МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.
«_____»_____2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
«БАКАЛАВР»

Тема: «Програмний модуль моніторингу апаратних та програмних ресурсів комп'ютера на основі *Arduino*»

Виконавець: _____ Радченко А.С.

Керівник: _____ Нечипорук О.П.

Нормоконтролер: _____ Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри

Литвиненко О.Є.
«_____» _____ 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи (проєкту)

Радченко Анастасії Сергіївни

1. Тема дипломної роботи (проекту) «Програмний модуль моніторингу апаратних та програмних ресурсів комп'ютера на основі Arduino»

затверджена наказом ректора від « 04 » лютого 2021 р. № 135/ст.

2. Термін виконання роботи (проєкту): з 17 травня 2021 р. по 20 червня 2021 р.

3. Вихідні дані до роботи (проєкту): постановка задачі до виконання роботи, плата *Arduino* мікроконтролера на базі, мова програмування *C#, C++,* середовище програмування *MS Visual Studio 2017* та *Qt Creator*

4. Зміст пояснювальної записки:

1) Створення програмних модулів моніторингу апаратних та програмних ресурсів ПК

2) Проектування програмного модуля моніторингу апаратних та програмних ресурсів на базі *Arduino*

3) Реалізація програмного модуля моніторингу апаратних та програмних ресурсів комп'ютера на основі *Arduino*

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Скріншот вкладки «*Overview*»

2) Скріншот вкладки «*Monitoring Tools*»

3) Скріншот вкладки «*Monitoring Tools Global*»

4) Фото додаткового монітору на базі *Arduino*

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Ознайомитись з постановкою задачі дипломного проектування	17.05.2021	
2.	Вивчити спеціальну літературу і технічну документацію	17.05.2021-19.05.2021	
3.	Огляд і порівняння систем відслідковування задач у проектах	20.05.2021-24.05.2021	
4.	Розробка архітектури програмного модуля моніторингу апаратних та програмних ресурсів комп'ютера на основі *Arduino*	28.05.2021-31.05.2021	
5.	Створення програмного модуля моніторингу апаратних та програмних ресурсів комп'ютера на основі *Arduino*	31.05.2021-06.06.2021	
6.	Написання пояснювальної записки	07.06.2021-14.06.2021	
7.	Підготувати графічний демонстраційний матеріал та доповідь	9.06.21-11.06.21	

7. Дата видачі завдання: «_17_» _травня_ 2021 р.

Керівник дипломного проекту _____ д.т.н., доц. Нечипорук О.П._
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Радченко А.С. _____
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Програмний модуль моніторингу апаратних та програмних ресурсів комп’ютера на основі *Arduino*»: 60 с., 46 рис., 3 табл., 26 літературних джерела, 3 додатка.

ПРОГРАМНИЙ МОДУЛЬ, МОНІТОРИНГ, ПЛАТФОРМА *ARDUINO*.

Об’єктом дипломного проектування є процес моніторингу ресурсів ПК за допомогою програмного забезпечення.

Предмет дипломного проектування – програмний модуль моніторингу апаратних та програмних ресурсів комп’ютера на основі *Arduino*.

Мета дипломного проекту: розробити програмний модуль апаратних та програмних ресурсів ПК для їхнього моніторингу.

Метод дослідження – при вирішенні поставлених завдань було використано теоретичні знання та практичні надбання в галузі програмного забезпечення моніторингу ресурсів ПК; для дослідження проектування програми, було використано знання у галузі потоків, у галузі теорії щодо навантаження ЦП та ОЗП; знання у галузі отримання інформації щодо комплектуючих та їх характеристик.

Прогнозні припущення про розвиток об’єкту та предмету дослідження – створення робочого зразка програми та використання його для проведення моніторингу ресурсів персонального комп’ютера.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ……..…..7
ВСТУП……..8
РОЗДІЛ 1 СТВОРЕННЯ ПРОГРАМНИХ МОДУЛІВ МОНІТОРИНГУ АПАРАТНИХ ТА ПРОГРАМНИХ РЕСУРСІВ ПК…….....….....10
1.1. Теоретичні основи створення програмних модулів моніторингу апаратних та програмних ресурсів ПК……..….10
1.2. Огляд та порівняння програмних модулів моніторингу апаратно-програмних ресурсів ПК…….……20
1.3. Висновки до розділу……..…..28
РОЗДІЛ 2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ МОНІТОРИНГУ АПАРАТНИХ ТА ПРОГРАМНИХ РЕСУРСІВ НА БАЗІ *ARDUINO*……29
2.1. Теоретичні основи створення програмних модулів моніторингу апаратних та програмних ресурсів ПК……...29
2.2. Технології основної програми……..……...…30
2.3. Технології програми для отримання температур…….……...… 39
2.4. Технології програми для прошивки мікроконтролера на базі плати *Arduino*……..……...……41
2.5. Висновки до розділу……...……..42
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ МОНІТОРИНГУ АПАРАТНИХ ТА ПРОГРАМНИХ РЕСУРСІВ КОМП’ЮТЕРА НА ОСНОВІ *ARDUINO*……...…….…..44
3.1. Функціонал готового програмного модуля моніторингу апаратних та програмних ресурсів комп’ютера на основі *Arduino*…….....…...44
3.2. Принцип роботи програмного модуля моніторингу апаратних та програмних ресурсів комп’ютера на основі *Arduino*……..……...46
3.3. Принцип роботи програми отримання температур……..…….53

3.4. Принцип роботи прошивки для плати *Arduino* та схеми на базі *Arduino*……..…..53
3.5. Висновки до розділу……....……55
ВИСНОВКИ……...56
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ……...……..58
Додаток А……...61
Додаток Б……..…..67
Додаток В……..…….72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПММАПР – програмний модуль моніторингу апаратно-програмних ресурсів.

ПК – персональний комп’ютер.

ЦП – центральний процесор.

ГП – графічний процесор.

BIOS – Basic Input/Output System

ОЗП – оперативний запам’ятовуючий пристрій.

БЖ – блок живлення.

DHCP - Dynamic Host Configuration Protocol.

ОС – операційна система.

ООП – об’єктно орієнтоване програмування.

ККД – коєфіцієнт корисної дії.

Arduino – open-source платформа на базі мікроконтролеру, створена для розробки пристроїв з мікроконтролерами.

Снапшот – статичний зріз стану деякого пристрою, механізму (наприклад, бази даних або процесора) в певні момент часу.

Лог – файл із записами про події в хронологічному порядку, найпростіший засіб забезпечення журналювання.

Сеттер – функція, що приймає дані як аргументи та записує дані у відповідні зміні класу, реалізується для збереження принципів інкапсуляції.

ВСТУП

Актуальність теми проекту. У наш час розвиток комп'ютерних технологій сягнув цілковито нових масштабів. Електронно-обчислювальні машини увійшли у всі сфери людської діяльності. Майже кожне підприємство, установа або ж організація незалежно від сфери діяльності вже неможливо уявити без комп'ютерної техніки. Наразі таке устаткування є такою ж невід'ємною частиною будь-якого офіса як і, наприклад, телефон. З огляду на це виникла необхідність моніторингу ресурсів комп'ютерів, оскільки вони виходили із ладу або не були оптимізовані. Принцип моніторингу залишається незмінним – це здійснення контролю за комп'ютером. У зв'язку з цим з'явилась і нова професія – системний адміністратор, функцією якої є налагодження комп'ютера для максимально стабільної безперебійної його роботи без загрози пошкодження його компонентів. Раніше таку діяльність не так і легко було здійснити, оскільки не існувало необхідного програмного забезпечення, яке задовольняло би зазначені потреби, але завдяки тому, що наука не стоїть на місці, з'явились сучасні технології та програми. Моніторинг ресурсів персонального комп'ютера став набагато простішим. У всіх сучасних операційних системах існують стандартні програми моніторингу, проте і досі є аспекти, які потребували б вдосконалення.

Об'єктом дослідження є процес моніторингу ресурсів ПК за допомогою програмного забезпечення.

Предмет дослідження – програмний модуль моніторингу апаратних та програмних ресурсів комп'ютера на основі *Arduino*.

Мета дипломного проекту: розробка програмного модулю для моніторингу апаратних та програмних ресурсів ПК.

Виходячи з поставленої мети роботи, для її досягнення необхідно вирішити такі завдання:

– аналіз комплектуючих ПК;
– аналіз характеристик комплектуючих ПК;
– розробка методів моніторингу характеристик комплектуючих;

– розробка методів отримання характеристик комплектуючих;

– розробка методів моніторингу навантаження на основні комплектуючі (ЦП та ОЗП);

– реалізація методів моніторингу навантаження на ЦП та ОЗП (глобально та від конкретних процесів)

– реалізація методів отримання характеристик комплектуючих;

Методи дослідження. При вирішенні поставлених завдань було використано теоретичні знання та практичні надбання в галузі програмного забезпечення моніторингу ресурсів ПК; для дослідження проектування програми, було використано знання у галузі потоків, у галузі теорії щодо навантаження ЦП та ОЗП; знання у галузі отримання інформації щодо комплектуючих та їх характеристик.

Одержані висновки та їх новизна. В результаті досліджень було досягнуто таких нововведень: моніторинг навантаження на ЦП та ОЗП від кожного процесу та оповіщення користувача щодо цієї події; моніторинг навантаження на ЦП та ОЗП в цілому та оповіщення користувача щодо цієї події; можливість не тримати програму відкритою на основному екрані ПК за рахунок виводу інформації на зовнішній дисплей через *Arduino*.

Результати досліджень можуть бути застосовані як у повсякденному житті для пересічного користувача ПК, так і для професійної роботи.

РОЗДІЛ 1
СТВОРЕННЯ ПРОГРАМНИХ МОДУЛІВ МОНІТОРИНГУ АПАРАТНИХ ТА ПРОГРАМНИХ РЕСУРСІВ ПК

1.1. Теоретичні основи створення програмних модулів моніторингу апаратних та програмних ресурсів ПК

Програмні модулі моніторингу апаратно-програмних ресурсів ПК існують з метою надання користувачу ПК інформації щодо стану програмних та/або апаратних ресурсів даного ПК. До апаратних ресурсів відносять такі ресурси як: назва компонентів ПК, їх характеристики, температура та ін. Наприклад, апаратними ресурсами ПК буде назва ЦП, його частота, модель, кількість ядер, температура ЦП на поточний момент або назва та місткість жорсткого диску, його температура.

Основні елементи сучасного ПК:

1. ЦП;
2. ГП;
3. ОЗП;
4. Жорсткий диск;
5. Материнська/головна плата;
6. БЖ;
7. Система охолодження.

Нище наведено опис основних елементів.

Центральний процесор (ЦП) – електронний блок (або інтегральна схема), що виконує машинні інструкції (тобто безпосередньо код програми), головна частина апаратного забезпечення комп’ютера. До складу процесора входять: пристрій управління (УП), арифметично-логічний пристрій (АЛУ),

запам’ятовуючий пристрій (ЗП) на основі регістрів процесорної пам’яті та кеш-пам’яті процесора, генератор тактової частоти (ГТЧ) [10].

Пристрій управління організовує процес виконання програм та координує взаємодію всіх пристроїв ЕОМ під час її роботи [10].

В даний час немає строгого визначення АЛП, що викликає деяку плутанину при користуванні різною літературою [3, с. 176]. АЛП виконує арифметичні та логічні операції над даними: складання, віднімання, множення, ділення, порівняння тощо [10].

Запам’ятовуючий пристрій – це внутрішня пам’ять процесора. Регістри слугують проміжною швидкою пам’яттю, використовуючи яку процесор виконує розрахунки та зберігає проміжкові результати. Для пришвидшення роботи з оперативною пам’яттю використовують кеш-пам’ять, в яку з випередженням підкачуються команди і дані з оперативної пам’яті, необхідні процесору для наступних операцій [10].

Генератор тактової частоти генерує електричні імпульси, що синхронізують роботу всіх вузлів комп’ютера. У ритмі ГТЧ працює центральний процесор [10].

Рис. 1.1. Зовнішній вигляд сучасних ЦП

Зовнішній вигляд сучасних ЦП наведено на рис. 1.1. На верхній частині рис. 1.1 зображено ЦП фірми *“Intel”*, на нижній – ЦП фірми *“AMD”*.

Основні характеристики ЦП:

а) тактова частота – вимірюється у мега- та гігагерцах (*МГц, ГГц*), являє собою кількість тактів (обчислень) у секунду. Чим вища частота процесора, тим вища його продуктивність [11];

б) частота шини – тактова частота, з якою відбувається обмін даними між процесором та системною шиною материнської (головної) плат [11];

в) розрядність – максимальна кількість біт інформації, що процесор може обробити і передавати одночасно. Процесори з підтримкою *64-bit* здатні адресувати понад 4 *Гб* оперативної пам'яті, чого не можуть *32-bit* процесори [11];

г) кеш-пам'ять – це понадоперативна пам'ять, яку ЦП використовує для зменшення часу доступу до ОЗП. Кеш використовує невелику кількість пам'яті, але вона є надзвичайно швидкою [11];.

д) *Socket* (укр. сокет) – роз'єм, розташований на материнській (головній) платі ПК, до якого приєднується центральний процесор. Сокети для сучасних ЦП *"Intel"* найчастіше бувають гніздовими, в яких контактні штирі розташовані в самому роз'ємі (на рис. 1.2 праворуч), а для ЦП *"AMD"* – щілинними, коли штирі припаяні до процесора, а в роз'ємі розташовані щілини, в які ці штирі вставляються (на рис. 1.2 ліворуч) [12];;

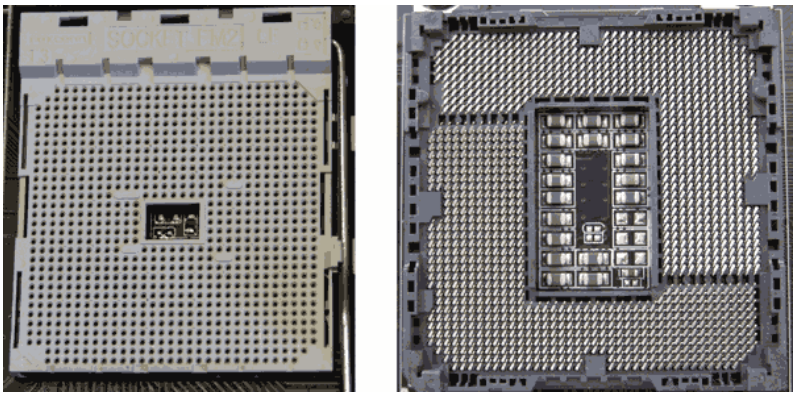

Рис. 1.2. Сокети сучасних ЦП

е) ядро – частина процесора, здатна виконувати один потік команд. Ядра відрізняються за розміром кеш пам'яті, частотою шини, технологією виготовлення тощо. Багатоядерні процесори зможуть швидше впоратися з архівацією, декодуванням відео, роботою сучасних відеоігор тощо [13].

