розробка та впровадження методів моніторингу та оптимізації ключових компонентів операційної системи для забезпечення її продуктивності, надійності та стійкості в умовах підвищеного навантаження та обмежених апаратних ресурсів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні метрики моніторингу операційних систем

В сучасних інформаційних системах операційна система (ОС) виступає центральним елементом, що забезпечує взаємодію між апаратним забезпеченням та прикладними програмами. Від ефективності її роботи залежить стабільність, продуктивність та безперервність функціонування всієї обчислювальної інфраструктури. Зростання обсягів даних, підвищені вимоги до швидкості обробки інформації, багатозадачність, віртуалізація та інтенсивне використання мережних сервісів створюють суттєві навантаження на системні ресурси. В таких умовах зростає важливість засобів моніторингу та оптимізації функціонування ОС [1, 2].

Моніторинг операційної системи – процес постійного спостереження за її станом та продуктивністю, який дозволяє своєчасно виявляти можливі проблеми, аналізувати поведінку системи в різних умовах і приймати обґрунтовані рішення для підвищення ефективності її роботи.

Серед ключових параметрів моніторингу стану ОС рівень використання центрального процесора (ЦП), оперативної пам’яті, дискового простору та мережного трафіку. Також важливими є час відгуку системи та аналіз журналів подій, які допомагають виявляти помилки та несправності.

До базових показників належать характеристики центрального процесора, оперативної пам’яті, дискової підсистеми та мережного інтерфейсу. Для процесора важливими є завантаження, яке відображає частку часу, протягом якого він виконує обробку завдань, а також тактова частота, що характеризує швидкість його роботи та безпосередньо впливає на продуктивність системи.

Параметр використання оперативної пам’яті включає обсяги використовуваної та вільної пам’яті, що дозволяє оцінити ризики

перевантаження, які можуть призводити до деградації швидкодії системи. Моніторинг дискової підсистеми включає вимірювання ступеня наповненості дискового простору та часу відгуку, що відображає затримку при отриманні даних і є критичним показником для вводу/виводу. Продуктивність мережного трафіку оцінюється через пропускну здатність та мережну затримку. Ці параметри визначають ефективність мережної взаємодії та впливають на загальну швидкодію системи. В таблиці 1.1 наведені основні метрики продуктивності операційної системи.

Таблиця 1.1 – Основні метрики продуктивності операційної системи

Метрика	Опис	Одиниця вимірювання
Завантаження процесора	Відсоток часу, коли процесор активно виконує інструкції	%
Частота процесора	Швидкість виконання інструкцій залежно від моделі та навантаження	ГГц
Використання пам’яті	Обсяг оперативної пам’яті, що використовується системою	МБ/ГБ
Вільна пам’ять	Обсяг доступної пам’яті для нових процесів	МБ/ГБ
Використання диска	Частка дискового простору, зайнятого файлами	%
Час доступу до диска	Середній час отримання даних з дискової підсистеми	мс
Пропускна здатність мережі	Обсяг даних, що передається через мережу за одиницю часу	Мбіт/с
Затримка мережі	Час передачі пакета між пристроями в мережі	мс

На рисунку 1.1 наведено схему аналізу метрик продуктивності операційної системи, який виконується за допомогою моніторингу стану ОС.

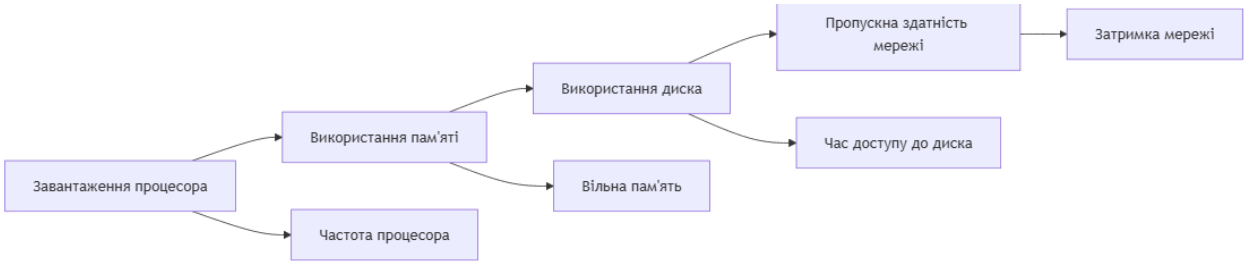


Рисунок 1.1 – Схема аналізу метрик продуктивності ОС

Отже, аналіз ключових показників стану операційної системи, зокрема параметрів центрального процесора, оперативної пам’яті, дискової підсистеми та мережного інтерфейсу, становить одне з основних завдань моніторингу її функціонування.

1.2 Компоненти системи моніторингу операційної системи

Система моніторингу операційної системи є сукупністю апаратних та програмних засобів, призначених для збору, обробки, візуалізації та аналізу інформації про стан обчислювального середовища в режимі реального часу або з певною періодичністю. Основна мета такої системи полягає у забезпеченні безперервного контролю за роботою ОС, своєчасному виявленні аномалій, прогнозуванні можливих збоїв та оптимізації використання системних ресурсів. До основних функціональних компонентів, які формують архітектуру системи моніторингу, належать:

- модуль збору даних, який відповідає за безперервний збір метрик з операційної системи. Для різних ОС доступні різні інтерфейси доступу до цих даних, зокрема API, системні файли, а саме /proc, sysctl, wmic, командні утиліти, а саме top, vmstat, iostat, Get–Process, або інтерфейси SNMP, WMI, NetFlow тощо. Модуль збору даних повинен підтримувати налаштування

інтервалів опитування, фільтрацію визначених параметрів для подальшого аналізу;

- модуль аналізу та виявлення аномалій, який виконує обробку зібраних даних з метою виявлення нестандартних або критичних ситуацій. Основні функції якого полягають в аналізі трендів, порівнянні з допустимими межами, побудові профілів «нормальної поведінки» системи, виявленні стрибків навантаження, які не характерні для поточного періоду часу. Залежно від складності реалізації, цей модуль може використовувати як прості порогові правила, так і елементи машинного навчання, зокрема кластеризацію, ізоляційні дерева, регресійний аналіз, для виявлення аномалій у патернах споживання ресурсів [3];

- модуль реагування виконує автоматичне застосування дій на основі висновків модуля аналізу. Реакції можуть бути як жорсткими, зокрема вимкнення процесів, перезапуск служб, так і м’якими, а саме вивільнення кешу, повідомлення адміністратору, зниження пріоритетів завдань;

- модуль взаємодії з користувачем попри високий рівень автоматизації, є важливою складовою системи моніторингу та надає користувачеві або адміністратору вичерпної інформації про стан системи, прийняті рішення та поточні рекомендації;

- модуль налаштувань та політик відповідає за збереження конфігурацій системи, зокрема порогові значення для параметрів, правила реагування, частоту оновлення даних, обмеження щодо автоматичних втручань.

Для моніторингу застосовують як вбудовані інструменти ОС, зокрема «Диспетчер завдань» у Windows, «Монітор системи» в macOS або утиліти top та htop в Linux, так і спеціалізовані сторонні програми, зокрема Zabbix, Nagios, Prometheus. Вони надають розширені можливості збору та аналізу даних, що дозволяє оперативно реагувати на зміни в роботі системи [4].

Незважаючи на різноманітність, сучасні системи моніторингу часто мають критичні обмеження. Більшість з них або надають лише статистичні

дані без аналітичної обробки, або не підтримують автоматичне втручання в роботу системи. Особливо це актуально для критичних систем, зокрема серверних інфраструктур, медичних інформаційних систем та автоматизованих систем управління виробництвом.

Крім того, стрімке поширення віддаленої роботи збільшує навантаження на кінцеві пристрої. В умовах обмежених можливостей масштабування обладнання ефективне управління наявними ресурсами є головним чинником до забезпечення прийнятної продуктивності [5].

Моніторинг включає постійний контроль стану ОС та ефективності використання ресурсів, що допомагає вчасно виявляти перевантаження, помилки та потенційні загрози. Основними напрямами моніторингу є:

- контроль продуктивності системи, який включає збір та аналіз показників завантаженості ЦП, обсягу використаної оперативної пам’яті, дискового простору, мережного трафіку, часу відгуку. Вбудовані засоби надають базову інформацію, тоді як спеціалізовані програми забезпечують розширений аналіз;

- моніторинг мережі, який забезпечує визначення доступності вебресурсів, оцінку швидкості з'єднання, виявлення затримок та втрат пакетів, а також контроль безпеки мережного середовища. Зокрема утіліти ping, traceroute та програмне забезпечення Wireshark дозволяють глибоко аналізувати трафік [6];

- моніторинг програмного забезпечення, який полягає в аналізі часу відгуку додатків, використання ресурсів та частоти помилок. Для глибокого аналізу застосовують системи збору логів, зокрема ELK Stack;

- контроль безпеки, який полягає у виявленні потенційних загроз і попередженні несанкціонованого доступу. Це дозволяє здійснити виявлення шкідливого ПЗ, аналіз вразливостей, контроль підозрілої активності. Для цього застосовуються антивірусні програми, системи виявлення вторгнень (IDS) та комплекси управління інформаційною безпекою (SIEM) [7].

Ефективний моніторинг та оптимізація операційної системи неможливі без чіткої структуризації її основних компонентів та засобів, які беруть участь в процесі контролю за ресурсами та автоматизованого втручання в роботу системи.

Методи моніторингу різняться залежно від цілей. Активний метод передбачає надсилання запитів до об'єкта моніторингу для отримання зворотного зв'язку. Пасивний базується на зборі даних без втручання в процеси моніторингу. Локальний моніторинг здійснюється на рівні окремого пристрою, а віддалений моніторинг здійснюється через мережу. Виважений вибір методів та інструментів моніторингу є ключовим для забезпечення стабільної та безпечної роботи операційної системи, а регулярний контроль дозволяє своєчасно реагувати на проблеми та оптимізувати роботу системи, що забезпечує її ефективність та стабільність.

1.3 Фактори впливу на продуктивність операційної системи

Типова операційна система містить сотні процесів, служб і драйверів, які динамічно взаємодіють між собою та з користувачем. Сучасні ОС оперують значною кількістю паралельних процесів, які конкурують за обмежені ресурси, зокрема центральний процесор, оперативна пам'ять, дискова підсистема та мережні канали. За відсутності ефективного моніторингу спостерігається ситуація, коли ресурси розподіляються неефективно: деякі процеси можуть споживати надмірний обсяг пам'яті чи обчислювальних потужностей, не виконуючи критично важливих функцій. Це призводить до зниження загальної продуктивності системи, збільшення часу відгуку програм і навіть до виникнення збоїв у її роботі [8].

Дослідження в галузі продуктивності комп'ютерних мереж переважно фокусуються на розв'язанні ключових завдань, пов'язаних із забезпеченням високої ефективності, стабільності, безпеки та масштабованості систем в умовах інтенсивного навантаження на обчислювальні ресурси. Широке

застосування технологій віртуалізації та контейнеризації, а також потреба в ефективному розгортанні ізольованих середовищ зумовлюють необхідність постійного спостереження за станом систем і динамічного балансування навантаження для раціонального використання доступних ресурсів [9].

Однією з найпоширеніших загроз для стабільної роботи ОС є перевантаження системних ресурсів, що може призводити до різкого зниження продуктивності або навіть до аварійного завершення роботи. Типові причини включають надмірне споживання процесора, оперативної пам’яті, дискової підсистеми та мережних інтерфейсів, що часто є наслідком неконтрольованої активності процесів або неправильної конфігурації системи. Типові фактори перевантаження системних ресурсів наведені в таблиці 1.2. Основними джерелами перевантаження операційної системи є надмірне завантаження ключових компонентів, зумовлене як апаратними, так і програмними чинниками. Високе завантаження процесора часто виникає через одночасне виконання великої кількості процесів, запуск ресурсоємних додатків або активність фонових сервісів, які споживають обчислювальні ресурси без безпосередньої користі для кінцевого користувача. Дефіцит оперативної пам’яті може бути спричинений надмірною кількістю запущених програм а також витоками пам’яті, коли додатки не звільняють зайняті ресурси після завершення роботи, поступово виснажуючи системні резерви. Ще одним критичним чинником є навантаження на дискову підсистему, що зумовлюється інтенсивними операціями читання або запису, наявністю фрагментованих файлів чи використанням повільних носіїв. Це може призвести до переповнення черг вводу/виводу, що суттєво знижує швидкодію підсистем зберігання даних і призводить до затримок у роботі додатків. Схожі ефекти спостерігаються також в умовах перевантаження мережі, коли високий обсяг вхідного або вихідного трафіку спричиняє затримки обробки пакетів, розриви з’єднань або нестабільність в роботі мережних сервісів.

Окрему загрозу для стабільності системи становить неефективне планування задач, коли операційна система неправильно розподіляє ресурси між процесами, внаслідок чого виникає їх блокування, конкуренція за ресурси або загальне зниження продуктивності. Всі ці фактори, особливо при їх одночасному прояві, можуть призвести до критичних станів системи, порушення її стабільної роботи та необхідності втручання адміністратора.

