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 перший (бакалаврський)

Пристрій дистанційного моніторингу та керування ресурсами комп’ютера на базі мікроконтролера ESP32
(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-6
Олександр САВЧЕНКО
(власне ім’я, прізвище)

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

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

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Володимир АРГУНОВ
(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 перший (бакалаврський)

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

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

Освітня програма Комп'ютерна інженерія
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Савченку Олександру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Пристрій дистанційного моніторингу та керування ресурсами комп'ютера на базі мікроконтролера ESP32

затверджена наказом по університету від " 26 " травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) мікроконтролер: ESP32; 2) стек протоколів: TCP/IP, TLS; 3) дані для моніторингу: завантаження процесора, пам'яті, мережі, дисків, список процесів

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз проблеми та огляд існуючих рішень
2) вибір технології розробки та інструментальних засобів
3) розробка серверної частини
4) розробка клієнтської частини
5) відлагодження проєкту
6) висновки

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

Слайд-презентація – 32 слайди

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
1	Аналіз проблеми та огляд існуючих рішень	27.05.25 - 30.05.25	Виконано
2	Вибір технології розробки та інструменталь-	31.05.25 - 02.06.25	Виконано
	них засобів		
3	Розробка серверної частини	03.06.25 – 05.06.25	Виконано
4	Розробка клієнтської частини	06.06.25 – 09.06.25	Виконано
5	Відлагодження проєкту	10.06.25 – 11.06.25	Виконано
6	Оформлення матеріалів кваліфікаційної роботи	12.06.25 – 13.06.25	Виконано
7	Подання кваліфікаційної роботи керівникові та	14.06.25 – 15.06.25	Виконано
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	15.06.25 – 17.06.25	Виконано
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____ (підпис)

Керівник роботи _____ (підпис) ас. Володимир АРГУНОВ (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 102 с., 36 рис., 1 табл., 4 дод., 13 джерел.

ESP32, МІКРОКОНТРОЛЕР, МОНІТОРИНГ, КЕРУВАННЯ СИСТЕМОЮ, CPU, TLS, WI-FI, МЕРЕЖА, КЛІЄНТ, СЕРВЕР, ВІДДАЛЕНЕ КЕРУВАННЯ, ПРОЦЕСИ, ЗАХИСТ ДАНИХ, ДЕСКТОПНИЙ ЗАСТОСУНОК.

Метою кваліфікаційної роботи є використання мікроконтролера ESP32 для розробки пристрою для збору даних про навантаження комп'ютера(відсоток завантаженості процесора, пам'яті, завантаження дисків та мережі, список процесів виконуваних на комп'ютері) та їх надсилання на десктопний додаток, а також можливість дистанційного завершення процесів, керування живленням, запуску скриптів.

У ході виконання кваліфікаційної роботи було розроблено програмно-апаратний комплекс для дистанційного моніторингу та керування комп'ютерною системою з використанням мікроконтролера ESP32. Система забезпечує збір ключових параметрів стану комп'ютера(Навантаження окремих компонентів, наприклад CPU, пам'ять, дисковий простір і т.д), їх обробку та передачу до клієнтського застосунку в реальному часі, а також реалізує можливість керування процесами та живленням з боку користувача. Клієнт-серверна комунікація використовує TLS для створення захищеного каналу зв'язку. Отримане рішення працює автономно і незалежно від системи і має систему сповіщень користувача у разі виявленя збоїв роботи системи, знеструмлення цільової обчислювальної машини або аномального використання ресурсів.

ABSTRACT

Bachelor’s thesis: 102 pages, 36 figures, 1 tables, 4 appendices, 13 sources.

ESP32, MICROCONTROLLER, MONITORING, SYSTEM MANAGEMENT, CPU, TLS, WI-FI, NETWORK, CLIENT, SERVER, REMOTE MANAGEMENT, PROCESSES, DATA PROTECTION, DESKTOP APPLICATION.

The major goal of this thesis is to use the ESP32 microcontroller to develop a device for collecting data on the computer load (percentage of processor utilization, memory, disk and network load, list of processes running on the computer) and sending them to a desktop application, as well as the ability to remotely terminate processes, power management, and run scripts.

In order to achieve this goal a hardware-software complex was developed for remote monitoring and management of a computer system using the ESP32 microcontroller. The system provides collection of key parameters of the computer state (load of individual components, such as CPU, memory, disk space, etc.), their processing and transmission to the client application in real time, and also implements the possibility of controlling processes and power supply by the user. Client-server communication uses TLS to create a secure communication channel. The resulting solution works autonomously and independently of the system and has a user notification system in case of system failures, power outage of the target computer or abnormal resource use.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОЗП – оперативний запам’ятовуючий пристрій

ОС – операційна система

IP – інтернет протокол (англ., Internet Protocol)

SSID – ідентифікатор набору послуг (англ., Service Set Identifier)

TCP – протокол керування передачею (англ., Transmission Control Protocol)

UART – універсальний асинхронний приймач-передавач (англ., Universal Asynchronous Receiver-Transmitter)

UDP – протокол користувацьких дейтаграм(англ., User Datagram Protocol)

ВСТУП

Моніторинг та керування ресурсами комп’ютера – одна з невід’ємних частин користування сучасними обчислювальними машинами. Окрім можливості виявлення проблем(наприклад позаштатної поведінки застосунків, витоку пам’яті або нескінченних циклів) та оптимізації системи(завершення окремих застосунків, запуск скриптів для очищення місця на жорсткому диску, тощо) моніторинг та керування ресурсами комп’ютера забезпечують користувача необхідною інформацією, яку можна застосувати для планування ресурсів(наприклад апгрейд ОЗП або дисків) та економію ресурсів, що є важливим не лише для простих користувачів але й для адміністраторів великих дата-центрів. Більшість операційних систем мають у своїй стандартній конфігурації вбудовані інструменти для моніторингу, проте для їх застосування потрібен фізичний доступ до обчислювальної системи або налаштування віддаленого підключення. Це може викликати такі проблеми як затримки в реагуванні та необхідність постійного нагляду. Якщо проблему не виявити вчасно через відсутність доступу це може призвести до простоїв або втрати даних, а без автоматичного сповіщення адміністратору доводиться регулярно перевіряти стан системи вручну. Окрім цього, користування вбудованими інструментами моніторингу не дозволяє оцінювати стан комп’ютерної системи автономно, незалежно від стану самої обчислювальної машини, наприклад якщо операційна система зависла або відсутнє живлення.

Метою роботи є використання мікроконтролера ESP32 для розробки пристрою для збору даних про навантаження комп’ютера(відсоток завантаженості процесора, пам’яті, завантаження дисків та мережі, список процесів виконуваних на комп’ютері) та їх надсилання на десктопний додаток, а також можливість дистанційного завершення процесів, керування живленням, запуску скриптів. Для досягнення даної цілі необхідно вирішити

наступні завдання:

- здійснити огляд наявних технологій та принципів моніторингу та керування ресурсами комп’ютера;
- розробити прошивку для мікроконтролера ESP32, що буде збирати дані з цільового комп’ютера та надсилатиме їх до клієнтського застосунку;
- організувати комунікацію між цільовим комп’ютером та мікроконтролером;
- розробити клієнтський застосунок із зручним інтерфейсом для користування;
- перевірити працездатність рішення при різних умовах.

Об’єкт дослідження – процес розробки пристрою для дистанційного моніторингу та керування ресурсами комп’ютера на базі мікроконтроллера ESP32.

Предмет дослідження – використання мікроконтроллера ESP32 у контексті збору, обробки та пересилання даних між клієнтським та цільовим комп’ютерами.

Інформаційною базою досліджень є навчальна та методична література, державні стандарти і ресурси Інтернету, що надають відкритий доступ до даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд інструментів моніторингу ресурсів комп’ютера

Моніторинг комп’ютерних ресурсів — це процес безперервного спостереження за станом та використанням ключових компонентів комп’ютерної системи. До таких компонентів належать центральний процесор (CPU), оперативна пам’ять (RAM), дискова підсистема (HDD або SSD), мережеві інтерфейси, відеокарта (GPU), системні шини, а також інші апаратні й програмні ресурси.

Цей процес дозволяє виявити аномалії, контролювати продуктивність, аналізувати навантаження та своєчасно реагувати на потенційні загрози або збої у роботі системи. Моніторинг особливо важливий у випадках використання серверів, віртуальних машин, критичних інфраструктур або систем з високими вимогами до надійності й доступності. Таким чином, моніторинг комп’ютерних ресурсів є невід’ємною частиною забезпечення стабільної та ефективної роботи як окремих пристроїв, так і великих IT-інфраструктур.

Кожна операційна система (наприклад, Windows, Linux, macOS) надає власні вбудовані засоби моніторингу.

1.1.1 Інструменти моніторингу в ОС Windows

Performance Monitor(PerfMon) є стандартним засобом для моніторингу продуктивності системи. Він дозволяє відстежувати різноманітні лічильники, такі як завантаження процесора, використання пам’яті, активність дисків та мережі. Цей інструмент надає можливість створення графіків, збору даних у реальному часі та аналізу історичних даних, що є корисним для виявлення вузьких місць у системі (рисунок 1.1).

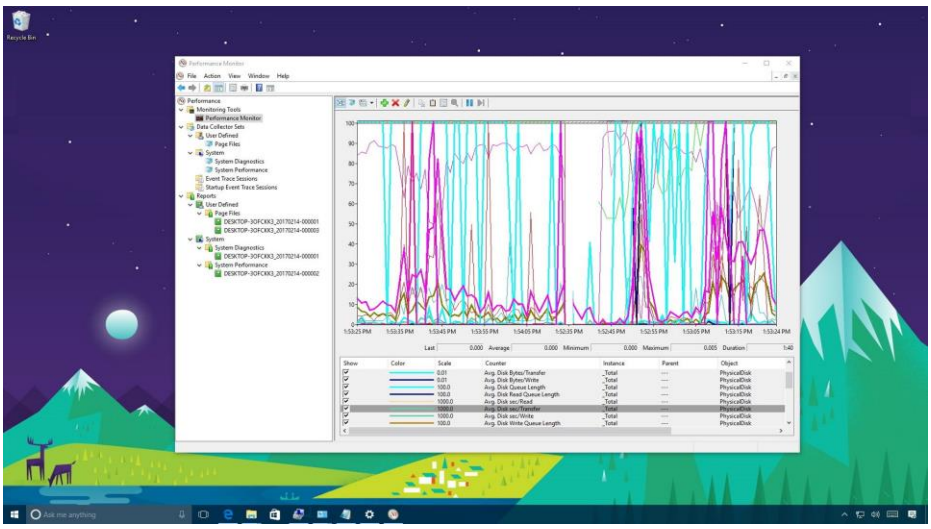

Рисунок 1.1 – Performance Monitor

Task Manager забезпечує швидкий огляд активних процесів, використання ресурсів та дозволяє завершувати непотрібні або завислі програми. Він також надає інформацію про запущені служби та їхній стан (рисунок 1.2).

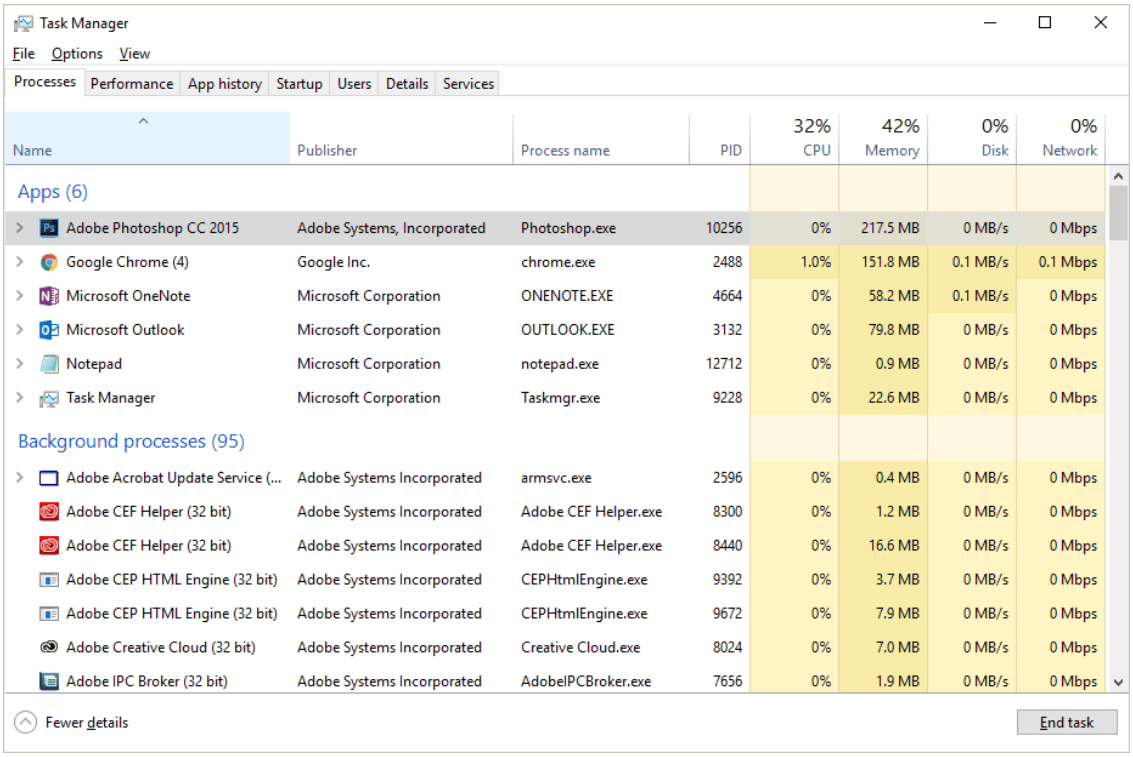

Рисунок 1.2 – Task Manager

1.1.2 Інструменти моніторингу в ОС Linux

top/htop є базовими утилітами для моніторингу процесів у реальному часі. Вони відображають інформацію про завантаження процесора, використання пам’яті та активні процеси. Htop має зручний інтерфейс та додаткові функції, такі як сортування процесів та управління ними (рисунок 1.3).

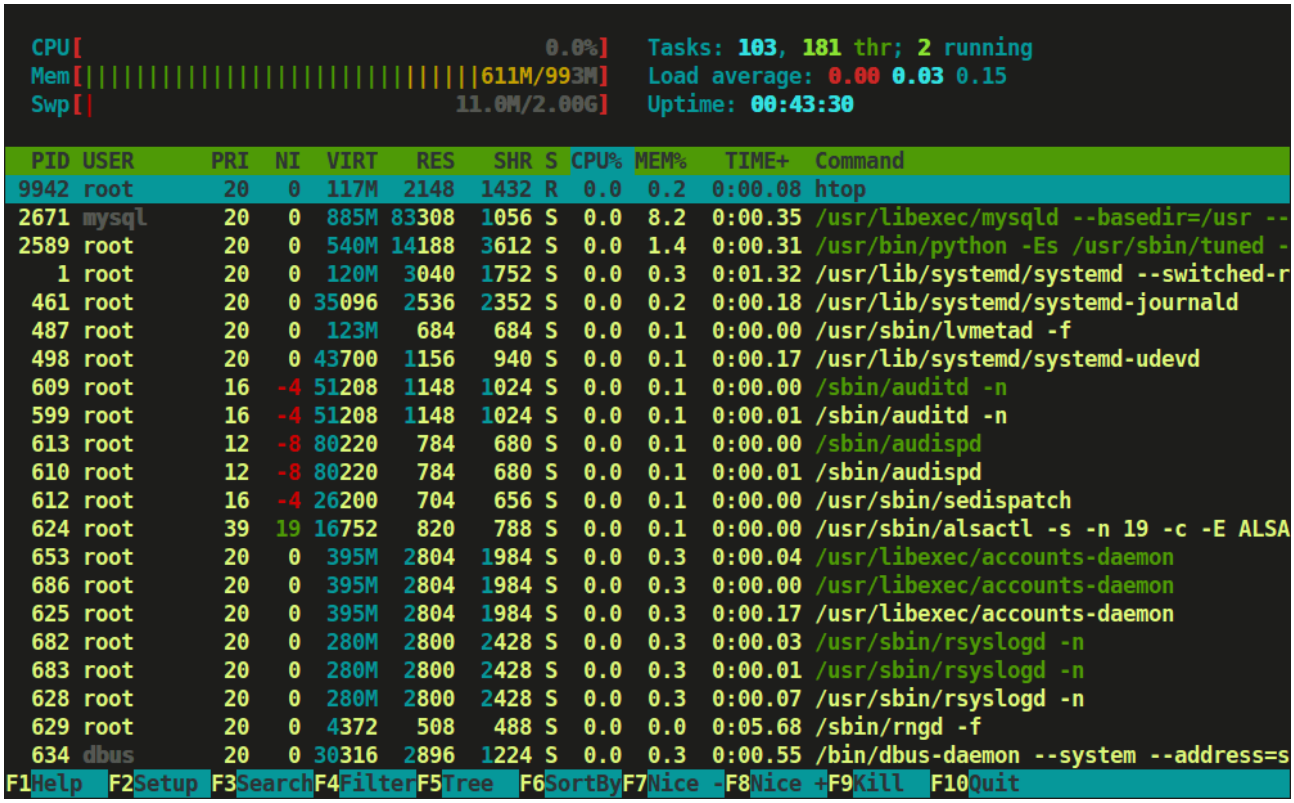

Рисунок 1.3 – Вивід команди htop

vmstat, iostat, netstat та інші утиліти надають детальну інформацію про стан пам’яті, вводу-виводу та мережевої активності відповідно. Ці інструменти є корисними для глибшого аналізу системних ресурсів.

1.1.3 Інструменти моніторингу в ОС macOS

Activity Monitor є стандартним інструментом macOS для моніторингу системних ресурсів. Він надає інформацію про використання процесора,

пам’яті, енергії, диску та мережі. Також дозволяє завершувати процеси та переглядати відкриті файли (рисунок 1.4).