ГП – графічний процесор, окремий пристрій комп’ютера, що виконує графічний рендерінг; ГП призначений виключно для операцій з обробки графіки і обчислень з плаваючою точкою. У першу чергу існує для того, щоб полегшити роботу основного процесора. Сучасні графічні процесори дуже ефективно обробляють і відображають комп’ютерну графіку, завдяки спеціалізованій конвеєрній архітектурі вони набагато ефективніше в обробці графічної інформації, ніж типовий ЦП [14]. Зовнішній вигляд сучасного ГП наведено на рис. 1.3:

Рис. 1.3. Зовнішній вигляд сучасного ГП

Сучасні ГП зазвичай встановлюються у окремий модуль ПК – відеокарту. Відеокарта включає у себе такі елементи:

1) ГП

2) відеопам’ять – є ОЗУ відеокарти. У ній знаходяться інформаційні дані образу, що йде на дисплей;

3) відеоконтролер – створює образ в оперативній пам’яті відеокарти, обробляє отримані дані від ЦП, виконує роботу в частині створення сигналів розгортки для монітора [15];

4) цифро-аналоговий перетворювач – модифікує зображення, яке генерує відеоконтролер, бере участь у регулюванні кольорової гами на екрані ПК [15];

5) відео-ПЗП (постійний запам’ятовуючий пристрій) – зберігає в собі БІОС відеокарти, доступ до нього має лише ЦП [15];

6) система охолодження – засоби для підтримки оптимального рівня температурного показника відеокарти [15];

Зовнішній вигляд відеокарти наведено на рис. 1.4.

Рис. 1.4. Зовнішній вигляд відеокарти

Основні характеристики ГП дещо схожі з характеристиками ЦП, до них можна віднести наступні:

а) ядро;

б) тактова частота;

в) частота шини;

г) кеш-пам’ять;

д) кількість обчислювальних (шейдерних) блоків – виконують спеціальні програми, відомі як шейдери [15];

е) швидкість заповнення (філлрейт) – показує, з якою швидкістю ГП здатний промальовувати пікселі. [15].

Більшість ПММАПР ПК надає інформацію про характеристики саме відеокарти:

1) тактова частота ГП;

2) швидкість заповнення (філлрейт);

3) обсяг відеопам’яті – це власна пам’ять, що використовується ГП для зберігання необхідних даних: текстур, вершин, даних буферів і т. ін.;

4) ширина шини пам’яті – шина пам’яті відповідає за передачу інформації від пам’яті до ГП та назад. Велика ширина дозволяє передавати більшу кількість інформації за одиницю часу**;**

5) типи пам’яті: існує декілька типів пам’яті, що встановлюються на відеокарту, детально розглядатися в даній роботі вони не будуть, оскільки для аналізу різних ПММАПР ПК достатньо знати, що існують наступні типи: *DDR, DDR2, GDDR3, GDDR4, GDDR5.*

ОЗП – це енергозалежний тип пам’яті з довільним доступом. В цій пам’яті зберігається виконуваний машинний код, вхідні, проміжкові та вихідні дані. Мікросхема ОЗП повинна бути постійно підключена до джерела живлення і тому може використовуватися тільки як тимчасова пам’ять [3, с. 97].

Зовнішній вигляд ОЗП наведено на рис. 1.5.

Рис. 1.5. Зовнішній вигляд ОЗП

Основні характеристики ОЗП:

1) тип пам’яті – тип технологій, що був використаний для виготовлення ОЗП. На даний момент найчастіше використовують такі типи: *DDR, DDR2, DDR3, DDR4*;

2) форм-фактор модуля пам’яті – залежить від того для якого типу ПК буде використовуватись планка пам’яті – для ноутбука форм-фактор буде *SO-DIMM*, а для стаціонарного ПК – *DIMM*. На рис. 1.6 наведено зовнішній вигляд ОЗП різних форм-факторів: ліворуч для стаціонарного ПК – *DIMM*, праворуч для ноутбука – *SO-DIMM*.

Рис. 1.6. ОЗП різних форм-факторів

3) об’єм модуля пам’яті – обсяг одного модуля оперативної пам’яті, що залежить від типу пам’яті, який подано у табл. 1.1:

Таблиця 1.1

Об’єм модуля пам’яті в залежності від типу пам’яті

Тип пам’яті	Об’єм модуля пам’яті	
	Мінімальний	Максимальний
DDR	256 *МБ*	1 *ГБ*
DDR2	512 *МБ*	4 *ГБ*
DDR3	1 *ГБ*	16 *ГБ*
DDR4	4 *ГБ*	128 *ГБ*

4) тактова частота оперативної пам’яті – кількість операцій з передачі даних в секунду через обраний канал. Тактова частота також залежить від типу пам’яті, що наглядно можна зобразити у табл. 1.2:

Таблиця 1.2

Тактова частота модуля пам’яті в залежності від типу пам’яті

Тип пам’яті	Тактова частота модуля пам’яті, *МГц*	
	Мінімальна	Максимальна
DDR	100	350
DDR2	200	600
DDR3	800	2400
DDR4	1600	3200

Жорсткий диск – запам’ятовуючий пристрій, місце зберігання даних у більшості ПК. Жорсткий диск є енергонезалежним типом пам’яті на відміну від ОЗП [10].

Існує два найбільш популярних жорстких дисків: *HDD* та *SSD*. *HDD* складаються з одного або декількох магнітних дисків і зчитуючих головок. Зовнішній вигляд *HDD* зображено на рис. 1.7: зліва без захисної кришки, справа – із захисною кришкою (звичайний вигляд *HDD*).

Рис. 1.7. Зовнішній вигляд *HDD*

SSD складається з великої кількості окремих *Flash*-накопичувачів, які вбудовані в диск за тим же принципом, що і в *USB*-флешках. Це означає, що в *SSD* немає рухомих механічних частин [11]. Зовнішній вигляд *SSD* зображено на рис. 1.8: зліва без захисної кришки, справа – із захисною кришкою (звичайний вигляд *SSD*).

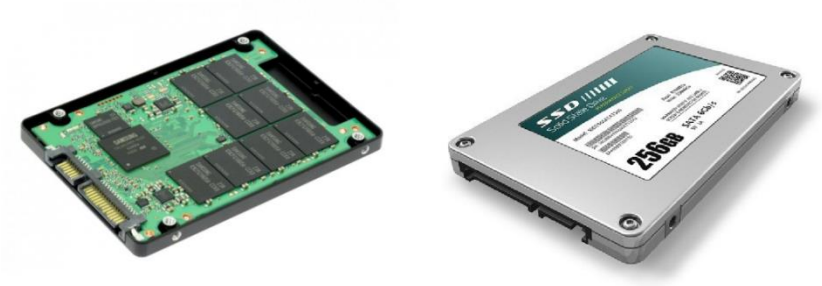

Рис. 1.8. Зовнішній вигляд *SSD*

Основні характеристики жорсткого диску:

1) інтерфейс – це підтримуваний стандарт обміну даними з накопичувачами інформації: *ATA (IDE, PATA), SATA* [17];

2) ємність – обсяг даних, які може зберігати жорсткий диск (*ГБ, ТБ*) [17];

3) форм-фактор – фізичний розмір диска з феромагнітним покриттям: 3,5 або 2,5 дюйма [17];

4) час доступу – час, за який жорсткий диск гарантовано виконає операцію читання або запису на будь-якій ділянці магнітного диска (діапазон від 2,5 до 16 *мс*) [17];

5) швидкість обертання шпинделя – параметр, від якого залежить час доступу і середня швидкість передачі даних [17] ;

6) введення-виведення – кількість операцій введення-виведення в секунду. Зазвичай жорсткий диск виробляє близько 50 операцій в секунду при довільному доступі і близько 100 при послідовному [17].

Материнська (головна) плата – це основна плата комп'ютера. На ній розташовані слоти і роз'єми для підключення комплектуючих ПК, таких як: відеокарти, оперативної пам'яті, процесора, накопичувачів даних, а також периферії [18]. Зовнішній вигляд материнської плати подано на рис. 1.9:

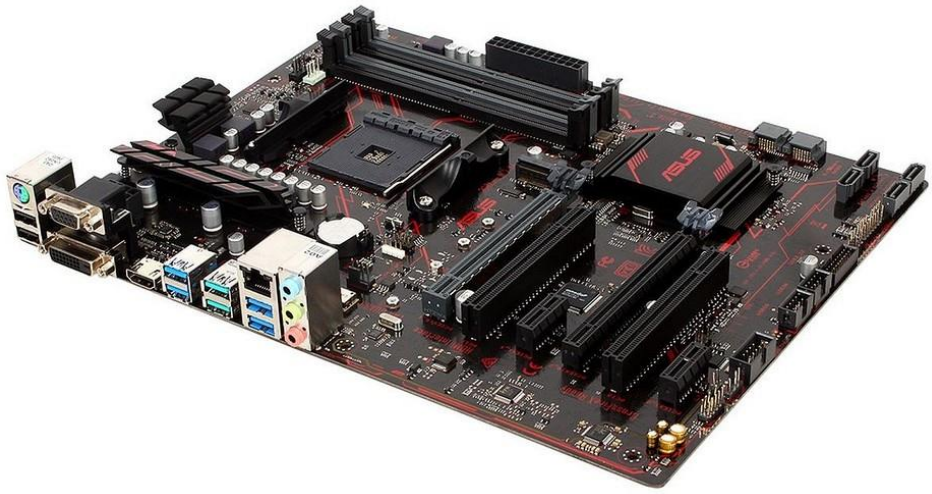

Рис. 1.9. Зовнішній вигляд материнської (головної) плати

Основні характеристики материнської (головної) плати:

1) форм-фактор – цей параметр включає в себе розмір, місця кріплення материнської плати, а також роз'єми для додаткових пристроїв;

2) тип сокета – тип роз'єму для ЦП;

3) наявність вбудованої звукової карти;

4) наявність вбудованого мережевого контролера;

5) слоти *PCI* і *PCI Express* – призначені для установки плат розширення (звукова, мережева, відеокарта, модем, різні контролери) [19].

Блок живлення – це електронний пристрій, що формує напругу, необхідне певного компоненту ПК, з напруги електричної мережі. З БЖ виходять кабелі до материнської (головної) плати, відеокарти, жорсткого диска, кулерів і вентиляторів, до інших пристроїв [20].На рис. 1.10 наведені БЖ різних форм-факторів:

Рис. 1.10. Блоки живлення різних форм-факторів

Основні характеристики БЖ:

1) потужність – скільки енергії, яку БЖ зможе віддати компонентам, що підключені до нього [21];

2) ККД – параметр, що позначає, наскільки ефективно блок живлення може перетворювати енергію для потреб комплектуючих [21];

3) наявність *PFC – Power Factor Correction* (корекція фактора потужності) – спеціальний елемент, призначений для корекції коефіцієнта потужності і спрямований на захист мережі. *PFC* умовно ділиться на активний *(Active)* і пасивний *(Passive)* [21];

4) форм-фактор.

Система охолодження – набір засобів для відводу тепла від компонентів, що нагріваються в процесі роботи ПК.

До програмних ресурсів можна віднести навантаження на ЦП від конкретного процесу ПК (процес – це просто екземпляр виконуваної програми, включаючи поточні значення лічильника команд, регістрів і змінних [7]), загальне навантаження на систему, список процесів, що протікає у системі на поточний момент, характеристики системи(наприклад назва та версія), моніторинг трафіку Інтернет з'єднання – кількість отриманих байт/секунду, кількість надісланих байт/секунду, *IP*-адреса, яка була надана ПК *DHCP*-сервером та інше.

Існують також програми, що сфокусовані на тому чи іншому виді моніторингу, тобто займаються лише апаратною або лише програмною частиною. Деякі з програм, що спеціалізуються на апаратній частині напрямлені на моніторинг лише одного з компонентів ПК, але такий моніторинг є більш глибоким.

1.2. Огляд та порівняння програмних модулів моніторингу апаратних та програмних ресурсів ПК

Прикладами програмних модулів моніторингу апаратно-програмних ресурсів ПК є такі програми:

- *Aida64;*
- *Speccy;*
- *OpenHardware Monitor;*
- *CPU-Z;*
- *ProcessHacker;*

Aida64 – утиліта для тестування і ідентифікації компонентів персонального комп'ютера під управлінням операційних систем *Windows*, що надає детальні відомості про апаратне і програмне забезпечення. Інтерфейс програми *Aida64* подано на рис. 1.11:

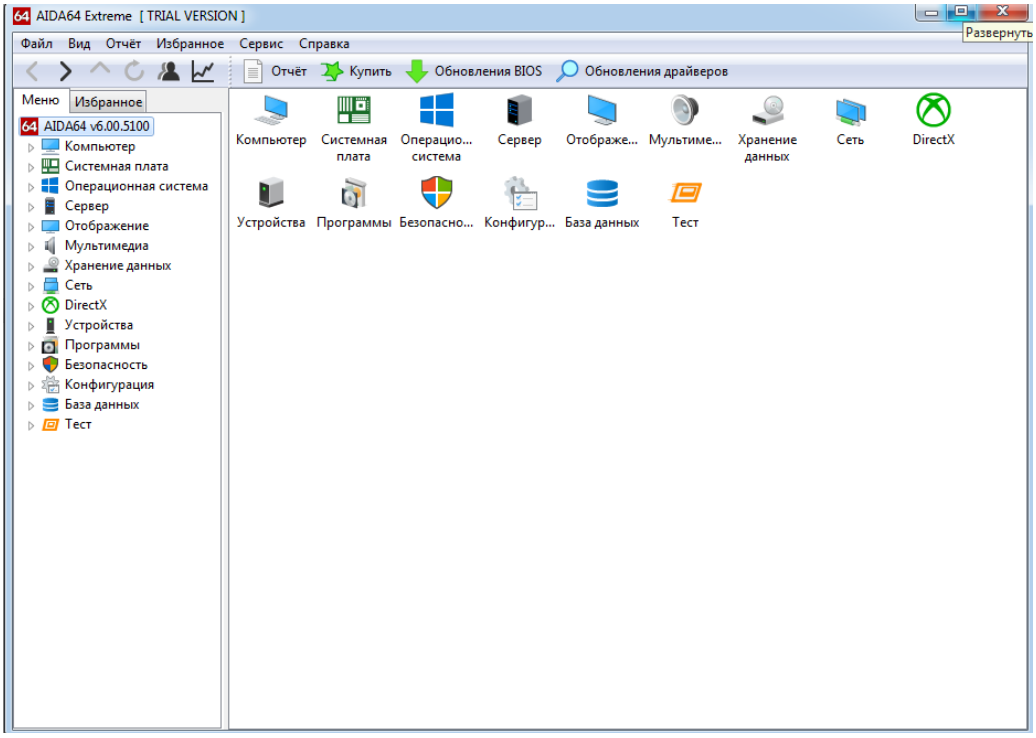

Рис. 1.11. Інтерфейс програми *Aida64*

Програма надає глибоку інформацію як про апаратну частину так і про програмну складову ПК. Наприклад, можна переглянути детальну інформацію про ЦП, як зображено на рис. 1.12:

Поле	Значение
Свойства ЦП	
Тип ЦП	QuadCore Intel Core i5-4460, 3200 MHz (32 x 100)
Псевдоним ЦП	Haswell-DT
Степпинг ЦП	C0
Наборы инструкций	x86, x86-64, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AVX...
Исходная частота	[TRIAL VERSION]
Мин./макс. множитель ЦП	8x / 32x
Engineering Sample	Нет
Кэш L1 кода	32 КБ per core
Кэш L1 данных	[TRIAL VERSION]
Кэш L2	256 КБ per core (On-Die, ECC, Full-Speed)
Кэш L3	6 МБ (On-Die, ECC, Full-Speed)
Физическая информация о ЦП	
Тип корпуса	1150 Contact FC-LGA
Размеры корпуса	37.5 mm x 37.5 mm

Рис. 1.12. Інформація про ЦП у програмі *Aida64*

Також детальну інформацію можна переглянути і про ОЗП, материнську плату, відеокарту.