Таблиця 1.2 – Типові проблеми перевантаження системних ресурсів

Проблема	Причина	Наслідки
Високе процесорне навантаження	Велика кількість активних процесів	Гальмування роботи системи, підвищене енергоспоживання
Дефіцит оперативної пам’яті	Витоки пам’яті, надмірне споживання ресурсів	Затримки виконання програм, зниження продуктивності
Навантаження на дискову підсистему	Інтенсивне читання/запис	Збільшення часу доступу до файлів, знос диску
Перевантаження мережі	Високий рівень трафіку	Втрата пакетів, затримки, розриви з'єднань
Переповнення черги вводу/виводу	Велике навантаження на диск/мережу	Зниження загальної швидкодії системи
Витоки пам’яті	Некоректна робота програм	Поступове уповільнення, аварійні збої
Неправильне планування задач	Неоптимальний розподіл ресурсів	Довгі затримки у виконанні процесів

В роботі [10] звертається увага на зростання енергоспоживання через надмірне навантаження центрів обробки даних, особливо в хмарних обчислювальних середовищах. Зокрема, зазначено збільшення споживання електроенергії з 200 ТВт•год у 2016 році до очікуваних 2967 ТВт•год до 2030 року. Для підвищення енергоефективності та зменшення викидів CO_2 рекомендується впровадження засобів віртуалізації та автоматизації управління ресурсами на всіх рівнях – від прикладного ПЗ до інфраструктури віртуалізації та операційної системи.

Додатки, орієнтовані на роботу з великими обсягами даних, потребують значного обчислювального потенціалу, а також відповідної інфраструктури для зберігання, обробки та аналітики інформації у розподілених середовищах. Як показано в дослідженні [11], використання контейнеризації, ефективних механізмів планування, інструментів керування, автоматизованого масштабування і розгортання дозволяє значно зменшити навантаження на сервери, забезпечуючи при цьому збалансоване використання ресурсів, що є критично важливим для побудови гнучких масштабованих серверних архітектур.

Водночас, хоча зазначені технічні рішення мають значний вплив на продуктивність серверного середовища, серйозним бар'єром часто залишаються внутрішні недоліки самої операційної системи. До таких належать накопичення тимчасових або непотрібних файлів, неефективне керування фоновими процесами, надмірне використання ресурсів окремими службами та відсутність оптимального механізму розподілу ресурсів між системними компонентами.

В роботі [12] обґрунтовано, що впровадження ієрархічних моделей управління ресурсами, орієнтованих на багаторівневу оптимізацію, дозволяє підвищити продуктивність ОС в серверному середовищі. Зокрема, пропонується використання планувальника регулярного технічного обслуговування (ROS) та модуля раннього виявлення збоїв (ETM). Перший відповідає за автоматичне виконання завдань з очищення, діагностики та профілактики, запобігаючи накопиченню помилок, а другий за безперервний моніторинг і виявлення відхилень в роботі системи на ранніх етапах. Їх комплексне застосування дозволяє досягти вищої стабільності, зменшити енергоспоживання та покращити загальну продуктивність ОС, що особливо важливо в умовах високих навантажень.

Як додатково зазначено в роботі [13], перспективним напрямом є впровадження інтелектуальних механізмів самоналаштування та автономного управління, що використовують методи штучного інтелекту та машинного

навчання. Наявні рішення здатні реагувати на збої та виконувати базову конфігурацію, проте їхня автономність є обмеженою через відсутність гнучкого моніторингу в реальному часі та недостатню реалізацію механізмів зворотного зв’язку, що ускладнює створення повністю адаптивних систем керування.

Автори робіт [14, 15] також підкреслюють важливість впровадження системного моніторингу стану ОС. Зокрема, демонструється ефективність використання вбудованих утиліт в UNIX-подібних системах для контролю за споживанням ресурсів та загальним станом системи. Такий підхід дозволяє підвищити прозорість функціонування без необхідності встановлення стороннього програмного забезпечення, що є особливо актуальним у середовищах з обмеженими ресурсами. Отже, стандартні інструменти моніторингу становлять важливий компонент у створенні ефективної системи адміністрування серверних ОС.

Наявні інструменти моніторингу, що входять до стандартного функціоналу ОС, зокрема top, htop, Task Manager, здебільшого орієнтовані на ручну діагностику. Їх функціональність обмежується виявленням вже існуючих проблем без можливості їх прогнозування або адаптивного реагування на зміни. Як вказується в дослідженні [16], ці засоби не здатні забезпечити повноцінний контроль над системою в умовах динамічних навантажень та високої складності сучасних обчислювальних середовищ.

Ще однією проблемою є відсутність вбудованих механізмів самостійної оптимізації. У випадку виявлення перевантаження системи, рішення щодо подальших дій приймається адміністратором. Це потребує глибокої технічної експертизи, оперативності та значного часу на аналіз і прийняття рішень. В критичних системах з високими вимогами до доступності така залежність від людського втручання є неприйнятною [17].

Крім того, ручний підхід в моніторингу та оптимізації істотно підвищує ризик людських помилок. Постійна потреба у фаховому супроводі також знижує загальну ефективність управління IT-інфраструктурою.

Особливо гостро ці проблеми постають в масштабованих архітектурах, зокрема в хмарних обчисленнях, мікросервісах та контейнеризованих середовищах [18]. В таких системах ресурси розподіляються динамічно, а компоненти можуть швидко змінюватися, що ускладнює застосування традиційних інструментів моніторингу, які не враховують виконання процесів, що призводить до хибних позитивних або негативних сигналів. Наприклад, інтенсивне використання ресурсів в певні періоди може бути цілком очікуваним, однак сприймається як відхилення [19].

Комплексна система моніторингу та оптимізації функціонування ОС повинна включати чітко структуровані та взаємопов’язані компоненти, кожен з яких виконує свою функцію в процесі аналізу, прийняття рішень та втручання в роботу системи. Забезпечення взаємодії між цими компонентами, гнучкість налаштувань і адаптивність до нових умов експлуатації є запорукою створення ефективного методу підтримки високої продуктивності та стабільності ОС [20].

Аналіз сучасного стану управління операційними системами демонструє суттєвий дефіцит інструментів, здатних до самостійного аналізу стану, прогнозування потенційних відмов та автоматичної корекції параметрів функціонування. Це зумовлює потребу в розробці та впровадженні інтелектуальних методів моніторингу та оптимізації, що забезпечують адаптивне, контекстно-залежне та проактивне керування ресурсами ОС. Такий підхід дозволить підвищити стабільність, надійність та продуктивність системи в умовах постійно зростаючих вимог до ефективності IT-інфраструктури.

1.4 Постановка задачі

Стрімкий розвиток інформаційних технологій та постійне зростання обсягів обчислювальних навантажень вимагає ефективного функціонування операційних систем застосовуваного обладнання. Зростання складності

інформаційних середовищ вимагає від ОС високої продуктивності, надійності, адаптивності до змін параметрів навантаження в режимі реального часу.

Попри наявність значного арсеналу інструментів моніторингу, більшість з них обмежуються лише реєстрацією поточного стану системи. Це призводить до неефективного використання обчислювальних ресурсів та недостатньої адаптивності до динаміки робочих процесів.

Сучасні методи оптимізації роботи ОС включають як апаратні засоби, так і програмні рішення, інтеграція яких в комплексну систему адміністрування сприяє суттєвому підвищенню ефективності функціонування IT-інфраструктури.

На основі вищезазаначеного, метою кваліфікаційної роботи є аналіз, розробка та реалізація методів моніторингу та оптимізації ключових компонентів операційної системи з метою забезпечення її стабільної, надійної та продуктивної роботи в умовах високого навантаження та обмежених обчислювальних ресурсів.

2 МЕТОДИ ТА ЗАСОБИ ОПТИМІЗАЦІЇ ФУНКЦІОНУВАННЯ ОПЕРАЦІЙНИХ СИСТЕМ

Ключовим етапом забезпечення стабільного та ефективного функціонування комп’ютерних систем є аналіз продуктивності операційної системи. Продуктивність ОС визначається сукупністю взаємопов’язаних факторів, кожен з яких відіграє важливу роль у загальній ефективності обчислювального середовища.

Насамперед, фундаментальне значення має апаратне забезпечення, що формує технічну основу для обробки даних. Центральний процесор (CPU) виконує обчислення та керує процесами, оперативна пам’ять (RAM) забезпечує тимчасове зберігання даних для активних завдань, а накопичувачі типу HDD або SSD визначають швидкість доступу до інформації. Якість та конфігурація цих компонентів безпосередньо впливають на багатозадачність і швидкодію системи. Наприклад, недостатній обсяг RAM або використання повільного жорсткого диска може значно знижувати продуктивність, що особливо критично для систем реального часу або ресурсомістких застосунків.

Конфігурація самої операційної системи також є важливим чинником. До неї належать вибір файлової системи, механізми керування процесами, кешування даних, розподіл ресурсів між задачами. Невдале налаштування параметрів ОС, зокрема фонових служб або системних буферів, здатне призводити до перевитрати ресурсів, зростання навантаження на CPU та пам’ять, що призводить до зниження загальної ефективності функціонування.

Не менш важливе значення має програмне забезпечення, яке встановлене та використовується в системі. Кількість активних процесів, ресурсоємні застосунки, велика кількість одночасно відкритих вкладок у веббраузерах спричиняють зростання навантаження на обчислювальні

ресурси, зменшуючи доступну продуктивність для інших задач. У випадку мобільних пристроїв додатковим обмеженням є енергоспоживання, що полягає в недосконалій реалізації фонових процесів чи надмірній активності додатків, що призводить до швидкої розрядки акумулятора.

Наявність шкідливого програмного забезпечення, включно з вірусами та іншими видами атак, часто залишається непомітною для користувача, але створює приховане навантаження на систему. Зловмисне ПЗ здатне запускати додаткові фонові процеси, активно використовувати ресурси центрального процесора та оперативної пам’яті, що суттєво впливає на продуктивність та стабільність роботи ОС.

Сукупність вищезгаданих чинників потребує постійного аналізу з метою виявлення та усунення вузьких місць. У цьому контексті особливого значення набувають як вбудовані, так і сторонні інструменти моніторингу, які забезпечують збір, обробку та візуалізацію даних про стан системних компонентів. Ефективне використання таких засобів дозволяє своєчасно виявляти відхилення, оптимізувати роботу ресурсів та підвищити загальну ефективність операційної системи.

2.1 Методи оптимізації функціонування ОС

Оптимізація операційної системи передбачає впровадження змін, спрямованих на підвищення її продуктивності, стабільності та безпеки. Вона базується на аналізі даних моніторингу та має на меті усунути виявлені проблеми та вузькі місця у роботі системи [21].

Процес – це виконувана програма, яка має власний адресний простір та ресурси. Взаємодія між процесами здійснюється через механізми міжпроцесної комунікації (IPC), зокрема черги повідомлень, роздільна пам’ять або сокети. В межах одного процесу можуть виконуватися потоки, тобто легковагові одиниці виконання, які поділяють спільний адресний

простір, що дозволяє ефективніше організовувати паралельне виконання та обмін даними.

Управління процесами в ОС охоплює кілька ключових аспектів. Створення та завершення процесів здійснюється автоматично при запуску та завершенні програм, а також у випадках аварійного припинення їх виконання. Планування процесів визначає порядок їх виконання згідно з певними алгоритмами, серед яких найпоширенішими є First-Come-First-Served (FCFS), Shortest Job First (SJF), Round Robin (RR) та пріоритетне планування. Для синхронізації потоків в межах одного або кількох процесів застосовуються механізми взаємного виключення, такі як м'ютекси, семафори та умовні змінні, що забезпечують безпечний доступ до спільних ресурсів.

До основних методів забезпечення продуктивності ОС належать:

- оптимальне використання оперативної пам'яті (ОЗП). Недостатній обсяг ОЗП призводить до уповільнення роботи, що особливо помітно під час одночасного запуску кількох ресурсомістких програм, зокрема відеоредактори або сучасні ігри. Регулярне оновлення програмного забезпечення та драйверів сприяє зменшенню навантаження, оскільки нові версії оптимізовані для кращого використання ресурсів;

- застосування антивірусного програмного забезпечення (ПЗ). Операційна система може бути вразливою до шкідливого програмного забезпечення, яке використовує ресурси комп'ютера без відома користувача. Тому важливо регулярно сканувати систему за допомогою сучасного антивірусного ПЗ;

- перехід на використання твердотільного накопичувача (SSD) замість традиційного жорсткого диска дозволяє значно пришвидшити доступ до даних завдяки вищій швидкості обміну між пам'яттю та сховищем [22];

- збільшення обсягу оперативної пам'яті. В разі регулярної роботи з великою кількістю програм доцільно збільшити обсяг оперативної пам'яті,

встановивши додаткові модулі. Це дозволяє уникнути ситуацій нестачі ресурсів та забезпечити плавну роботу системи.

- використані програми для очищення пам’яті, що вивільняють ресурси від тимчасових або зайвих даних;
- налаштування параметрів віртуальної пам’яті, що дозволяє гнучкіше розподіляти ресурси системи.