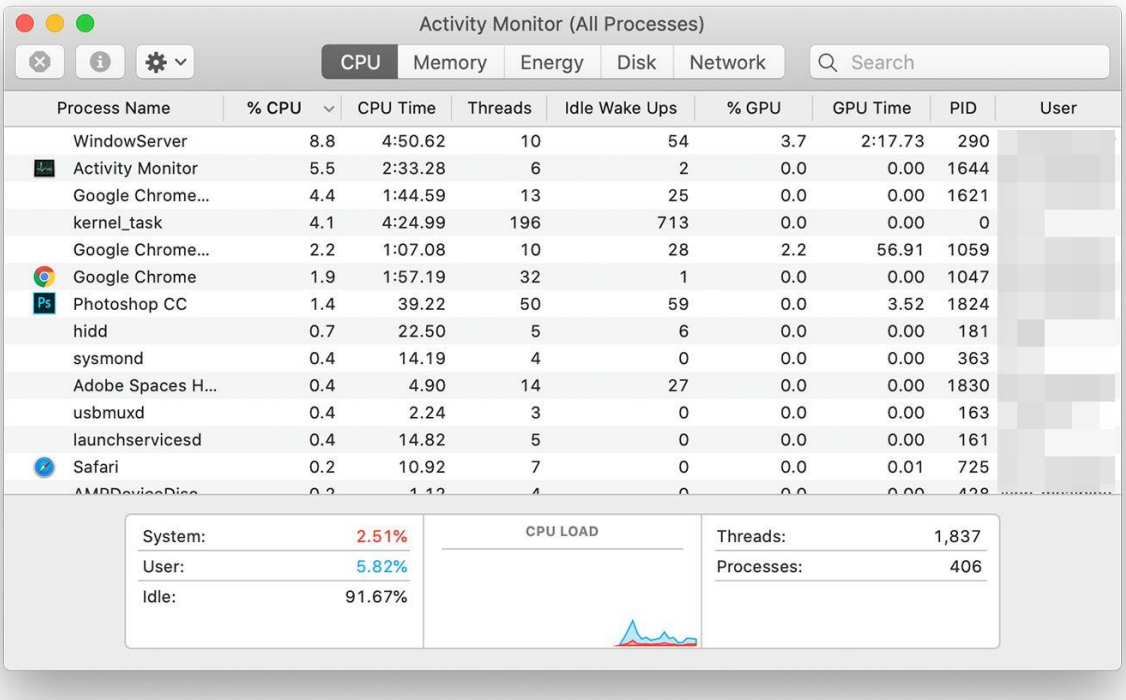

Рисунок 1.4 – Activity Monitor

Powermetrics є командною утилітою, яка надає детальну інформацію про енергоспоживання системи, включаючи використання процесора, GPU та інших компонентів (рисунок 1.5).

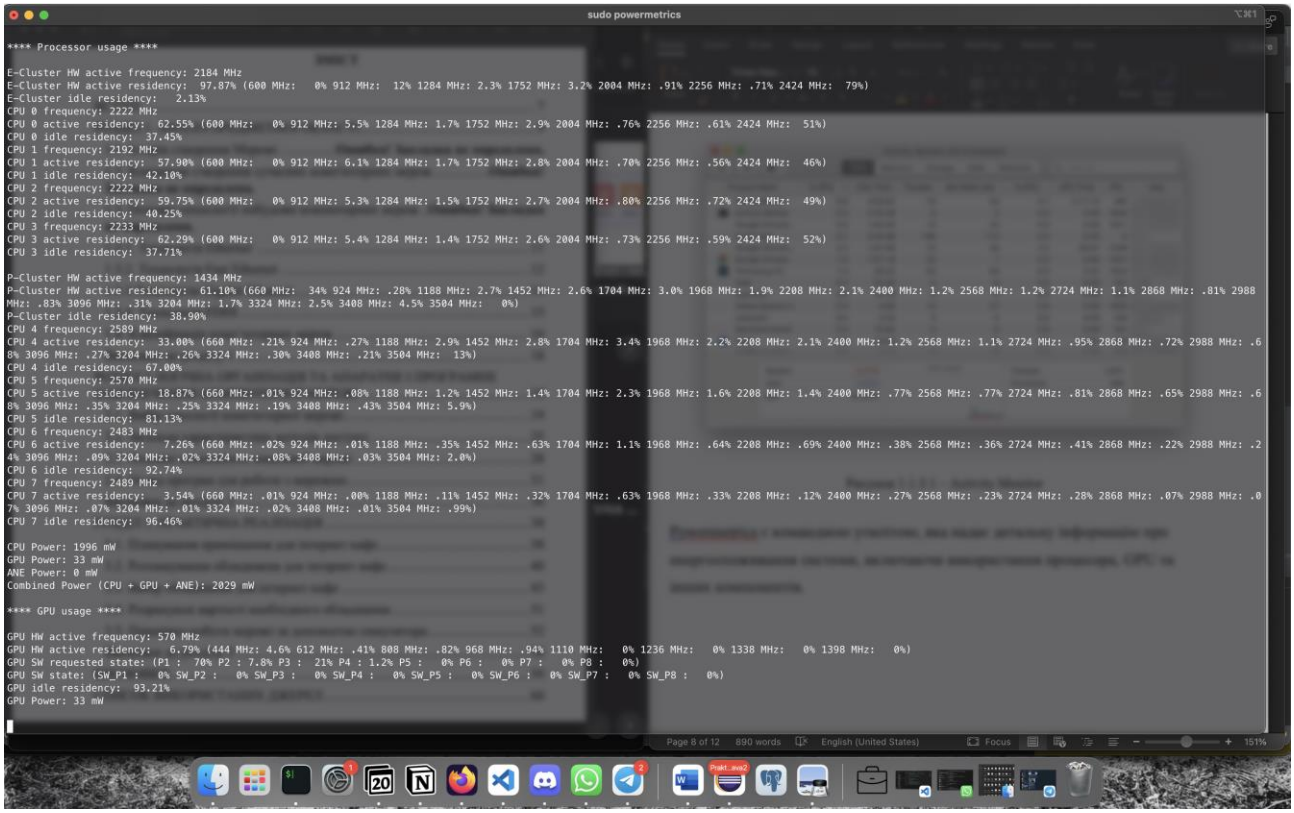

Рисунок 1.5 – powermetrics

1.2 Огляд інструментів дистанційного моніторингу

Дистанційний моніторинг(Remote Monitoring and Management, RMM) – це сукупність технологій та програмних рішень, що дозволяють IT-фахівцям та керівникам систем забезпечувати контроль, обслуговування та безпеку комп’ютерних систем, мереж і пристроїв без фізичної присутності на місці. Це особливо актуально в умовах зростаючої популярності віддаленої роботи та розподілених IT-інфраструктур.

До основних функцій RMM-систем належать:

- моніторинг стану систем у реальному часі;
- виконання віддалених дій для усунення пробле;
- передачі файлів та спрощення процесу усунення несправностей;
- забезпечення безперервного контролю за IT-операціями.[1]

До популяних інструментів RMM можна віднести NinjaOne, ConnectWise Automate і Atera.

NinjaOne – Це комплексна платформа для віддаленого моніторингу та управління, яка поєднує потужні функції з простим інтерфейсом. Серед ключових можливостей – автоматизоване управління оновленнями для Windows, Mac та сторонніх додатків, віддалений доступ у реальному часі, автоматизація сценаріїв та робочих процесів, а також вбудовані протоколи безпеки, що відповідають стандартам GDPR та HIPAA (рисунок.1.6).

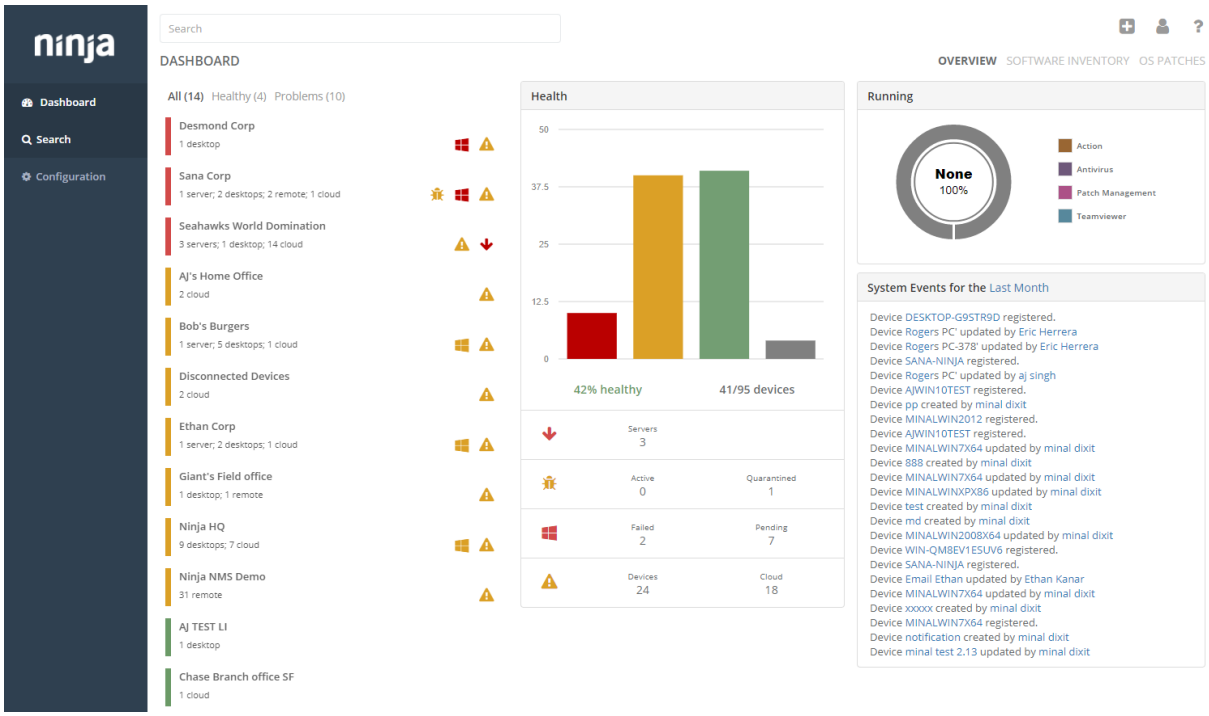

Рисунок 1.6 – Приклад роботи з NinjaOne

ConnectWise Automate надає можливість автоматизації рутинних завдань, моніторингу систем та управління оновленнями, що дозволяє зменшити навантаження на IT-персонал та підвищити ефективність роботи (рисунок 1.7).

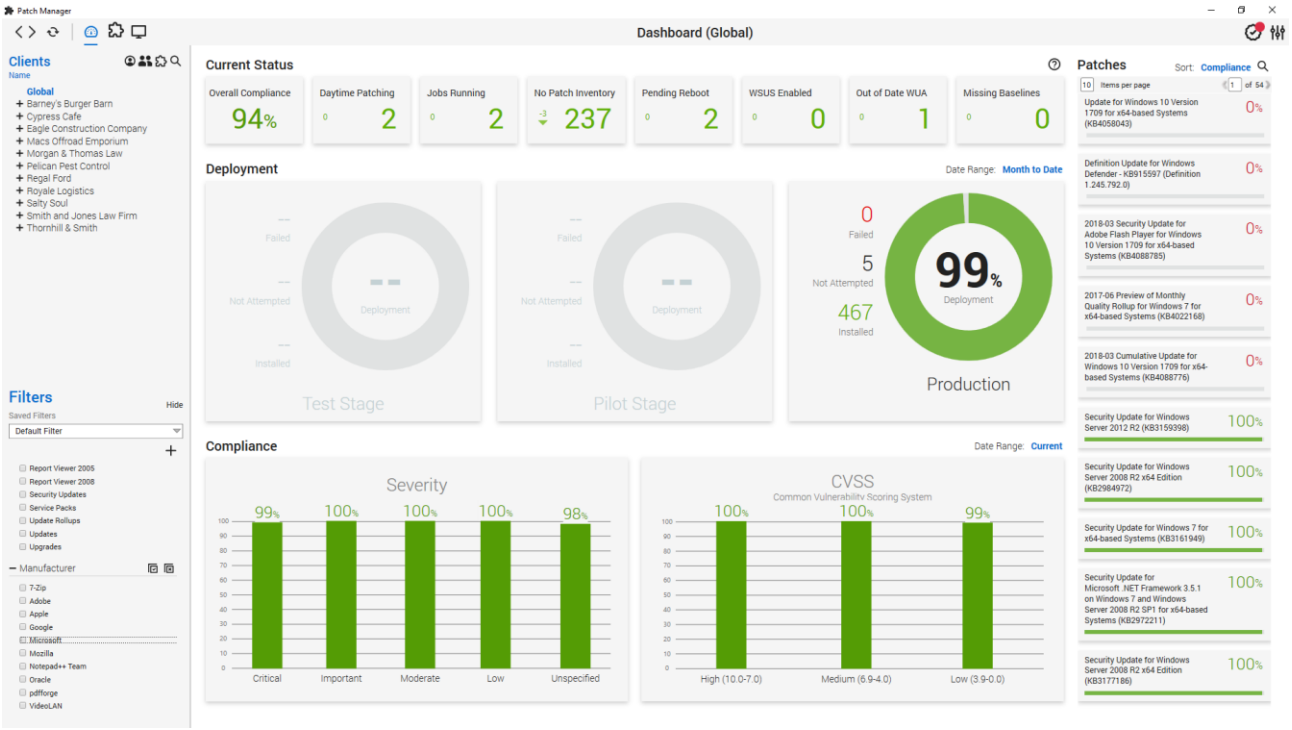

Рисунок 1.7 – Приклад роботи з ConnectWise Automate

Atera – це хмарна платформа, яка об’єднує функції RMM, Service Desk та управління активами, що дозволяє IT-фахівцям ефективно обслуговувати клієнтів та управляти їхніми системами з єдиного інтерфейсу (рисунок 1.8).

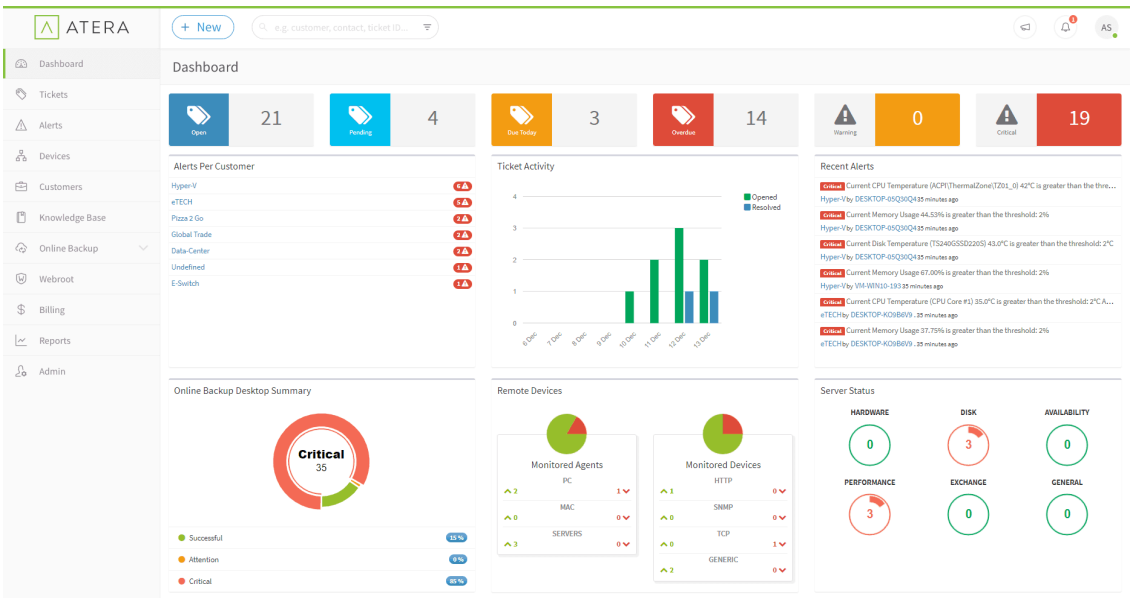

Рисунок 1.8 – Atera RMM

1.3 Вибір мови програмування та засобів розробки

1.3.1 C/C++ та середовище програмування для ESP32

Для написання прошивки для таких апаратно-програмних платформ як Arduino або ESP32 найчастіше використовується C/C++ з використанням додаткових бібліотек. Здебільшого програмування відбувається у середовищі ArduinoIDE який є найбільш поширеним кросплатформеним рішенням написаним на Java і має всі необхідні інструменти для написання коду, його перевірки, встановлення бібліотек і завантаження прошивки безпосередньо у плату[2]. Однак у цьому середовищі розробки все ще є недоліки, наприклад відсутність підклеслювання синтаксичних помилок у режимі реального часу, натомість на помилку буде вказано лише на етапі компіляції коду, що може тормозити процес розробки програми.

Алтернативою середовищу розробки ArduinoIDE може стати редактор Visual Studio Code із встановленим на нього розширенням PlatformIO. Це кросплатформений, кросархітектурний професійний інструмент написання коду для вбудованих систем, що розширює функціонал Visual Studio Code і відповідно дозволяє користуватись усіма перевагами цього редактора, спрощуючи та прискорюючи роботу розробників (рисунок 1.9).[3]

В контексті даної роботи PlatformIO був використаний як основний редактор для написання прошивки і конфігурації проєкту для мікроконтролера ESP32 з урахуванням наведених вище переваг.

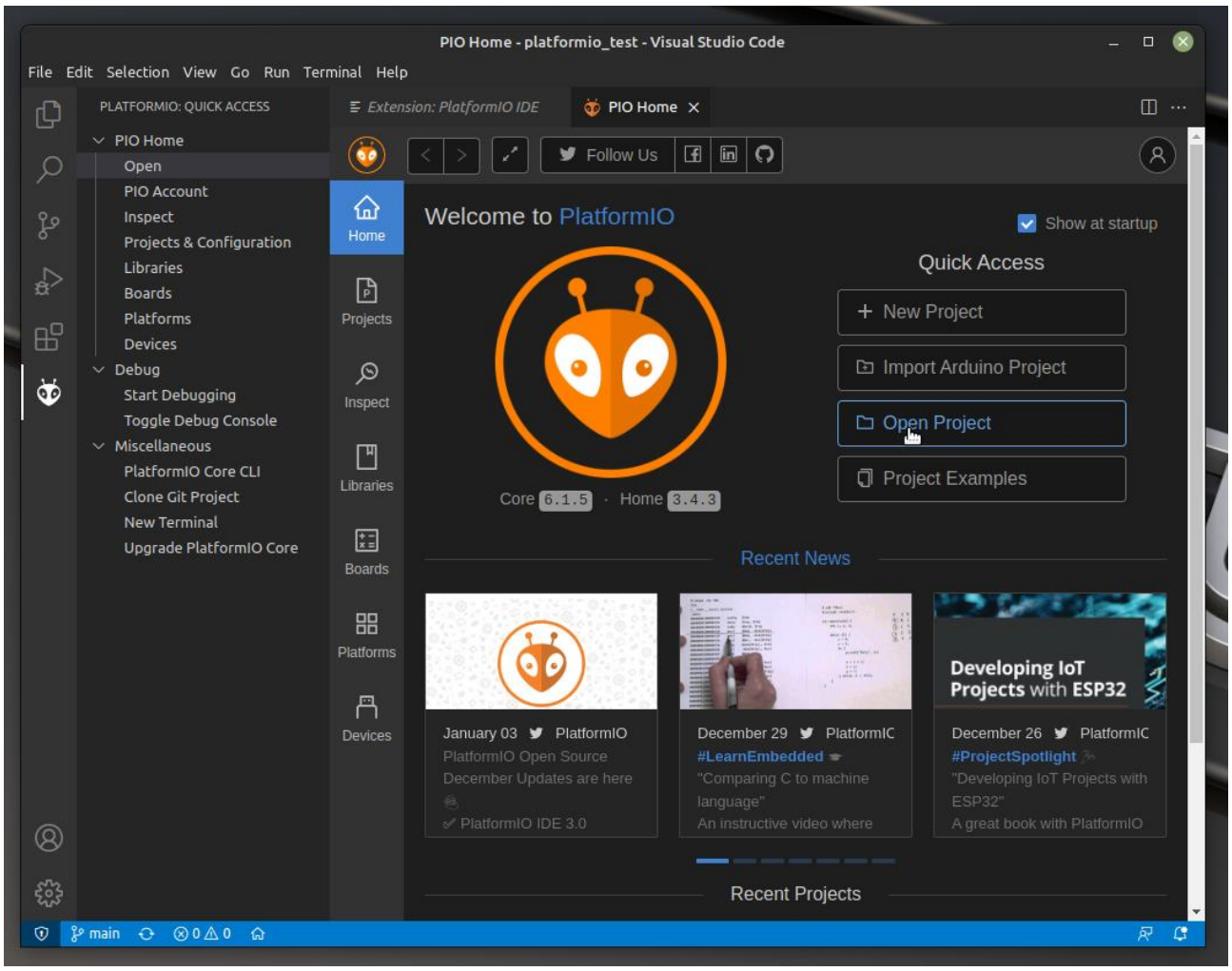

Рисунок 1.9 – PlatformIO встановлений поверх Visual Studio Code

1.3.2 Кросплатформена розробка засобами мови програмування Java

У контексті кросплатформеної розробки віконних застосунків мова програмування Java є одним із провідних інструментів розробки, яка завдяки своїй архітектурі забезпечує незалежність від операційної системи, має обширну екосистему бібліотек та активну спільноту розробників.