Програма надає інформацію і про програмні ресурси – процеси, що протікають у системі, служби системи, *DLL*-файли, програми

автозавантаження, встановлені програми та навіть ліцензії різних програм. Також можна переглянути інформацію про інтернет-з'єднання.

Окрім того *Aida64* надає можливість проводити стрес-тести ПК. Приклад вікна стрес-тесту зображено на рис. 1.13:

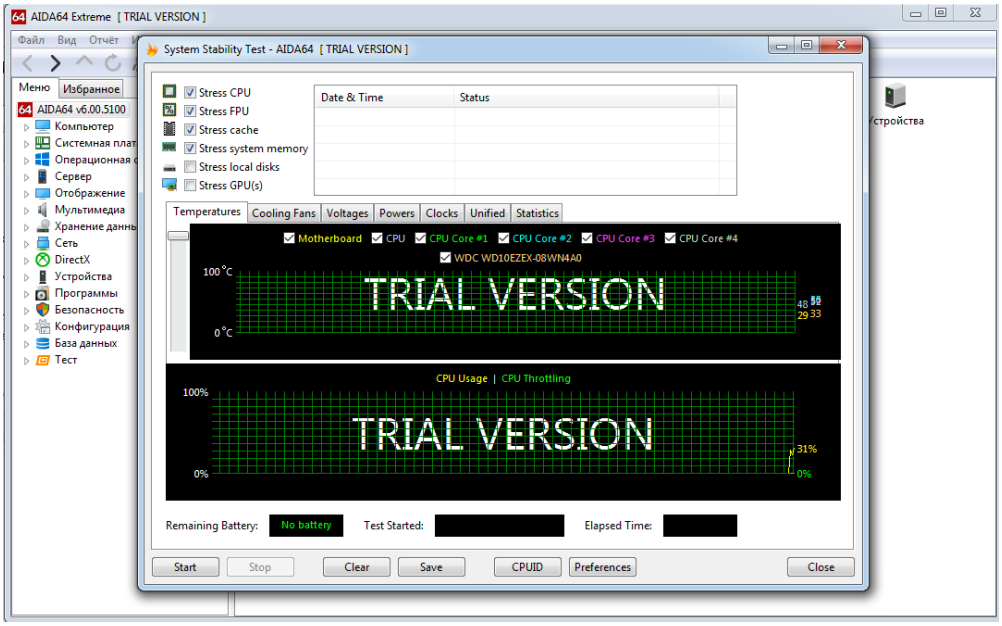

Рис. 1.13. Вікно для стрес-тестів у програмі *Aida64*

Aida64 надає інформацію про стан ПК у обширному обсязі, інформація надається як про апаратну частину ПК так і про програмну.

Speccy – утиліта, яка надає користувачам потужний і простий у використанні інструмент для відображення детальної системної інформації. Інтерфейс програми можна побачити на рис. 1.14.

На головній сторінці програми наводиться скорочена інформація про всі компоненти ПК – ЦП, ОЗП, материнську плату, відеокарту, жорсткі диски, дисководи та аудіопристрій. Отримати детальну інформацію можна перейшовши по відповідній вкладці, наприклад, переглянути інформацію про ЦП, що зображено на рис. 1.15.

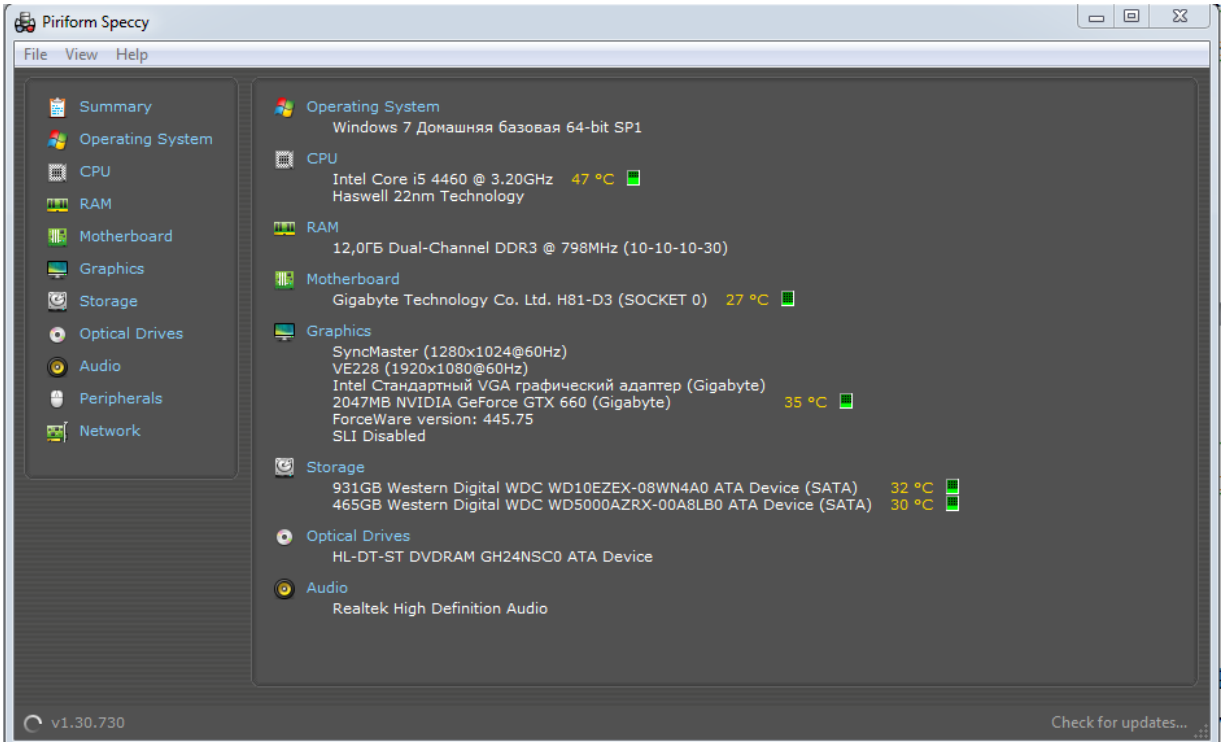

Рис. 1.14. Інтерфейс програми *Speccy*

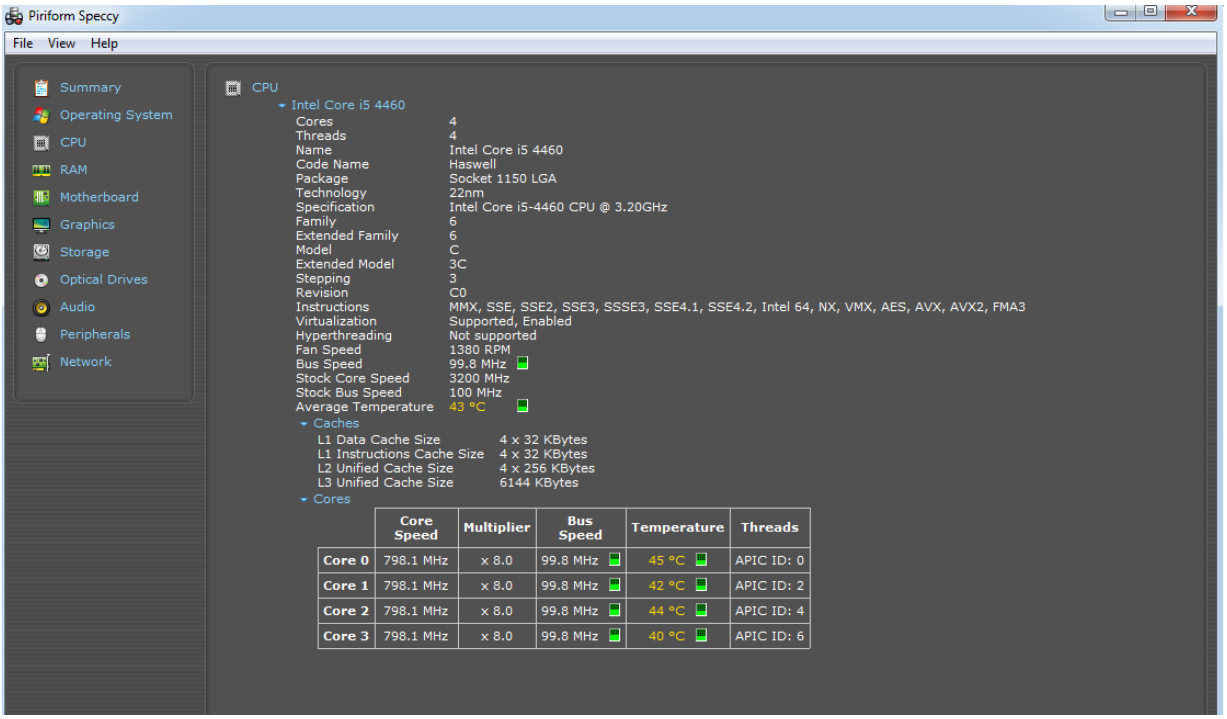

Рис. 1.15. Інтерфейс програми *Speccy*

Було отримано більш детальну інформацію щодо ЦП. Так само можно отримати детальну інформацію щодо всіх компонентів, що наведені зліва (див. рис. 1.14 або рис. 1.15).

На відміну від програми *Aida64*, дана програма не надає інформації про програмні ресурси ПК, одна надає досить глибоку інформацію про апаратні ресурси.

Open Hardware Monitor – утиліта з відкритим вихідним кодом, яка забезпечує централізований інтерфейс, де можна легко контролювати різні аспекти продуктивності обладнання, включаючи швидкість обертання вентиляторів, температурні датчики, споживання напруг, навантаження тощо. Саме бібліотека *Open Hardware Monitor* була застосована у дипломному проекті. Як видно з наведеного рис. 1.16, дана програма надає лише основні чисельні характеристики, такі як температура, напруга, навантаження та ін., більш глибокої інформації як у програмах *Aida64* та *Speccy* у даній програмі не наведено.

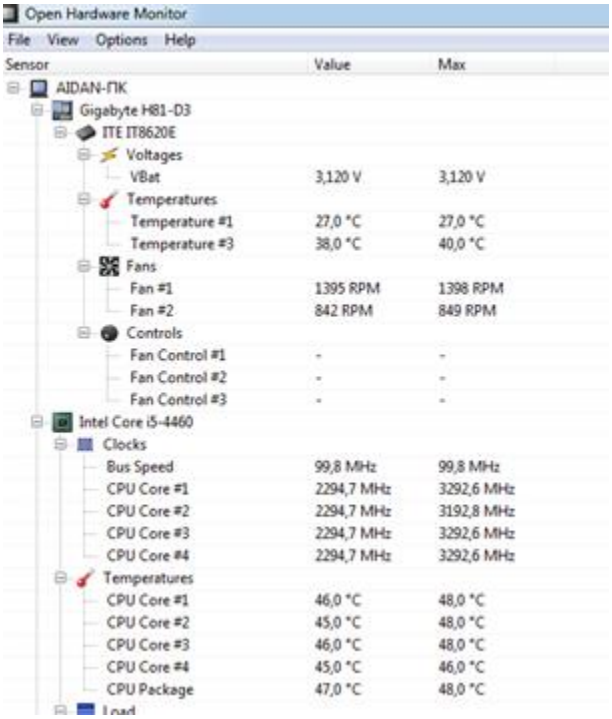

Рис. 1.16. Інтерфейс програми *Open Hardware Monitor*

CPU-Z – прикладна програма-утиліта для відображення технічної інформації про персональний комп'ютер користувача, що працює під ОС *Microsoft Windows.* Інтерфейс даної програми подано на рис. 1.17:

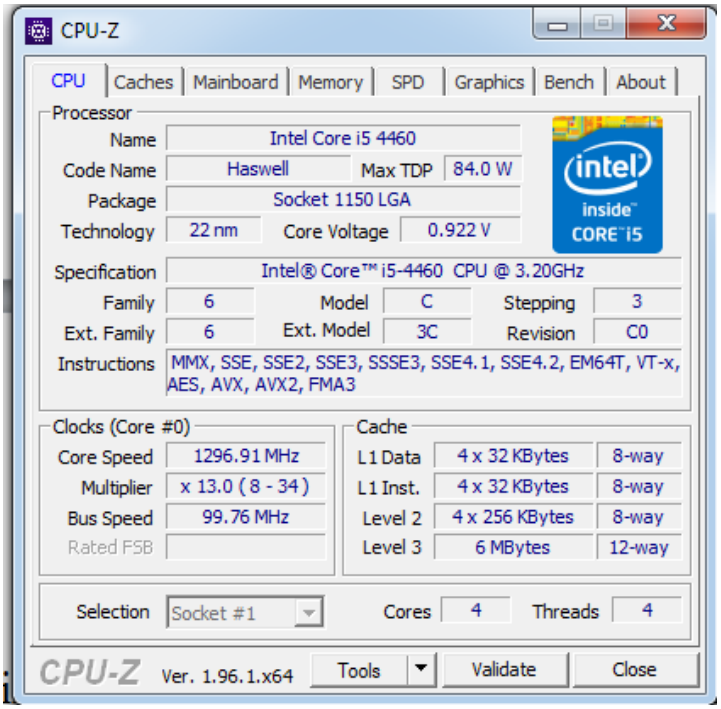

Рис. 1.17. Інтерфейс програми *CPU-Z*

Дана програма надає досить глибоку інформацію про ЦП, менш глибоку про материнську плату, ОЗП та відеокарту. Однак ця програма є досить невимогливою та займає мало місця (близько 5 *МБ*), проте не надає інформації про температуру ЦП.

ProcessHacker – утиліта з відкритим кодом для моніторингу системних процесів і служб, запущених під керуванням 32-бітних і 64-розрядних операційних систем сімейства *Windows*, яка використовується як заміна або доповнення диспетчера задач *Windows*. Її інтерфейс можна побачити на рис. 1.18.

Перша відмінність даної програми від диспетчера задач *Windows* заключається у тому, що програма показує одразу все дерево процесів як наведено на рис. 1.19, де зліва – *ProcessHacker*, справа – диспетчер задач *Windows*.