Окрім цього, важливим є також управління файловою системою та дисковими операціями. Ефективне зберігання й обробка даних залежать від вибору відповідної файлової системи, зокрема NTFS для Windows або ext4 для Linux [23]. Для традиційних жорстких дисків рекомендована періодична дефрагментація, яка забезпечує компактне розміщення фрагментів файлів, скорочуючи час доступу до них. Натомість для SSD ця операція є шкідливою та не рекомендується через зношування комірок пам’яті.

Налаштування кешування також впливає на ефективність системи, оскільки швидкий доступ до часто використовуваних даних дозволяє знизити затримки. Окрім того, звільнення місця на диску шляхом видалення тимчасових файлів, кешів і дублікатів допомагає покращити загальну стабільність роботи. Додатково доцільно перевіряти диски на наявність помилок і своєчасно оновлювати драйвери для дискових контролерів, що позитивно впливає на стабільність зберігання даних.

Для підвищення надійності систем зберігання використовуються RAID-масиви, які дозволяють поєднувати кілька фізичних дисків в логічний том. Це забезпечує як підвищення швидкості доступу до даних, так і збереження інформації в разі відмови одного з дисків.

Ще одним напрямом оптимізації є мережні налаштування. Вони передбачають підвищення швидкості, надійності та безпеки з'єднання. Для цього проводять оновлення застарілого мережного обладнання, зокрема маршрутизаторів, комутаторів, мережних карт, на моделі, що підтримують сучасні стандарти. Важливо водночас використовувати якісні кабелі та з’єднання для зменшення втрат сигналу. Маршрутизатори потрібно

налаштовувати з урахуванням розподілу трафіку. Прошивки пристроїв варто регулярно оновлювати, а функцію QoS використовувати для пріоритетного обслуговування критичного трафіку. Також необхідно забезпечити належний рівень захисту Wi–Fi з використанням сучасних протоколів шифрування, щоб гарантувати безпеку з'єднання [24]. Налаштування параметрів TCP/IP на комп'ютері користувача та оновлення драйверів мережної карти дозволяє забезпечити стабільну та ефективну роботу в мережі. Оптимізація програмного забезпечення включає використання сучасних браузерів з увімкненим кешуванням, що скорочує час завантаження вебсторінок. Програми, що оптимізують мережний трафік, також допомагають зменшити його витрати.

Для прискорення завантаження контенту доцільно використовувати CDN – мережу серверів, розміщених по всьому світу, що дозволяє користувачам отримувати дані з найближчих географічно точок. Водночас систематичний моніторинг мережного трафіку за допомогою спеціалізованих інструментів дає змогу виявляти збої, перевіряти стабільність з'єднання та своєчасно реагувати на можливі порушення.

Таким чином, методи оптимізації функціонування ОС забезпечують стабільну, безпечну та ефективну роботу операційної системи.

2.2 Засоби для моніторингу та оптимізації ОС

Реалізація ефективної системи моніторингу та оптимізації функціонування операційної системи є складним завданням, що потребує вибору та інтеграції відповідних програмних засобів, здатних здійснювати збір метрик, аналіз системного навантаження та прийняття обґрунтованих рішень щодо коригування конфігурації або виконання оптимізаційних дій. Для більш системного підходу до організації моніторингу доцільно класифікувати інструменти за кількома категоріями, зокрема вбудовані засоби операційної системи, сторонні утиліти, комплексні системи

моніторингу та спеціалізовані фреймворки для обробки та аналізу даних [25]. З огляду на функціональне призначення, інструменти моніторингу можна поділити на дві основні групи: вбудовані засоби, які є невід'ємною частиною операційної системи і забезпечують базові можливості контролю за станом системи та використанням її ресурсів без необхідності додаткового встановлення, та сторонні утиліти, що пропонують розширений набір функцій, включаючи масштабованість, інтеграцію з іншими системами, а також більш детальну візуалізацію та аналіз отриманих даних.

2.1.1 Вбудовані засоби моніторингу та оптимізації ОС

Вбудовані засоби моніторингу операційних систем є ключовими інструментами для спостереження за станом системних ресурсів і управління ними. Вони забезпечують базовий рівень контролю, дозволяючи користувачам оцінювати навантаження на процесор, використання пам'яті, дискову активність та мережеву взаємодію, а також отримувати статистику про активні процеси, системні виклики та інші параметри продуктивності.

В операційній системі Windows основним засобом моніторингу є Task Manager, який дає змогу переглядати список поточних процесів, рівень використання процесора, оперативної пам'яті, дискової підсистеми та мережі [26]. Його функціональність включає вкладки з відображенням продуктивності в реальному часі, диспетчер служб, а також історію використання ресурсів. Цей інструмент є зручним для базового аналізу, однак має обмежені можливості щодо глибокого дослідження системної продуктивності. Для розширеного моніторингу Windows також використовує такі засоби, як Resource Monitor, який надає детальний аналіз ресурсів у розрізі процесів і служб, Performance Monitor для збору метрик з гнучкими параметрами візуалізації, а також Windows Management Instrumentation (WMI) – API, що забезпечує доступ до системних параметрів через скрипти або зовнішні програми.

В середовищі Linux поширеним інструментом є htop – вдосконалена версія стандартної утиліти top, яка надає зручний інтерфейс з можливістю сортування, фільтрації, динамічного налаштування відображення метрик, а також інтерактивного керування процесами [27]. Візуалізація навантаження на процесор і пам’ять у вигляді графіків полегшує оперативний аналіз [28]. Хоча htop є потужним у своєму класі, його функціональність обмежується Linux та деякими Unix-подібними системами. Для моніторингу вводу/виводу в Linux застосовується iostat, який надає детальну статистику взаємодії з дисковими пристроями та мережевими інтерфейсами, дозволяючи оцінити ефективність дискових операцій і передачі даних [29]. Ще одним ефективним інструментом є perf, що призначений для глибокого аналізу продуктивності на рівні ядра та користувацького простору. Він підтримує профілювання коду, моніторинг системних викликів, аналіз кешу процесора та виявлення критичних ділянок навантаження [30]. Утиліта perf` характеризується високою гнучкістю, але вимагає певного рівня технічної підготовки через складність інтерпретації результатів.

Додатково в Linux–середовищі застосовуються утиліти vmstat, mpstat для аналізу пам’яті та процесорних ядер, dstat для комбінованої статистики в реальному часі, а також systemd-analyze та journalctl для аналізу часу завантаження та журналів системи.

В Unix-подібних системах та macOS вбудованими засобами моніторингу є Activity Monitor, який забезпечує функціональність, аналогічну до Task Manager, з додатковими можливостями щодо перегляду енергоспоживання, роботи з мережею та дисками. Для глибшого профілювання додатків в macOS доступний інструмент Instruments з комплекту Xcode.

Загалом вбудовані утиліти операційних систем становлять основу для первинної діагностики продуктивності, надаючи доступ до базових метрик і можливість оперативного реагування на системні проблеми. Проте їх функціональність є обмеженою в аспектах масштабованості, збереження

історичних даних, автоматизації аналізу та інтеграції з централізованими системами моніторингу. Тому для більш комплексного аналізу доцільним є використання зовнішніх програмних рішень або поєднання вбудованих інструментів з інтерфейсами автоматизації.

2.2.2 Сторонні засоби для моніторингу та оптимізації ОС

В процесі моніторингу системних ресурсів та інфраструктури, крім вбудованих засобів операційних систем, широкого застосування набули сторонні утиліти та централізовані системи моніторингу, які забезпечують більшу гнучкість, масштабованість та точність збору метрик. Сторонні утиліти, зокрема Process Explorer, Netdata, Glances, AIDA64, HWMonitor та Open Hardware Monitor, забезпечують розширену деталізацію процесів, реальний моніторинг ресурсів в текстовому або вебінтерфейсі, а також контроль за апаратною частиною системи. Вони ефективні для індивідуального або локального застосування, але часто не підтримують централізованого контролю чи складної логіки автоматизованого реагування.

Для великих інфраструктур або багатосегментних мереж доцільним є використання централізованих систем моніторингу, зокрема Prometheus, Grafana, Zabbix, Nagios та Elastic Stack (ELK). Ці рішення забезпечують збір, обробку та візуалізацію даних з великої кількості джерел, а також мають потужні механізми створення тригерів, звітності та інтеграції з іншими системами.

Prometheus – система моніторингу та збору метрик, оптимізована для роботи з мікросервісною архітектурою та контейнеризованими середовищами, такими як Kubernetes [31]. Вона використовує pull-модель збору даних, мову запитів PromQL для гнучкого аналізу метрик, а також підтримує зберігання даних у вигляді часових рядів [32]. Prometheus легко інтегрується з іншими інструментами, зокрема з Grafana, яка використовується для візуалізації зібраних метрик. Основними перевагами

Prometheus є простота налаштування, масштабованість та придатність для динамічних середовищ, проте для ефективного використання потрібні знання PromQL, а також окреме налаштування системи візуалізації.

Grafana – потужна платформа для візуалізації, яка підтримує велику кількість джерел даних, зокрема Prometheus, Elasticsearch, MySQL, PostgreSQL, і дозволяє створювати інтерактивні панелі моніторингу з функціями сповіщення та кастомізації [33]. Хоча Grafana не виконує збір метрик самостійно, вона забезпечує високий рівень інтерактивності та зручності у представленні даних, що робить її незамінним компонентом моніторингового стеку.

Zabbix – система моніторингу з відкритим кодом, орієнтована на корпоративні середовища. Вона підтримує як pull-, так і push-моделі збору даних, складну логіку тригерів, систему сповіщень, а також вбудовані засоби візуалізації та звітності. Zabbix дозволяє здійснювати моніторинг великої кількості хостів і пристроїв, як фізичних, так і віртуальних, проте потребує ретельного налаштування і значних ресурсів для масштабованого використання [34].

Nagios – один з найстаріших інструментів моніторингу, що підтримує широкий спектр плагінів та інтеграцій. Його основні функції включають перевірку доступності мережевих сервісів, виявлення збоїв та надсилання сповіщень. Nagios є гнучким завдяки можливості розширення, однак потребує специфічних знань для налаштування в складних середовищах і може споживати значні ресурси при моніторингу великої кількості об'єктів [35].

Крім того, ELK використовується як потужний інструмент для збору, обробки та аналізу логів, який може доповнювати або інтегруватися з іншими системами моніторингу.

2.1.3 Логування та аудит

Логування та аудит є ключовими складовими механізмів забезпечення безпеки та стабільної роботи операційних систем і програмного забезпечення. Логування охоплює процес автоматичного фіксування подій, що відбуваються в системі, з метою подальшого моніторингу, діагностики несправностей, виявлення аномалій та оцінки функціонального стану [36]. Зібрані дані можуть стосуватися помилок, попереджень, спроб доступу, змін конфігурації, використання ресурсів тощо [37].

Аудит є процесом систематичної перевірки журналів подій з метою виявлення потенційних загроз, несанкціонованих дій, а також для оцінки відповідності встановленим політикам безпеки. Це дає змогу підвищити прозорість операцій, забезпечити контроль за користувацькою активністю та сприяти вдосконаленню заходів кіберзахисту [38].

Основні типи журналів включають:

- системні журнали, які містять інформацію про роботу операційної системи та її компонентів, наприклад, помилки завантаження драйверів або підключення до зовнішніх пристроїв;

- журнали безпеки, які фіксують події, пов’язані з аутентифікацією, авторизацією та іншими аспектами контролю доступу, зокрема спроби несанкціонованого входу до системи;

- аудиторські журнали, які відображають дії користувачів та адміністраторів, пов’язані зі змінами налаштувань, конфігурацій або об'єктів доступу.

Процес логування та аудиту включає кілька етапів:

- збір даних, що полягає в автоматичному фіксуванні подій за допомогою вбудованих або спеціалізованих інструментів операційної системи;

- зберігання, що забезпечується архівацією журналів у спеціальних лог-файлах або базах даних для подальшого аналізу;

- аналіз, який полягає в обробці зібраних даних з метою виявлення аномалій, інцидентів безпеки або невідповідностей політикам;

- сповіщення, яке в разі виявлення підозрілої активності або порушень, система моніторингу надсилає повідомлення адміністраторам або відповідальним особам.

В таблиці 2.1 представлено основні типи логів, які збираються в процесі роботи операційної системи.

Таблиця 2.1 – Основні типи логів

Тип логу	Опис
Системні логи	Події, пов’язані з роботою ОС, її компонентів та ресурсів
Логи безпеки	Дані про автентифікацію, авторизацію, доступ до системи
Аудиторські логи	Дії користувачів або адміністраторів, зміни конфігурацій

Таким чином, логування та аудит є важливими інструментами контролю та забезпечення інформаційної безпеки. Вони дозволяють своєчасно виявляти та реагувати на інциденти, зменшувати ризики витоку даних, підвищувати стійкість до загроз, а також формують базу для подальшого вдосконалення політик безпеки та процедур управління інфраструктурою.

В даному розділі було розглянуто основні методи та засоби моніторингу та оптимізації функціонування операційних систем, що є невід’ємною частиною забезпечення стабільності, безпеки та продуктивності IT-інфраструктури. Ефективний моніторинг базується на систематичному зборі, зберіганні та аналізі метрик роботи ОС і додатків, що дозволяє виявляти аномалії, оцінювати навантаження на ресурси та прогнозувати можливі проблеми.