Однією з ключових переваг Java є принцип “Write Once, Run Anywhere(WORA)”, що реалізується через компіляцію коду у байт-код, який виконується на віртуальній машині Java(JVM). Це дозволяє запускати один і той самий застосунок на різних операційних системах без необхідності змін у коді. Такий підхід значно спрощує процес розробки та підтримки програмного забезпечення (рисунок 1.10).[4]

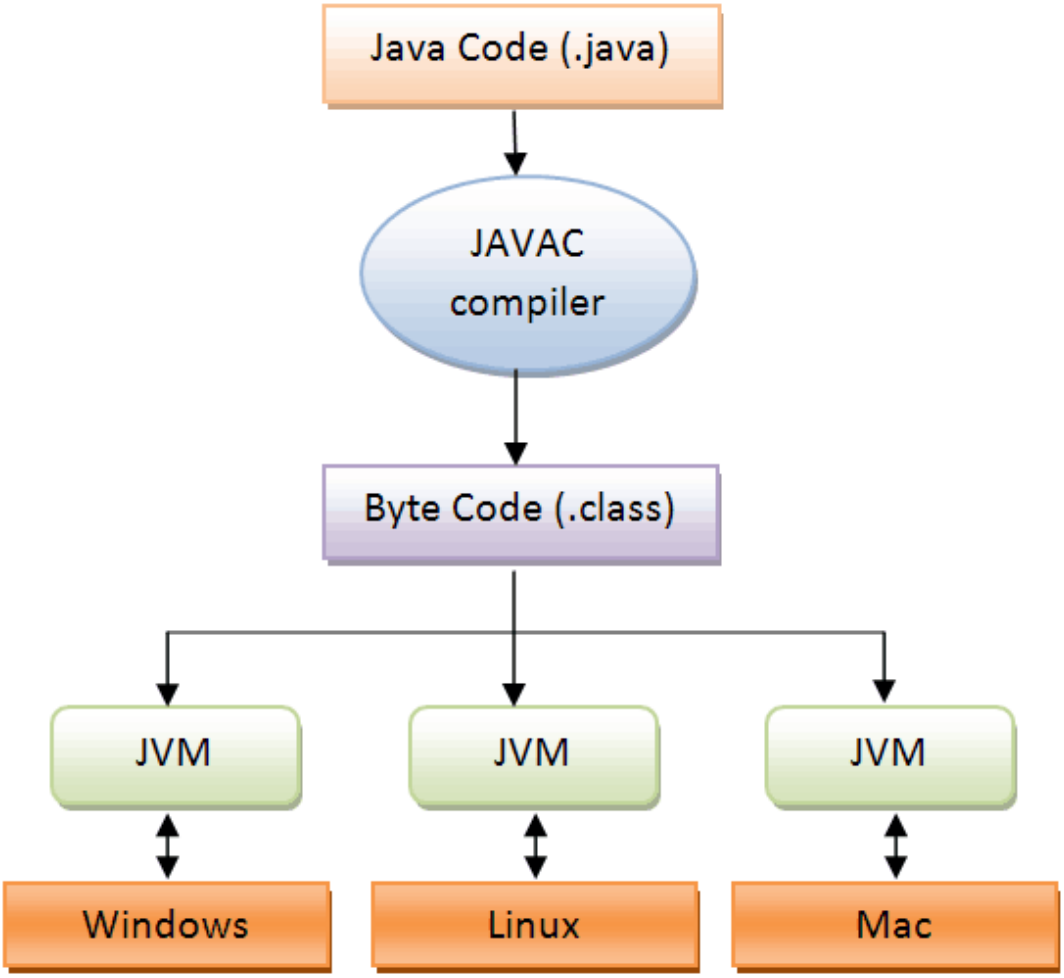

Рисунок 1.10 – Принцип компіляції і виконання коду Java-програми

Java пропонує декілька фреймворків для створення графічного інтерфейсу користувача, які забезпечують кросплатформену сумісність. Серед них в особливості можна виділити фреймворк Swing, що належить до Java Foundation Classes(JFC), яка надає широкий набір компонентів для створення інтерфейсів. Swing є повністю написаним на Java, що забезпечує однаковий вигляд застосунків на різних платформах.

Не дивлячись на те що Swing є не самим новим фреймворком для розробки віконних застосунків, його простота використання і стабільність роботи на всіх операційних системах обґрунтовують його використання у цій роботі для створення клієнтського додатку.

1.3.3 Створення скриптів за допомогою мови Python

Для створення комунікації між мікроконтролером ESP32 та цільовим комп’ютером, та отримання даних про використання ресурсів, комп’ютер повинен вміти оброблювати запити мікроконтролера за допомогою попередньо написаного скрипта. В свою чергу мова програмування Python є одним із найпопулярніших інструментів для створення скриптів завдяки своїй простоті, гнучкості та широкій підтримці бібліотек. До переваг використання Python для створення скриптів можна віднести:

- простота та читабельність коду;
- кросплатформеність;
- активна спільнота;
- широка стандартна бібліотека.

Python є потужним інструментом для створення скриптів, що робить його ідеальним вибором для автоматизації завдань, обробки даних та інших сфер. До його недоліків можна віднести відносно низьку швидкість виконання коду, тому що Python є інтерпретованою мовою програмування, цей недолік нівелюється високою швидкістю розробки, саме тому його було обрано для створення скрипта який буде забезпечувати обмін даними між комп’ютером та мікроконтролером.

1.4 Складання вимог та постановка завдання

Отже, враховуючи функціонал наявних інструментів моніторингу і керування ресурсами комп’ютера, до програмно-апаратного рішення що підлежить розробці можна висунути наступні вимоги:

а) збір моніторингових даних з цільового комп’ютера, таких як:

1) навантаження центрального процесора;
2) об’єм вільної оперативної пам’яті;
3) навантаження мережі;

4) об’єм вільного дискового простору;

5) список виконуваних процесів.

б) отримання загальних характеристик системи;

в) передача моніторингових даних через зашифрований канал зв’язку до клієнта;

г) відображення моніторингових даних у клієнті у вигляді графіків та діаграм, можливість збереження даних в окремий файл;

д) можливість надсилати команди на цільовий комп’ютер з метою знищення процесів та запуску скриптів;

е) сповіщення користувача у разі виявлення аномального використання ресурсів;

є) Незалежність від основної системи; сповіщення користувача у разі відсутності напруги живлення цільового комп’ютера; можливість керування живленням комп’ютера.

Відповідно завдання на роботу складається з:

- розробки прошивки для мікроконтроллера;
- розробки скриптів для коммунікації;
- розробки клієнтського застосунку;
- тестування отриманого рішення.

2 ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Мережева комунікація за допомогою TCP Socket

TCP/IP сокети є базовим механізмом обміну даними між пристроями в мережі, який широко використовується в сучасних комп’ютерних системах або вбудованих пристроях.

Сокет(Socket) – це абстракція, яка представляє кінцеву точку з’єднання між двома пристроями. В контексті TCP/IP, сокети дозволяють встановлювати надійне двостороннє з’єднання між сервером та клієнтом через мережу. TCP(Transmission Control Protocol) гарантує доставку пакетів у правильному порядку без втрат, що робить його ідеальним для додатків, де важлива цілісність переданої інформації (рисунок 2.1).

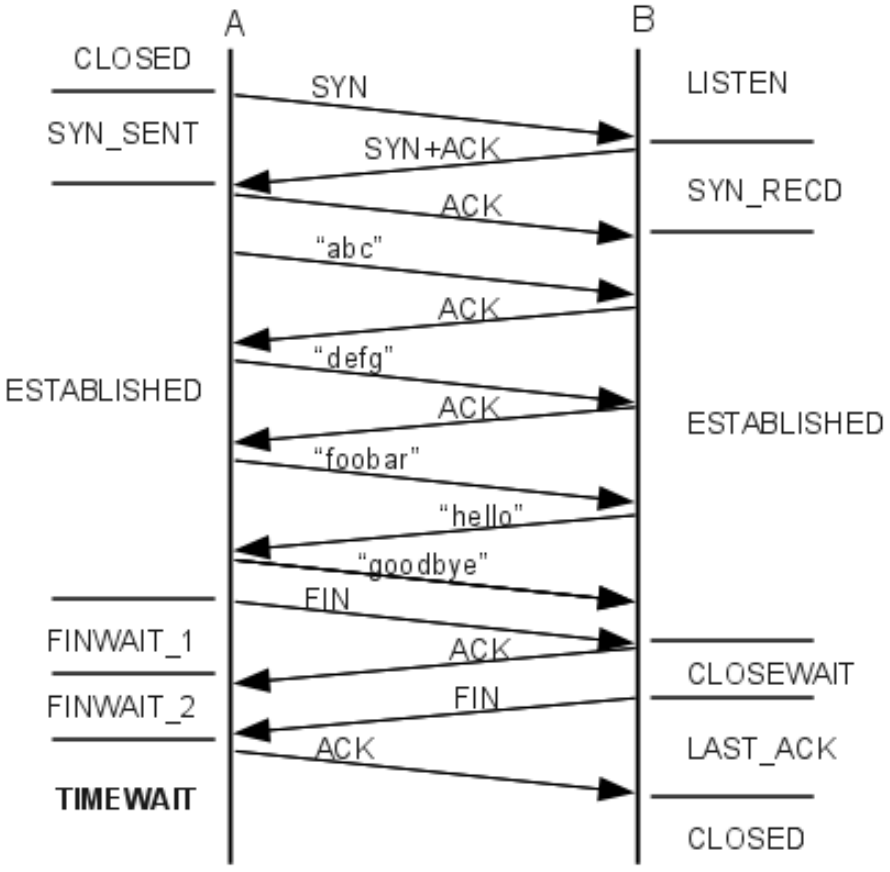

Рисунок 2.1 – Flow-діаграма TCP комунікації

Сокетна модель включає:

- IP-адресу(ідентифікатор пристрою в мережі);
- порт(ідентифікатор сервісу або програми);
- протокол(TCP або UDP).

ESP32 підтримує як TCP так і UDP з’єднання завдяки вбудованому стеку протоколів(часто реалізованому через IwIP у фреймворку ESP-IDF або Arduino Core for ESP32). Це дозволяє створювати як сервери, так і клієнти, що обмінюються даними з іншими пристроями в локальній мережі або через Інтернет.

В свою чергу Java має вбудовану підтримку роботи з TCP/IP сокетами через пакети java.net.Socket та java.net.ServerSocket. Вона дозволяє розробляти кросплатформені додатки, які можуть виступати як у ролі клієнтів, так і серверів.

Використання TCP Socket забезпечує надійний обмін без втрат, що особливо важливо для систем моніторингу або керування. До переваг такого типу з’єднання можна віднести надійність передачі даних, простоту реалізації та широку підтримку в різних мовах програмування. Можливі виклики для розробника полягають у необхідності ручної обробки підключень і потоків, потенційної затримки(latency) порівняно з UDP а також потребу в обробці помилок та перепідключення.

В контексті даної роботи TCP Socket є ключовим компонентом комунікації між клієнтським Java-застосунком та ESP32.[5]

2.2 Обмін даними між комп’ютером і мікроконтролером через USB-to-UART

Послідовна передача даних здійснюється за допомогою протоколу UART (Universal Asynchronous Receiver-Transmitter). UART — це апаратний модуль, який забезпечує асинхронну передачу даних без окремого тактового

сигналу. Передача відбувається у вигляді кадрів, що складаються з:

- стартового біта**:** сигналізує початок передачі;
- бітів даних**:** зазвичай 5–9 бітів, що представляють передану інформацію;
- біта парності (опційно)**:** використовується для перевірки цілісності даних;
- стоп-бітів**:** сигналізують завершення передачі кадру.

Кожен біт передається протягом фіксованого інтервалу часу, визначеного швидкістю передачі (baud rate). Для успішної комунікації обидва пристрої повинні бути налаштовані на однакову швидкість передачі, кількість бітів даних, парність та кількість стоп-бітів.

Більшість розробницьких плат ESP32 (наприклад, DevKitC) мають на борту чіпи, такі як CP2102, CH340 або FTDI, які перетворюють UART-сигнали ESP32 у USB-сигнали для комп’ютера. При підключенні через USB, комп’ютер розпізнає пристрій як віртуальний COM-порт (рисунок 2.2).[6]

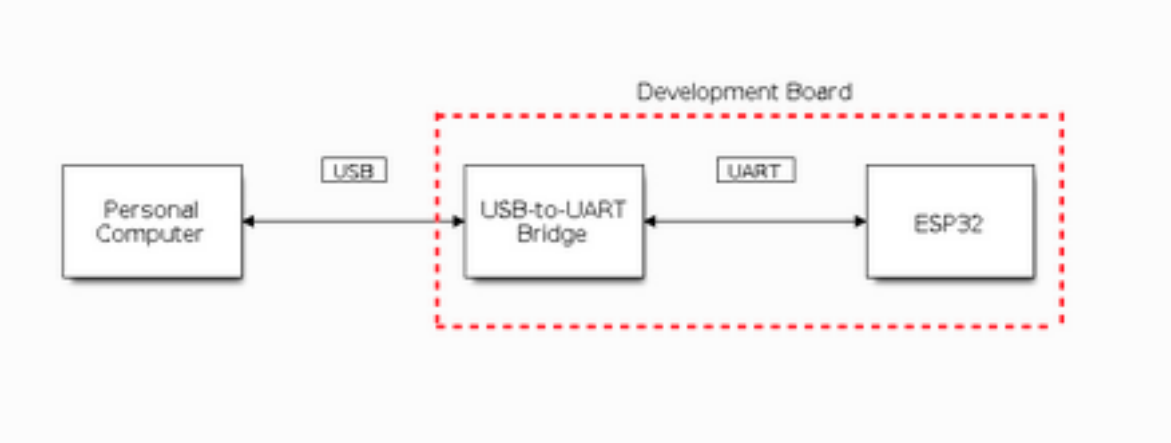

Рисунок 2.2 – Підключення до комп’ютера через вбудований USB-to-UART

Комунікація через послідовний порт (serial port) — це метод передачі даних, при якому інформація передається по одному біту за раз через один провідник. Цей спосіб широко використовується для з'єднання мікроконтролерів, комп'ютерів та периферійних пристроїв.

Зі сторони мікроконтролера комунікація через послідовний порт

здійснюється за використання функцій з файлу заголовків driver/uart.h, наприклад uart_driver_install(), uart_read_bytes(), uart_write_bytes()[7].

Для організації передачі даних з комп’ютера на мікроконтролер є доцільним використання модуля pySerial для мови програмування Python. Використання цього модуля дозволяє ефективно організувати обмін даними між комп’ютером і мікроконтролером через послідовний порт.

В даній роботі інтерфейс UART використовується для передачі даних про систему між цільовим комп’ютером і мікроконтролером через кабель USB-C для їх подальшої обробки і передачі клієнту.

2.3 Форматування даних за допомогою JSON

JSON (JavaScript Object Notation) - простий формат обміну даними, що є зручним як для читання та написання людиною, так і для парсінгу та генерації комп'ютером. Він базується на підмножині мови програмування JavaScript стандарту ECMA-262 3rd Edition - December 1999. JSON - це текстовий формат, повністю незалежний від мови реалізації, але він використовує конвенції, знайомі програмистам С-подібних мов, таких як С, С++, С#, Java, JavaScript, Perl, Python та багатьох інших. Ці властивості роблять JSON ідеальною мовою для обміну даними.

JSON базується на двох структурах даних:

- колекція пар ключ/значення. У різних мовах ця концепція реалізована як *об'єкт*, запис, структура, словник, хеш, іменований список або асоціативний масив;

- упорядкований список значень. У більшості мов це реалізовано як *масив*, вектор, список або послідовність.

Це універсальні структури даних. У тому чи іншому вигляді їх підтримують майже усі сучасні мови програмування. Є сенс будувати формат даних, що є незалежним від мови програмування, саме на цих структурах (рисунок 2.3).[8]

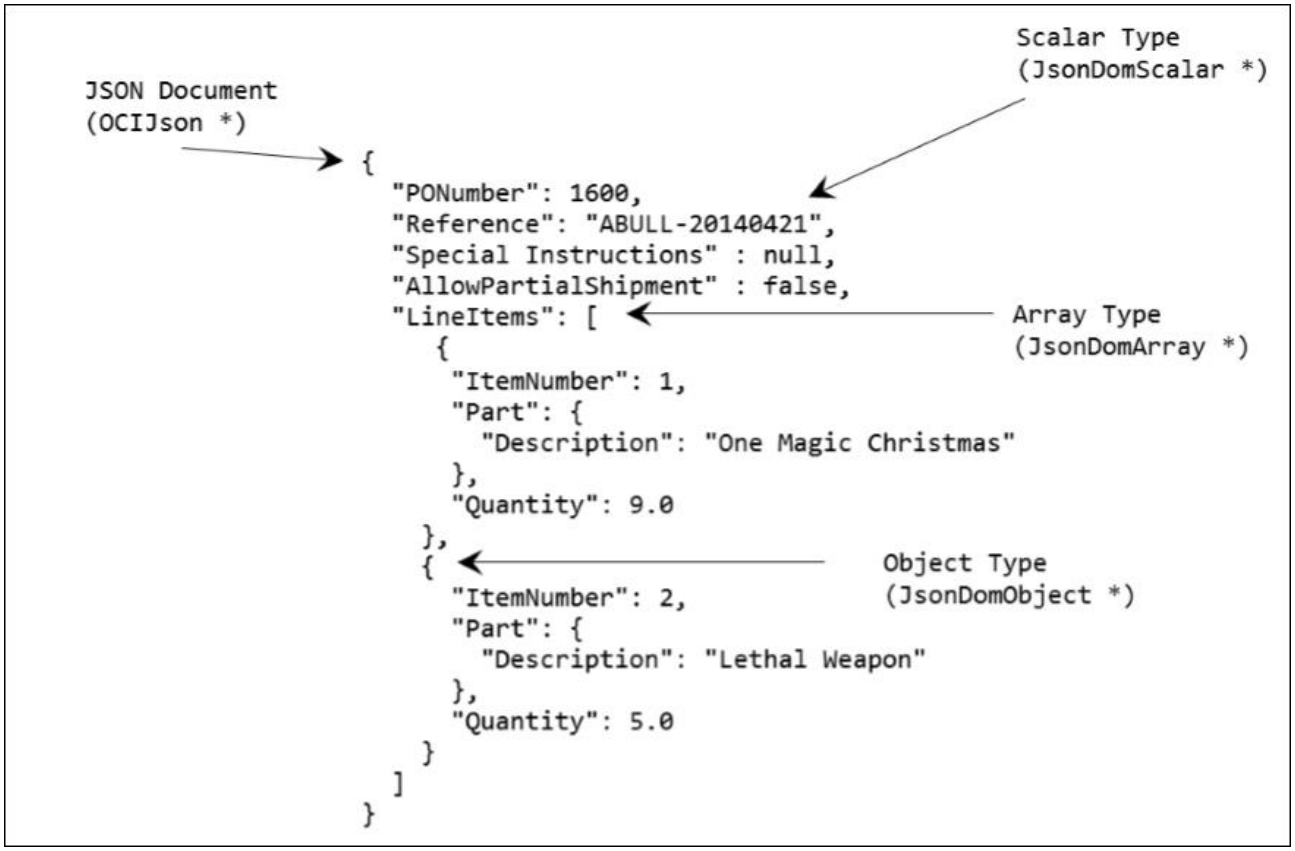

Рисунок 2.3 – Приклад формату JSON

Для використання даного формату у різних мовах програмування є власні бібліотеки. Таким чином, у мові програмування Java є зручна бібліотека Gson.