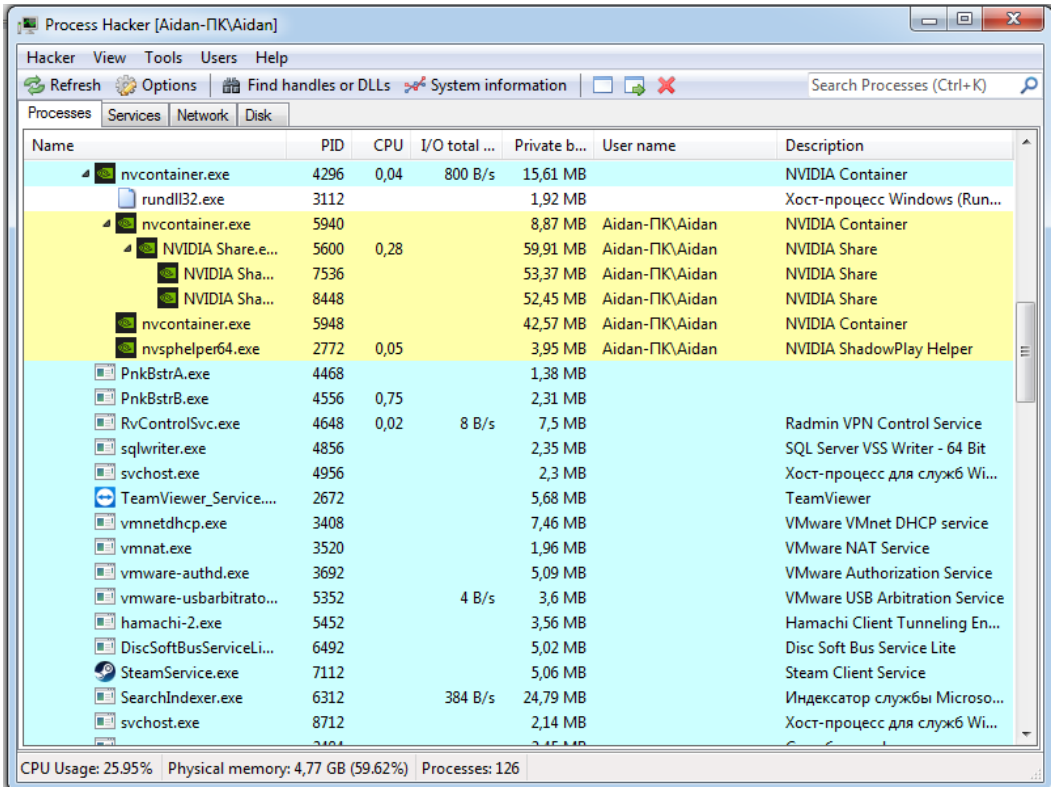

Рис. 1.18. Інтерфейс програми *ProcessHacker*

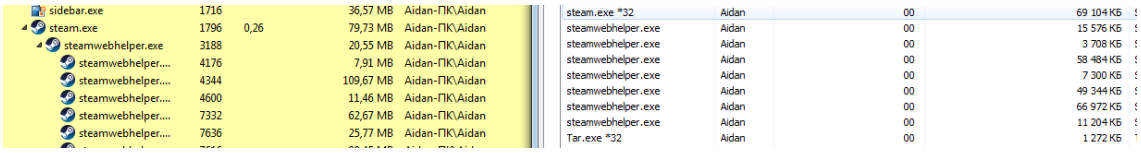

Рис. 1.19. Порівняння відображення списку процесів

Окрім того, у *ProcessHacker* існує система розподілу типу процесів, для зручності кожен тип процесу має підсвітку своїм кольором. Наприклад, у базових налаштуваннях темно-голубий колір позначає системний процес, жовтий – процес, що був запущений тим самим користувачем, що запустив *ProcessHacker*.

Також програма надає інформацію про навантаження на ЦП, ГП, ОЗП та прилади вводу/виводу (рис. 1.20).

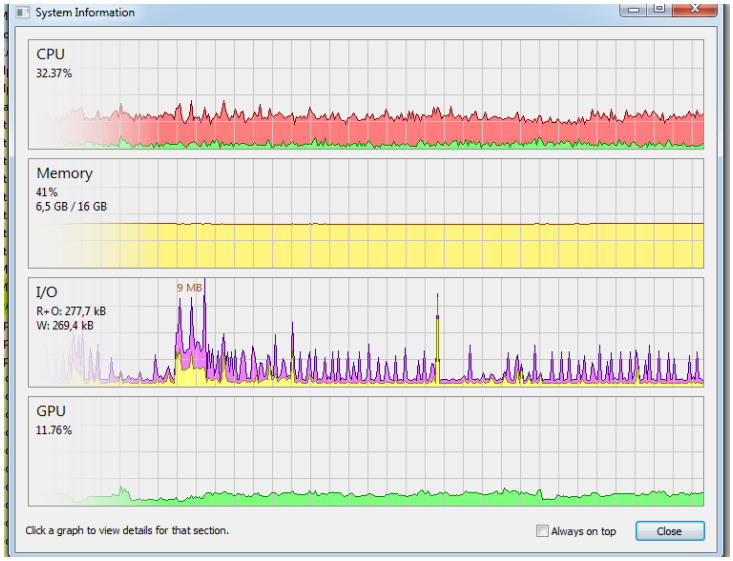

Рис. 1.20. Вікно *ProcessHacker*, що надає інформацію про навантаження на апаратну частину ПК

ProcessHacker надає інформацію лише про програмну частину ПК, на відміну від наведених вище програм, у *ProcessHacker* ми можемо отримати детальну інформацію про процеси, що на даний момент відбуваються у системі, навантаження ж на апаратну частину ПК також надає програма *Aida64.*

Порівняльну характеристику поданих вище програм наведено у табл. 1 додатку А.

Існує і більший спектр програмного забезпечення, але він настільки обширний, що описати увесь не вдасться можливим, тому у цій дипломній роботі описано лише найпопулярніші з усього загалу.

Окрім наведених вище програмних модулів моніторингу існує ряд програмних модулів для тестів та стрес-тестів ПК, але вони виходять за межі модулів моніторингу апаратно-програмних ресурсів ПК, тому вони в межах даної роботи розглядатись не будуть. До таких програм можна віднести такі програмні модулі як:

– *Victoria* – тестування жортких дисків*;*
– *OCCT* – тестування відеокарт та ЦП*;*
– *IntetlBurnTest* – тестування ЦП*;*
– *Prime 95* – тестування ЦП*;*
– *Aida64* – тестування ЦП, ГП, ОЗП та жорстких дисків.

1.3. Висновки до розділу

У даному розділі було проведено аналіз комплектуючих сучасних ПК, наведено характеристики комплектуючих для подальшого більш детального аналізу ПММАПР ПК. Таким чином було описано основну комплектуючу ПК – ЦП, наведено такі його характеристики як тактова частота, частота шини, розрядність, кеш-пам'ять, *socket*, ядро; наведено «допоміжний» процесор, а саме ГП, який був створений для обробки графічних даних та покликаний зняти частину навантаження з ЦП. Розглянуто такі характеристики ГП як ядро, тактова частота, частота шини, кеш-пам’ять, кількість обчислювальних (шейдерних) блоків, швидкість заповнення, також розглянуто такі характиристики відеокарт як тактова частота ГП, швидкість заповнення (філлрейт), обсяг відеопамяті, ширина шини памяті, типи памяті. Розглянуто компонент ОЗП (оперативний запамятовуючий пристрій), такі характеристики як тип памяті, форм-фактор, об’єм модуля пам’яті, тактова частота оперативної памяті. Розглянуто компонент жорсткий диск, та такі його характеристики як інтерфейс, ємність, форм-фактор, час доступу, швидкість обертання шпинделя, введення-виведення. Розглянуто компонент материнська/головна плата, такі її характеристики як форм-фактор, типа сокета, наявність вбудованої звукової карти, наявність вбудованого мережевого контролера, слоти *PCI* та *PCI Express*. Розглянуто компонент блок живлення, такі характеристики як потужність, ККД, наявність *PFC*. Останнім розглянутим компонентом був компонент система охолодження, та такі його характеристики як тип утилізації тепла.

Проведений аналіз був здійснений для кращого розуміння інформації, що надають ПММАПР ПК.

Було розглянуто різноманітні версії ПММАПР ПК, проведено їх аналіз та порівняння у виді таблиці. Розглянуто програми *Aida64, Speccy, OpenHardwareMonitor, CPU-Z, ProcessHacker* та проведено їх порівняння.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ МОНІТОРИНГУ АПАРАТНО-ПРОГРАМНИХ РЕСУРСІВ НА БАЗІ ARDUINO

2.1. Технології, застосовані при створенні програмного модуля моніторингу апаратно-програмних ресурсів на базі *Arduino*

Готовий програмний модуль складається з трьох підмодулів: основної програми, програми для отримання температур та програми для прошивки мікроконтролера на базі *Arduino*.

Основна програма включає в себе такий набір технологій:

1. *WinApi;*
2. *Windows Management Instrumentation;*
3. *Qt;*
4. Мова програмування *C++*.

Програма для отримання температур включає в себе такий набір технологій:

1. Відкрита бібліотека *Open Hardware Monitor;*
2. *Microsoft Visual Studio;*
3. Мова прогрмуваня *C#.*

Програма для прошивки мікроконтролера на базі *Arduino* включає в себе такий набір технологій:

1. *Arduino;*
2. *Arduino IDE;*
3. *COM*-порти;
4. Мова прогрмуваня *Arduino C.*

У результаті маємо таку схему набору технологій (рис. 2.1):

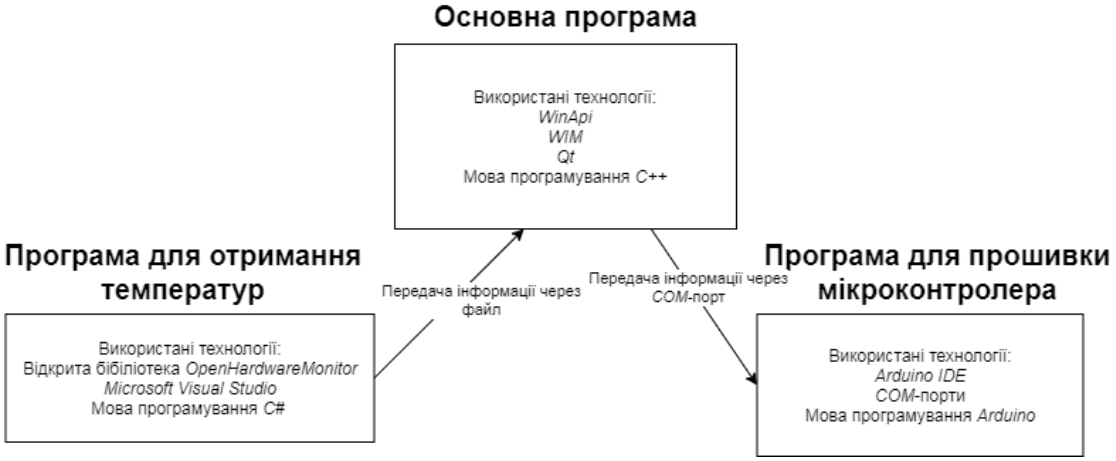

Рис. 2.1 Схема набору технологій

2.2. Технології основної програми

WinApi – Windows Application programming interfaces – загальна назва набору базових функцій інтерфейсів програмування додатків операційних систем сімейств *Microsoft Windows.* Надає прямий спосіб взаємодії додатків користувача з операційною системою *Windows*. Для створення програм, що використовують *WinAPI. WinAPI* спроектований для використання в мовах *C/C++* для написання прикладних програм, призначених для роботи під управлінням ОС *Windows*. Робота через *WinAPI* – це найбільш близький до операційної системи спосіб взаємодії з нею з прикладних програм. Більш низький рівень доступу, необхідний тільки для драйверів пристроїв, в поточних версіях Windows надається через *Windows Driver Model.*

У даному проєкті *WinAPI* було використано для створення потоків, отримання деякої системної інформації (список драйверів, процесів, навантаження на ОЗП, час роботи системи і т. д.), також для типів змінних були використані типи даних *WinAPI*, наприклад, *DWORD* – 32-бітове беззнаковое ціле, аналог: *unsigned long int*; *ULONGLONG* – 64-бітове беззнаковое ціле; *LONG* – 32-бітове знаковое ціле, та інші. Також використовуються структури *WinAPI*, такі як *FILETIME, PROCESS_MEMORY_COUNTERS_EX* та інші.

Windows Management Instrumentation (WMI) – інструментарій управління *Windows*. *WMI* – це одна з базових технологій для централізованого управління і стеження за роботою різних частин комп'ютерної інфраструктури під управлінням платформи *Windows*. *WMI* є відкритою уніфікованою системою інтерфейсів доступу до будь-яких параметрах операційної системи, пристроїв і додатків, які функціонують в ній. Важливою особливістю *WMI* є те, що об'єкти, які в ньому зберігаються, відповідають динамічним ресурсам, тобто параметри цих ресурсів постійно змінюються, тому параметри таких об'єктів не зберігаються постійно, а створюються за запитом споживача даних. Сховище властивостей об'єктів *WMI* називається репозиторієм і розташоване в системній папці операційної системи Windows: *%SystemRoot%\System32\WBEM\Repository*.

Так як *WMI* побудований за об'єктно-орієнтованим принципом, то всі дані операційної системи представлені у вигляді об'єктів і їх властивостей і методів. Всі класи групуються в простору імен, які ієрархічно впорядковані і логічно пов'язані один з одним за певною технологією або галузю управління.

Використовуючи *WMI* технологію можна отримати інформацію про апаратну частину ПК, застосувавши консольну команду, наприклад, було отримано інформацію про виробника та модель материнської плати, результат виконаної команди наведено на рис. 2.2.

```
C:\Users\Aidan>wmic baseboard get product, manufacturer
Manufacturer                  Product
Gigabyte Technology Co., Ltd.  H81-D3
```

Рис. 2.2. Результат виконання команди *wmic baseboard get product, manufacturer*

У даному проєкті *WMI* використовується для отримання інформації про апаратну частину ПК (про виробника та модель материнської плати (рис. 2.3), назву, тип, об'єм ОЗП (рис. 2.4), назву та об'єм жорстких дисків (рис. 2.5), назви аудіо- та мережевих плат (рис. 2.6 та 2.7 відповідно)). Однак для

отримання інформації про ЦП та відеокарту *WMI* не використовується, замість цього використовується *WinAPI*.