Значну роль в процесі оптимізації відіграють автоматизовані засоби реагування, які на основі отриманих даних можуть запускати сценарії корекційних дій, що підвищує швидкість та ефективність усунення збоїв та підтримує належний рівень продуктивності системи.

Особливу увагу приділено методам ведення журналів та аудиту, які дозволяють не лише контролювати стан системи, але також забезпечувати інформаційну безпеку, своєчасно виявляючи потенційні загрози та несанкціоновані дії. Аналіз логів є важливою складовою підтримки стабільності роботи та оперативного реагування на інциденти.

Загалом, інтеграція комплексного моніторингу з автоматизованими механізмами оптимізації та аудиту формує ефективний підхід до управління операційними системами, що відповідає сучасним вимогам надійності, безпеки та продуктивності корпоративних інформаційних систем.

3 МЕТОД ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ФУНКЦІОНУВАННЯ UNIX-ПОДІБНИХ СИСТЕМ

Стрімкий розвиток інформаційно-комунікаційних технологій спричиняє формування якісно нового цифрового середовища, в якому домінують сучасні засоби обробки, зберігання та передавання інформації, що витісняють традиційні підходи. Одним з ключових наслідків цього процесу є експоненційне зростання обсягів даних, що може бути забезпечено застосуванням надійної серверної інфраструктури [39]. В сучасних умовах така інфраструктура створюється на основі віртуалізації та контейнеризації, що забезпечує високу гнучкість, масштабованість та продуктивність [40].

В контексті зростаючих вимог до обробки великих обсягів інформації особливої актуальності набуває проблема підтримки високої продуктивності серверних систем, яка залежить від ефективності механізмів управління ресурсами, реалізованих на рівні операційної системи [41].

Сучасні концепції підвищення ефективності функціонування мережної та серверної інфраструктури ґрунтуються на комплексному підході, що передбачає інтеграцію методів моніторингу, діагностики та динамічної оптимізації системних ресурсів [42]. Зокрема, в UNIX-подібних операційних системах активно використовуються вбудовані утиліти для збору телеметричних показників, що дозволяє ідентифікувати аномальні стани, критичні події, а також потенційні «вузькі місця» в роботі системи [43].

Аналіз зібраних даних створює основу для впровадження рішень, спрямованих на підвищення стабільності та безперервності функціонування ОС, а також дає змогу реалізовувати механізми прогнозування, скорочення затримок обробки запитів і зниження загального навантаження на ресурси. Крім того, автоматизація процесів управління сприяє зменшенню потреби в ручному адмініструванні та підвищенню загальної ефективності управління інфраструктурою.

Для реалізації запропонованого методу було розроблено алгоритм, що ґрунтується на комплексному підході до забезпечення стабільності, надійності та енергоефективності функціонування операційної системи. Основу цього підходу становить інтеграція процедур безперервного моніторингу, автоматизованого управління апаратними та програмними ресурсами, а також механізмів своєчасного виявлення та локалізації збоїв в роботі системи. Розроблений алгоритм включає наступні етапи:

1. Ініціалізація системного моніторингу.

На даному етапі ініціюється виконання скриптів, які забезпечують первинне збирання та регулярне оновлення інформації про стан ключових обчислювальних ресурсів системи, зокрема центрального процесора, оперативної пам’яті та підсистеми зберігання даних. Збирання відповідних метрик здійснюється за допомогою стандартних утиліт, інтегрованих в UNIX-подібні операційні системи. Отримані дані реєструються у спеціалізованих лог-файлах, що формуються у виділеному каталозі для подальшого аналізу.

2. Автоматизоване управління ресурсами.

На другому етапі реалізується автоматизоване керування системними ресурсами шляхом впровадження механізмів, що забезпечують технічне обслуговування та оперативне реагування на події, пов’язані з деградацією критичних компонентів UNIX-подібної операційної системи. Основним інструментом, який забезпечує автоматизацію цих процесів, виступає вбудований планувальник завдань cron, що дозволяє запускати відповідні скрипти з заданою періодичністю у фоновому режимі без втручання адміністратора.

Для підвищення функціональної гнучкості в структуру скриптів можуть бути інтегровані логічні модулі, які, окрім стандартної періодичної перевірки стану ресурсів, забезпечують автоматичне виявлення збоїв та ініціацію процедур перезапуску критично важливих сервісів. Це актуально для служб резервного копіювання, засобів віддаленого доступу та інших

компонентів, що забезпечують безперервність функціонування інфраструктури.

Кількість та склад логічних модулів у скриптах визначається поточним рівнем навантаження, функціональним призначенням системи та характеристиками застосовуваних сервісів. Частота виконання окремих модулів відповідає рівню критичності відповідних процесів, що дозволяє мінімізувати потребу в ручному адмініструванні, знижує ймовірність помилок та сприяє стабільному функціонуванню серверного середовища в умовах динамічних змін навантаження.

3. Контроль системного журналювання.

На цьому етапі здійснюється контроль за системним журналюванням шляхом безперервного аналізу логів операційної системи та повідомлень ядра з метою своєчасного виявлення аномальної активності, критичних помилок або ознак несанкціонованого доступу. Моніторинг реалізується в режимі реального часу або з мінімальною затримкою, що є ключовим чинником для ранньої діагностики потенційних відмов програмного забезпечення та ідентифікації загроз інформаційній безпеці.

Інциденти, що класифікуються як нетипові або потенційно небезпечні, фіксуються в спеціалізованому журналі подій, який, виконує функцію тригера для автоматичного інформування системного адміністратора. Такий підхід дозволяє не лише оперативно реагувати на загрози, але й забезпечити первинну діагностику виявлених відхилень у роботі системи.

Інтеграція механізмів контролю журналювання з попередніми етапами моніторингу та управління ресурсами створює цілісну систему базового рівня захисту інфраструктури, що поєднує реактивні та проактивні методи забезпечення стабільності функціонування серверного середовища.

4. Механізм сповіщення та реагування.

На даному етапі реалізується впровадження механізмів сповіщення, що забезпечують оперативне реагування на виявлені збої, критичні помилки або аномальні події в роботі операційної системи. Це значно скорочує час

простою, мінімізує ризик втрати даних та сприяє підтриманню безперервності функціонування сервісів.

Крім того, на цьому етапі можлива інтеграція з централізованими системами моніторингу, які функціонують в межах корпоративної мережі, що забезпечує єдиний простір для спостереження та управління. Такий підхід створює передумови для автоматизації реакцій на критичні події, що підвищує загальну ефективність адміністрування.

Таким чином, систематизоване впровадження сповіщень в поєднанні з попередніми етапами дозволяє суттєво підвищити рівень надійності, інформаційної безпеки та стабільності функціонування серверної інфраструктури, а також сприяє забезпеченню сталого режиму роботи всієї корпоративної мережі.

5. Циклічний аналіз та адаптація.

На цьому етапі відбувається завершення формування повного циклу моніторингу, який включає послідовну реалізацію чотирьох ключових компонентів, зокрема діагностики поточного стану системи, реагування на виявлені події, адаптації конфігурації або параметрів функціонування відповідно до здійснених змін, а також прогнозування поведінки системи з урахуванням можливих коливань у навантаженні.

Завдяки цьому забезпечується не лише підтримка стабільної роботи операційного середовища, а й створюються умови для проактивного управління ресурсами, що є важливим для сучасних високонавантажених IT-інфраструктур.

Псевдокод, наведений в Додатку В, деталізує логіку реалізації запропонованого методу, зокрема умови виконання, ітераційні конструкції, виклики функцій та механізми обробки вхідних та проміжних даних, що сприяє глибшому розумінню послідовності дій, що реалізуються в межах методу.

Загальна ефективність методу оптимізації функціонування Unix-подібних операційних систем визначається через багатокритеріальну

оптимізацію основних параметрів, зокрема максимізацію стабільності та мінімізацію частоти системних збоїв, енергоспоживання та скорочення затримок в реагуванні на події. Ефективність методу F_{eff} визначається у вигляді багатофакторної оптимізаційної задачі наступного вигляду:

$$\max_t E(t) = w_1 \cdot S(t) - w_2 \cdot R(t) - w_3 \cdot P(t) - w_4 \cdot L(t), \quad (3.1)$$

де w_1, w_2, w_3, w_4 – вагові коефіцієнти, які визначають пріоритетність відповідного критерію задачі.

Компоненти оптимізаційної задачі (1) представлені нижче:

1. Оцінка стабільності системи $S(t)$ тривалістю безперервної роботи операційної системи (аптаймом), яку необхідно максимізувати. Оцінка стабільності системи може бути обчислена як:

$$S(t) = \frac{T_{uptime}}{T_{total}}, \quad (3.2)$$

де T_{uptime} – тривалість безперервної роботи системи без перезавантажень чи критичних збоїв, с;

T_{total} – загальний час моніторингу, с.

2. Оцінка надійності системи $R(t)$ кількістю виниклих збоїв або критичних помилок протягом періоду спостереження. Зниження цього показника є важливим чинником підвищення працездатності системи. Оцінка надійності системи може бути обчислена як:

$$R(t) = 1 - \frac{F(t)}{F_{\max}}, \quad (3.3)$$

де $F(t)$ – кількість зареєстрованих збоїв або критичних помилок за час t, од.;

$F_{\max}$ – максимально допустима кількість збоїв в інтервалі спостереження, с.

3. Енергоефективність $P(t)$ відображає рівень споживання енергії операційною системою під час роботи. Мінімізація цього параметру сприяє зменшенню теплового навантаження та продовженню ресурсу серверного обладнання. Енергоефективність може бути обчислена як:

$$P(t) = 1 - \frac{W(t)}{W_{\max}}, \tag{3.4}$$

де $W(t)$ – фактичне енергоспоживання системи, Вт;

$W_{\max}$ – максимально допустиме енергоспоживання системи, Вт.

4. Швидкість реагування $L(t)$ характеризує затримки системи при обробці подій, зокрема запуску скриптів, активації тригерів моніторингу чи обробці помилок служб. Зменшення затримок є індикатором оперативності роботи системи. Швидкість реагування на події може бути обчислена як:

$$L(t) = 1 - \frac{D(t)}{D_{\max}}, \tag{3.5}$$

де $D(t)$ – фактична середня затримка відповіді, мс;

$D_{\max}$ – максимально допустима затримка, мс.

Запропонований підхід забезпечує комплексну кількісну оцінку ефективності методу підвищення продуктивності Unix-подібних систем, базуючись на інтеграції ключових показників, зокрема стабільності, надійності, енергоефективності та швидкості реагування. Така модель

дозволяє враховувати взаємозв’язок між продуктивністю операційної системи та основними параметрами її функціонування із застосуванням пріоритетного вагового розподілу.

На початковому етапі було здійснено аналіз апаратних вимог для операційної системи Ubuntu 18.04 LTS, обраної як тестове середовище для лабораторного обладнання кафедри електронних обчислювальних машин ХНУРЕ. Визначено, що для забезпечення стабільної роботи системи достатньо процесора з двома ядрами та тактовою частотою 2,9 ГГц, а також 4 ГБ оперативної пам’яті. Обсяг дискового простору, виділеного операційній системі, становить 20 ГБ, що є достатнім для підтримки стабільної роботи ОС та розгортання необхідних сервісів, зокрема веб-сервера Apache 2 для хостингу веб-сайту. Пропускна здатність каналу зв’язку в тестовому середовищі становить не менше 1 Гбіт/с, що забезпечує належний рівень передачі даних.

Для створення реального навантаження було розгорнуто веб-сайт, а також згенеровано додаткове навантаження у вигляді фіктивних логів розміром по 200 МБ кожен.

В межах розробки методу підвищення ефективності функціонування операційної системи було створено та запущено автоматизований скрипт моніторингу стану тестового середовища, який виконується за розкладом із використанням утиліти cron. На рисунку 4.1 наведено узагальнену схему основних компонентів операційної системи, що підлягають моніторингу в межах розробленого методу.

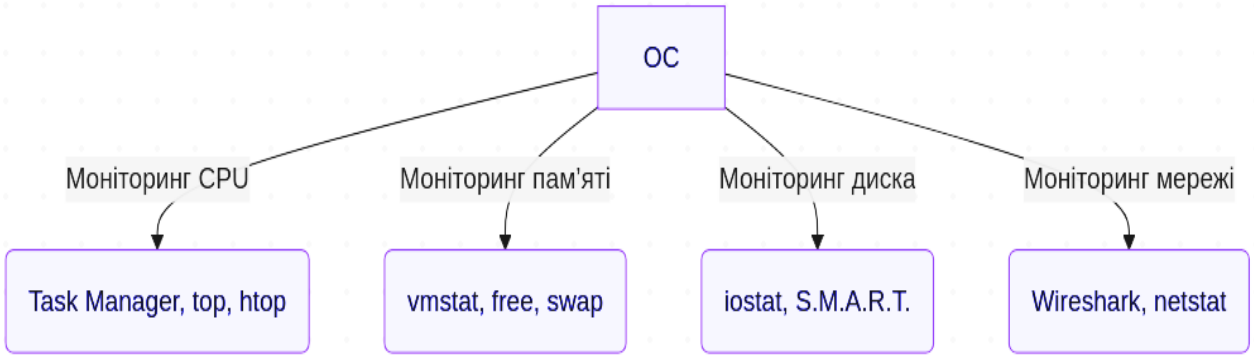


Рисунок 4.1 – Основні компоненти операційної системи, які підлягають моніторингу

Розроблений скрипт має модульну архітектуру, що забезпечує його адаптацію до специфічних вимог моніторингу, а також можливість масштабування шляхом додавання відповідних логічних блоків. Автоматизований скрипт охоплює основні компоненти операційної системи, зокрема центральний процесор (CPU), оперативну пам’ять та дисковий простір. За результатами виконання скрипта автоматично формуються сповіщення у вигляді повідомлень до месенджера адміністратора про критичні події або результати роботи скрипта, наприклад, досягнення критичного рівня заповнення диску чи видалення застарілих лог-файлів.