Gson — це бібліотека для Java, яка забезпечує зручне перетворення Java-об'єктів у формат JSON, а також обернене перетворення — з рядків JSON у відповідні об'єкти Java. Однією з ключових переваг Gson є здатність працювати з будь-якими Java-об'єктами, включно з тими, вихідний код яких недоступний розробнику.

Існує кілька відкритих проєктів, які дозволяють серіалізувати Java-об’єкти у JSON. Проте більшість з них вимагають використання анотацій у класах, що є неможливим у випадках, коли вихідний код недоступний. Крім того, багато таких рішень не забезпечують повноцінної підтримки дженериків (узагальнених типів) Java.

На відміну від них, Gson розроблено з урахуванням двох ключових

принципів:

- можливість роботи без модифікації вихідного коду класів (тобто без використання анотацій);
- повноцінна підтримка Java Generics.

Завдяки цим властивостям Gson є ефективним і гнучким інструментом для роботи з JSON у Java-застосунках, особливо в умовах обмеженого доступу до вихідного коду або при роботі з узагальненими структурами даних[9].

В свою чергу при розробці застосунків на C/C++ і для вбудованих систем є доцільним використання бібліотеки cJSON.

Бібліотека cJSON є легкою та ефективною бібліотекою для роботи з JSON у середовищах з обмеженими ресурсами, таких як ESP32. Вона широко використовується у проєктах на ESP-IDF, особливо коли йдеться про обмін структурованими даними. У контексті ESP32 cJSON зазвичай використовується для:

- формуватування вихідних даних. При надсиланні інформації з пристрою, наприклад, показників системного моніторингу(CPU, пам’ять, мережа та ін.), cJSON дозволяє зручно створити JSON-об’єкт і перетворити його у строку;
- оптимізація пам’яті. cJSON має невеликий розмір та дозволяє контролювати виділення та звільнення пам’яті, що критично для мікроконтролерів із обмеженими ресурсами.[10]

З урахуванням описаних переваг кожної з бібліотек, було обґрунтовано використання cJSON для форматування даних про систему в мікроконтролері, для їх подальшої відправки до Java-клієнту, де завдяки бібліотеці Gson отримані дані можна легко десеріалізувати в відповідні класи.

2.4 Шифрування даних за допомогою Transport Layer Security(TLS)

Протоколи TLS (Transport Layer Security) використовуються для забезпечення захищеного обміну даними в широкому спектрі онлайн-транзакцій. До них належать фінансові операції (наприклад, банківські перекази, торгівля акціями, електронна комерція), медичні сервіси (перегляд медичних записів, запис до лікаря), а також соціальні взаємодії (електронна пошта, соціальні мережі). Будь-який мережевий сервіс, який обробляє чутливу або цінну інформацію — зокрема персональні дані (PII), фінансові дані чи облікові дані користувача — має забезпечити належний захист цієї інформації.

TLS створює захищений канал зв’язку між клієнтом і сервером, запобігаючи перехопленню, підробці або модифікації даних під час передачі. У більшості випадків клієнтом є веббраузер, однак TLS може використовуватись і в інших типах клієнтів[11].

Схема встановлення TLS-сесії наведена на рисунку 2.4.

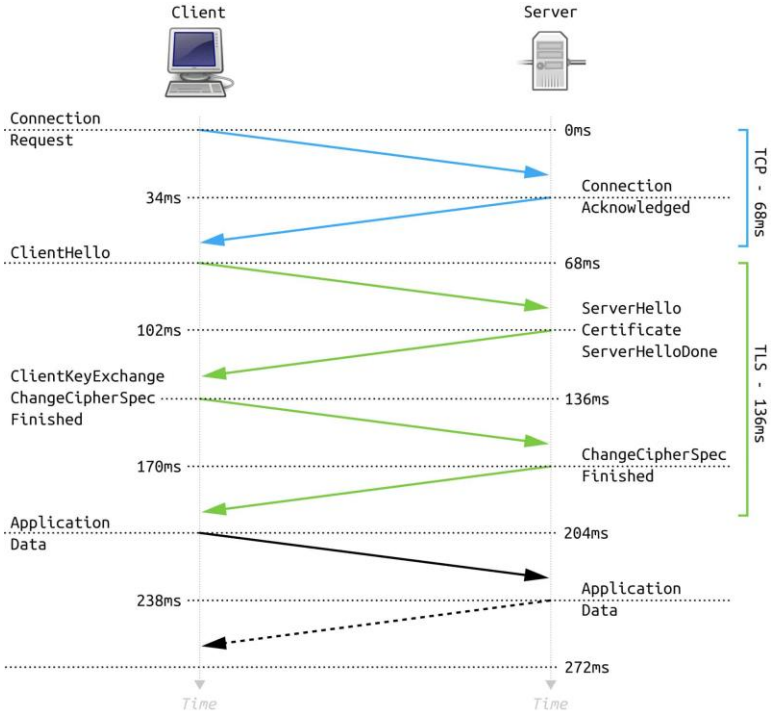

Рисунок 2.4 – Встановлення TLS сесії

Налаштування TLS-з’єднання у клієнті Java можливий за використання вбудованних бібліотек, які підтримують створення SSLSocket. Для використання TLS у мікроконтролера ESP32 існує окремий компонент ESP-TLS.

ESP-TLS — це компонент фреймворку ESP-IDF, який надає спрощений інтерфейс програмування (API) для роботи з TLS-з’єднаннями на мікроконтролерах серії ESP32. Його основне призначення — забезпечити зручний доступ до найбільш поширених функцій захищеної передачі даних через TLS, без необхідності прямого використання складних криптографічних бібліотек, таких як mbedTLS.

Компонент ESP-TLS підтримує ключові функції, зокрема:

- валідацію сертифікатів центру сертифікації (CA);
- підтримку Server Name Indication (SNI) — необхідну для хостингу кількох доменів на одному сервері;
- погодження ALPN (Application-Layer Protocol Negotiation) — для вибору протоколу прикладного рівня (наприклад, HTTP/2);
- неблокуюче з’єднання, що важливо для роботи з асинхронною логікою або при обмежених ресурсах;
- інші параметри безпеки та гнучкість налаштувань.

Усі параметри конфігурації TLS-з’єднання задаються через спеціальну структуру даних esp_tls_cfg_t. Після налаштування, встановлення TLS-з’єднання та обмін зашифрованими даними здійснюється за допомогою відповідних функцій API ESP-TLS, таких як:

- esp_tls_init();
- esp_tls_conn_new();
- esp_tls_read();
- esp_tls_write();
- esp_tls_conn_delete().

Таким чином, ESP-TLS забезпечує зручну, гнучку та безпечну платформу для реалізації захищеної передачі даних на пристроях з

обмеженими обчислювальними ресурсами, зберігаючи при цьому високу сумісність з сучасними вимогами до безпеки в IoT-системах[12].

В контексті даної роботи TLS-з’єднання використовується для створення захищеного потоку даних між клієнтом та сервером, що дозволяє забезпечити безпеку комунікації при передачі даних по Wi-Fi.

2.5 Технології багатопоточності у процесорах і мікроконтролерах

У паралельному програмуванні існують дві базові одиниці виконання — процеси та потоки. У мові програмування Java паралельне програмування здебільшого стосується саме потоків. Проте, процеси також відіграють важливу роль.

Сучасна комп’ютерна система зазвичай виконує велику кількість активних процесів та потоків. Це справедливо навіть для систем із єдиним ядром виконання, де в кожен момент часу реально працює лише один потік. Час процесора в таких системах розподіляється між процесами та потоками за допомогою механізму операційної системи, що зветься тайм-слайсінгом (time slicing).

В сучасних реаліях є дуже поширенним використання комп’ютерних систем з кількома процесорами або процесорами, що мають декілька обчислювальних ядер. Це значно підвищує здатність системи до паралельного виконання процесів і потоків. Однак, паралелізм можливий навіть у простих системах без багатоядерних процесорів [13].

У мові програмування Java є окремий клас Thread, використання якого полягає або у спадкуванні цього класа власним класом або у імплементації інтерфейсу Runnable. У будь якому випадку обов’язковим є власна імплементація методу run() в якому безпосередньо описується поведінка потоку.

В свою чергу, мова програмування Python реалізує багатопоточність за допомогою модуля threading. При створенні ногово потоку констркуктор

threading.Thread() приймає функцію та аргументи до неї та створює новий поток, який запускається методом start() та завершується методом join().

Багатопоточність у мікроконтролері ESP32 реалізується завдяки використанню операційної системи реального часу FreeRTOS, яка вбудована в середовище розробки ESP-IDF. Вона дозволяє запускати кілька завдань (tasks), які можуть виконуватися паралельно на двох ядрах мікроконтролера. Це дає змогу ефективно розподіляти обчислювальні ресурси між різними функціями, наприклад, обробкою даних із сенсорів, роботою з мережею та керуванням пристроями. Завдання можна пріоритезувати, а також налаштовувати синхронізацію між ними за допомогою черг, семафорів і м’ютексів. Такий підхід забезпечує високу продуктивність і гнучкість під час розробки складних вбудованих систем.

В даному проєкті багатопоточність використовується як в клієнтській так і серверній частині. Сервер сворює декілька окремих потоків, один для роботи TLS-сервера, один для організації періодичного збору моніторингових даних цільової системи кожні 5 секунд та ще один для постійного контролю за станом підключення мікроконтролера до цільової системи і вчасного оповіщення користувача у випадку будь-яких системних збоїв.

Оскільки в даному проєкті обмін даними між клієнтом та сервером базується не на моделі request/response, а є постійним потоком інформації у межах сокету, клієнт повинен мати окремий поток що постійно читає вхідний буфер даних і завчасно реагує на нові повідомлення, викликаючи певні функції клієнта.

3 СТРУКТУРА ТА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

3.1 Структура Proxy-скрипта на Python

Скрипт, що запускається на цільовому комп’ютері повинен виконувати наступні функції:

- ідентифікація мікроконтролера при його підключенні та відкриття з’єднання за послідовним портом;
- організація підключення пристрою до мережі Wi-Fi;
- виконання команд отриманих від мікроконтролера та передача результатів виконання.

Для ідентифікації пристрою була створена функція ESP32_detect(). У ній була використана функція comports() модуля serial.tools.list_ports бібліотеки pyserial яка повертає список всіх доступних послідовних портів на комп’ютері. Після чого у кожному з портів перевіряються поля desc та manuf, що містять опис та виробника пристрою підключеного до порта. Мікроконтролер ідентифікується за ключовими словами ch340, cp210, silicon labs та ftdi. Якщо ESP32 було знайдено, функція повертає номер порту.

Після ідентифікації відкривається послідовний порт, встановлюється швидкість передачі даних у 115200 бод та таймаут 3 секунди (лістинг 3.1).

Лістинг 3.1 – Функції пошуку та підключення мікроконтролеру

```
import serial
import serial.tools.list_ports
import time
import subprocess

def ESP32_detect():
    ports = serial.tools.list_ports.comports()
    for port in ports:
        desc = port.description.lower()
        manuf = (port.manufacturer or "").lower()
        if "ch340" in desc or "cp210" in desc or "silicon labs"
in manuf or "ftdi" in desc:
```

```
            return port.device
    return None

def wait_for_device():
    port = ESP32_detect()
    while not port:
        input("\nPlease insert the device and press ENTER\n")
        port = ESP32_detect()
    print(f"ESP32 detected on port {port}!\n")
    return port

def connect_to_serial(port):
    while True:
        try:
            ser = serial.Serial(port=port, baudrate=115200,
timeout=3)
            time.sleep(2)
            return ser
        except serial.SerialException:
            print(f"Failed to open port {port}. Retrying...")
            time.sleep(2)
            port = wait_for_device()
```

Після успішного підключення пристрою виконується перевірка підключення пристрою до Wi-Fi шляхом надсилання повідомлення check_connection через послідовний порт. Якщо пристрій ще не підключений до мережі йому передаються отримані від користувача ssid та пароль Wi-Fi точки, після чого підключення перевіряється знову (лістинг 3.2).

Лістинг 3.2 – Функції забезпечення Wi-Fi з’єднання для мікроконтролера

```
     def check_wifi_connection(ser):
    time.sleep(0.5)
    while True:
        try:
            ser.write(b"check_conn\n")
            response =
ser.readline().decode(errors="replace").strip()
        except serial.SerialException:
            raise ConnectionError("Lost connection while
checking Wi-Fi.")
        print(f"Initial response: {response}")
        if "WiFiConnected" in response:
            print("ESP32 is already connected to Wi-Fi.")
            return True
        elif "WiFiDisconnected" in response:
            return False
def provision_wifi(ser):
```

```
    while True:
        wlan = input("\nPlease write your WLAN Name: ")
        password = input("\nPlease write your WLAN Password: ")
        ser.write(f'{wlan},{password}\r\n'.encode())
        time.sleep(1)

        while True:
            try:
                response =
ser.readline().decode(errors="replace").strip()
            except serial.SerialException:
                raise ConnectionError("Lost connection during
Wi-Fi provisioning.")
            print(response)
            if 'Connected to Wi-Fi' in response:
                print("Connected successfully")
                return
            elif 'Wi-Fi connection failed' in response:
                print('Failed to connect. Please try again.')
                break
```

Коли ESP32 було успішно підключено до Wi-Fi починається основний цикл виконання команд. Послідовний порт постійно перевіряється на наявність нових повідомлень, і у разі знаходження ключового слова COMMAND розташована за ним команда виконується завдяки функції subprocess.run() модуля subprocess. Результат виконання зберігається та построково записується у послідовний порт. Якщо під час виконання команди виникла помилка вона також будет записана у результат (лістинг 3.3).

Лістинг 3.3 – Основний цикл виконання команд

```
def main_loop(ser):
    while True:
        try:
            line =
ser.readline().decode(errors="replace").strip()
        except serial.SerialException:
            raise ConnectionError("Lost connection during
command loop.")
        if line:
            print(line)
        if line.lower() in ['exit', 'quit']:
            print('\nExiting command mode.\n')
            break
        if "COMMAND" in line:
```

```
            idx = line.find("COMMAND:")
            cmd = line[idx + len("COMMAND: "):].strip()
            print(f"Executing command: {cmd}")
            try:
                result = subprocess.run(cmd, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True,
timeout=10)
                output = result.stdout + result.stderr
                if not output.strip():
                    output = "Command executed, but no output\n"
                output += "$"
                for line in output.splitlines():
                    ser.write((line[:250] + '\n').encode())
            except subprocess.TimeoutExpired:
                ser.write("Error executing command: Command
timed out.\n").encode()
            except Exception as e:
                ser.write(f"Error executing command:
{str(e)}\n").encode()
```

Якщо під час виконання будь-якої частини скрипта з’єднання з ESP32 буде втрачено виконання скрипта почнеться з початку. Алгоритм роботи скрипта наведено на рисунку 3.1.

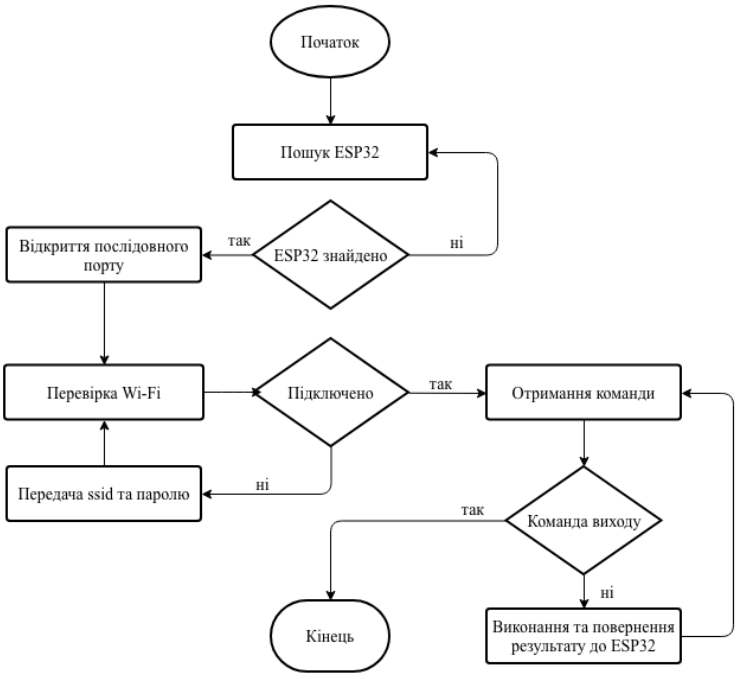

Рисунок 3.1 – Алгоритм роботи Proxy-скрипта

3.2 Структура прошивки ESP32-сервера

3.2.1 Опис функціонала прошивки

До основного функціоналу прошивки належить:

- відкриття послідовного з’єднання з цільовим комп’ютером;
- підключення до мережі Wi-Fi за отриманими від комп’ютера ssid та паролем;
- створення TLS-сокета;
- організація підключення клієнта та створення зашифрованого каналу зв’язку;
- обробка отриманих від клієнта команд;
- отримання даних про навантаження окремих частей комп’ютера та загальної інформації системи шляхом читання розташованих у /proc/ файлів та виконання таких команд як uname та df;
- парсінг отриманої інформації та конвертування даних у JSON-формат;
- надсилання інформації до клієнта;
- постійний моніторинг системи на збої: таймаут на виконання команд, перевірка фізичного підключення пристрою до цільового комп’ютера;
- надсилання повідомлень до клієнта у разі виявлення збою роботи.