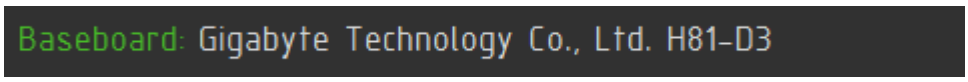

Рис. 2.3. Виробник та модель материнської плати у програмі даного проєкту

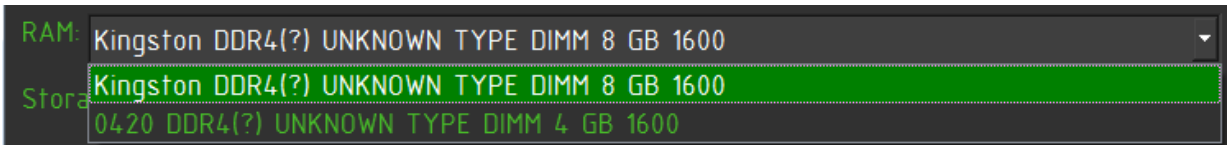

Рис. 2.4. Назва, тип, об’єм ОЗП у програмі даного проєкту

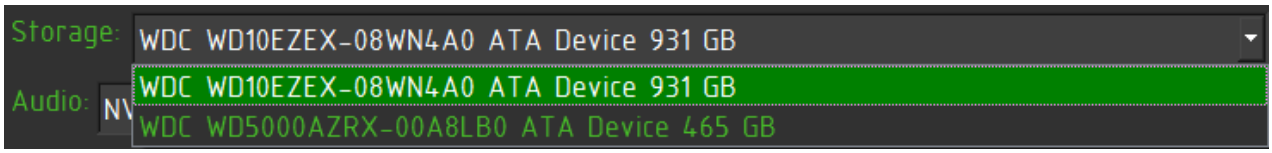

Рис. 2.5. Назва та об’єм жорстких дисків у програмі даного проєкту

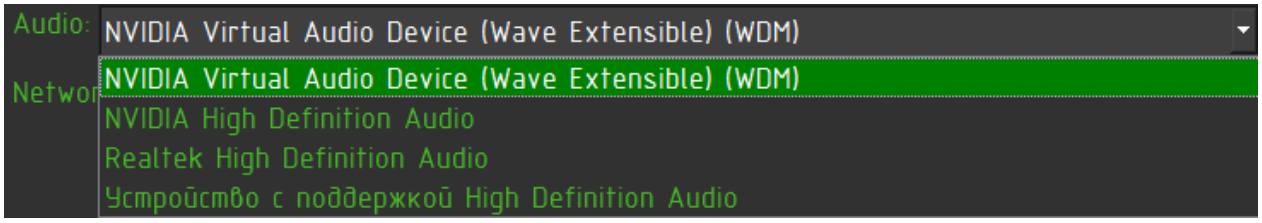

Рис. 2.6. Назви аудіо пристроїв у програмі даного проєкту

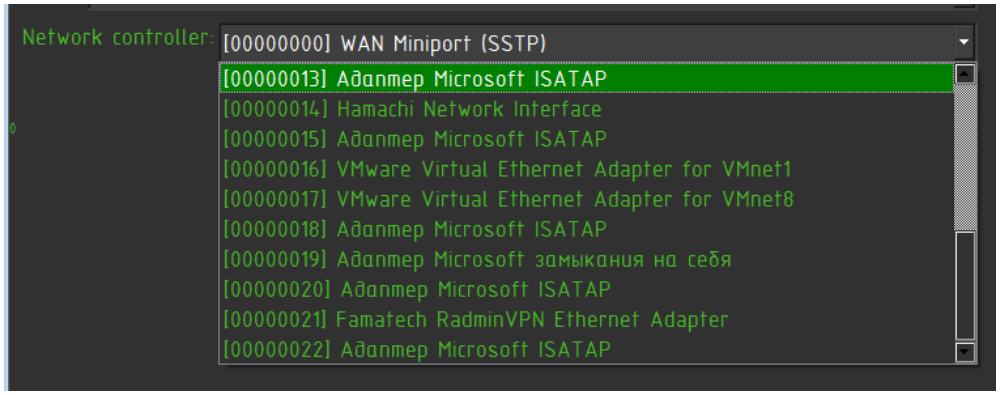

Рис. 2.7. Назви мережевих пристроїв у програмі даного проєкту

Qt – фреймворк для розробки кроссплатформенного програмного забезпечення на мові програмування *C++*. *Qt* дозволяє запустити написане з його допомогою програмне забезпечення в більшості сучасних операційних систем, що дозволяє просту компіляцію програми для кожної системи без змін вихідного коду. Включає в себе всі основні класи, які можуть бути необхідними при розробці програмного забезпечення, починаючи елементами графічного інтерфейсу та закінчуючи класами для роботи з мережею, базами даних та *XML*. Комплектується візуальним середовищем розробки графічного інтерфейсу *Qt Designer*, що дозволяє створювати діалоги і форми в режимі *WYSIWYG* (*WYSIWYG – What You See Is What You Get* – властивість прикладних програм або веб-інтерфейсів, в яких вміст відображається в процесі редагування і виглядає максимально близько схожим на кінцеву продукцію). Починаючи з версії 4.5.0 в комплект включена середовище розробки *Qt Creator*, яка включає редактор коду, довідку, графічні засоби *Qt Designer* і можливість налагодження додатків. *Qt Creator* може використовувати *GCC* або *Microsoft VC++* як компілятора і *GDB* як відладчика. Для *Windows*-версій бібліотека комплектується компілятором, заголовними і об'єктними файлами *MinGW*.

Бібліотека *Qt* – це безліч класів (понад 500), які охоплюють велику частину функціональних можливостей операційних систем, надаючи розробнику потужні механізми. При цьому не порушується ідеологія операційної системи. *Qt* не є єдиним цілим, він розбит на модулі. Основні модулі наведено у табл. 2.1.

Таблиця 2.1

Основні модулі *Qt*

Бібліотека	Позначення в проєктному файлі	Призначення
QtCore	*core*	Основний модуль, що складається з класів, не пов'язаних з графічним інтерфейсом
QtGui	*gui*	Модуль для програмування графічного інтерфейсу

Продовження таблиці 2.1

QtNetwork	*network*	Модуль для програмування мережі
QtOpenGL	*opengl*	Модуль для програмування графіки *OpenGL*
QtSql	*sql*	Модуль для програмування баз даних
QtSvg	*svg*	Модуль для роботи з *SVG (Scalable Vector* Graphics, масштабована векторна графіка)
QtXml	*xml*	Модуль підтримки *XML*, класи, що відносяться до *SAX* і *DOM*
Qt3Support	*qt3support*	*qt3support*
QtScript	*script*	Модуль підтримки мови сценаріїв
Phonon	*phonon*	Модуль мультимедіа
QtWebKit	*webkit*	Модуль для створення веб-додатків
QtScriptTools	*scripttools*	Модуль додаткових можливостей підтримки мови сценарію. На даний момент надає відладчик
QtTest	*test*	Модуль, що містить класи для тестування коду

Діаграму деяких модулів наведено на рис. 2.8:

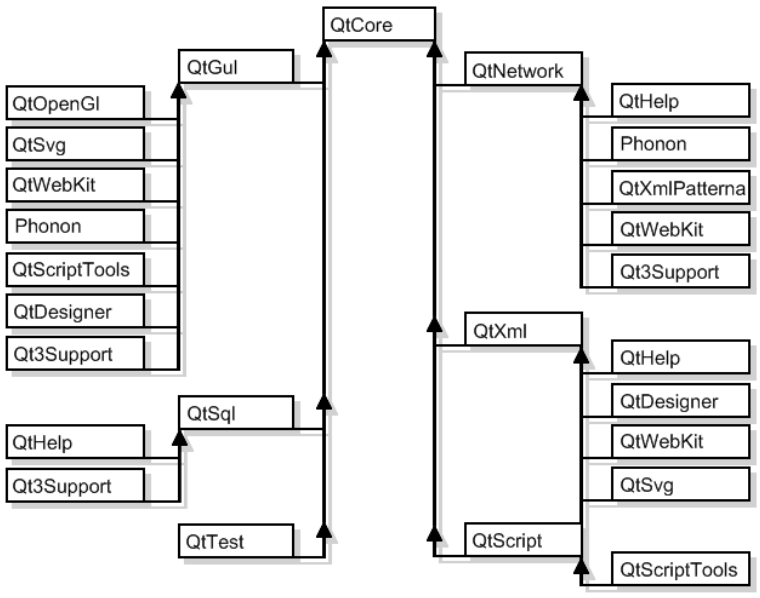

Рис. 2.8. Діаграма модулів *Qt*

Будь-яка *Qt*-програма так чи інакше повинна використовувати хоча б один з модулів, в більшості випадків це *QtCore* і *QtGui*, тому ці два модуля визначені в програмі створення *make*-файлів за замовчуванням. Для використання інших модулів в своїх проєктах необхідно перерахувати їх в проєктному. Наприклад, щоб додати модулі, потрібно написати:

Qt += opengl network sql

А щоб виключити модуль з проєкту:

Qt - = gui

У даному проєкті було використано наступні класи: *QWidget, QObject, QMessageBox, QDebug, QDir, QCharts, QFileDialog.*

QWidget – базовий клас для всіх об’єктів користувацького інтерфейсу. Віджет – це елементарний об’єкт користувацького інтерфейсу: він отримує події миші, клавіатури та інші події від віконної системи і малює своє зображення на екрані. Віджет обмежений своїм батьківським класом і іншими віджетами, розташованими перед ним.

У даному проєкті клас *QWidget* було застосовано у його стандартному призначенні, тобто для створення користувацького інтерфейсу за допомогою віджетів.

QObject – базовий клас для всіх об’єктів *Qt*. Головна особливість в цій моделі – це дуже потужний механізм для зв'язку об'єктів, званий сигналами і слотами.

У проєкті *QObject* також було використано як базовий клас для об’єктів.

QMessageBox – клас, що відображає повідомлення користувачу. Повідомлення може складатися з тексту, іконки та кнопок.

У проєкті *QMessageBox* використовується для створення повідомлення про перенавантаження ЦП чи ОЗП від процесу або в цілому. При деякій умові (перенавантеження) створюється об’єкт *QMessageBox.* Повідомлення має наступний вигляд (рис. 2.9).

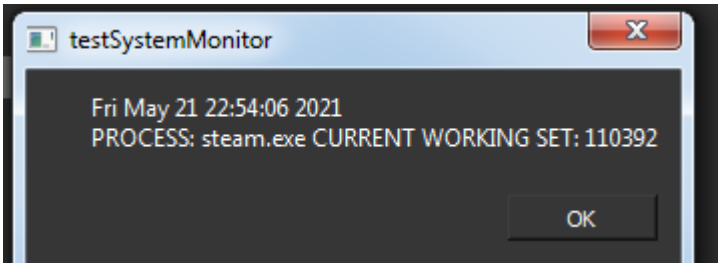

Рис. 2.9. Вигляд повідомлення *QMessageBox*

Текст повідомлення задається програмістом.

QDebug забезпечує вихідний потік для інформації відладки. *QDebug* використовується, коли розробнику потрібно записати інформацію про відладку. У даному проєкті *QDebug* також використовується для виводу інформації відладки на екран.

QDir використовується для маніпулювання іменами шляхів, доступу до інформації про шляхи та файли та маніпулювання базовою файловою системою.

У проєкті *QDir* використовується для оримання темпепатур, а саме для отримання поточної папки програми та запуску *temperatureGetter.exe*.

QChart надає набір простих у використанні компонентів діаграм. Він використовує *Qt Graphics View Framework*, тому діаграми можна легко інтегрувати в сучасні користувальницькі інтерфейси. Діаграми *Qt* можна використовувати як типи *QWidgets*, *QGraphicsWidget* або *QML*.

У даному проєкті *QChart* використовується для створення графіків температур, наприклад для виведення графіків темпратур ЦП, приклад наведено на рис. 2.10:

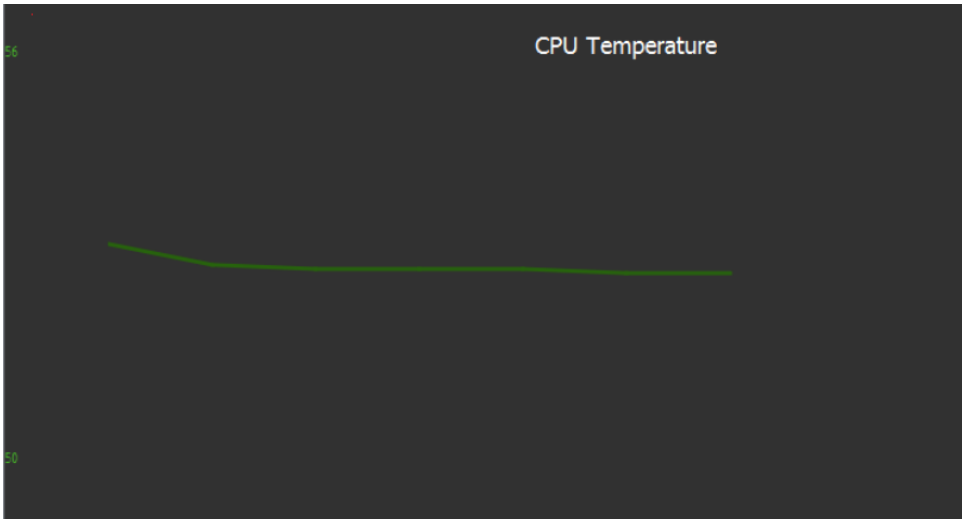

Рис. 2.10. Використання *QChart* у даному проєкті

QFileDialog – клас надає діалогове вікно, яке дозволяє користувачам вибирати файли або каталоги. Клас *QFileDialog* дозволяє користувачеві обходити файлову систему, щоб вибрати один або багато файлів або каталог.

У даному проєкті *QFileDialog* використовується для обрання ігноруємих процесів для моніторингу навантаження від процесів на ЦП та ОЗП. Застосування *QFileDialog* наведено на рис. 2.11.

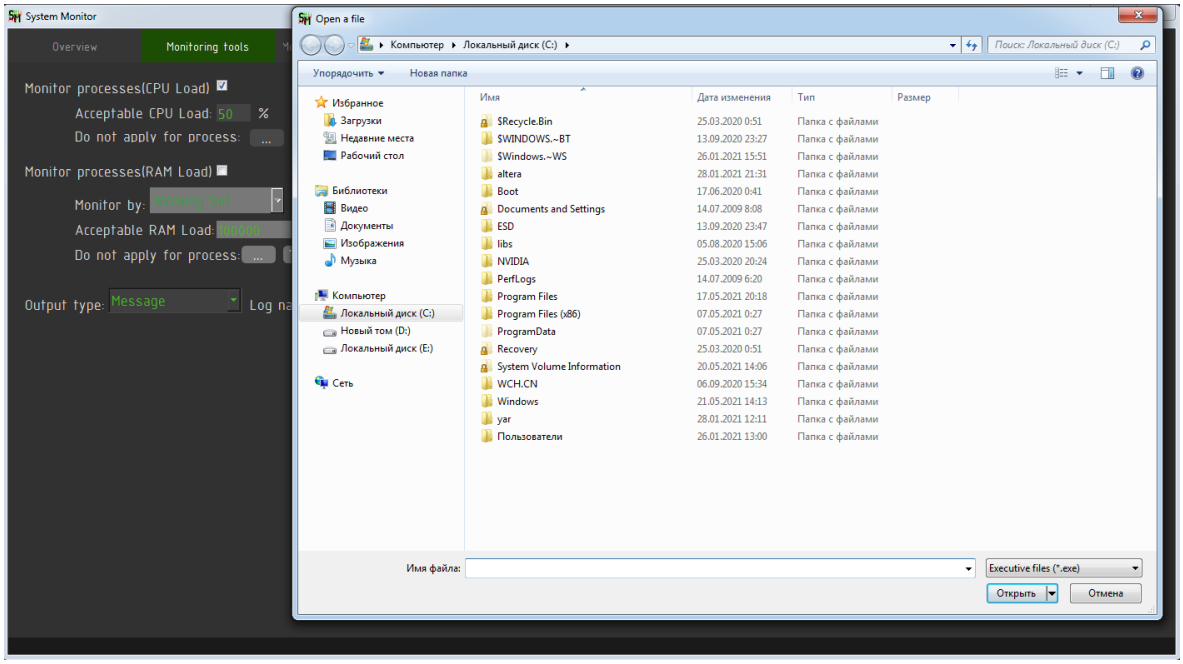

Рис. 2.11. Використання *QFileDialog*, обрання *.exe* файлу

Окрім того для створення графічного користувацького інтерфейсу було використано *Qt Designer*. У *Qt Designer* було побудовано весь інтерфейс та

додано більша частина стилів, менша частина стилів додається безпосередньо у коді. Вигляд осовного вікна даного проєкту у *Qt Designer* наведено на рис. 2.12.