Архітектура скрипта включає такі функціональні блоки:

1. Блок ініціалізації. В цьому модулі визначаються змінні середовища, виконується перевірка прав доступу, створюються лог-файли та налаштовується збереження результатів моніторингу. Також фіксується поточний час виконання з метою формування звітності.

2. Блок збору метрик. Даний модуль відповідає за діагностику стану тестового середовища за допомогою вбудованих утиліт. Кожна команда виконується в окремому процесі, що дозволяє забезпечити асинхронну обробку даних і зменшити затримку. Утиліта free використовується для отримання актуальної інформації про використання оперативної пам’яті, що дозволяє виявити потенційний дефіцит пам’яті, здатний знижувати

продуктивність. Утиліта vmstat забезпечує розширений моніторинг основних параметрів ядра ОС, включаючи статистику процесів, навантаження на CPU, обмін пам’яттю, активність вводу/виводу, що дозволяє оперативно ідентифікувати перевантаження або «вузькі місця» в роботі системи. Утиліта htop застосовується для візуалізації загального стану системи та активних процесів у режимі реального часу.

3. Блок обробки подій. В цьому модулі виконується аналіз критичних подій на основі виявлених підозрілих записів у лог-файлах, контроль заповнення дискового простору та очищення застарілих даних. Автоматичне видалення тимчасових файлів зменшує навантаження на файлову систему та підвищує швидкість доступу до актуальних даних. Контроль обсягу вільного простору здійснюється за допомогою утиліт df та du. Це дозволяє реалізувати порогові правила для виявлення перевантажених розділів. Критичним порогом вважається заповнення диску на 80%, після чого скрипт виконує очищення кешу та тимчасових логів. Також реалізовано перевірку стану основних сервісів, зокрема Apache2, MySQL, SSH, які є критичними для функціонування вебсайту. В разі виявлення їх деградації виконується автоматичний перезапуск через утиліту systemctl.

4. Блок сповіщень. В разі фіксації інцидентів, таких як зміна стану системи, критичне заповнення або очищення диску, активується механізм оперативного інформування адміністратора через встановлений канал сповіщень (рисунок 4.2).

5. Блок логування та збереження статистики. Зібрані метрики записуються в локальні файли з відповідним іменуванням, що забезпечує структурованість зберігання інформації. За визначеним часовим розкладом автоматично формується зведений звіт, який відображає рівень навантаження системи за певний період. Це дозволяє здійснювати ретроспективний аналіз продуктивності та своєчасно виявляти негативні тенденції в роботі системи.

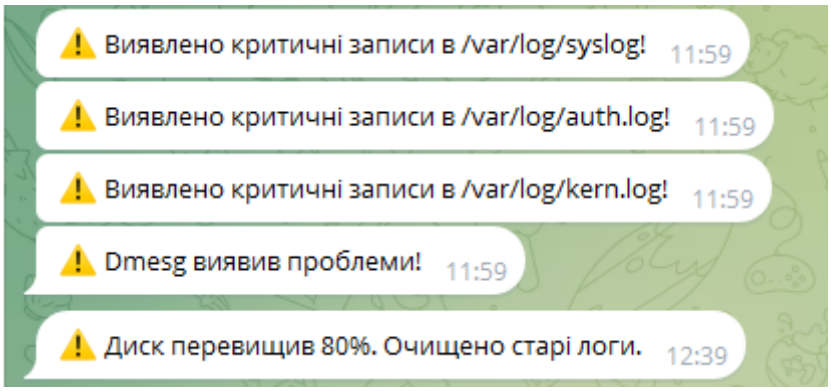


Рисунок 4.2 – Сповіщення в месенджері адміністратора про результати роботи автоматизованого скрипту

6. Блок автоматичного виконання. Даний модуль забезпечує інтеграцію роботи скрипта із системним планувальником завдань — утилітою cron, що дозволяє реалізувати періодичне виконання функцій моніторингу. Для тестування було обрано наступні періоди запуску моніторингу:

- monitor – збір метрик та запис у лог-файли про стан системних процесів кожні 5 хвилин;

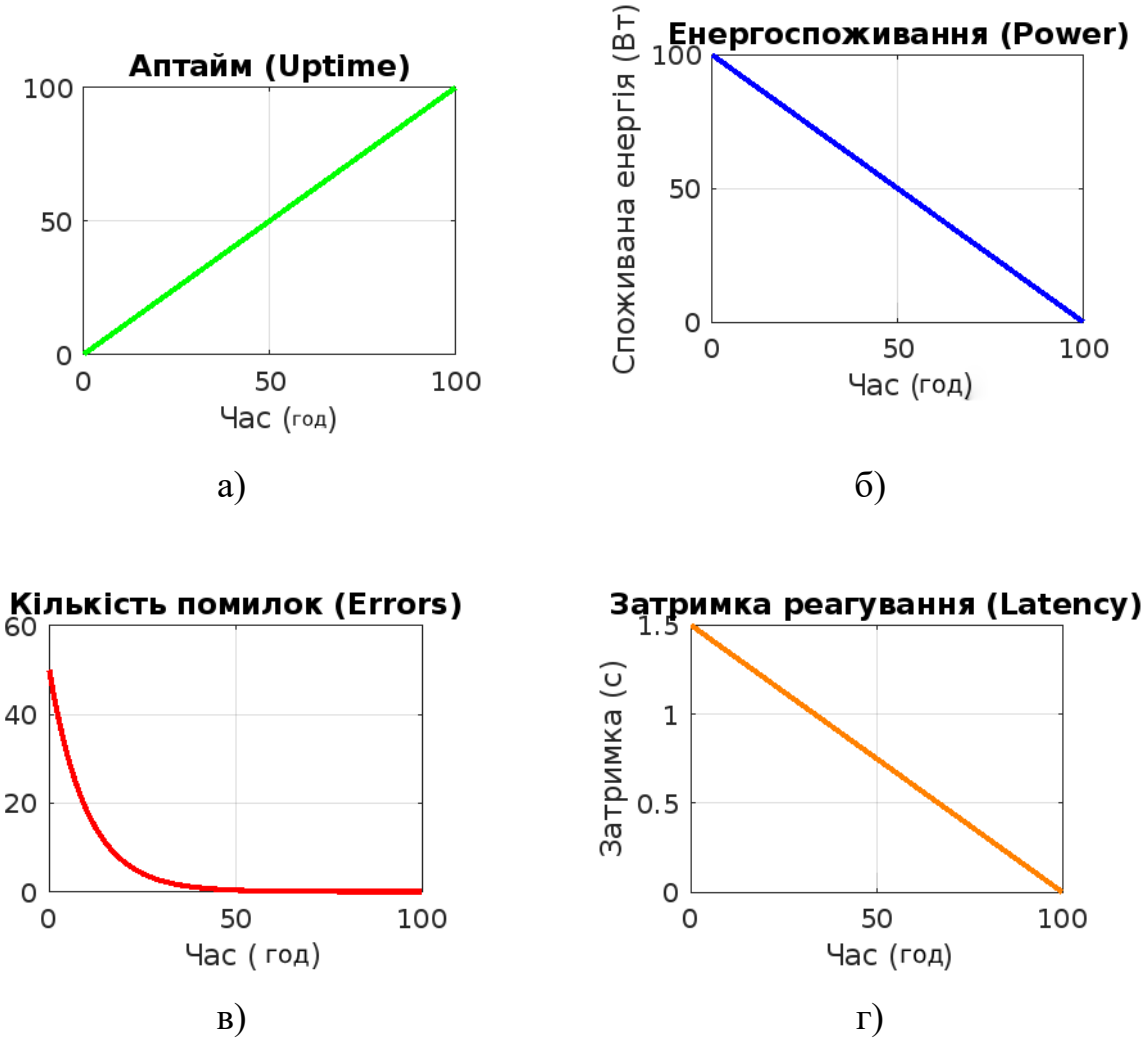
- daily – щоденне очищення тимчасових файлів, контроль використання дискового простору та стану системних служб;

- hourly – щогодинна перевірка лог-файлів на наявність критичних записів і надсилання відповідних сповіщень;

- weekly – щотижневе формування аналітичного звіту про навантаження на CPU.

Для реалізації інтеграції з cron, відповідні записи було додано до конфігураційного файлу планувальника, що формують політику моніторингу та правила генерації звітів.

За результатами тестування автоматизованого скрипта для моніторингу стану операційної системи було отримано динаміку зміни досліджуваних параметрів ОС Ubuntu 18.04, яка представлена на рисунку 4.3.



а) б)

в) г)

Рисунок 4.3 – Зміни показників ОС Ubuntu 18.04 LTS за часом:
а) час аптайму; б) енергоспоживання ОС; в) кількість виявлених та виправлених помилок роботи; г) затримка реагування на події операційною системою

В дослідженні детально проаналізовано метод підвищення ефективності функціонування Unix-подібних операційних систем серверної інфраструктури. Запропонований підхід базується на використанні автоматизованого скрипту для моніторингу системних показників у режимі реального часу, що забезпечує стабільну та безперервну роботу ОС із мінімальним втручанням адміністратора. Метод дозволяє автоматизувати контроль стану системи та оркестрацію ключових системних процесів, знижуючи вплив людського фактора і підвищуючи надійність роботи.

Наукова новизна полягає у впровадженні адаптивного механізму моніторингу, який фіксує поточний стан системи та дає можливість прогнозувати подальше функціонування під навантаженням. Це забезпечує динамічне балансування між продуктивністю, енергоспоживанням і безпекою.

Результати моделювання демонструють, що автоматизація та постійний моніторинг призводять до зниження латентності, кількості помилок і енергоспоживання, водночас підвищуючи час безперервної роботи системи. Це сприяє підвищенню ефективності серверної інфраструктури, забезпечує довготривалу стабільність ОС, знижує ризики збоїв та мінімізує необхідність втручання адміністратора.

Отже, запропонований метод є доцільним для застосування в серверній інфраструктурі корпоративних комп’ютерних мереж, особливо в умовах динамічної зміни навантаження або обмежених апаратних ресурсів. Подальші дослідження варто зосередити на інтеграції цього методу із сучасними системами інтелектуального моніторингу, які використовуються для створення автономних систем адміністрування в корпоративних мережах.

4 РОЗРОБКА ЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ СТАНУ ОС

4.1 Побудова архітектури застосунку

Для тестування методів моніторингу та оптимізації стану операційної системи в роботі було розроблено вебзастосунок для моніторингу стану операційної системи в режимі реального часу.

Архітектура розробленого вебзастосунку побудована на основі клієнт–серверної моделі, яка забезпечує чітке розділення функціональних компонентів системи на три основні рівні: інтерфейс користувача (Frontend), серверну частину (Backend) та програмний інтерфейс прикладного рівня (API), що забезпечує гнучкість масштабування, полегшує інтеграцію з іншими інформаційними системами та підвищує зручність користування.

Frontend-компонент відповідає за візуалізацію інформації в веб-інтерфейсі та реалізований за допомогою HTML, CSS та JavaScript. Його функціональність включає інтерактивне відображення даних, зокрема оновлення в режимі реального часу через звернення до API. Також інтерфейс підтримує налаштування теми оформлення, що підвищує комфорт користувача.

Backend реалізує серверну логіку та відповідає за обробку запитів, що надходять від Frontend або через API. Він написаний мовою програмування Python з використанням бібліотеки aiohttp, яка забезпечує асинхронну обробку запитів, що є важливим для підвищення продуктивності системи в умовах багатокористувацького середовища. Основні функції Backend включають збір та обробку інформації про стан операційної системи, яка згодом передається на клієнтську частину або через API. Для отримання системних метрик застосовується бібліотека psutil, що забезпечує доступ до детальних показників використання апаратних ресурсів, зокрема процесора, оперативної пам’яті, дискових накопичувачів та мережних інтерфейсів.

Компонент API реалізований у вигляді REST API на основі бібліотеки aiohttp, що забезпечує уніфікований доступ до системних даних у форматі JSON. Це дозволяє інтегрувати систему моніторингу з іншими інструментами керування, автоматизації та аналітики. API підтримує отримання актуальної інформації про стан системи через стандартні HTTP-запити, що забезпечує платформонезалежний механізм взаємодії.