Для забезпечення клієнта необхідною інформацією мікроконтролер повинен бути спроможнім оброблювати наступні команди:

- /get-sys-info – Отримання загальної інформації про систему;
- /start-monitor – Запуск моніторингу, надсилання пакету даних про навантаження системи кожні 5 секунд.;
- /stop-monitor – Зупинка моніторингу;
- /get-proc-info <pid> - Отримання більш детальної інформації про процес. Приймає pid процесу як параметр;

- /kill-proc <pid> - Знищення процесу. Приймає pid процесу як параметр;

- /reset-conn – Команда для скидання флагів таймауту та відключення від USB;

- /execute <path> - виконання sh-скрипта, приймає абсолютний шлях скрипта як параметр;

- /reboot-system – Виконує перезавантаження системи.

3.2.2 Використані бібліотеки

Під час розробки прошивки були використані наступні бібліотеки:

- esp_system.h – надає функції для керування системою ESP32, наприклад перезавантаження, інформація про чип та ін.;

- esp_wifi.h – API для налаштування і управління Wi-Fi;

- esp_flash.h – доступ до вбудованої SPI-флеш-пам’яті ESP32;

- nvs_flash.h – робота з енергонезалежним сховищеми(Non-Volatile Storage, NVS);

- sdkconfig.h – автоматично сгенерований файл конфігурації проєкта;

- esp_event.h – управління подіями та їх обробниками(event loop);

- cJSON.h – бібліотека для створення, серіалізації та десеріалізації JSON-об’єктів на C;

- cstring – C++ заголовок для роботи з C-строками;

- esp_log.h – логування в ESP-IDF;

- driver/gpio.h – керування GPIO;

- driver/uart.h – робота з UART-інтерфейсом;

- mbedtls/net_sockets.h – абстракція сокетов для TLS-з’єднань;

- mbedtls/ssl.h – API TLS/SSL-з’єднань(ініціалізація, шифрування, рукостискання);

- mbedtls/entropy.h – генерація ентропії для криптографічних операцій;

- mbedtls/ctr_drbh.h – генератор випадкових числед на базі

CTR_DRBG(використовується mbedTLS);

- mbedtls/x509_crt.h – робота з X.509 сертифікатами;
- mbedtls/pk.h – підтримка публічних ключей(парсінг, підписи, шифрування);
- freertos/FreeRTOS.h – базовий заголовок FreeRTOS, підключає ядро та базові типи;
- freertos/task.h – керування задачами(створення, видалення, затримки і т.д.);
- string – C++ стандартна строка;
- vector – C++ контейнер динамічного масиву;
- map – C++ асоціативний контейнер ключ-значення;
- sstream – C++ потоки для роботи зі строками;
- freertos/semphr.h – семафори і м’ютекси у FreeRTOS;
- freertos/event_groups.h – події(бітові флаги) для синхронізації задач.

3.2.3 Архітектура проєкту

Проєкт прошивки для ESP32 складається з файлу main.cpp, файлу зброки CMakeLists.txt, директорій certs, headers та modules (рисунок 3.2).

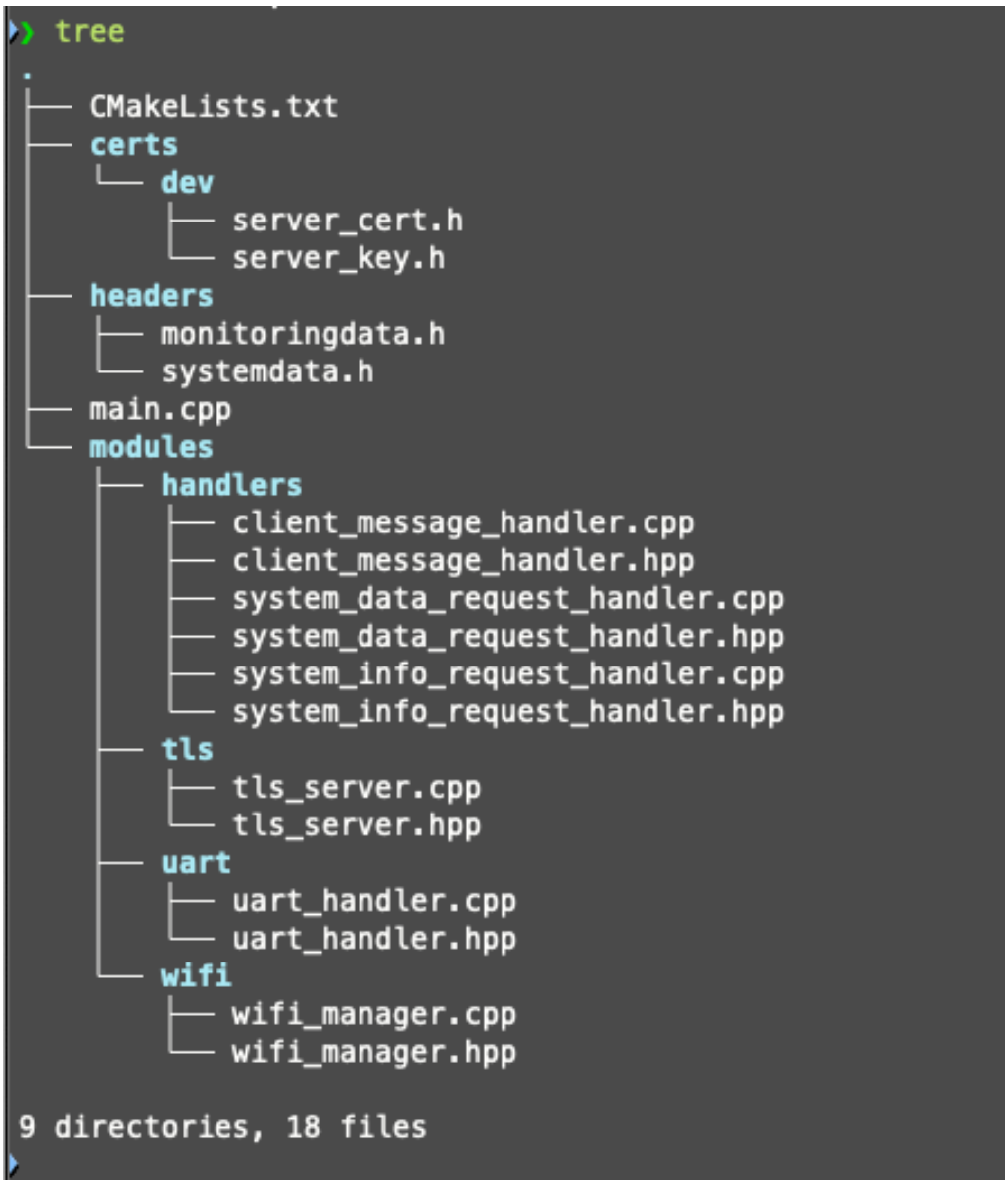

Рисунок 3.2 – Ієрархія файлів та директорій проєкту

Директорія certs містить конвертовані у header-файли ssl-сертифікат та private key. У директорії headers містяться файли monitoringdata.h та systemdata.h що містять відповідні структури. У директорії modules містяться окремі модулі функціоналу, такі як tls_server, wifi_manager, uart_handler та директорія handlers у якій знаходяться основні обробники повідомлень. Кожен модуль містить .hpp-файл заголовків та .cpp файл з імплементацією функцій.

3.2.4 Опис модулів та main – функція

Ініціалізація та всі операції що виконуються за участі UART-інтерфейсу описані у файлах uart_handler.hpp та uart_handler.cpp (лістинг 3.4).

Лістинг 3.4 – вміст файлу uart_handler.hpp

```
#pragma once
#include "driver/gpio.h"
#include "driver/uart.h"

#define VBUS_GPIO GPIO_NUM_4
#define UART_PORT UART_NUM_0
#define BUF_SIZE 4096

extern bool uart_connected;
extern TickType_t last_uart_activity;

void uart_init(void);
void init_vbus_gpio(void);
int uart_readline(char* buf, int max_len);
bool uart_check_connection();
```

Функція uart_init(void) ініціалізує UART-порт з наступними параметрами:

- швидкість 115200 бод;
- 8 біт даних, без біта парності, 1 стоп-біт;
- без апаратного контроля контроля потоку.

Також налаштовує піни(не змінюючи їх) і встановлює драйвер UART з буфером для прийому.

Функція uart_readline(char* buf, int max_len) читає строку з UART побайтно до символу \n або \r, або поки не досягуто max_len – 1, з таймаутом 100 мс на кожен байт. Повертає довжину строки або 0 при помилці або таймауті.

Функція init_vbus_gpio(void) налаштовує GPIO-пін VBUS_GPIO як вхід, використовується для визначення підключення живлення по USB.

Функція uart_check_connection() визначає чи підключений ESP32 фізично до комп’ютера, читає рівень на піні VBUS_GPIO, повертає true якщо

рівень високий або false якщо рівень низький.

Модуль wifi_manager відповідний за ініціалізацію та підключення до Wi-Fi а також обробку Wi-Fi подій і перевірку підключення (лістинг 3.5).

Лістинг 3.5 – вміст файлу wifi_manager.hpp

```
#pragma once
#include "esp_event.h"
#include "esp_wifi.h"
#include "cstring"
#include "esp_log.h"

#define WIFI_CONNECTED_BIT BIT0
#define TAG "WiFi"

extern EventGroupHandle_t s_wifi_event_group;

bool check_connection();
esp_err_t wifi_init_sta(const char*, const char*);
void wifi_event_handler(void* arg, esp_event_base_t event_base,
                                int32_t event_id, void*
event_data);
```

Функція wifi_event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data) обробляє події Wi-Fi:

- при старті(WIFI_EVENT_STA_START) – підключається до Wi-Fi;
- при відключені(WIFI_EVENT_STA_DISCONNECTED) – намагається підключитись знову;
- при отриманні IP-адреси(IP_EVENT_STA_GOT_IP) – виводить IP в лог та встановлює флаг підключення в групі подій;

Функція wifi_init_sta(const char* ssid, const char* password) ініціалізує Wi-Fi в режимі клієнта:

- перевіряє ssid та пароль, налаштовує стек мережевих інтерфейсів та Wi-Fi;
- реєструє обробники подій;
- встановлює конфігурацію Wi-Fi та запускає підключення;
- очікує до 10 секунд на підключення і повертає ESP_OK при успіху, або ESP_FAIL.

Функція check_connection() перевіряє чи встановлений флаг підключення Wi-Fi у групі подій(WIFI_CONNECTED_BIT) і повертає відповідно true або false.

Модуль tls_server відповідний за налаштування TLS-з’єднання з клієнтом і обробка нових поступаючих з’єднань (лістинг 3.6).

Лістинг 3.6 – вміст файлу tls_server.hpp

```
#pragma once
#include "mbedtls/net_sockets.h"
#include "mbedtls/ssl.h"
#include "mbedtls/entropy.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/x509_crt.h"
#include "mbedtls/pk.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "string"
#include "esp_log.h"
#include "cstring"
#include "certs/dev/server_cert.h"
#include "certs/dev/server_key.h"
#include "esp_system.h"

#define SERVER_PORT "4433"
#define TAG_SERVER "TLS_SERVER"

extern SemaphoreHandle_t ssl_mutex;
extern mbedtls_ssl_context* monitor_ssl;
extern char *message_handler(const std::string &msg,
mbedtls_ssl_context *ssl);
int ssl_write_all(mbedtls_ssl_context *ssl, const unsigned char
*buf, size_t len);

bool setup_tls_config(mbedtls_ssl_config* conf,
                      mbedtls_ctr_drbg_context* ctr_drbg,
                      mbedtls_x509_crt* cert,
                      mbedtls_pk_context* pkey);
bool setup_listening_socket(mbedtls_net_context* listen_fd,
const char* port);
void handle_tls_client(mbedtls_net_context* listen_fd,
                       mbedtls_ssl_context* ssl,
                       mbedtls_ssl_config* conf);
void client_communication_loop(mbedtls_ssl_context* ssl);
void cleanup_tls_context();
void tls_server_task(void* pvParameters);
```

Функція init_tls_context(…) ініціалізує контексти TLS і завантажує сертифікат і приватний ключ сервера. Також налаштовує генератор випадкових чисел. Повертає false при помилці ініціалізації.

Функція setup_tls_config(…) налаштовує TLS-конфігурацію для режиму сервера(MBEDTLS_SSL_IS_SERVER), прив’язує сертифікат, приватний ключ і генератор випадкових чисел. Повертає false при помилці конфігурації.

Функція setup_listening_socket(…) відкриває TCP-сокет і починає слухати вказаний порт SERVER_PORT. Повертає true при успіху, інакше false.

Функція client_communication_loop(…) обробляє цикл комунікації з клієнтом:

- читає дані через TLS;
- викликає message_handler();
- відсилає результат клієнту через захищене з’єднання.

Завершується за командою /disconnect або при помилці з’єднання

Функція handle_tls_client(…) приймає вхідне TCP-підключення, виконує TLS-рукостискання і запускає обробку повідомлень від клієнта. Після завершення комунікації – закриває сесію та вивільнює ресурси клієнта

Функція cleanup_tls_context(…) очищує і вивільнює всі структури, що використовуються для TLS: сокет, конфігурація, контексти, ключі, генератори та сертифікати.

tls_server_task(void* pvParameters) – FreeRTOS-задача, яка:

- ініціалізує TLS-сервер;
- запускає цикл обробки підключень від клієнтів;
- після завершення очищує ресурси.

Файли systemdata.h та monitoringdata.h містять відповідні простори імен, описи структур інформації про систему та поточне завантаження системи, а також методи серіалізації цих даних у JSON. Опис цих файлів наданий у додатках Б та В.

Обробник повідомлень, та споміжні функції описані у модулі client_message_handler (лістинг 3.7).

Лістинг 3.7 – Вміст заголовку client_message_handler.hpp

```
#pragma once
#include <string>
#include <cstring>
#include"mbedtls/ssl.h"
#include <esp_log.h>
#include <cJSON.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "headers/monitoringdata.h"
#include "headers/systemdata.h"
#include "system_data_request_handler.hpp"
#include "system_info_request_handler.hpp"
#include <freertos/semphr.h>
#include "driver/uart.h"

const char* COMM_TAG = "COMMAND";
#define TAG_SERVER "TLS_SERVER"
#define UART_PORT UART_NUM_0
extern int uart_readline(char*, int);

extern TaskHandle_t monitor_task_handle;
extern mbedtls_ssl_context* monitor_ssl;
extern SemaphoreHandle_t ssl_mutex;
extern SemaphoreHandle_t serial_mutex;
extern bool conn_timeout;
extern bool usb_disconnected;

extern int ssl_write_all(mbedtls_ssl_context *ssl, const
unsigned char *buf, size_t len);
void monitor_task(void* arg);
char* message_handler(const std::string& msg,
mbedtls_ssl_context* ssl);
std::string executeCommand(const std::string& command);
```

Функція execute_command(const std::string& command) надсилає команду через UART до цільового комп’ютера, збирає текстову відповідь до маркера $, при досягнені таймаута у 10 секунд надсилає відповідне повідомлення клієнту і блокує подальше виконання команд, встановлюючи флаг conn_timeout. Ця функція використовує serial_mutex для синхронізації

доступу до UART. Якщо при виконанні команди виникла помилка повертає ERROR.

Функція message_handler(const std::string msg, mbedtls_ssl_context* ssl) є головною функцією обробки повідомлень від клієнта. Вона визначає команду, виконує відповідну дію та повертає JSON-рядок або повідомлення. Список команд, що оброблюються:

- /get-sys-info – збирає загальну системну інформацію;
- /get-proc-info <pid> - збирає детальну інформацію про процесс;
- /start-monitor; /stop-monitor – запускає або зупиняє задачу моніторингу;
- /kill-proc <pid> - надсилає команду на вбивство процесу;
- /reboot-system – перезавантажує хост;
- /reset-conn – скидає внутрішній стан з’єднання;
- /execute <script> - запускає shell-скрипт через sudo sh.

monitor_task(void* arg) – задача FreeRTOS, яка щоп’ять секунд надсилає дані про завантаження системи. Вона генерує JSON за допомогою serializeMonitoringData і відправляє відповідь через захищений канал, використовуючи ssl_mutex для синхронізації запису.

Отримання необхідно інформації від системи, парсінг даних та заповнення відповідних структур реалізують модулі system_data_request_handler (лістинг 3.8) і system_info_request_handler (лістинг 3.9).

Лістинг 3.8 – Вміст файлу заголовків system_data_request_handler.hpp

```
#pragma once
#include "headers/monitoringdata.h"
#include "esp_log.h"
#include <string>
#include <sstream>
#include <vector>
#include <map>

extern std::string execute_command(const std::string& command);
monitoringdata::MemoryInfo get_mem_info();
```

```
std::vector<monitoringdata::DiskUsage> get_disk_usage();
monitoringdata::ProcessInfo get_process_info(std::string pid);
bool killProcess(std::string pid);
std::map<int, std::string> get_processes();
std::vector<monitoringdata::NetworkInterface>
get_network_info();
monitoringdata::CPUStat get_cpu_stat();
monitoringdata::SystemLoadData get_system_load_data();
```

Лістинг 3.9 – Вміст файлу заголовків system_info_request_handler.hpp

```
#pragma once
#include "headers/systemdata.h"
#include <string>
#include <vector>
#include <sstream>

extern std::string execute_command(const std::string& command);
systemdata::kernel_info get_kernel_info();
std::vector<systemdata::cpu_info> get_cpu_info();
systemdata::os_info get_os_info();
systemdata::system_info get_system_info();
```

У файл main.cpp включаються всі необхідні модулі, встановлюються глобальні змінні. Окрім функції app_main() присутня також функція задачі void device_state_task(void* arg), яка постійно перевіряє UART-порт на наявність check_conn, та інформує скрипт про підключення до Wi-Fi, у разі необхідності приймає SSID та пароль мережі і ініціалізує з’єднання. Окрім цього, задача перевіряє чи підключений ESP32 до комп’ютера перевіряючи джерело живлення, і у разі відсутності підключення повідомляє клієнт(якщо наявний) і блокує виконання execute_command(), встановлюючи флаг usb_disconnected.

Точка входу програми, функція app_main() виконує ініціалізацію всіх модулів, запускає задачу device_state_task(), чекає на підключення з Wi-Fi, після чого вмикає синій LED-індикатор на платі та запускає tls_server_task(). Вміст файлу main.cpp наведено у додатку Г.

3.3 Додаткові модифікації ESP32

Для можливості окремо фізично відстежувати відсутність живлення на мікроконтролері(що свідчить про відсутність живлення системи або відсутність підключення по USB) необхідно додати зовнішнє джерело живлення у вигляді акумулятора на 3.7В, та підключити його до піну 5В на ESP32. Сам акумулятор повинен бути підключений через спеціальну плату захисту від перерозряду/перезаряду акумулятора, у випадку з одним Li-Ion акумулятором формату 18650 доцільно використовувати плату BMS 1S. Детектувати джерело живлення буде GPIO4-пін на платі. На пін подається напруга живлення V_{in} ESP32 через окремий дільник напруги, який забезпечить напругу на піні у 3.3В у разі живлення від 5В через USB-C порт або напругу <2.5В у разі живлення від зовнішнього акумулятора. Таким чином мікроконтролер зможе ідентифікувати джерело живлення і відправляти повідомлення клієнту у разі відключення цільового комп'ютера.