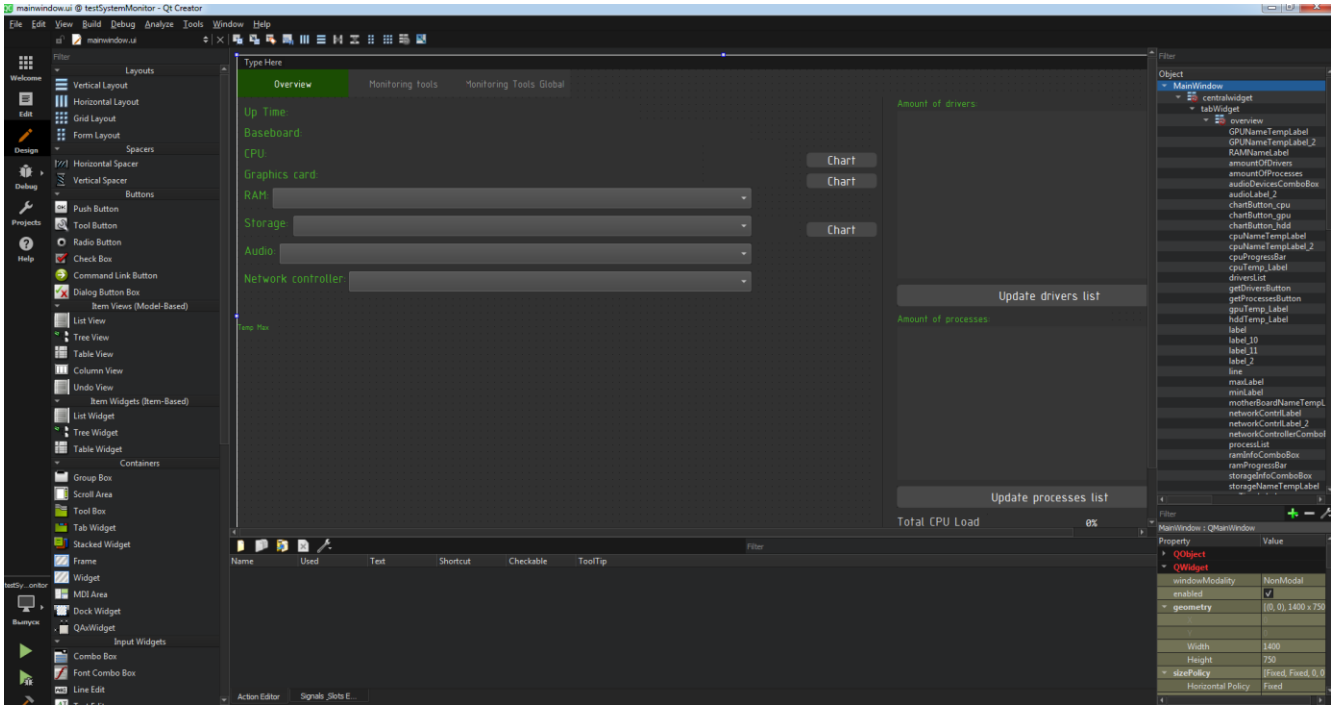

Рис. 2.12. Вигляд основного вікна даного проєкту у *Qt Designer*

Мова програмування *C++* була розроблена Бьуорном Страуструпом в підрозділі *Bell Labs* компанії *AT & T* в якості доповнення до мови *C* в 1979 р. Він додав безліч нових можливостей в мову *C*. Її популярність була викликана об'єктно-орієнтованістю мови. На *C++* створюють операційні системи, різноманітні прикладні програми, драйвера пристроїв, ігри тощо.

До переваг *C++* можна віднести такі механізми мови: нединамічну типізацію, свободу у роботі за пам'яттю, вказівники, посилання, побітові оператори, об'єктно-орієнтованість мови, шаблони, стандартна бібліотека *C++* (включає в себе різноманітні класи, що спрощують написання кода. До таких класів можна віднести контейнерні класи як *std::vector*, *std::map*, *std::deque* та багато інших). Також дуже зручним інструментом зі стандартної бібліотеки є *std::string*, що дозволяє вести роботу зі строками [8, с. 1224]. Окрім цих механізмів *C++* має перевагу, тому що існує безліч готових бібліотек, які досить відключити у свою програму, щоб почати роботу з ними.

У проєкті *C++* використовується для написання всієї програми, взаємодії мови з бібліотеками. Програма написана у стилі ООП і включає в себе 6 класів.

2.3. Технології програми для отримання температур

Розробники утиліти *Open Hardware Monitor* представили відкриту однойменну бібліотеку, інструменти якої дозволяють отримувати дані про апаратної складової комп'ютера. Дані виходять використанням *WMI*-запитів, однак процес їх вилучення помітно спрощений. Основними типами даних є:

1. *HardwareType* – перерахування, призначене для розрізнення частин обладнання друг від друга. Ця структура не покладається на конкретний тип, так як отримання даних залежить від датчиків, що фіксують параметри, а такі можуть бути відсутніми. У проєкті використані деякі значення перерахування: *CPU, RAM* і *HDD*.

2. *SensorType* – перерахування для розрізнення датчиків певних типів параметрів. Використано *Voltage, Clock, Temperature, Load, Data, Level*.

Технологія запитів *Open Hardware Monitor* заснована на взаємозв'язку примірників датчиків і апарату: апаратна компонента має кілька датчиків, кожен датчик має один батьківський апарат. Властивість *Hardware* повертає колекцію доступних апаратних елементів. Властивості *Name, HardwareType* дозволяють отримати і розмежувати одержувані параметри. Властивість *Sensors* повертає колекцію параметрів датчиків. Інформація про кожен показник отримується за допомогою властивостей *Name* і *Value*.

Таким чином, процес отримання даних про апаратний стан являє собою наступний алгоритм.

У якості батьківського програмного компонента виступає *Hardware.computer*, який налаштовується в залежності від необхідних даних за допомогою властивостей: *RAMEnabled*, *HDDEnabled*, *CPUEnabled*.

Властивість *Computer.Hardware* повертає всі доступні апаратні компоненти, які фільтруються по типу за допомогою властивості *HardwareType*.

У кожного елемента *Computer.Hardware* викликається властивість *Sensors*. Отримана колекція типів датчиків фільтрується по типу реєстрованих даних. Значення даних передаються в результуючі списки.

У даному проєкті бібліотека *OpenHardwareMonitor* використовується для отримання температур ЦП, ГП та жорстких дисків.

Microsoft Visual Studio — серія продуктів фірми *Microsoft*, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології *Windows Forms*, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються *Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight* [4, с. 12].

На рис. 2.13 наведено інтерфейс *Microsoft Visual Studio.*

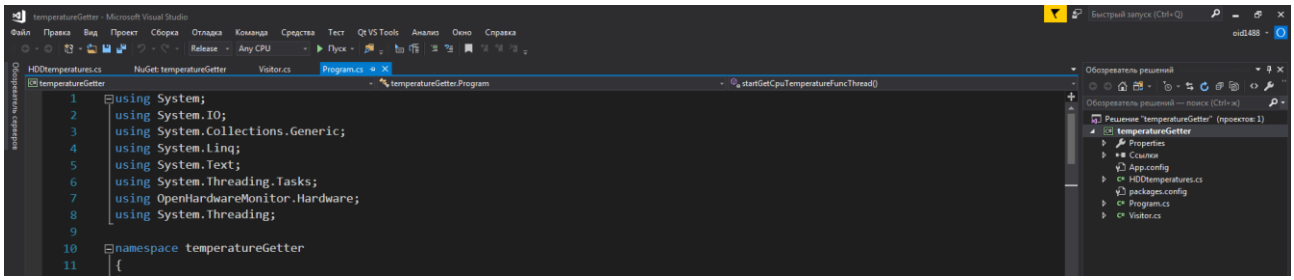

Рис. 2.13. Інтерфейс *Microsoft Visual Studio 2017*

Мова програмування *C#* – це об’єктно-орієнтована мова програмування, розроблена в 1998-2001 рр. групою інженерів під керівництвом Андерса Хейлсберга в компанії *Microsoft* як мова розробки додатків для платформи *Microsoft .NET Framework* [4, с. 12].

У дипломному проєкті мова *C#* використовувалась для створення програмного модуля, що взаємодіє з відкритою бібліотекою *Open Hardware Monitor.*

2.4. Технології програми для прошивки мікроконтролера на базі плати *Arduino*

Наявність в одному корпусі більшості системних пристроїв зробило мікроконтролер подібним до звичайного комп'ютера. Відповідно і бажання використовувати мікроконтролери як звичайні комп'ютери з'явилося практично з їх появою. Але бажання це стримувалося багатьма факторами, наприклад, щоб зібрати пристрій на мікроконтролері, необхідно знати основи схемотехніки, пристрій і роботу конкретного процесора, вміти програмувати на асемблері і виготовляти електронну техніку. Також будуть потрібні програматори, відладчики і інші допоміжні пристрої. В результаті без величезного обсягу знань і дорогого обладнання не обійтися. Зараз, з появою пристроїв, що дають можливість працювати з мікроконтролерами без наявності серйозної матеріальної бази і знання багатьох предметів, все змінилося. Прикладом такого пристрою може бути проєкт *Arduino* італійських розробників, *Arduino* і його клони являють собою набори, до складу готового електронного блоку і програмного забезпечення. Електронний блок тут – це друкована плата з встановленим мікроконтролером і мінімумом елементів, необхідних для його роботи. Фактично електронний блок *Arduino* є аналогом материнської плати сучасного комп'ютера. На ньому є роз'єми для підключення зовнішніх пристроїв, а також роз'єм для зв'язку з комп'ютером, по якому і здійснюється програмування мікроконтролера. Особливості використовуваних мікроконтролерів *ATmega* фірми *Atmel* дозволяють виконувати програмування без застосування спеціальних програматорів [5, с. 19].

У цьому проєкті *Arduino* було використано для створення схеми з додатковим екраном та потенціометром, екран використовується для додаткового виводу інформації.

Arduino IDE – інтегроване середовище розробки призначена для створення і завантаження програм на *Arduino*-сумісні плати, а також на плати інших виробників. *Arduino IDE* дозволяє писати код, компілювати його та завантажувати на обрану плату. Також є можливість моніторингу *Serial*-порту.

У даному проєкті *Arduino IDE* використовувалась для написання, перевірки та завантаження коду на плату.

Послідовний порт або *COM*-порт – двонаправлений послідовний інтерфейс, призначений для обміну байтовою інформацією. Основною одиницею зберігання даних в пам'яті є двійковий розряд, який називається бітом. Біт може містити 0 або 1. Це найменша одиниця пам’яті [4, с. 94]. Послідовний тому, що інформація через нього передається по одному біту, біт за бітом (на відміну від паралельного порту).

У даному проєкті *COM*-порт було використано для зв’язку ПК з платою *Arduino* та побайтовій передачі даних з ПК на плату *Arduino.*

Мова програмування *Arduino* називається *Arduino C* і являє собою мову *C++* з фреймворком *Wiring*, вона має деякі відмінності за частиною написання коду, який компілюється і збирається за допомогою *avr-gcc*, з особливостями, які полегшують написання працюючої програми є набір бібліотек, що включає в себе функції і об'єкти. При компіляції програми *IDE* створює тимчасовий файл з розширенням **.cpp*. Програми, написані програмістом *Arduino*, називаються скетчами (транслітерація від англ. *Sketch*) і зберігаються в файлах з розширенням **.ino*. Ці файли перед компіляцією обробляються препроцесором *Arduino*. Також існує можливість створювати і підключати до проєкту стандартні файли *C++*.

У даному проєкті мова програмування *Arduino C* була використана для написання скетчу для прошивки мікроконтролера плати *Arduino.*

2.5. Висновки до розділу

У даному розділі було проведено аналіз використаних технологій проєкту. Було розписано технології використані для кожного з описаних модулів. Було визнавчено три модулі – основна програма, програма отримання температур та програма прошивки плати *Arduino*.

У розділі було описано технології, що були вкиористані для основної програми: *WinAPI, WMI, Qt*, мова програмування *C++*.

Також було проаналізовано, що для програми отримання температур було використано такі технології як *Open Hardware Monitor* та мову програмування *C#.*

Було розглянуто прогруму для прошивки плати на базі *Arduino*, для програми були використані такі технології як *Arduino IDE* та мова програмування *Arduino C.*

Окрім того було використано стандартну технологію *C++* для запису у файл для передачі інформації від програми отримання температур до основної програми; *COM*-порти для передачі інформації від основної програми до програми прошивки *Arduino*.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ МОНІТОРИНГУ АПАРАТНИХ ТА ПРОГРАМНИХ РЕСУРСІВ КОМП'ЮТЕРА НА ОСНОВІ ARDUINO

3.1. Функціонал готового програмного модуля моніторингу апаратних та програмних ресурсів комп'ютера на основі *Arduino*

Як раніше було зазначено, готовий проєкт складається з трьох підмодулів. Перший модуль – це основна програма, яка включає в себе основний інтерфейс та функціонал. Програма складається з трьох вкладок – *"Overview", "Monitoring Tools", "Monitoring Tools Global"*.

Вкладка *"Overview"* складається з інформації про апаратну частину ПК – назва компонентів, виробники, характеристики компонентів, температури де це можливо; графік температур обраного користувачем компоненту; список драйверів, список процесів; поточне навантаження на ЦП та ОЗП. Саме з цією програмою взаємодіє користувач.

Вкладка *"Monitoring Tools"* надає можливість відстежувати перенавантаження ЦП та ОЗП від конкретного процесу. Користувач задає допустиме навантаження та запускає процес моніторингу. Окрім того користувач задає бажаний тип відклику – повідомлення, або лог. Якщо навантаження перевищило допустиме навантаження, то користувачу буде надіслане повідомлення, або записано інформацію в лог (в залежності від обраного користувачем типу відклику).

Вкладка *"Monitoring Tools Global"* надає можливість відстежувати перенавантаження ЦП та ОЗП. На цій вкладці також можна обрати тип виводу. Таким чином при перевищенні допустимого навантаження користувачу буде надіслане повідомлення, або записано інформацію в лог (в залежності

від обраного користувачем типу відклику).

Окрім того основна програма надсилає дані третьому модулю (плата *Arduino*) для виводу на екран.

Другий модуль – програма, що отримує значення температур. Ця програма отримує температури зазначених пристроїв (ЦП, ГП та жорстких дисків), потім записує їх у текстовий файл для зчитування основною програмою.

Третій модуль – прошивка для плати *Arduino* та сама схема на базі *Arduino.* Схема отримує інформацію від ПК (передається від основної програми) та виводить її на власний екран. У схемі використовується плата *Arduino Uno*, дісплей 1602 з інтерфейсною шиною *IIC/I2C*, потенціометр на 10 кОм, 4 світлодіоди.