Таким чином, побудова додатку за принципами модульності, асинхронності та стандартизованої взаємодії між компонентами забезпечує його гнучкість, масштабованість та надійність в умовах функціонування в серверному середовищі. Загальна структура побудованого вебзастосунку наведена на рисунку 4.1.

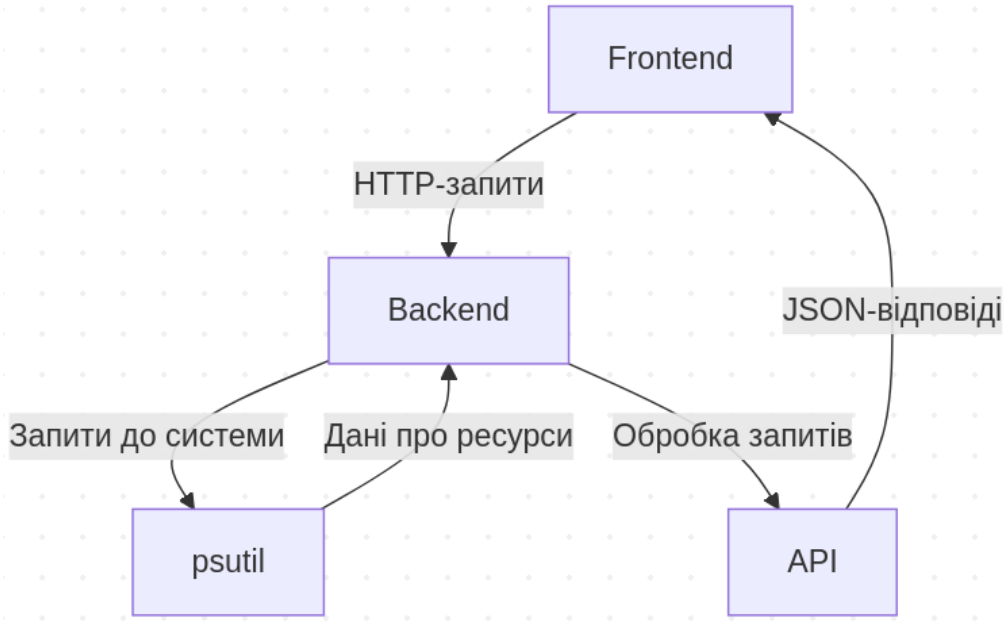


Рисунок 4.1 – Загальна архітектура застосунку для моніторингу

В серверній частині розробленого додатку реалізується комплексний підхід до збору інформації про стан операційної системи, зокрема отримання релевантних характеристик процесора (CPU), оперативної пам'яті, дискових ресурсів та мережного інтерфейсу з високою точністю та мінімальним навантаженням на систему. Для цього використовуються два ключові джерела системних даних – бібліотека psutil та віртуальні файлові системи

Linux proc та sys. Бібліотека psutil дозволяє отримати дані про апаратні ресурси та системні показники, віртуальні файлові системи Linux proc та sys надають можливість прямого зчитування низькорівневих системних параметрів безпосередньо з ядра операційної системи, що дозволяє отримувати більш детальну та достовірну інформацію про стан системи. На рисунку 4.2 наведено основні методи та джерела збору системних метрик, які використовуються у серверній частині.

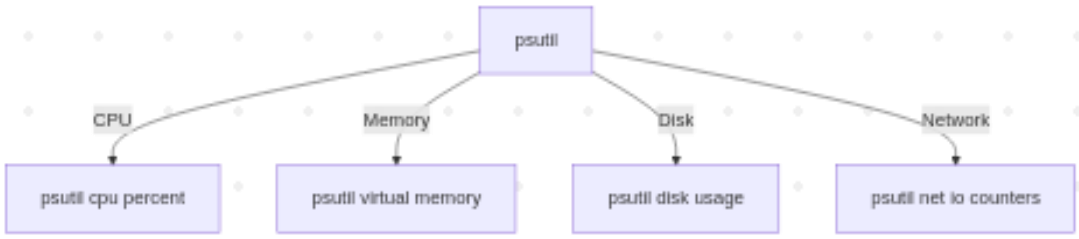


Рисунок 4.2 – Методи та джерела збору системних метрик

Для визначення навантаження на процесор у відсотковому співвідношенні застосовується метод psutil.cpu_percent(). Поточна тактова частота CPU отримується за допомогою psutil.cpu_freq(). Додатково використовується psutil.cpu_count() для визначення кількості фізичних ядер та логічних потоків, що забезпечує повне уявлення про ресурсну конфігурацію процесора. При застосуванні віртуальної файлової системи Linux, дані про завантаження CPU зчитуються з файлу /proc/stat, де накопичується статистика за всіма CPU-ядрами.

Поточна частота процесора отримана безпосередньо з файлів /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq, які відображають динамічні параметри керування живленням процесорних ядер (лістинг 4.1).

Лістинг 4.1 – Отримання статистики CPU через /proc/stat

```
with open('/proc/stat', 'r') as f:
    cpu_stats = f.readline()
    print("CPU Stats:", cpu_stats)
```

Для моніторингу оперативної пам'яті використовується метод psutil.virtual_memory(), що надає відомості про загальний обсяг пам'яті, використану та доступну пам'ять. Для аналізу використання swap застосовується метод psutil.swap_memory(). В Linux ці дані доступні у файлі meminfo, який містить докладну інформацію про стан пам'яті, включаючи кеш і буфери (лістинг 4.2).

Лістинг 4.2 – Отримання інформації про оперативну пам’ять через psutil

```
import psutil
memory = psutil.virtual_memory()
print("Total Memory:", memory.total)
print("Used Memory:", memory.used)
print("Available Memory:", memory.available)
```

Інформація про використання дискового простору збирається за допомогою методу psutil.disk_usage(), який дозволяє отримати дані про загальний обсяг, використаний та вільний простір на дисках. Для збору статистики вводу/виводу використовується метод psutil.disk_io_counters(). В Linux ці дані отримано з файлу diskstats, що містить інформацію про активність дисків, а з директорії /sys/block/*, що надає інформацію про фізичні диски (лістинг 4.3).

Лістинг 4.3 – Отримання інформації про диски через psutil

```
import psutil
disk_usage = psutil.disk_usage('/')
print("Total Disk Space:", disk_usage.total)
print("Used Disk Space:", disk_usage.used)
print("Free Disk Space:", disk_usage.free)
```

Для моніторингу мережної активності застосовується метод psutil.net_io_counters(), що надає статистику мережної активності, включно з обсягом переданих та отриманих даних, а також кількістю пакетів. Відомості про мережні інтерфейси збираються за допомогою методу psutil.net_if_addrs(). В Linux ці дані доступні з файлу dev, що містить

статистику мережних інтерфейсів, або з директорії /sys/class/net/*, що надає інформацію про стан мережних інтерфейсів (лістинг 4.4).

Лістинг 4.4 – Отримання статистики мережі через /proc/net/dev

```
with open('/proc/net/dev', 'r') as f:
    network_stats = f.readlines()
    for line in network_stats[2:]:  # Пропускаємо заголовки
        print("Network Interface Stats:", line.strip())
```

Вебінтерфейс програми розроблено з орієнтацією на комфорт користувача. Він гарантує відображення інформації в режимі реального часу. Інтерфейс було створено з використанням HTML, CSS та JavaScript. На рисунку 4.3 представлено схему роботи вебінтефейсу користувача.

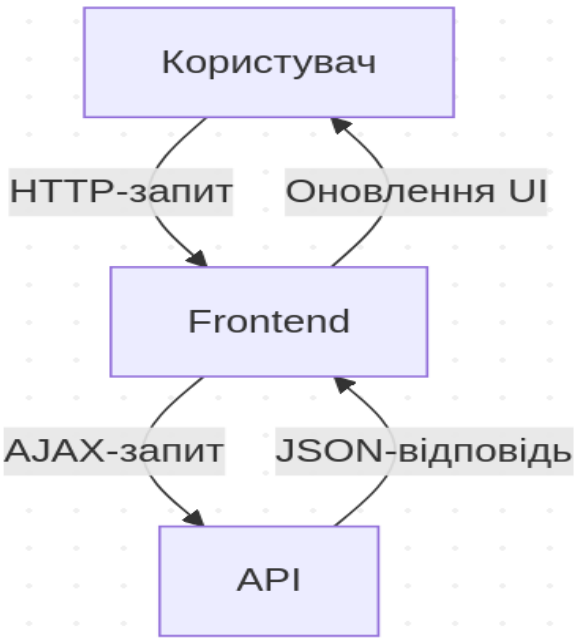


Рисунок 4.3 – Схема роботи вебінтерфейсу

Інтерфейс реалізовано в мінімалістичному стилі з акцентом на ключові системні показники, зокрема рівень завантаження процесора, обсяг використаної оперативної пам’яті, поточний стан дискових ресурсів, параметри мережної активності (рисунок 4.4).

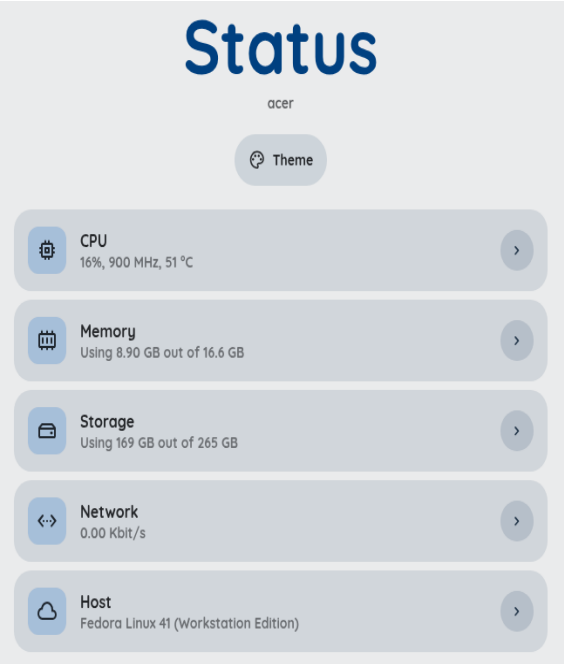


Рисунок 4.4 – Головне меню вебзастосунку

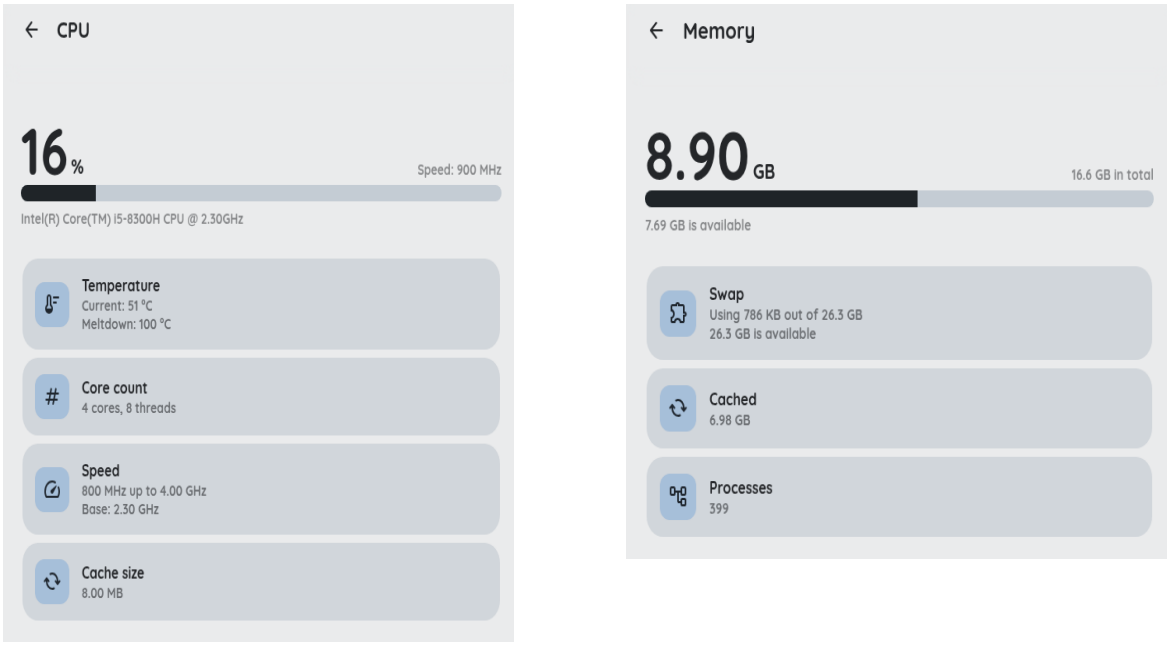
На рисунку 4.5 представлено компоненти вебінтерфейсу, зокрема завантаження процесору та обсяг операційної пам’яті.

Для забезпечення оновлення даних в режимі реального часу використовується AJAX. JavaScript-функції періодично відправляють запити до API додатку, отримують актуальні дані в форматі JSON та оновлюють відповідні елементи інтерфейсу без перезавантаження сторінки (лістинг 5.5).