Розрахунок дільника напруги:

$$V_{\text{out}} = V_{\text{in}} \cdot \frac{R_2}{R_1 + R_2}$$

$$R_2 = 0.66R_1 + 0.66R_2 \Rightarrow R_2 - 0.66R_2 = 0.66R_1$$

$$\Rightarrow 0.34R_2 = 0.66R_1 \Rightarrow \frac{R_2}{R_1} = \frac{0.66}{0.34} \approx 1.94$$

При V_{in} = 3.3В:

$$V_{\text{out}} = 3.3 \cdot \frac{1.94}{1 + 1.94} = 3.3 \cdot \frac{1.94}{2.94} \approx 2.18\,\text{В}$$

Приклад з номіналами резисторів R_1 = 10 kΩ, R_2 = 18 kΩ:

$$\frac{18}{10 + 18} = \frac{18}{28} \approx 0.643$$

Отже, результуюча напруга:

$$V_{\text{out}} = V_{\text{in}} \cdot \frac{R_2}{R_1 + R_2} \Rightarrow \begin{cases} V_{\text{in}} = 5\,\text{В} \Rightarrow V_{\text{out}} \approx 3.21\,\text{В} \\ V_{\text{in}} = 3.7\,\text{В} \Rightarrow V_{\text{out}} \approx 2.38\,\text{В} \end{cases}$$

Схема підключення компонентів до ESP32 наведена на рисунку 3.3.

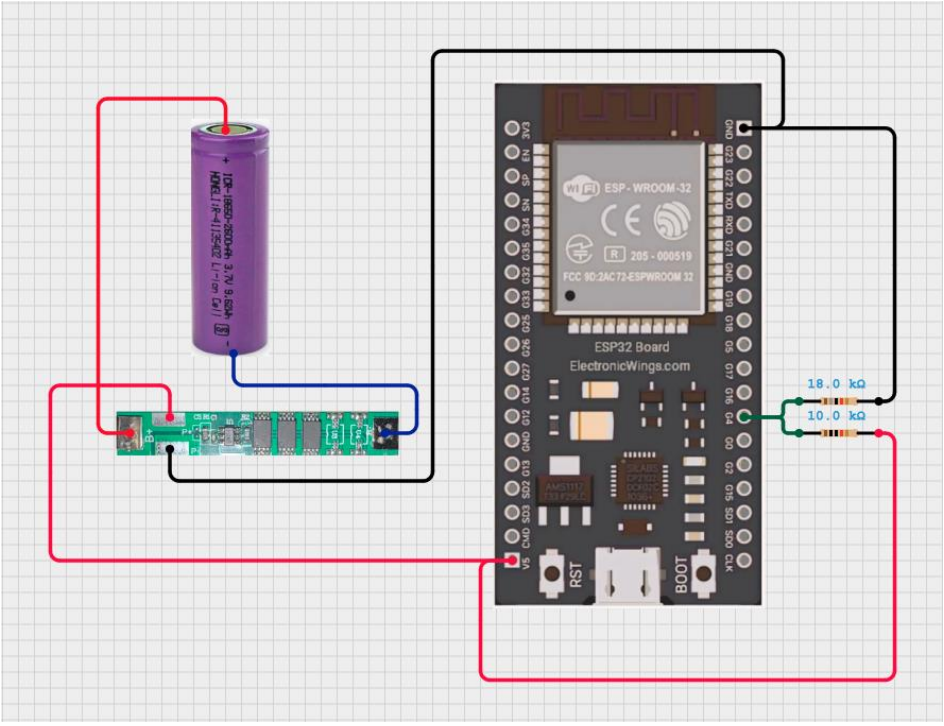

Рисунок 3.3 – Підключення компонентів до ESP32

Розмір отриманої прошивки для ESP32 становить 1079696 байт, що трохи більше за стандартний розмір виділеної під прошивку пам’яті(1 Мб), тому для збільшення простору сховища необхідно створити окрему таблицю розділів, що використовуватиметься при запису прошивки (таблиця 3.1).

Таблиця 3.1 – таблиця розділів(файл partitions.csv)

Name	Type	SubType	Offset	Size
nvs	data	nvs	0x9000	0x5000
otadata	data	ota	0xe000	0x2000
app0	app	ota_0	0x10000	0x1E0000
spiffs	data	spiffs	0x1F0000	0x10000

Таким чином досягається розмір сховища у 2 Мб, чого цілком достатньо для розміщення прошивки на мікроконтролері.

4 СТРУКТУРА ТА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

4.1 Структура класів Java-додатку

Клієнтський додаток складається з декількох пакетів: SystemInfo, SystemLoad, Tabs, net, ui і common. Пакет net містить класи ConnectionManager, який відповідає за створення TLS-з’єднання з сервером, DataReceiverThread що спадкує клас Thread і створює поток який постійно читає буфер даних і оброблює події в залежності від отриманих повідомлень і класу CommandHandler який надсилає до сервера отримані від користувача команди.

Пакет ui містить імплементацію основних компонентів графічного інтерфейсу користувача, такі як ControlsPanel(Панель елементів керування), TabsPanel(Панель інформаційних вкладок), MainClientFrame(Основне вікно програми) і SaveLoad(Вікна збереження та завантаження даних).

TabsPanel є контейнером що структурує отриману від сервера інформацію про систему в окремі вкладки. Імплементацію цих вкладок містить пакет Tabs. Для десеріалізації отриманої від сервера інформації і збереження у вигляді класів створені пакети SystemInfo та SystemLoad, більш детальний опис класів у цих пакетах наведено у додатках Д та Е.

Пакет common містить клас ESP32MonitorClient, що містить у собі точку входу застосунку(функція main) а також клас ESP32Message який використовується для десеріалізації отриманих від сервера системних повідомлень.

Повна ієрархія файлів і директорій проєкту наведена на рисунку 4.1.

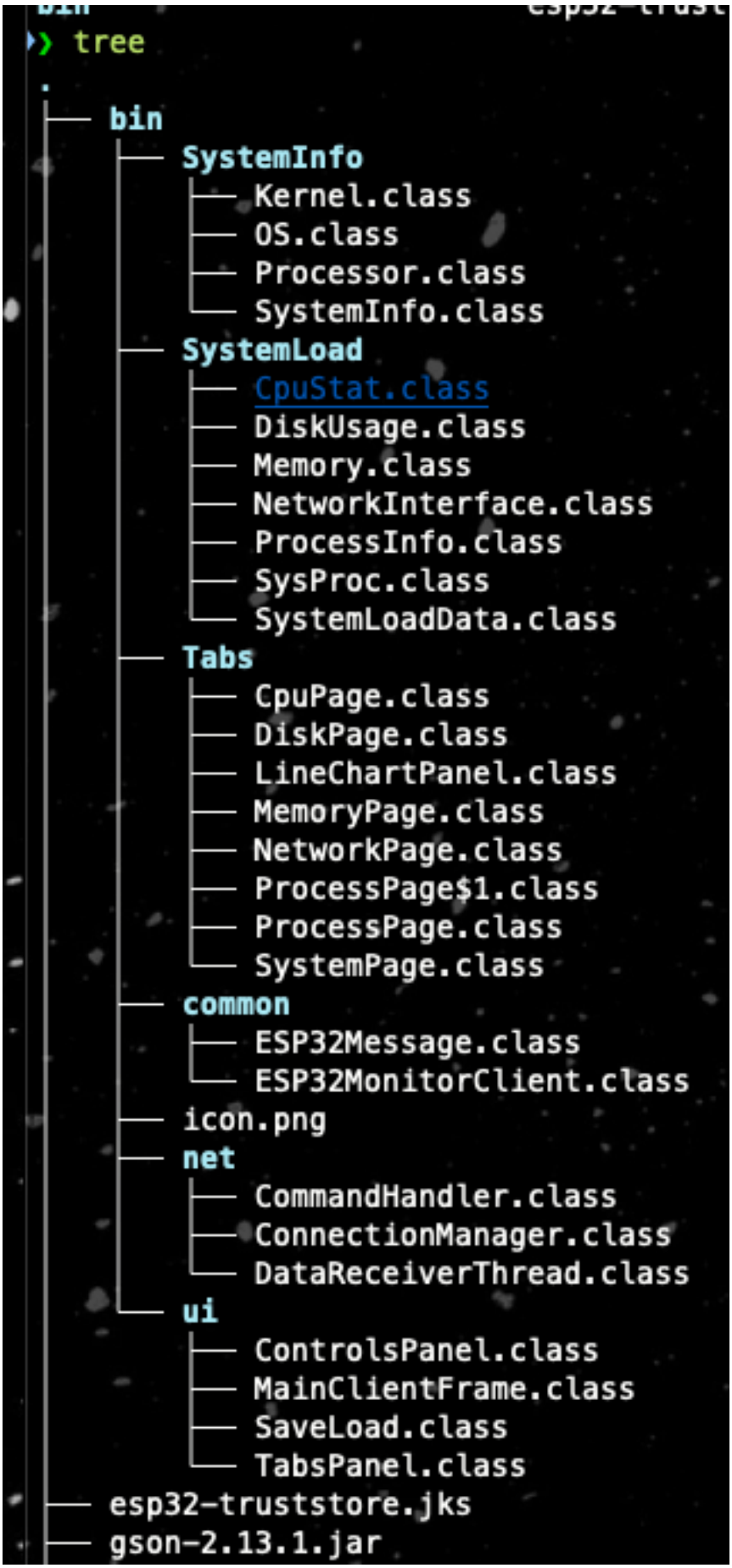

Рисунок 4.1 – Ієрархія файлів і директорій проєкту

4.2 Компоненти графічного інтерфейсу користувача

Графічний інтерфейс користувача складається з основного клієнтського вікна та споміжних діалогових вікон, що з’являються у процесі користування.

При старті програми створюється діалогове вікно з полем введення IP адреси та двома кнопками Connect яка починає TLS рукостискання з сервером за введеною адресою та Exit, яка завершує виконання програми (рисунок 4.2).

Рисунок 4.2 – Вікно вводу адреси мікроконтролера

Якщо з’єднання встановлено успішно запускається основне клієнтське вікно.

Клієнтське вікно поділене на панель керування та інформаційну панель. Інформаційна панель поділена на окремі вкладки, такі як System, Cpu, Memory, Network, Disk та Processes (рисунок 4.3).

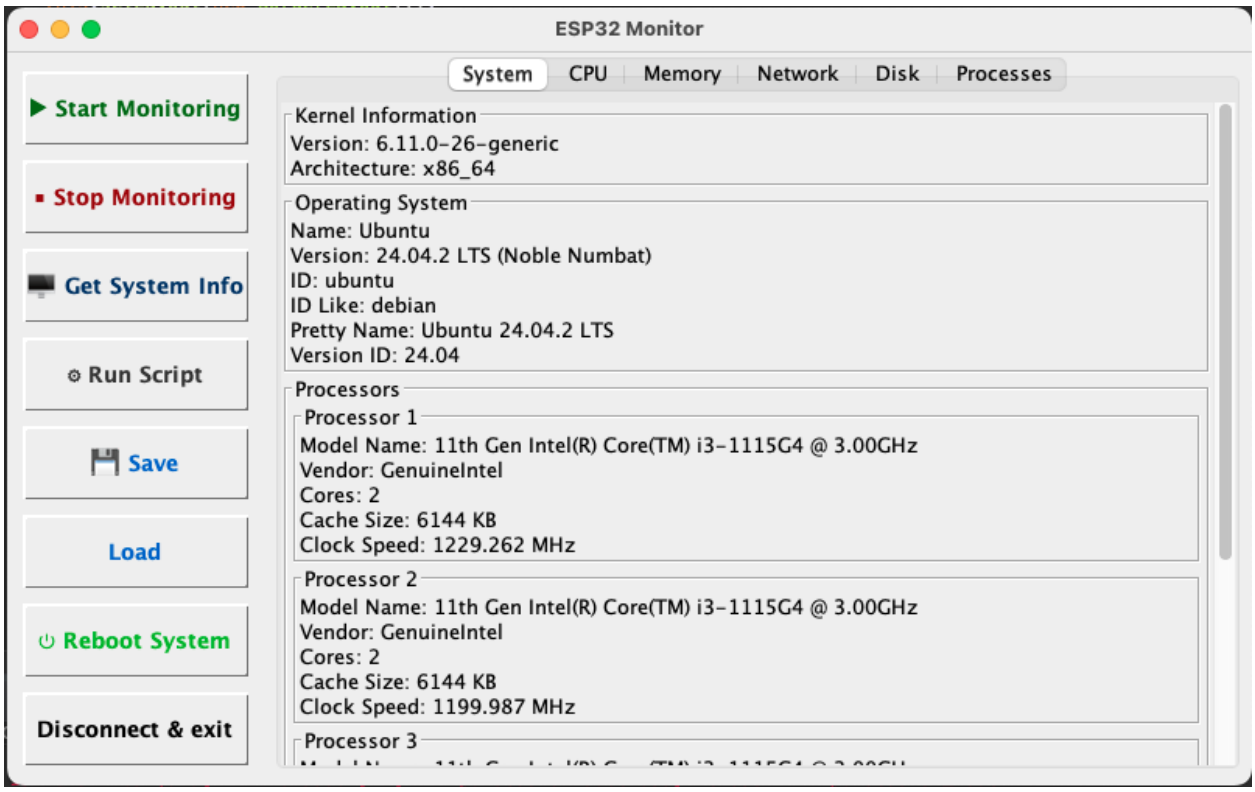

Рисунок 4.3 – Клієнтське вікно програми

Програмно це реалізовано шляхом створення JPanel controlsC та JPanel tabsC. За створення відповідають функції createControls() та createTabs() відповідно. Вікно має BorderLayout, controlsC задана прив’язка до західної частини вікна, для tabsC - центральна частина.

Кожна вкладка містить відповідну до назви отриману від сервера інформацію про цільову систему. Інформація розділена на панелі та має обрамлення з підписом(TitledBorder) для візуального розмежування блоків інформації.

Вкладки CPU (рисунок 4.4) та Memory (рисунок 4.5) містять графіки завантаженності процесора та пам’яті, які представлені об’єктами LineChartPanel.

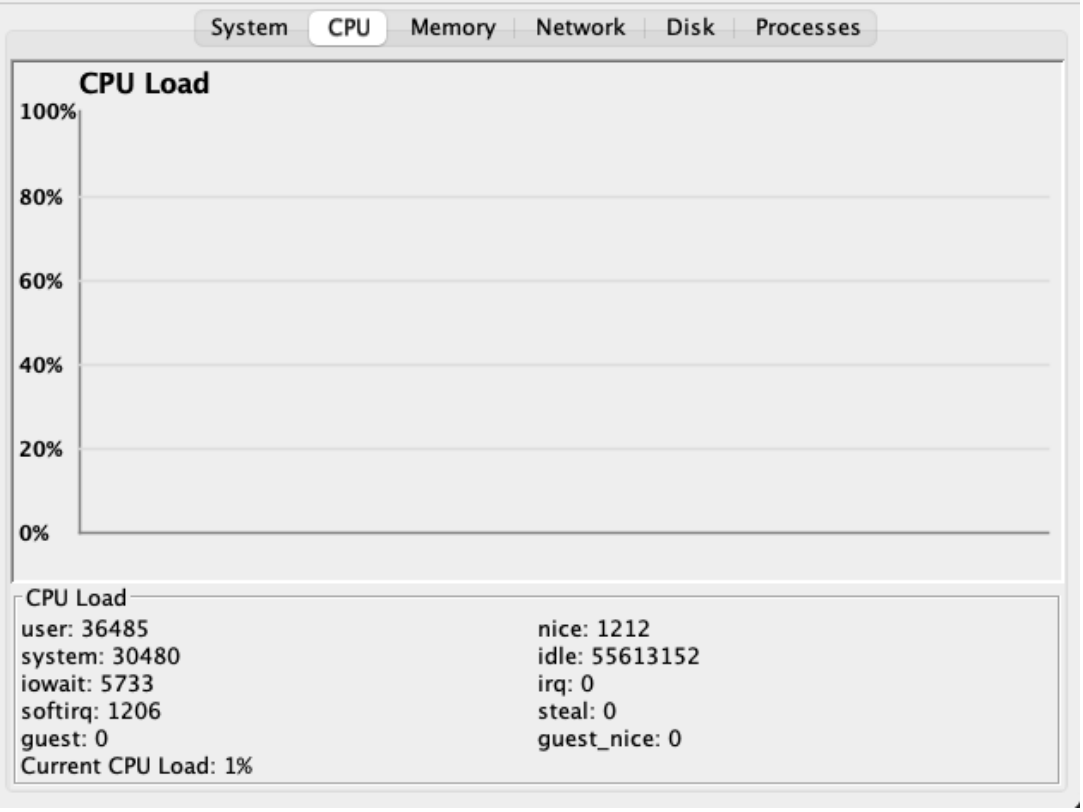

Рисунок 4.4 – Вкладка CPU

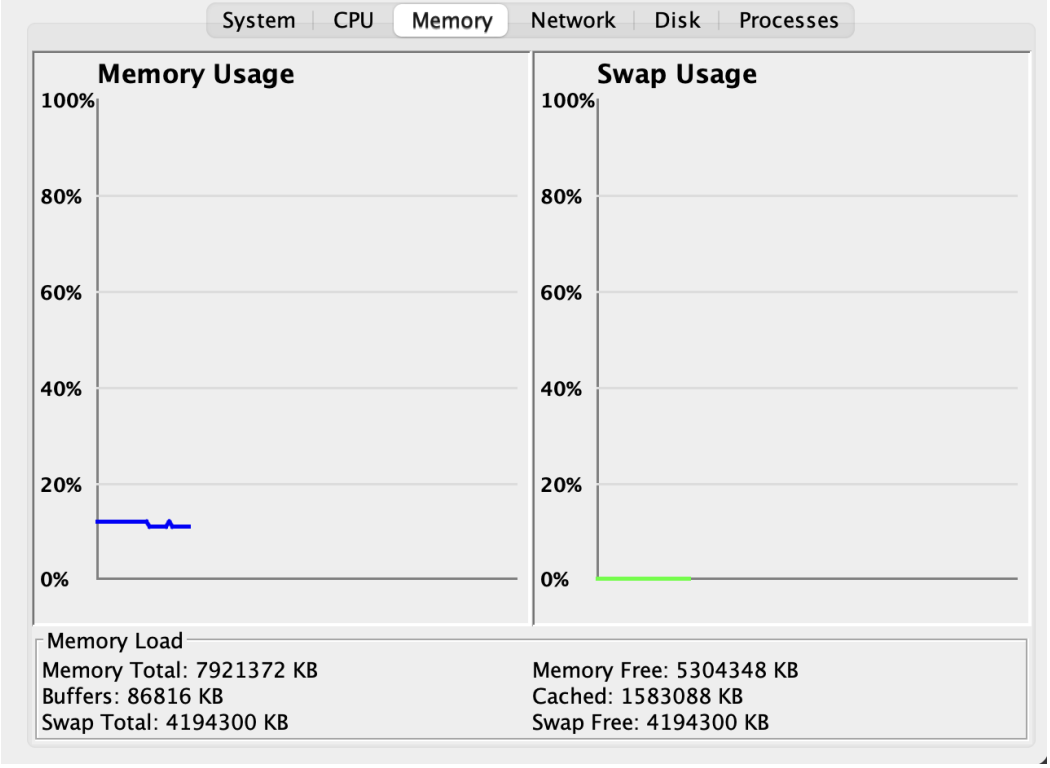

Рисунок 4.5 – Вкладка Memory

Вкладка Disk виводить інформацію про всі диски наявні в системі.