У результаті було отримано діаграму наведену на рис. 3.1.

Блок-сехму алгориту роботи програми наведено на рис. 3.14-3.17.

Рис. 3.1. Схема функціоналу всіх підмодулів

На рис. 1 – рис. 4 додатку В наведено блок-схему алгоритму програми. А на рис. 5 додатку В наведено схему структури програми.

3.2. Принцип роботи програмного модуля моніторингу апаратних та програмних ресурсів комп'ютера на основі *Arduino*

Як вже було раніше зазначено основний модуль складається з трьох підмодулів: основної програми, програми для отримання температур та прошивка для плати *Arduino* та сама схема на базі *Arduino.*

Принцип роботи основної програми: як вже було зазначено в основній програмі є 3 вкладки. У самій програмі є 6 класів, а саме: *arduinocomcenter, countingcenter, cpuusage, hardwareinformationcenter, hardwareinformationcenter, mainwindow, mainwindow.* Проаналізовано функціонал кожного з наведених класів.

Arduinocomcenter – клас, що встановлює контакт із платою *Arduino* за допомою функції *openComPort().*

Запускається функція *getInfoForPackage(int cpuLoad, int cpuTemp, int gpuTemp, int ramLoad, int hdd1Temp, int hdd2Temp, int hdd3Temp, int hdd4Temp, int upTimeHours, int upTimeMinuters, int upTimeSeconds).* Ця функція отримує потрібні значення від *mainwindow.* Після отримання потрібних даних запускається функція *createPackege(),* яка формує пакет для передачі. Пакет формується з наступних даних: *CPU_LOAD, CPU_TEMP, GPU_TEMP, RAM_LOAD, HDD1_TEMP, HDD2_TEMP, HDD3_TEMP, HDD4_TEMP, UPTIME_HOURS, UPTIME_MINUTES, UPTIME_SECONDS,* тобто до даних входить навантаження на ЦП, ОЗП, температура ЦП, ГП, доступних жорстких дисків, кількість годин, хвилин та секунд поки система працює. Після цього запускається функція *writeToComPort()* яка починає запис інформції у *COM*-порт. після запису основна програма має закрити *COM*-порт за допомогою *closeCom().*

На рис А.1. наведено *UML*-діаграму класу *Arduinocomcenter.*

Countingcenter – клас, що виконує більшу частину функціоналу програми. Цей клас містить функції для отримання списку драйверів, списку процесів, створення файлів для запису у лог. Також саме тут реалізовані функції

моніторингу навантаження на ЦП та ОЗП від конкретних процесів, та навантаження на ЦП та ОЗП в цілому.

Окрім того в класі реалізовано допоміжні функції – для запуску потоків моніторинга навантаження на ЦП та ОЗП (глобально та від кожного процесу), сеттери різноманітних сигналів та значень, обчислення тіків процесора, перевод від типу даних *FILETIME* до *int64.*

Детальний опис функцій:

void CountingCenter::createFile(int whichFile, bool global, QString fileName) – функція, що створює файл або файли для запису у лог. У разі, якщо в функцію було передано імя для файла від користувача, файл буде створено з цим іменем. Якщо імя не було зазначено, то функція визначить точний час початку створення файлу та візьме його за основу назви файлу, далі в залежності від того який моніторинг буде писатись у лог додається постфікс «*_CPU*» або «*_RAM*», якщо моніторинг проводиться для навантаження в цілому то ще буде додано постфікс «*_global*». В результаті файли будуть мати назви по типу тих, що наведено на рис. 3.2.

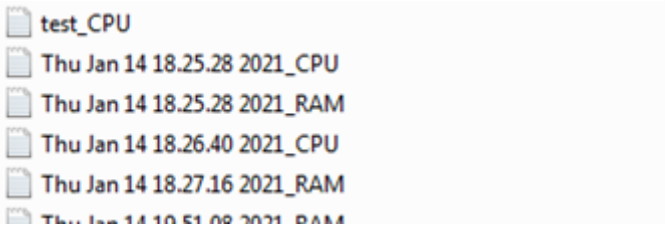

Рис. 3.2. Сформовані назви файлів-логів

bool CountingCenter::processCPUProcLoadMonitoring(DWORD processID, const WCHAR exeFile)* – функція, що виконує моніторинг навантаження на ЦП від кожного процесу. Функція отримує поточне навантаження на ЦП від конкретного процесу за допомогою методу *getUsage(processID)* класу *cpuUsage* (він буде розглянутий наступним). У *getUsage(processID)* передається *ID* процеса, у фунцію *processCPUProcLoadMonitoring* цей *ID* передається аргументом *DWORD processID.* Таким чином було отримано навантаження для конкретного процесу, такий процес проводиться для кожного процесу зі списку процесів *std::vector<processInfo> processInfosCPU*, цей список заповнюється за допомогою раніше описаної функції *getProcesses.* Після отримання поточного

навантаження проводиться порівняння його із значенням, що вказав користувач, у разі перевищення допустимого навантаження буде виведено повідомлення або зроблено запис у лог (в залежності від обраного користувачем методу). Текст повідомлення та запису у лог однаковий, він формується у цій функції у разі перевищення допустимого навантаження.

Нижче наведено код формування тексту:

```
auto timer = std::chrono::system_clock::now();
std::time_t end_time = std::chrono::system_clock::to_time_t(timer);
infoString.append(ctime(&end_time));
infoString.append( "PROCESS: ");
QString procNameBuff;
infoString.append(procNameBuff.fromWCharArray(exeFile));
infoString.append(" LOAD: ");
QString currentLoadString = QString::number(currentLoad);
infoString.append(currentLoadString);
```

Таким чином у текст повідомлення входить дата та час перенавантаження, назва процесу та навантаження на ЦП. Приклад такого повідомлення наведено на рис. 3.3:

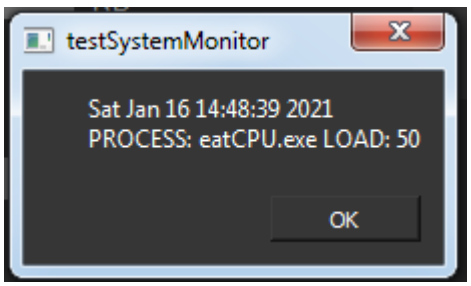

Рис. 3.3. Приклад повідомлення про перенавантаження на ЦП від конкретного процесу.

Приклад запису у лог можна побачити на рис. 3.4.

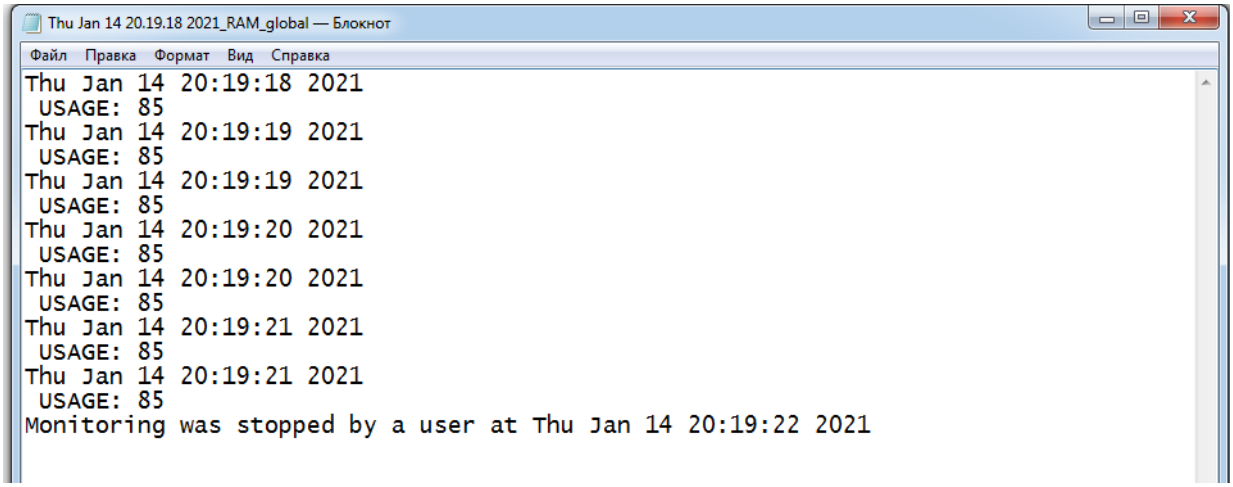

Рис. 3.4. Приклад запису у лог

Однак сам процес запису у лог або створення повідомлення відбувається не в цій функції, для запису у лог та створення повідомлення створений зв'язок за допомогою сигналів та слотів. При перевищенні допустимого навантаження випромінюється сигнал *emitMessage* який пов'язаний зі слотом *getMessage* із класа *MainWindow*, що буде описан нижче.

bool CountingCenter::processRAMProcLoadMonitoring(DWORD processID, const WCHAR exeFile)* – функція, що виконує моніторинг навантаження на ОЗП від кожного процесу. Спочатку використовується функція *OpenProcess* яка відкриває існуючий об'єкт локального процесу. Після цього використовується функція *GetProcessMemoryInfo*, яка отримує інформацію про використання пам'яті зазначеного процесу. Далі принцип роботи такий самий як і у функції *processCPUProcLoadMonitoring* – порівняння отриманого навантаження з зазначеним користувачем. У разі перенавантаження надсилається повідомлення або проводиться запис у лог.

float CountingCenter::getCPULoad() – функція отримання загального навантаженна на ЦП. Тут використовується функція *GetSystemTime,* яка повертає таймінги системи та допоміжну функцію *float CountingCenter::calculateCPULoad (unsigned long long idleTicks, unsigned long long totalTicks)*, яка проводить обчислення тактів процесора.

float CountingCenter::getRAMLoad() – функція отримання загального навантаженна на ОЗП. Використовує функцію *GlobalMemoryStatusEx*, яка отримує інформацію про поточне використання системою фізичної та

віртуальної пам'яті. Функція обраховує співвідношення використаного обєму памяті та усієї кількості обєму памяті та переводить цю інформацію у проценти для подальшого виводу користувачу.

На рис А.3. наведено *UML*-діаграму класу *Countingcenter.*

Cpuusage – клас, що вираховує навантаження на ЦП від конкретного процесу. Клас містить функцію *short cpuUsage::getUsage(DWORD processID),* функція приймає аргументом ID конкретного процесу та проводить обрахунок для цього процесу. Сама функція має такий принцип – отримання час простою системи, час роботи системи в режимі ядра, час проведений системою у режимі користувача, час створення процесу, час закриття процесу, час процесу проведений у режимі ядра, час процесу проведений у режимі користувача. Після чого за допомогою допоміжної функції *SubTime* проводиться віднімання отриманих показників від попередньо записаних (при минулій ітерації). Після чого йде розрахунок за формулою.

$$CPU\% = \frac{\text{Загальне навантаження від процеса}}{\text{Загальне навантаження системи}}$$

На рис А.4. наведено *UML*-діаграму класу *Cpuusage.*

HardwareInformationCenter – клас, що відповідає за отримання інформації про апаратну частину ПК.

Клас містить такі функції: *QString getCPUInfo(), QString getGPUInfo(), QString* getRAMInfo(int *amountOfBars), QString getBaseboardInfo(),QString* getStorageInfo(int *amountOfDisks), QString* getAudioDevicesInfo(int *amountOfDevices), QString* getNetworkControllers(int *amountOfControllers).

QString getCPUInfo() – отримує інформацію про ЦП за допомогою функції *__cpuid* яка створює інструкцію *cpuid*, доступну для x86 та x64. Ця інструкція запитує у процесора інформацію про підтримувані функції та тип процесора. Функція повертає *QString* з інформацією про ЦП.

QString getGPUInfo() – функція, що отримує інформацію про відеокарту, за допомогою функції *EnumDisplayDevices* яка дозволяє отримувати інформацію про пристрої відображення в поточному сеансі. Функція повертає *QString* з інформацією про відеокарту.

QString getRAMInfo(int *amountOfBars)* – функція, що повертає інформацію про ОЗП. Код функції наведено нижче:

```
initCOM();
WMI_getRAMInfo(ramInfo, amountOfBars);
cleanUpCOM();

return ramInfo;
```

initCOM() – відкриття *COM*-порту. Головну функцію виконує функція *WMI_getRAMInfo(ramInfo, amountOfBars)* – отримання інформації через *WMI* про ОЗП, а саме виробника, тип памяті, формфактор, об'єм пам'яті, швидкість.

cleanUpCOM() – закриття *COM*-порту.

QString getBaseboardInfo() – функція, що повертає інформацію про материнську плату. Код функції наведено нижче:

```
QString buffer;
initCOM();
buffer = WMI_getBaseboardInfo();
cleanUpCOM();

return buffer;
```

initCOM() та *cleanUpCOM()* вже були описані раніше, *WMI_getBaseboardInfo()* – отримання інформації через *WMI* про материнську плату, а саме виробника та модель.

QString getStorageInfo(int *amountOfDisks)* – функція, що повертає інформацію про жорсткі диски. Код аналогічний двом минулим функціям, отримання інформації відбувається також за допомогою *WMI*, до інформації належить назва пристроїв та об'єм пам'яті.

QString getAudioDevicesInfo(int *amountOfDevices)* – функція, що повертає інформацію про аудіо пристрої. Отримання інформації відбувається також за допомогою *WMI*, до інформації належить назва пристроїв.

QString getNetworkControllers(int *amountOfControllers)* – функція, що повертає інформацію про мережеві пристрої. Отримання інформації відбувається також за допомогою *WMI*, до інформації належить назва пристроїв.

Окрім наведених вище функцій у даному класі ще наявна функція отримання часу роботи системи та зчитування даних від модуля отримання температур.

void HardWareInformationCenter::getUptime(long &hours, long &minutes, long &seconds, long &millies, HardWareInformationCenter &hc) – функція, що отримує дані про час роботи системи за допомогою функції *std::chrono::milliseconds(GetTickCount64())*, після чого отримані мілісекунди переводяться у години, хвилини, секунди та мілісекунди.

std::vector<string>

HardWareInformationCenter::readTemperaturesFromFile() – відкриває файл створений модулем отримання температур, зчитує дані та зберігає їх у відповідних змінних для подальшого виводу для користувача.

UML-діаграму класу *HardwareInformationCenter* подано на рис рис А.5.

MainWindow – клас головного вікна програми, звідси відбувається майже всі виклики функцій описаних вище класів, клас містить в собі об’єкти класів *CountingCenter cc, HardWareInformationCenter hc, ArduinoComCenter acc*. Також тут прописується менша частина стилів. У цьому класі реалізовані зв’язки сигнал-слот, також з цього класу запускаються всі потоки – отримання часу активності системи, отримання навантаження на ЦП та ОЗП, отримання інформації про температури. Побудова графіків в залежності від обраного пристрою. Так як це клас основного функціоналу програми, тут описано всі алгоритми які повинні відбуватись при натисканні користувачем відповідних кнопок.

На рис А.6. наведено *UML*-діаграму класу *MainWindow.*

smartHandle – невеликий допоміжний клас, при створенні об’єкта передається інформація про хендл, при видаленні обєкта вся інформація видаляється.