Лістинг 4.5 – JavaScript-функції для оновлення даних

```
async function fetchData(endpoint) {
    const response = await fetch(endpoint);
    return response.json();
}
async function updateUI() {
    const cpuData = await fetchData('/api/cpu');
    document.getElementById('cpu-usage').textContent = `CPU
Usage: ${cpuData.usage}%`;
    const memoryData = await fetchData('/api/memory');
    document.getElementById('memory-usage').textContent =
`Memory Usage: ${memoryData.used} / ${memoryData.total} MB`;
    const diskData = await fetchData('/api/disk');
    document.getElementById('disk-usage').textContent = `Disk
Usage: ${diskData.used} / ${diskData.total} GB`;
    const networkData = await fetchData('/api/network');
    document.getElementById('network-activity').textContent =
```

```
`Download: ${networkData.download} KB/s, Upload:
${networkData.upload} KB/s`;
}
setInterval(updateUI, 5000);
updateUI();
```



а) б)

Рисунок 4.5 – компоненти вебінтерфейсу: а)завантаження процесору; б) обсяг оперативної пам’яті

Для реалізації функцій моніторингу та базової оптимізації роботи операційної системи було розроблено програмний модуль у вигляді класу Optimizer. Цей клас забезпечує контроль за станом критично важливих системних компонентів, зокрема CPU, оперативної пам’яті та мережної активності. Ініціалізація параметрів відбувається в конструкторі класу (лістинг 4.6), де встановлюються порогові значення для кожної метрики моніторингу, а також задається інтервал перевірки. На рисунку 4.6 наведено логічну схему роботи оптимізатора.

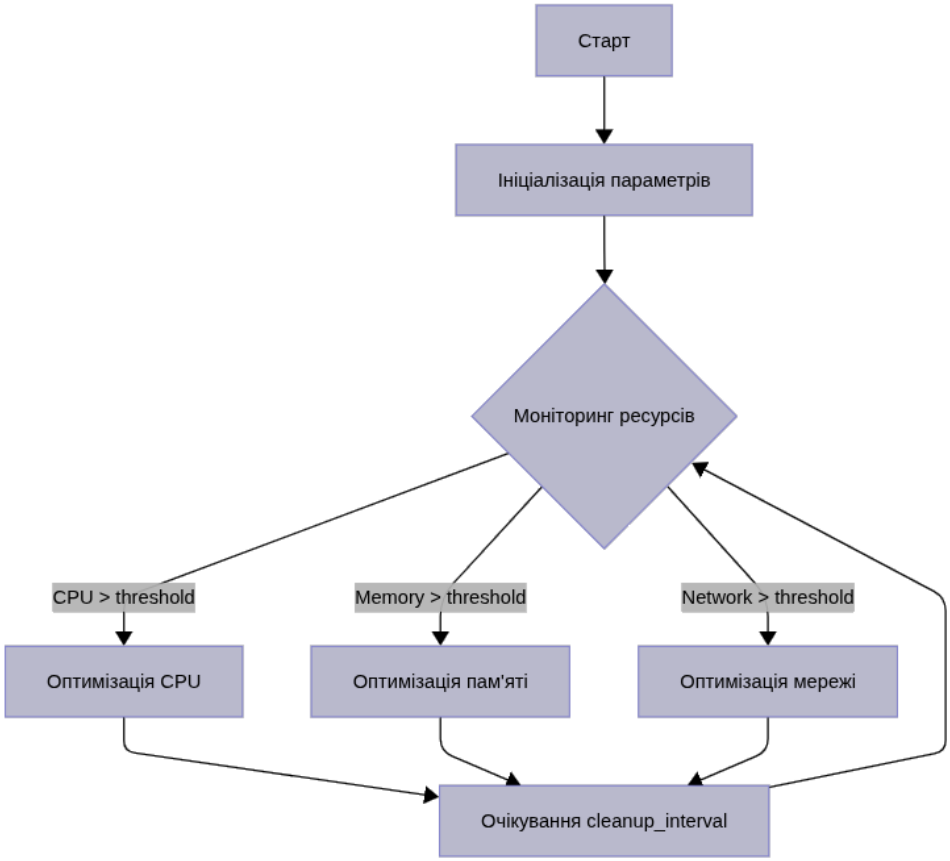


Рисунок 4.6 – Логічна схема роботи оптимізатора

Асинхронна реалізація гарантує, що під час збору інформації система не блокується, забезпечуючи стабільність та швидкодію.

Метод check_cpu() поточне завантаження CPU. В разі перевищення встановленого порогу, активується процедура оптимізації (лістинг 4.6).

Лістинг 4.6 – Перевірка рівня завантаженості CPU

```
async def check_cpu(self):
    cpu_usage = psutil.cpu_percent(interval=1)
    if cpu_usage > self.cpu_threshold:
        print(f"Виявлено високе навантаження CPU: {cpu_usage}%.
Виконуються дії...")
        self.optimize_cpu()
```

Метод optimize_cpu() перебирає активні процеси та намагається завершити ті, які мають незначне споживання ресурсів, що дозволяє знизити навантаження на систему (лістинг 4.7).

Лістинг 4.7 – Оптимізація використання CPU

```
def optimize_cpu(self):
    for proc in psutil.process_iter(['pid', 'name',
'cpu_percent']):
        if proc.info['cpu_percent'] < 1:
            try:
                proc.terminate()
                print(f"Завершено процес {proc.info['name']}
(PID: {proc.info['pid']})")
            except psutil.AccessDenied:
                print(f"Відмовлено у доступі для завершення
процесу {proc.info['name']}")
```

Метод check_memory використовує psutil.virtual_memory() для отримання даних про використання оперативної пам’яті. У разі перевищення порогу активується процедура очищення кешу (лістинг 4.8)

Лістинг 4.8 – Перевірка використання пам’яті

```
async def check_memory(self):
    memory = psutil.virtual_memory()
    if memory.percent > self.memory_threshold:
        print(f"Виявлено високе використання пам'яті:
{memory.percent}%. Виконуються дії...")
        self.optimize_memory()
```

Очищення кешу виконується шляхом запуску системної команди, що наведено в лістингу 4.9.

Лістинг 4.9 – Очищення кешу пам’яті

```
def optimize_memory(self):
    os.system("sync; echo 3 > /proc/sys/vm/drop_caches")
    print("Кеш пам'яті очищено.")
```

Моніторинг мережної активності здійснюється за допомогою методу check_network(), який порівнює загальний обсяг переданих та отриманих байтів з пороговим значенням, як наведено в лістингу 4.10.

Лістинг 4.10 – Оцінка мережного трафіку

```
async def check_network(self):
```

```
    net_io = psutil.net_io_counters()
    if net_io.bytes_sent + net_io.bytes_recv >
self.network_threshold:
        print(f"Виявлено великий мережевий трафік. Виконуються
дії...")
        self.optimize_network()
```

4.2 Оцінювання функціональності та ефективності вебзастосунку

Для комплексного аналізу функціональності розробленого додатку було проведено тестування на різноманітних апаратних платформах (таблиця 4.1). Основними завданнями тестування були перевірка точності обробки даних, оцінка швидкодії, аналіз впливу на системні ресурси, а також порівняння з іншими засобами моніторингу.

На кожному з пристроїв проводилася оцінка коректності функціонування, точності результатів та стабільності роботи в умовах підвищеного навантаженням. Узагальнена схема реалізації тестування наведена на рисунку 4.7.

Для верифікації точності показників додатку результати порівнювалися з вбудованими утилітами моніторингу, зокрема htop, free, df, iftop. Всі ключові метрики продемонстрували високу відповідність, з відхиленням не більше ніж 1–2% від значень, зафіксованих іншими інструментами.

Таблиця 4.1 – Апаратні конфігурації, використані для тестування вебзастосунку

Тип системи	Операційна система	Процесор	Оперативна пам’ять	Накопичувач
Серверна система	Fedora 41	8-ядерний x86_64	16 ГБ	SSD
Одноплатний комп’ютер	Raspberry Pi OS / Linux	4-ядерний ARM (Raspberry Pi 4)	4 ГБ	microSD
Віртуальна машина	Linux	2 ядра (x86_64)	2 ГБ	Віртуальний диск

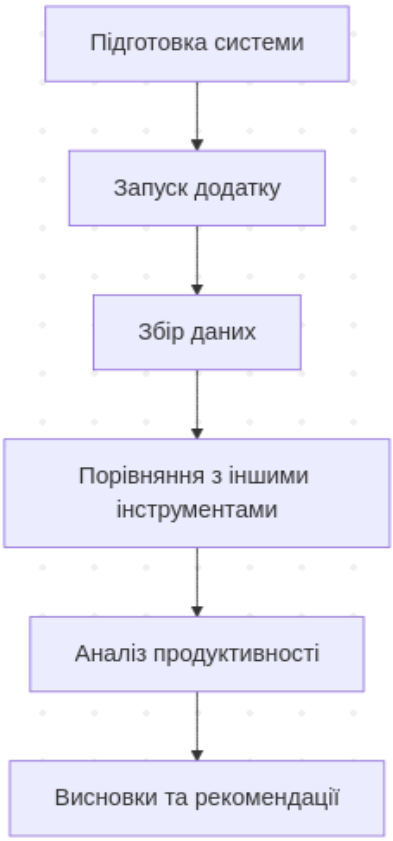


Рисунок 4.7 – Узагальнена схема процесу тестування

Під час тестування вебзастосунку було проаналізовано вплив додатку на системні ресурси. Зокрема, досліджувалося використання процесора, обсяг оперативної пам’яті та мережне навантаження під час активної роботи додатку. Результати тестування наведені в таблиці 4.2.

Таблиця 4.2 – Споживання системних ресурсів додатком під час тестування

Система	Середнє навантаження на CPU	Середнє споживання RAM	Примітка
Fedora 41	1–2 %	≈50 МБ	Помірне навантаження, характерне для серверних систем
Raspberry Pi 4	3–5 %	≈60 МБ	Прийнятне для обмежених за ресурсами пристроїв
Віртуальна машина	2–3 %	≈55 МБ	Низьке навантаження при роботі в обмеженому середовищі

При тестуванні розробленого додатку було проведено порівняльний аналіз функціонування вебзастосунку з іншими засобами моніторингу, зокрема Netdata. За результатами аналізу було встановлено, що додаток має значно нижчі показники споживання ресурсів. Зокрема, Netdata на аналогічних системах споживав близько 5–7% CPU та ≈150 МБ RAM.

Також в процесі тестування було виявлено недоліки розробленого додатку. На пристроях з низькою продуктивністю, зокрема Raspberry Pi спостерігалися періодичні затримки оновлення даних. Іншим недоліком є відсутність механізму автентифікації API, що підвищує ризики несанкціонованого доступу та витоку інформації.

В результаті тестування додаток демонструє високу стабільність, точність та ефективність на різних платформах. Він є придатним для використання на серверних системах та на пристроях з обмеженими апаратними ресурсами. Виявлені недоліки мають локальний характер і можуть бути усунені в процесі подальшого розвитку програмного продукту. Загальні результати тестування наведені в таблиці 4.3.

З метою оцінювання ефективності роботи серверної частини застосунку було розроблено спеціалізований модуль моніторингу, що інтегрується безпосередньо в бекенд. Цей модуль забезпечує збір ключових показників продуктивності в режимі реального часу без втручання в логіку основного коду.

З архітектурної точки зору, модуль побудований на основі класу BackendEfficiency, який відповідає за накопичення статистичних даних. Клас містить лічильник HTTP-запитів, сумарний час їх обробки та мітку запуску сервера. Для забезпечення потокобезпечного доступу до спільних змінних використовується механізм блокування threading.Lock.

Дані про кожен HTTP-запит фіксуються за допомогою проміжного програмного забезпечення middleware, яке обчислює час обробки запиту та передає його у відповідний метод класу record_request(). Зібрана інформація агрегується, а узагальнені метрики доступні через метод get_metrics(), який повертає поточні значення визначених параметрів, зокрема загальний час

безперервної роботи сервера, кількість оброблених запитів, середній час відповіді, рівень завантаження CPU, використання оперативної пам’яті. Спрощена схема взаємодії модуля для збору визначених метрик представлена на рисунку 4.8.

Таблиця 4.3 – Результати тестування вебзастосунку на різних апаратних платформах

Система	CPU (середнє)	RAM	Затримка оновлення	Стабільність	Порівняння з Netdata
Fedora 41	1–2%	50 МБ	<1 секунди	Висока	Менше споживання ресурсів
Raspberry Pi 4	3–5%	60 МБ	~2 секунди	Висока	Менше споживання ресурсів
Віртуальна машина	2–3%	55 МБ	< 1 секунда	Висока	Менше споживання ресурсів

Для збору системної інформації використовується бібліотека psutil. В лістингу 4.11 наведено опис класу BackendEfficiency, який призначений для накопичення статистичних даних.

Лістинг 4.11 – Клас BackendEfficiency

```
class BackendEfficiency:
    def __init__(self):
        self.request_count = 0
        self.total_response_time = 0.0
        self.lock = threading.Lock()
        self.start_time = time.time()
    def record_request(self, response_time):
        with self.lock:
            self.request_count += 1
            self.total_response_time += response_time
    def get_metrics(self):
        uptime = time.time() - self.start_time
        avg_response = (self.total_response_time /
self.request_count) if self.request_count else 0
        cpu = psutil.cpu_percent()
```

```
        mem = psutil.virtual_memory().percent
        return {
            "uptime_sec": uptime,
            "requests": self.request_count,
            "avg_response_time_ms": avg_response * 1000,
            "cpu_percent": cpu,
            "memory_percent": mem
        }
```

Рисунок 4.8 – Спрощена схема взаємодії модуля для збору метрик

Інтеграція модуля в серверну частину реалізована за допомогою middleware-компонента, що перехоплює всі HTTP-запити, вимірює час їх виконання та передає ці дані до відповідного методу класу BackendEfficiency (лістинг 4.12).