Якщо розмір вікна не дозволяє розмістити всі елементи додається полоса прокрутки (рисунок 4.6).

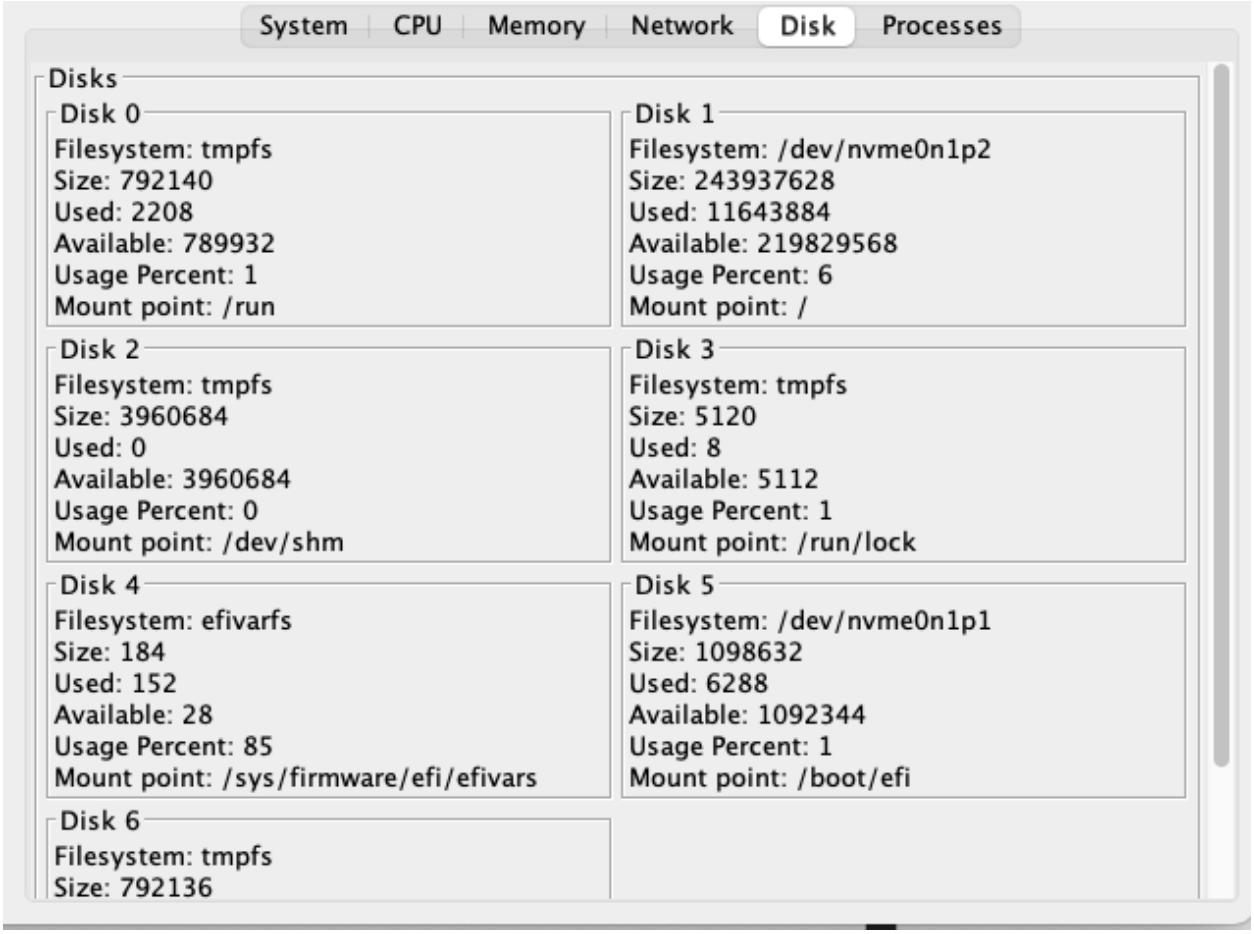

Рисунок 4.6 – Вкладка Disk

Вкладка Networks виводить інформацію про всі наявні мережеві інтерфейси (рисунок 4.7).

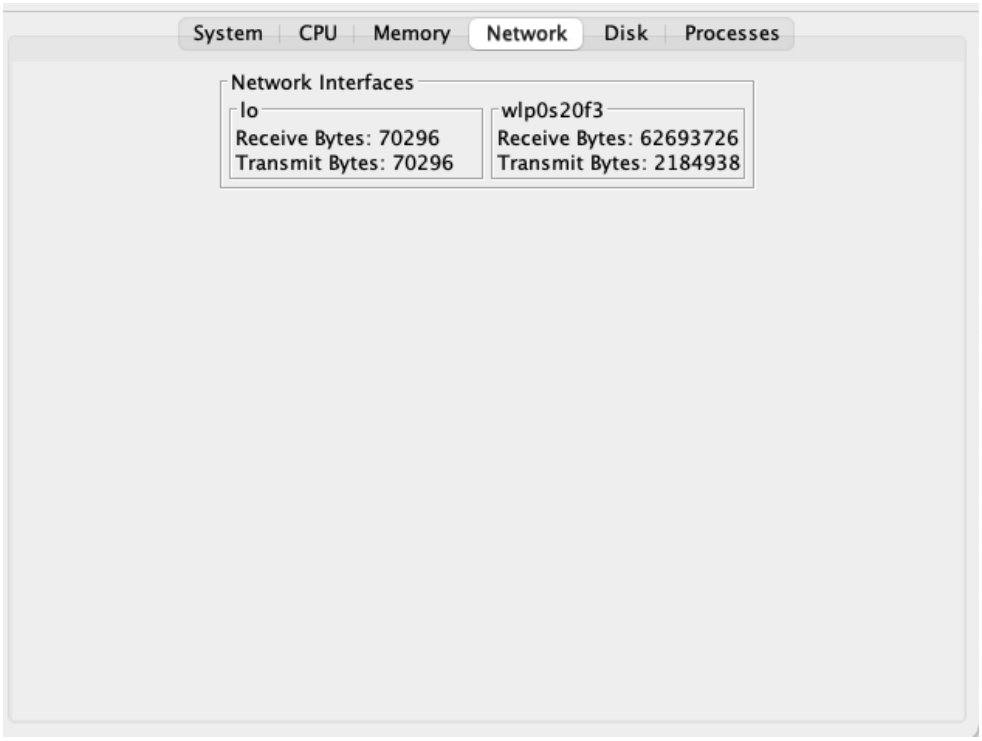

Рисунок 4.7 – Вкладка Network

Processes виводить список виконуваних на комп'ютері процесів у вигляді JList. При подвійному кліку на процесі створюється окреме вікно з більш детальною інформацією про процес, та кнопками Kill process, та ОК (рисунок 4.8).

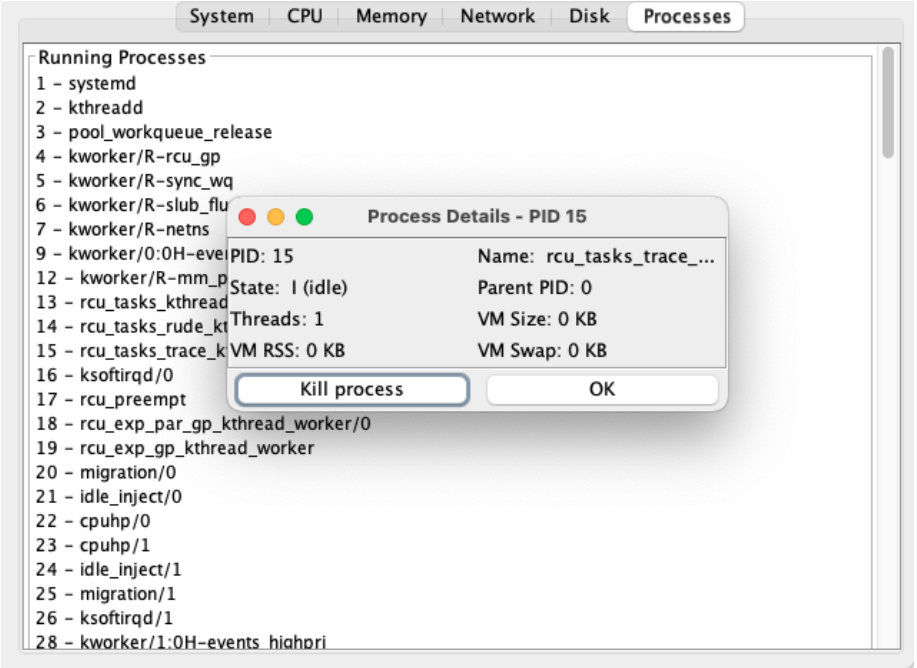

Рисунок 4.8 – Вкладка Processes

4.3 Реалізація розрахунку завантаженості системи та виявлення аномалій

Оскільки сервер повертає завантаженість системи лише в поточний момент часу, для розрахунку проценту завантаженості процесора необхідно розрахувати час використання процесора(t_{total}) та час простою(t_{idle}) і, використовуючи дві точки вимірювання обчислити процент завантаженості за формулою:

$$usage = \frac{\Delta t_{total} - \Delta t_{idle}}{\Delta t_{total}} * 100 \tag{4.1}$$

де t_{total} – загальний час використання процесору($t_{user} + t_{nice} + t_{system} + t_{idle} + t_{iowait} + t_{softirq} + t + t_{steal}$), t_{idle} – час простою($t_{idle} + t_{iowait}$).

В свою чергу інформація про використану пам’ять і swap містять поля total і free, тому розрахунок проценту використання пам’яті виконується за формулою:

$$MemUsage = \frac{M_{total} - M_{free}}{M_{total}} * 100 \tag{4.2}$$

де M_{total} - загальний обсяг пам’яті(Кб), M_{free} – обсяг вільної пам’яті(Кб).

Таким саме чином розраховується процент використання swap.

Програмна реалізація цих розрахунків наведена у класах CpuStat (лістинг 4.1) і Memory (лістинг 4.2).

Лістинг 4.1 – Розрахунок завантаженості процесора(файл CpuStat.java)

```
private long getIdleTime() {
        return idle + iowait;
    }
```

```
    private long getTotalTime() {
        return user + nice + system + idle + iowait + irq +
softirq + steal;
    }

    public static double computeCpuUsage(CpuStat prev, CpuStat
curr) {
        long idleDelta = curr.getIdleTime() -
prev.getIdleTime();
        long totalDelta = curr.getTotalTime() -
prev.getTotalTime();
        if (totalDelta == 0) return 0.0;
        return 100.0 * (totalDelta - idleDelta) / totalDelta;
    }
```

Лістинг 4.2 – Розрахунок завантаженості пам’яті(файл Memory.java)

```
public double getUsedMemoryPercent() {
        long used = memTotal - memFree - buffers - cached;
        return memTotal > 0 ? 100.0 * used / memTotal : 0;
    }

public double getUsedSwapPercent() {
        long usedSwap = swapTotal - swapFree;
        return swapTotal > 0 ? 100.0 * usedSwap / swapTotal : 0;
    }
```

В контексті даної роботи під аномалією використання ресурсів комп’ютера розуміється висока заванаженість процесора або пам’яті(>95%) протягом певного часу(> 1 хвилини). Тому клієнтський застосунок постійно перевіряє процент завантаженості на графіку і у разі виявлення більш ніж 12 записів(що дорівнює одній хвилині, оскільки нові записи з’являються кожні 5 секунд), де завантаженість перевищує 95% користувачу виводиться повідомлення про виявлену аномалію (лістинг 4.3).

Лістинг 4.3 – Функція перевірки аномалій(файл MainClientFrame.java)

```
public void checkForAnomalies()
     {
          List<Integer> cpuLoadPercents =
cpuLoadPanel.getDataPoints();
          List<Integer> memoryLoadPercents =
memUsagePanel.getDataPoints();
          List<Integer> swapUsagePercents =
swapUsagePanel.getDataPoints();
```

```
          if(cpuLoadPercents.size() > 12 &&
memoryLoadPercents.size() > 12 && swapUsagePercents.size() > 12)
          {
               int cpucnt = 0;
               for(int i = cpuLoadPercents.size(); i >
cpuLoadPercents.size() - 12; i--)
               {
                    if(cpuLoadPercents.get(i) > 95) cpucnt++;
               }
               if(cpucnt > 10 && !ignore)
               {
                    JOptionPane.showMessageDialog(null, "Cpu
load anomaly detected");
                    ignore = true;
               }
               else if (cpucnt < 6) ignore = false;

               int memcnt = 0;
               for(int i = memoryLoadPercents.size(); i >
memoryLoadPercents.size() - 12; i--)
               {
                    if(memoryLoadPercents.get(i) > 95) memcnt++;
               }
               if(memcnt > 10 && !ignore)
               {
                    JOptionPane.showMessageDialog(null, "Memory
usage anomaly detected");
                    ignore = true;
               }
               else if (memcnt < 6) ignore = false;

               int swpcnt = 0;
               for(int i = swapUsagePercents.size(); i >
swapUsagePercents.size() - 12; i--)
               {
                    if(swapUsagePercents.get(i) > 95) swpcnt++;
               }
               if(swpcnt > 10 && !ignore)
               {
                    JOptionPane.showMessageDialog(null, "Swap
usage anomaly detected");
                    ignore = true;
               }
               else if (swpcnt < 6) ignore = false;
          }
     }
```

5 ТЕСТУВАННЯ ТА ІНСТРУКЦІЯ КОРИСТУВАЧА

На цільовому комп’ютері повинен бути встановлений інтерпретатор мови Python а також модуль Pyserial. Необхідно запустити скрипт з правами адміністратора(sudo python3 script.py). Якщо пристрій вже під’єднано скрипт автоматично почне конфігурувати Wi-Fi, якщо ні то користувачу буде запропоновано під’єднати пристрій і натиснути клавішу Enter (рисунок 5.1).

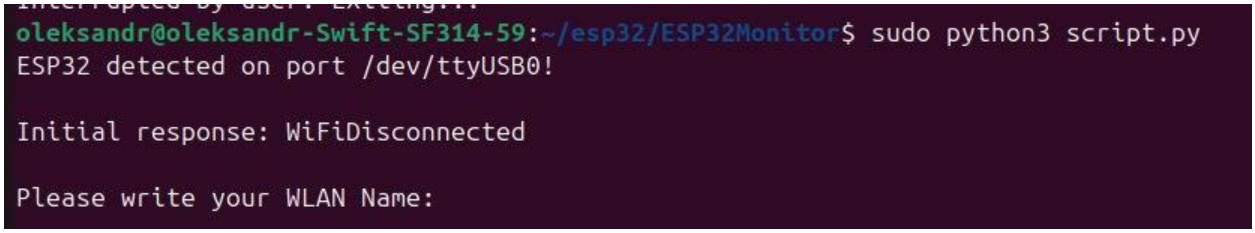

Рисунок 5.1 – Налаштування на цільовому комп’ютері

Після введення SSID та паролю мікроконтролер встановить підключення до мережі Wi-Fi, вбудований LED-індикатор засвітиться синім а в консоль виведеться IP адреса пристрою і повідомлення “Connected successfully” (рисунок 5.2).

Рисунок 5.2 – Результат успішного підключення до Wi-Fi

На цьому налаштування на цільовому комп’ютері завершені і можна перейти до клієнтського застосунку. Після старту клієнту відкриється діалогове вікно де користувачу запропонується ввести IP адресу пристрою (рисунок 5.3).

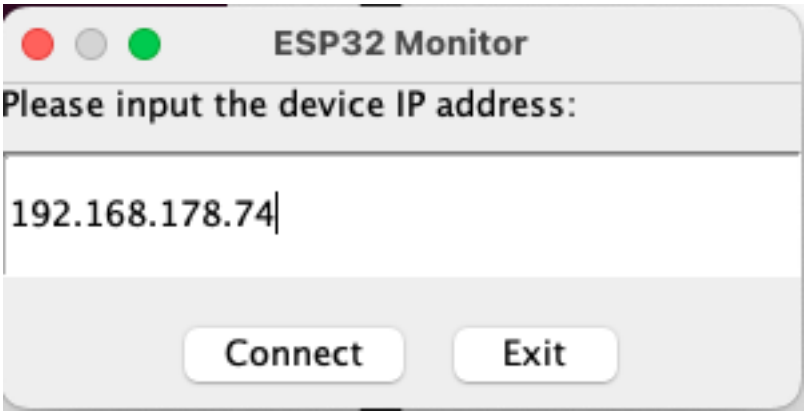

Рисунок 5.3 – Вікно введення IP адреси

У разі успішного з'єднання виведеться вікно з надписом «TLS connection established successfully» і відкриється основне клієнтське вікно (рисунок 5.4).

Рисунок 5.4 – Головне вікно програми

Керування виконується за допомогою розташованих у лівій частині вікна кнопок.

Кнопка Get System Info виводить загальну інформацію про систему,

таку як інформація про ядро системи, детальну інформацію про встановлену ОС, та характеристики процесора.

Кнопка Start Monitoring запускає моніторинг системи, інформація доступна у відповідних вкладках CPU, Memory, Network, Disk та Processes. Кнопка Stop Monitoring зупиняє процес моніторингу (рисунок 5.5).

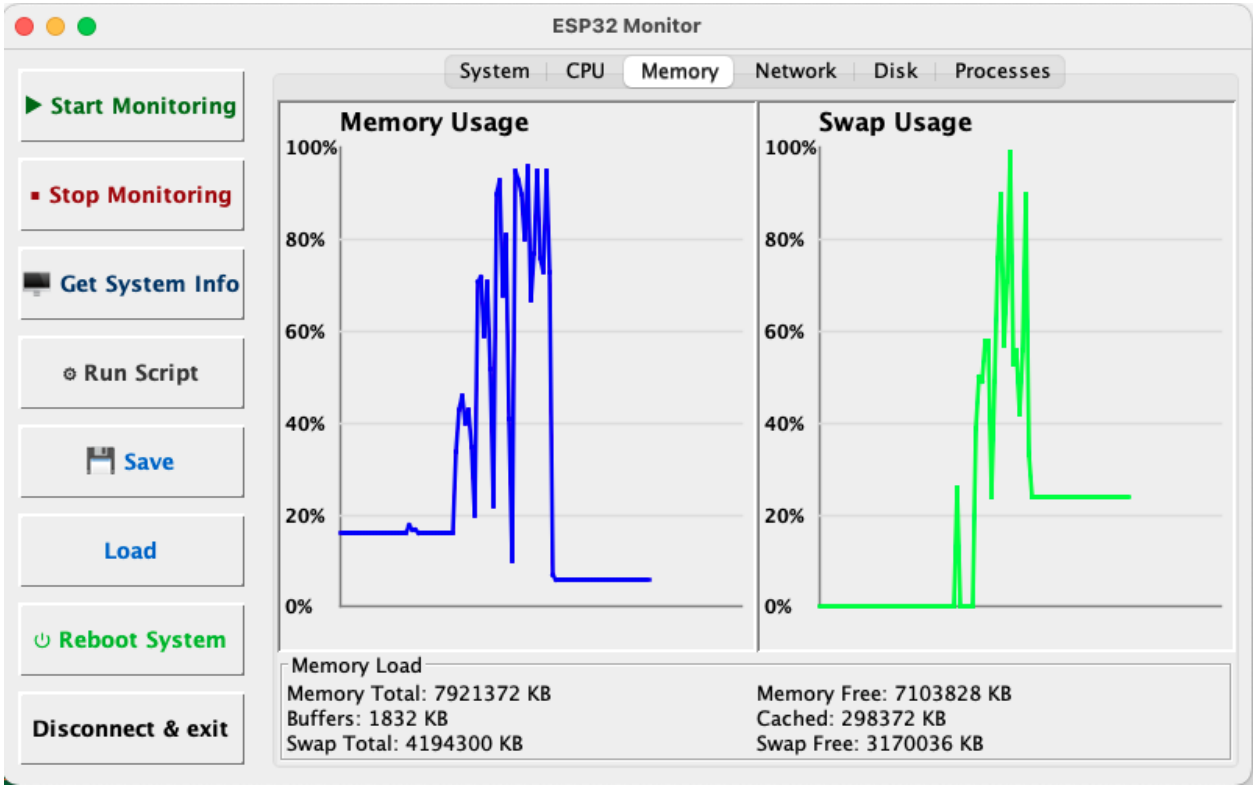

Рисунок 5.5 – Приклад моніторингу пам’яті цільової системи

У вкладці Processes знаходиться список виконуваних процесів. Подвійним кликом на процес можна отримати більш детальну інформацію або вбити процес (рисунок 5.6).