Лістинг 4.12 – Middleware-компонент для збору статистики

```
@web.middleware
async def efficiency_middleware(request, handler):
    start = time.time()
```

```
    response = await handler(request)
    elapsed = time.time() - start
    backend_efficiency.record_request(elapsed)
    return response
```

Для отримання зібраних метрик створено окремий HTTP endpoint /api/backend_efficiency, що повертає узагальнені дані в форматі JSON. Це дозволяє легко інтегрувати модуль з зовнішніми системами моніторингу або відображати дані у вебінтерфейсах (лістинг 4.13).

Лістинг 4.13 – Endpoint для отримання метрик

```
@routes.get("/api/backend_efficiency")
async def api_backend_efficiency(request):
    return web.json_response(backend_efficiency.get_metrics())
```

Для перевірки функціональної коректності модуля збору метрик продуктивності було проведено комплексне тестування на сервері зі стандартною апаратною конфігурацією, зокрема 4 ядра CPU та 4 ГБ оперативної пам’яті. Тестування охоплювало декілька сценаріїв навантаження з метою визначення швидкодії серверної частини за стосунку, зокрема часу обробки HTTP-запитів, навантаження на центральний процесор та обсягу використаної оперативної пам’яті під час функціонування системи.

Під час тестування до API бекенду надсилалися HTTP-запити з варіативною інтенсивністю, а спеціалізований модуль BackendEfficiency автоматично накопичував статистичні дані. Для доступу до узагальнених показників використовувався ендпоінт /api/backend_efficiency, який повертав метрики в форматі JSON. Крім того, тестувалися /api/status для отримання загальної інформації про систему та /api/processes для отримання списку запущених процесів, що дозволило оцінити вплив різних типів запитів на загальну продуктивність серверної частини. Отримана відповідь від модуля після 10 хвилин роботи при середньому рівні навантаження наведене в лістингу 4.14.

Лістинг 4.14 – Відповідь endpoint /api/backend_efficiency

```
{
    "uptime_sec": 600.5,
    "requests": 3000,
    "avg_response_time_ms": 18.7,
    "cpu_percent": 7.5,
    "memory_percent": 32.1
}
```

Для аналізу ефективності функціонування модуля було проведено тестування з різною інтенсивністю навантаження. В таблиці 4.4 наведено результати ефективності бекенду при різному навантаженні.

Таблиця 4.4 – Результати тестування ефективності бекенду при різному навантаженні

Кількість запитів	Середній час відповіді, мс	CPU, %	Пам’ять, %
600	15.2	3.1	30.5
3000	18.7	7.5	32.1
15000	27.4	18.9	35.8

Додатково було виконано тестування окремих ендпоінтів з метою оцінки їх впливу на загальну швидкодію системи. Зокрема, було проаналізовано середній час відповіді для запитів до /api/status, /api/processes та /api/backend_efficiency за умов різного навантаження. Результати тестування наведено в таблиці 4.5.

Проведений аналіз свідчить про стабільну роботу модуля збору метрик в режимі високого навантаження. Ендпоінт /api/backend_efficiency має найменший час відповіді, що підтверджує ефективність його реалізації та доцільність використання для моніторингу.

Таблиця 4.5 – Результати тестування ендпоінтів

Ендпоінт	Низьке навантаження, мс	Середнє навантаження, мс	Високе навантаження, мс
/api/status	16.1	19.3	29.2
/api/processes	14.8	17.5	25.7
/api/backend_efficiency	2.3	2.7	3.5

В цьому розділі було здійснено повний цикл розробки вебзастосунку, що включає розробку архітектури, реалізацію серверної логіки, створення API та інтеграцію інструментів моніторингу продуктивності. Основну увагу приділено забезпеченню модульності, розширюваності та стабільності системи. Для збору системної інформації реалізовано окремий програмний компонент, зокрема модуль BackendEfficiency, який дозволяє оперативно отримувати ключові метрики продуктивності в форматі JSON через відповідні API-ендпоінти. Застосовано сучасні підходи до роботи з багатопоточністю, управління процесами та оптимізації використання ресурсів, що сприяло зниженню середнього часу відповіді в умовах високого навантаження.

Отримані результати демонструють, що розроблений застосунок є ефективним, гнучким у налаштуванні та придатним до використання в середовищах з підвищеними вимогами до моніторингу та обробки запитів в режимі реального часу.

ВИСНОВКИ

В кваліфікаційній роботі здійснено всебічний аналіз сучасних методів моніторингу та оптимізації функціонування операційних систем. Основну увагу зосереджено на використанні вбудованих інструментів системного контролю, а також на концептуальних підходах до збору, обробки та інтерпретації метрик продуктивності в режимі реального часу.

В рамках кваліфікаційної роботи розроблено метод підвищення ефективності функціонування операційної системи в умовах інтенсивного навантаження та обмежених апаратних ресурсів. Запропонований підхід базується на автоматизації управління ключовими системними ресурсами та впровадженні оптимізаційних заходів, що реалізуються на основі результатів моніторингу та діагностики за допомогою вбудованих утиліт операційної системи.

Результати проведеного аналізу підтвердили, що запропонований метод дозволяє своєчасно виявляти та локалізувати потенційні збої у функціонуванні системи, мінімізувати ризики надмірного використання апаратних ресурсів та забезпечити стабільність та безперервність роботи серверної інфраструктури в умовах підвищеного навантаження.

Отримані результати можуть бути використані для побудови адаптивних систем моніторингу та оптимізації функціонування серверних операційних систем в корпоративному середовищі. Застосування розробленого підходу є доцільним для забезпечення стабільної роботи критично важливих програмних сервісів в режимі реального часу з урахуванням обмежень обчислювальних ресурсів та вимог до надійності та відмовостійкості інфраструктури корпоративних комп’ютерних мереж.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lee, C., Kim, D., & Choi, J. Intelligent Operating System Monitoring Using Machine Learning Techniques. Journal of Systems Architecture, 117, 2021p.

2. Zhang, Y., Wang, H., & Liu, Z. (2020). Adaptive Monitoring Framework for Cloud–Based Operating Systems. Future Generation Computer Systems, 105, 156–168, 2020 p.

3. Richardson, C. Microservices Observability. 1st ed. Sebastopol : O'Reilly, 2023. 314 p.

4. Burns, B., Grant, B., Oppenheimer, D. Kubernetes: Up and Running. 3rd ed. Sebastopol : O'Reilly Media, 2022. 390 p.

5. Gregg, B. Systems Performance: Enterprise and the Cloud. 2nd ed. Upper Saddle River : Addison–Wesley, 2022. 928 p.

6. Goasguen, S. Docker Cookbook. 2nd ed. Birmingham : Packt Publishing, 2023. 430 p.

7. Turnbull, J. The Art of Monitoring. 1st ed. Melbourne : Turnbull Press, 2021. 398 p.

8. Dobbelaere, M., Esmaili, H. Monitoring and Observability in DevOps. 1st ed. London : Apress, 2023. 265 p.

9. Sharma, R. Hands–On Linux Administration on Azure. 2nd ed. Birmingham : Packt Publishing, 2023. 502 p.

10. Papaioannou, C.; Dimara, A.; Papaioannou, A.; Tzitzios, I.; Anagnostopoulos, C.–N.; Krinidis, S. Hierarchical Resources Management System for Internet of Things–Enabled Smart Cities. Sensors 2025, 25, 616. https://doi.org/10.3390/s25030616

11. Singh, N., Hamid, Y., Juneja, S. et al. *Load balancing and service discovery using Docker Swarm for microservice based big data applications*. J Cloud Comp 12, 4 (2023). https://doi.org/10.1186/s13677–022–00358–7

12. Apriani, D., Afrijaldi, R., Auliya, N., & Darmawan, A. A. (2024). Operating System and Server Integration for Business Effectiveness. IAIC Transactions on Sustainable Digital Innovation (ITSDI), 5(2), 91–99. https://doi.org/10.34306/itsdi.v5i2.615

13. Gill, S. S., Xu, M., Ottaviani, C., Patros, P., Bahsoon, R., Shaghaghi, A., . Uhlig, S. (2022). AI for next generation computing: Emerging trends and future directions. Internet of Things, 19, 100514.

14. Umer, M. (2023). Architecture and Design of the Linux Storage Stack. Packt Publishing Ltd.

15. Davis, J. (2022). Modern System Administration. O'Reilly Media, Inc.

16. Love, R. Linux Kernel Development. 4th ed. Boston : Addison–Wesley, 2022. 496 p.

17. Bovet, D. P., Cesati, M. Understanding the Linux Kernel. 4th ed. Sebastopol : O’Reilly, 2022. 944 p.

18. Maher, R. Modern System Administration. 1st ed. Sebastopol : O’Reilly, 2022. 320 p.

19. Cain, D. Resource Optimization for Linux Servers. 1st ed. Birmingham : Packt Publishing, 2023. 366 p.

20. Hoskins, W. Linux Performance Observability. 1st ed. New York : No Starch Press, 2022. 384 p.

21. Hoskins, W. Linux Performance Observability. 1st ed. New York : No Starch Press, 2022. 384 p.

22. Beyer, B. et al. Site Reliability Engineering: How Google Runs Production Systems. Updated ed. Sebastopol : O’Reilly, 2022. 552 p.

23. Verma, S. Optimizing Linux Performance. 1st ed. New York : Apress, 2022. 350 p.

24. Gunther, N. J. Analyzing Computer System Performance with Perl::PDQ. 2nd ed. Heidelberg : Springer, 2022. 490 p.

25. Ross, J. Linux Command Line and Shell Scripting Bible. 5th ed. Indianapolis : Wiley, 2022. 912 p.

26. Tomsho, G. Guide to Operating Systems. 6th ed. Boston : Cengage Learning, 2023. 688 p.

27. Mircea, A. Advanced System Monitoring with Prometheus. 1st ed. Birmingham : Packt Publishing, 2022. 480 p.

28. Hasan, A. System Performance Tuning with BPF Tools. 1st ed. Sebastopol : O'Reilly, 2022. 342 p.

29. Harper, M. Practical Python System Administration. 1st ed. San Francisco : No Starch Press, 2022. 360 p.

30. Bhatt, R. Windows Performance Analysis Field Guide. 2nd ed. Redmond : Microsoft Press, 2022. 560 p.

31. Boutet, A. Real-Time Analytics with Grafana and Prometheus. 2nd ed. Birmingham : Packt Publishing, 2023. 330 p.

32. Mircea, A. Advanced System Monitoring with Prometheus. 1st ed. Birmingham : Packt Publishing, 2022. 480 p.

33. Allen, J. The DevOps Handbook. 3rd ed. Boston : IT Revolution, 2023. 480 p.

34. Zabbix Team. Zabbix 6.0 Network Monitoring Essentials. 1st ed. Riga : Zabbix SIA, 2022. 395 p.

35. Chan, G. Network Performance Monitoring and Troubleshooting. 1st ed. London : Apress, 2022. 332 p.

36. Quigley, E. Principles of Computer Security. 6th ed. New York : McGraw-Hill, 2023. 820 p.

37. Langley, J. Security Monitoring with OSSEC. 2nd ed. San Francisco : No Starch Press, 2023. 344 p.

38. Kim, M. Log Management and Analytics. 1st ed. San Francisco : No Starch Press, 2022. 370 p

39. Peñalvo, F. J. G., Sharma, A., Chhabra, A., Singh, S. K., Kumar, S., Arya, V., & Gaurav, A. (2022). Mobile cloud computing and sustainable development: Opportunities, challenges, and future directions. International Journal of Cloud Applications and Computing (IJCAC), 12(1), 1-20.

40. Arogundade, O. R., & Palla, K. (2023). Virtualization revolution: Transforming cloud computing with scalability and agility.

41. Apriani, D., Afrijaldi, R., Auliya, N., & Darmawan, A. A. (2024). Operating System and Server Integration for Business Effectiveness. IAIC Transactions on Sustainable Digital Innovation (ITSDI), 5(2), 91–99. https://doi.org/10.34306/itsdi.v5i2.615

42. Шостак, А. Р., Ткачов, В. М. (2023). Питання оптимізації бізнес-процесів з використанням сучасних інформаційних технологій.

43. Duin, J., McKeown, S., & Abubakar, M. (2024, June). Exploring DTrace as an Incident Response Tool for Unix Systems. In The International Conference on Cybersecurity, Situational Awareness and Social Media (pp. 169-193). Singapore: Springer Nature Singapore.

44. Чепурна, І.С., Шевченко І.О. (2025). Метод підвищення ефективності функціонування Unix-подібних систем. Вісник Херсонського Національного Технічного Університету, №2/2025 (прийнята до друку 27.05.2025).