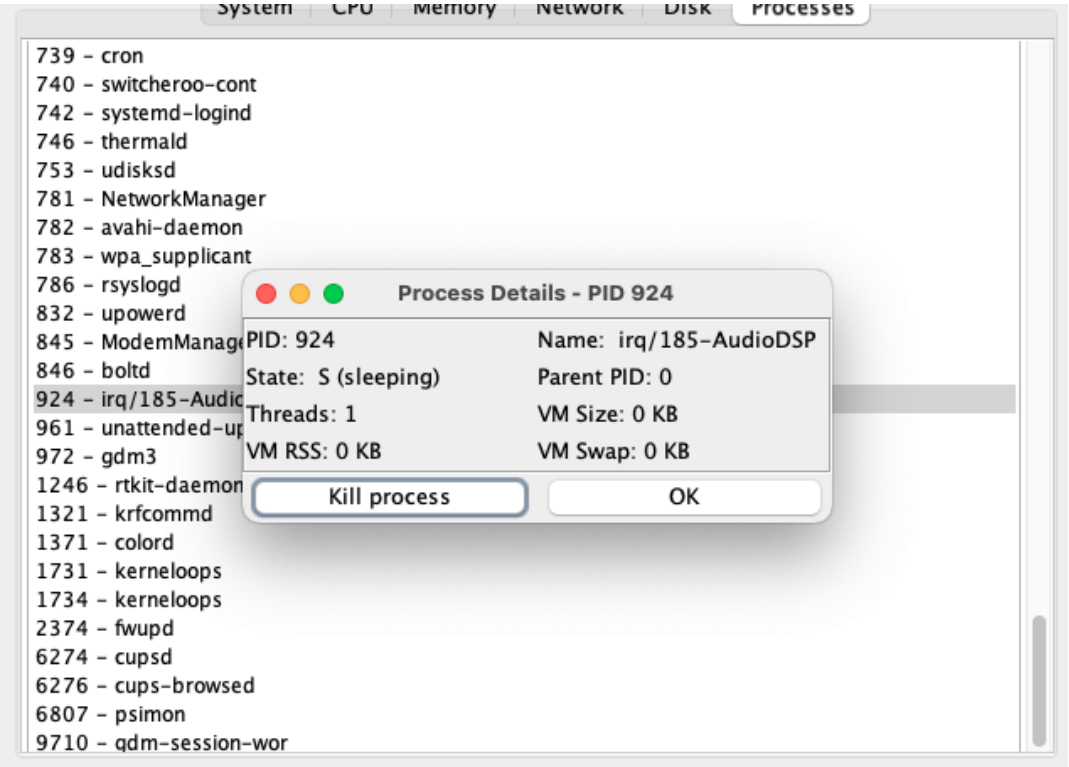

Рисунок 5.6 – Детальна інформація про обраний процес

Симулювати аномальне використання процесора можна за допомогою утиліти stress. Якщо процесор завантажений на 100 відсотків довше ніж хвилину на екран виведеться відповідне вікно (рисунок 5.7).

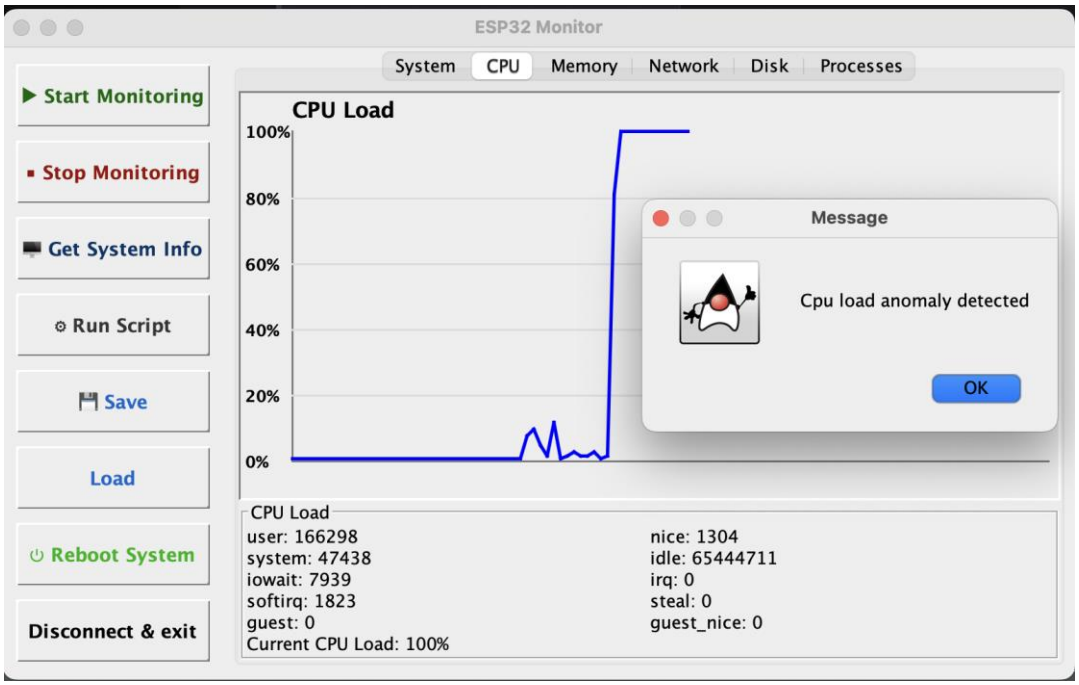

Рисунок 5.7 – Повідомлення про аномальне використання ресурсів

Таким саме чином у разі занадто високого використання пам’яті клієнт виведе відповідне повідомлення.

Якщо система перестала відповідати(виконання команди займає більше 10 секунд), виведеться вікно System not responding. Користувачу запропонується закрити застосунок або спробувати знову (рисунок 5.8).

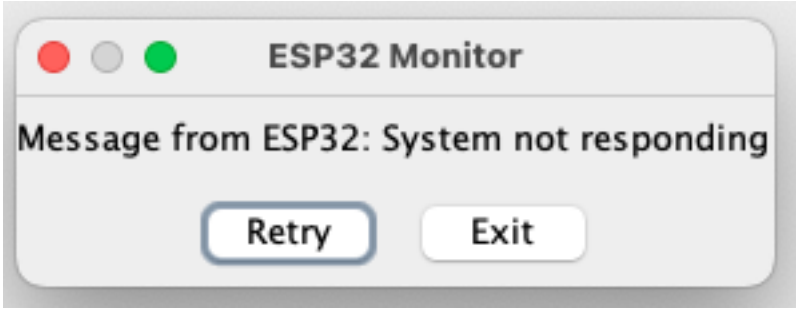

Рисунок 5.8 – Повідомлення від серверу про відсутність реакції системи

Якщо система знеживлена і пристрій переключився на живлення від акумулятору користувач буде також про це повідомлений (рисунок 5.9).

Рисунок 5.9 – Повідомлення від серверу про відсутність живлення

Кнопка Run Script дозволяє користувачу виконувати shell-скрипти. Після натискання на кнопку запускається діалогове вікно де користувачу пропонується ввести абсолютний шлях розташування скрипта. Після натискання на кнопку Execute скрипт будет виконано з адміністраторськими привілеями а користувач буде проінформований про успішне або неуспішне виконання скрипта (рисунок 5.10).

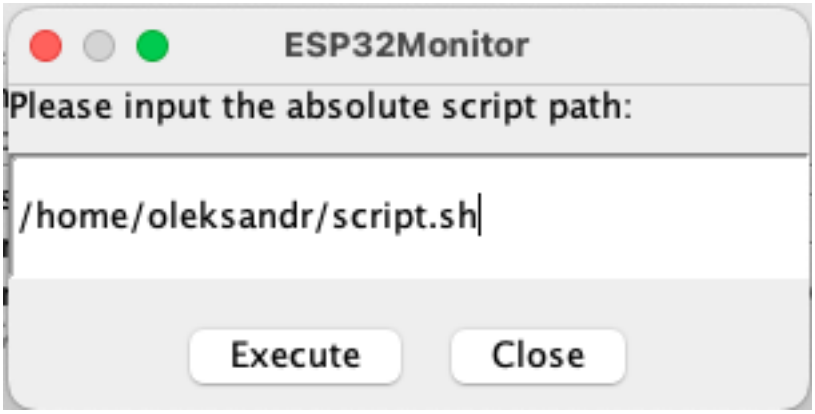

Рисунок 5.10 – Функція виконання скрипта

Кнопка Save відкриває стандартне діалогове вікно де можна вибрати шлях та ім’я файлу збереження. Після чого наявна інформація буде збережена у файл і може буде завантаження через кнопку Load (рисунок 5.11).

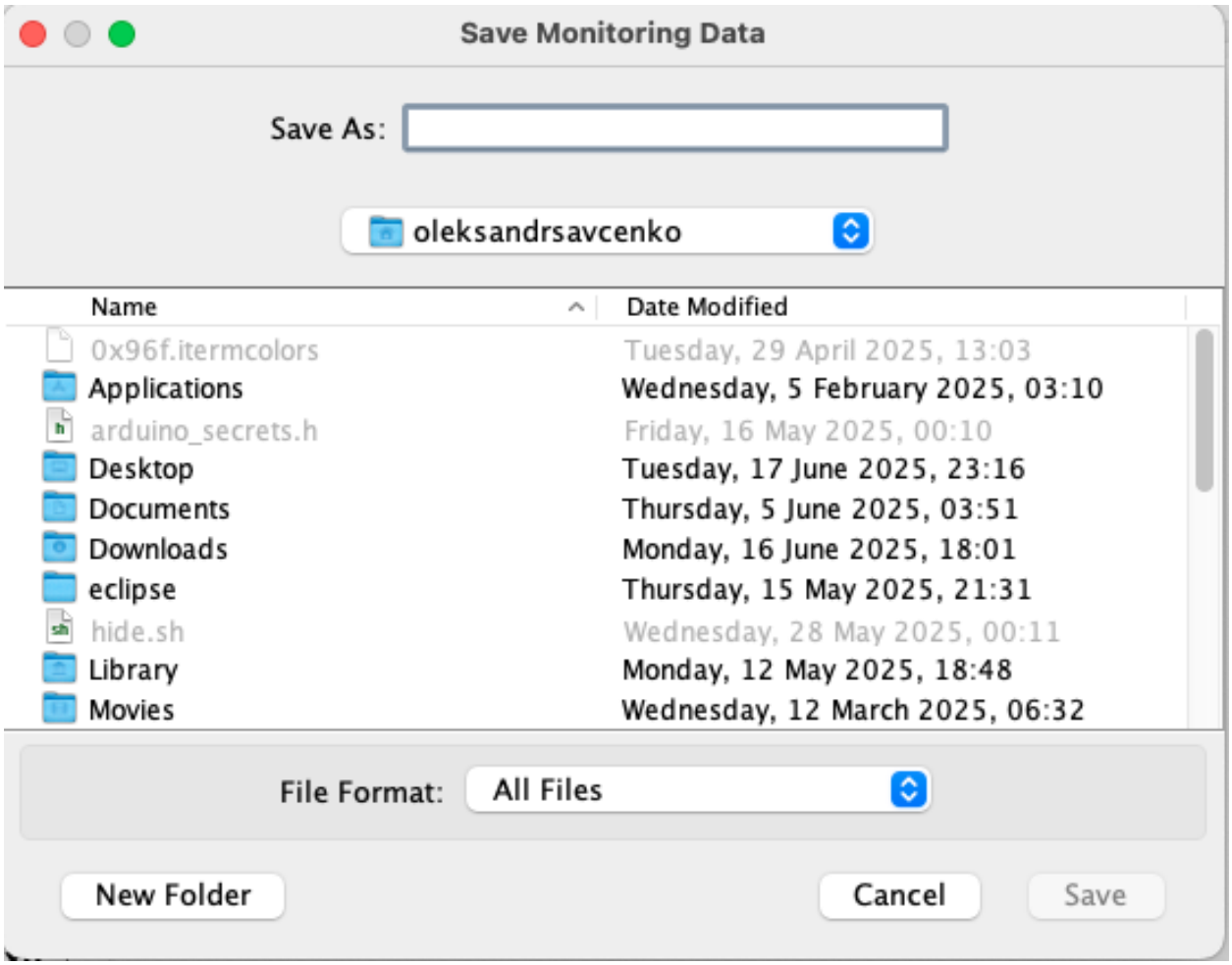

Рисунок 5.11 – Вікно збереження файлу

Кнопка Reboot System виконує перезавантаження комп’ютера, а кнопка Disconnect & Exit зупиняє моніторинг, виключає клієнт і завершує виконання програми.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи повністю досягнуто поставлену мету – розроблено програмно-апаратний комплекс для дистанційного моніторингу та керування комп’ютерною системою з використанням мікроконтролера ESP32. Система забезпечує збір ключових параметрів стану комп’ютера(Навантаження окремих компонентів, наприклад CPU, пам’ять, дисковий простір і т.д), їх обробку та передачу до клієнтського застосунку в реальному часі, а також реалізує можливість керування процесами та живленням з боку користувача.

У межах проєкту було реалізовано:

- створення скрипта на Python який забезпечує обмін даними між мікроконтролером та цільовою системою;
- розробку серверної частини на мікроконтролері ESP32 з використанням FreeRTOS, TLS, UART та JSON;
- створення клієнтського застосунку на Java з графічним інтерфейсом, що дозволяє візуалізувати моніторингові дані у вигляді таблиць та графіків;
- впровадження системи безпечного обміну даними через TLS та оптимізовану багатопоточну архітектуру як на стороні мікроконтролера, так і клієнтської програми.

У порівнянні з існуючими вітчізняними та зарубіжними аналогами, розроблена система має низку переваг:

- моніторинг здійснюється незалежно від системи, і пристрій здатний відстежувати аномальне використання ресурсів комп’ютера та інформувати користувача у випадках якщо немає відповіді з боку системи;
- має власне джерело живлення що дозволяє вчасно повідомляти користувача у випадках коли обчислювальна система була знеструмлена;
- застосунок має інтуітивно зрозумілий інтерфейс та є простим у використанні.

Подальша робота в цьому напрямку може включати:

- розширення функціональності клієнтського застосунку, зокрема, додавання сповіщень користувача у вигляді SMS-повідомлень або повідомлень через Email;

- додавання більш гнучких налаштувань системи моніторингу, наприклад, задання власного порогу аномального використання ресурсів, вибір власного інтервалу замірів для моніторинга;

- адаптацію системи для використання на серверному обладнанні або в умовах обмеженого доступу(наприклад дата-центри, промислові об’єкти).

Практичне використання результатів цієї роботи можливе як у повсякденному адмініструванні систем, так і в навчальному процесі. Розроблений комплекс може бути застосований як приклад побудови інтегрованих систем з використанням мікроконтролерів, засобів захищеної комунікації та міжплатформного програмного забезпечення. Зокрема, його доцільно використовувати в навчальних дисциплінах, пов’язаних із мікроконтролерами, системним програмуванням, мережевими технологіями та операційними системами.

Таким чином, виконана кваліфікаційна робота підтверджує практичну доцільність застосування мікроконтролера ESP32 для побудови систем моніторингу та дистанційного керування, а також демонструє високий рівень інтеграції апаратної та програмної складових у межах сучасних обчислювальних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is RMM (Remonte Monitoring and Management)? Amazon Web Services. URL: https://aws.amazon.com/what-is/remote-monitoring-and-management/. (дата звернення 18.05.2025)

2. Загальні відомості про Arduino. Державний університет «Житомирська політехника». URL: https://learn.ztu.edu.ua/pluginfile.php/336877/mod_folder/content/0/%D0%97%D0%B0%D0%B3%D0%B0%D0%BB%D1%8C%D0%BD%D1%96%20%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BE%D1%81%D1%82%D1%96%20%D0%BF%D1%80%D0%BE%20Arduino.pdf (дата звернення 20.05.2025)

3. What is PlatformIO. PlatformIO. URL: https://docs.platformio.org/en/latest/what-is-platformio.html (дата звернення 20.05.2025)

4. Write Once, Run Anywhere. Old Dominion Univ. URL: https://www.cs.odu.edu/~zeil/cs382/latest/Public/runAnywhere/index.html (дата звернення 20.05.2025)

5. A guide to Java Sockets. Baeldung. URL: https://www.baeldung.com/a-guide-to-java-sockets (дата звернення 18.06.2025)

6. Establish Serial Connection with ESP32. Espressif Systems(Shanghai) Co. URL: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/establish-serial-connection.html?utm_source=chatgpt.com (дата звернення 21.05.2025)

7. ESP32 UART Cpmmunication using ESP-IDF. ESP32Tutorials. URL: https://esp32tutorials.com/esp32-uart-tutorial-esp-idf/ (дата звернення 18.06.2025)

8. Вступ у JSON. JSON. URL: https://www.json.org/json-uk.html (дата звернення 21.05.2025)

9. Gson documentation. Gson. URL: https://google.github.io/gson/ (дата

звернення 22.05.2025)

10. cJSON documentation. Dave Gamble. URL: https://github.com/DaveGamble/cJSON (дата звернення 18.06.2025)

11. Kerry A. McKay, David A. Cooper. Guidelines for Selection, Configuration, and Use of Transport Layer Security(TLS) Implementations: NIST SP 800-52 Rev. 2 : U.S. Department of Commerce, 2019. 63 с.

12. ESP32: TLS(Transport Layer Security) And IoT Devices. Kedar Sovani. URL: https://developer.espressif.com/blog/esp32-tls-transport-layer-security-and-iot-devices/ (дата звернення 22.05.2025)

13. Processes and Threads. Oracle Java Documentation. URL: https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html (дата звернення 22.05.2025)