На рис А.2. наведено *UML*-діаграму класу *MainWindow.*

3.3 Принцип роботи програми отримання температур

Як вже було описано вище, програма отримання температур написана за допомогою мови *C#* та відкритої бібліотеки *OpenHardwareMonitor*. Сама програма складається з трьох класів – основного класу, класу, який потрібен для коректної роботи *OpenHardwareMonitor* та допоміжного класу. Вони будуть надалі розглянуті детальніше.

*Program.cs – о*сновний клас, у якому відбувається створення потоків отримання температур. Тут же описані функції які безпосередньо отримують температури за допомогою бібліотеки *OpenHardwareMonitor*. В класі отримується інформація про температури наступних пристроїв: ЦП, відеокарти та жорстких дисків. Логіка роботи у всіх функціях однакова: налаштування параметрів для роботи за *OpenHardwareMonitor*, перебор всіх пристроїв до співпадіння з потрібним (ЦП, відеокарта, жорсткий диск, в залежності від обраної функції), після цього перебір всіх сенсорів до співпадіння з сенсором температур. Після цього з сенсора копіюється інформамція про температуру та повертається до викликача, після чого інформація записується у текстовий файл для подальшого передання її у основну програму.

Visitor.cs – Клас для налаштування необхідних для *OpenHardwareMonitor* параметрів.

temperatures.cs – допоміжний клас, створений для повернення температур жорстких дисків.

3.4. Принцип роботи прошивки для плати *Arduino* та схеми на базі *Arduino*

Алгоритм прошивки наступний: плата постійно зчитує інформацію з *COM*-порту, якщо кількість зчитаної інформації дорівнює зазначеному в коді розміру пакету, то зчитана інформація записується у відповідні зміні. Також

постійно проводиться зчитування від підключеного потенціометру, це зроблено для переключення інформації на підєднаному дисплеї 1602 з інтерфейсною шиною *IIC/I2C*.

Сама схема складається з плати *Arduino Uno*, дисплея 1602 з інтерфейсною шиною *IIC/I2C*, потенціометру на 10 *кОм*. Схематичне зображення наведено на рис. 3.5:

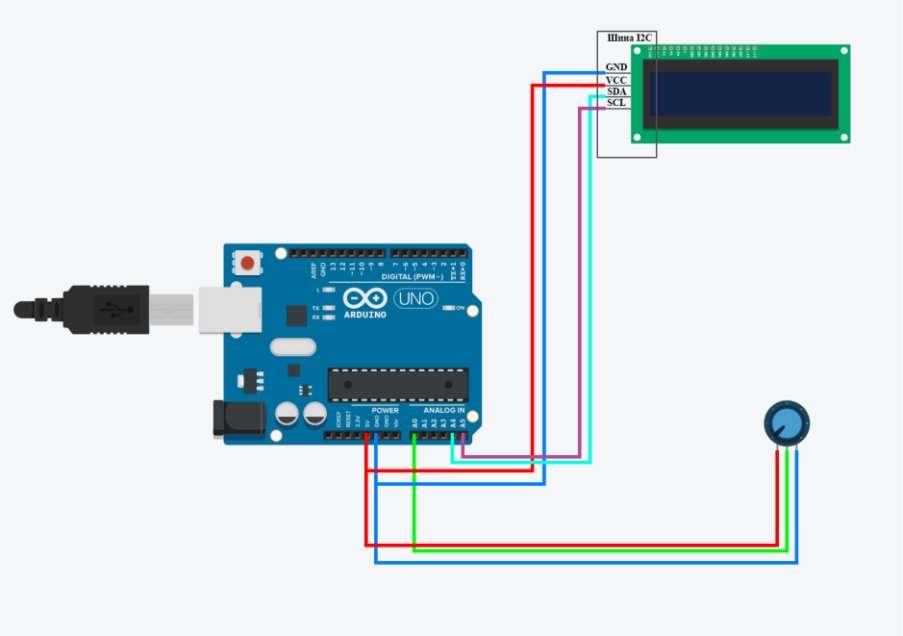

Рис. 3.5. Схематичне зображення

На рис. 3.6 наведено фізичну схему:

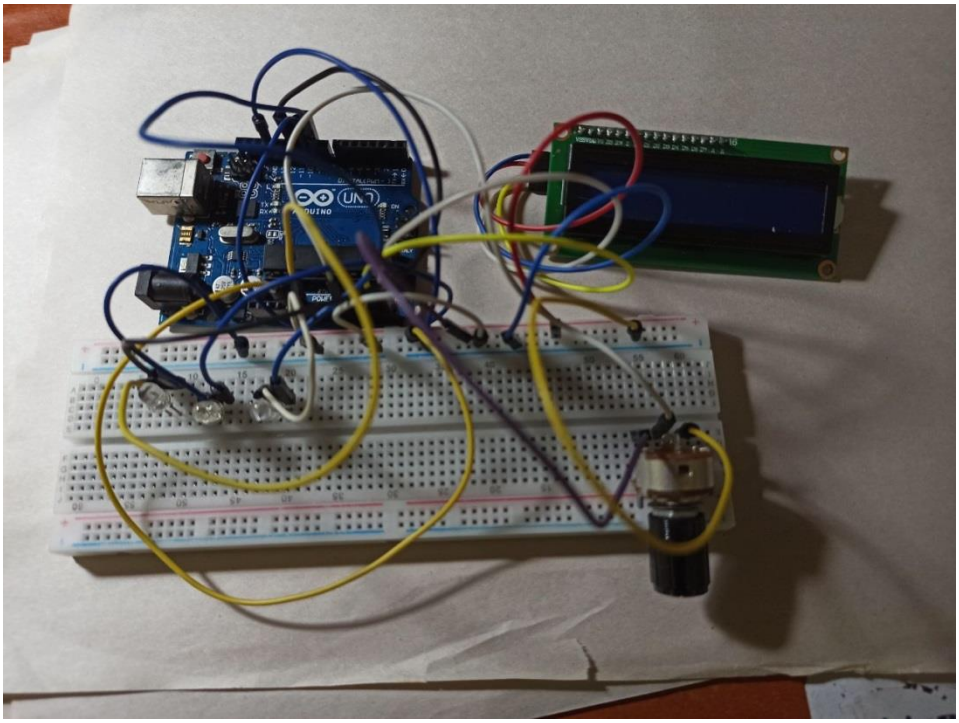

Рис. 3.6. Фізична схема на базі *Arduino*

3.5. Висновки до розділу

У розділі було наведено та описано класи програми та функціонал кожного класу. Розглянуто кожен підмодуль та його класи та у порівнянні із інформацією наведеною у другому розділі проведено більш детальний аналіз підмодулів та класів. Приведено діаграму функціоналу модулів програми. Для найбільших класів детально розписано частину функцій з поясненням алгоритму роботи описаної кожної функції, а саме:

– підмодулю «основна програма» та його класи – *Arduinocomcenter, Countingcenter, CpuUsage, HardwareInformationCenter, mainwindow, smartHandle*;

– підмодулю «програма для отримання температур», та його класи – *Program.cs, Visitor.cs, HDDtemperatures.cs.*

Було наведено логіку роботи прошивки для плати *Arduino* та схеми на базі *Arduino.* Також було наведено фізичну схему на базі *Arduino.* Наведено компоненти схеми на базі *Arduino.*

Представлено інтерфейс готового програмного модулю, описано кожну вкладку програмного модуля та показано вигляд зовнішнього монітору, який використовується для виводу інформації з плати *Arduino*.

ВИСНОВКИ

Дипломна робота присвячена темі «Програмний модуль моніторингу апаратних та програмних ресурсів комп’ютера на основі *Arduino*».

Під час виконання даної роботи було проведено аналіз комплектуючих сучасних ПК, наведено наступні компоненти ПК, та наведені їх основні характеристики:

- ЦП: тактова частота, частота шини, розрядність, кеш-пам’ять, *socket*, ядро;
- ГП: ядро, тактова частота, частота шини, кеш-пам'ять, кількість обчислювальних (шейдерних) блоків, швидкість заповнення;
- Відеокарта: тактова частота ГП, швидкість заповнення (філлрейт), обсяг відеопамяті, ширина шини памяті, типи памяті;
- ОЗП: тип памяті, форм-фактор, об’єм модуля пам’яті, тактова частота оперативної памяті;
- Жорсткий диск: інтерфейс, ємність, форм-фактор, час доступу, швидкість обертання шпинделя, введення-виведення;
- Материнська/головна плата: форм-фактор, типа сокета, наявність вбудованої звукової карти, наявність вбудованого мережевого контролера, слоти *PCI* та *PCI Express*;
- Система охолодження: утилізації тепла.

Проведений аналіз був здійснений для кращого розуміння інформації, що надають ПММАПР ПК.

Було розглянуто декілька версій ПММАПР ПК, проведено їх аналіз та порівняння у виді таблиці. Розглянуто програми *Aida64, Speccy, OpenHardwareMonitor, CPU-Z, ProcessHacker* та проведено їх порівняння.

Було проведено детальний огляд використаних технологій проєкту та обгрунтовано вибір технології для кожного з описаних модулів. Було визнавчено три модулі – основна програма, програма отримання температур та програма прошивки плати *Arduino*.

Також наведено аналіз та обгрунтовано вибір технологій, які були вкиористані для основної програми: *WinAPI, WMI, Qt*, мова програмування *C++*, а для програми отримання температур було використано такі технології як *OpenHardwareMonitor* та мову програмування *C#*. Наведено прогруму для прошивки плати на базі *Arduino*, для програми були використані такі технології як *Arduino IDE* та мова програмування *Arduino C*. Окрім того було зазначено факт використання стандартної технології *C++* для запису у файл для передачі інформації від програми отримання температур до основної програми; *COM*-порти для передачі інформації від основної програми до програми прошивки *Arduino*.

Було наведено та описано класи програми та функціонал кожного класу. Розглянуто кожен підмодуль та його класи та у порівнянні із інформацією наведеною у розділі 2 проведено більш детальний аналіз підмодулів та класів. Приведено діаграму функціоналу модулів програми. Для найбільших класів детально розписано частину функцій з поясненням алгоритму роботи описаної кожної функції, а саме:

– підмодулю «основна програма» та його класи – *Arduinocomcenter, Countingcenter, CpuUsage, HardwareInformationCenter, mainwindow, smartHandle*;

– підмодулю «програма для отримання температур», та його класи – *Program.cs, Visitor.cs, HDDtemperatures.cs.*

Було наведено логіку роботи прошивки для плати *Arduino* та схеми на базі *Arduino.* Також було наведено фізичну схему на базі *Arduino.* Наведено компоненти схеми на базі *Arduino.*

Представлено інтерфейс готового програмного модулю, описано кожну вкладку програмного модуля та показано вигляд зовнішнього монітору, який використовується для виводу інформації з плати *Arduino*.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С. В., Іванченко О. В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. Київ : НАУ, 2017. 63 с.

2. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : видання офіційне. Київ : Держстандарт України, 1995. 38 с.

3. Тарарака В. Д. Архітектура комп'ютерних систем: навч. посібник. Житомир : ЖДТУ, 2018. 383 с.

4. Труніна Г. О., Настенко Д. В., Нестерко А. Б. Обчислювальна техніка та програмування : конспект лекцій : навч. посіб. Київ : КПІ ім. І. Сікорського, 2020. 117 с.

5. Петлин В. А. Проекты с использованием контроллера Arduino. 2-е изд, перераб. и доп. СПб. : БХВ-Петербург, 2015. 464 с.

6. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. СПб. : Питер, 2013. 816 с.

7. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб. : Питер, 2015. 1120 с.

8. Страуструп Б. Программирование: принципы и практика с использованием C++. 2-е изд. : Пер. с англ. Москва, 2016. 1328 с.

9. Наука и техника : Энциклопедия Кругосвет. URL : https://www.krugosvet.ru/enc/tehnologiya-i promyshlennost/kompyuter?page=0%2C1#part-7 (дата звернення: 17.05.2021 р.).

10. Алексеев Е. Г., Богатырев С. Д. Информатика : Мультимедийный электронный учебник. URL : *http://inf.e-alekseev.ru/text/toc.html* (дата звернення: 17.05.2021 р.).

11. Онлайн справочник пользователя ПК. URL : *https://2hpc.ru/* (дата звернення: 17.05.2021 р.).

12. Socket (разъем) процессора: что это такое и как узнать его тип. URL : *https://www.chaynikam.info/socket_cpu.html* (дата звернення: 17.05.2021 р.).

13. Что такое центральный процессор? URL : https://mediapure.ru/matchast/chto-takoe-centralnyj-processor/ (дата звернення: 17.05.2021 р.).

14. Что такое видеокарта компьютера? URL : *https://geekkies.in.ua/pc/chto-takoe-videokarta-kompjutera.html* (дата звернення: 17.05.2021 р.).

15. Руководство покупателя игровой видеокарты. URL : *https://www.ixbt.com/video3/guide/guide-03.shtml* (дата звернення: 18.05.2021 р.).

16. Что такое жесткий диск? URL : *https://www.dropbox.com/ru/business/resources/what-is-a-hard-drive* (дата звернення: 18.05.2021 р.).

17. Дмитриева О. Накопители HDD и SSD: в чем разница? URL : *https://ichip.ru/sovety/nakopiteli-hdd-i-ssd-v-chem-raznica-194272* (дата звернення: 18.05.2021 р.).

18. Что такое материнская плата компьютера? URL : *https://geekkies.in.ua/pc/chto-takoe-materinskaja-plata-kompjutera.html* (дата звернення: 18.05.2021 р.).

19. Выбор и характеристики материнской платы. URL : *http://putevodytel.com/view_it_news.php?art=vibor_materinskoy_platy* (дата звернення: 18.05.2021 р.).

20. Блок питания. URL : *http://compolife.ru/ustrojstvo-kompjutera/blok-pitanija.html* (дата звернення: 19.05.2021 р.).

21. Как выбрать блок питания — критерии и характеристики. URL : *https://sonikelf.ru/vsya-pravda-o-bloke-pitaniya-skupoj-platit-dvazhdy/* (дата звернення: 19.05.2021 р.).

22. Системы охлаждения компьютера. URL : *https://it.ros-kit.ru/help/computers/sistemy-okhlazhdeniya-kompyutera/* (дата звернення: 19.05.2021 р.).

23. 4 лучших программы для стресс-тестирования процессора (диагностика работы ЦП). URL : *https://ocomp.info/stress-testirovaniya-protsessora.html* (дата звернення: .05.2021 р.).

24. About WMI : Microsoft Build. URL : *https://docs.microsoft.com/en-us/windows/win32/wmisdk/about-wmi?redirectedfrom=MSDN* (дата звернення: .05.2021 р.).

25. Уроки программирования на языке C++. URL : *https://ravesli.com/uroki-cpp/* (дата звернення: .05.2021 р.).

26. Open Hardware Monitor. Студенческая библиотека онлайн. URL : *https://studbooks.net/2271882/informatika/open_hardware_monitor* (дата звернення: .05.2021 р.).

Додаток А

Табл. 1.1

Порівняльна характеристика програм наведених у розділі 1.

Параметр	*Aida64*	*Speccy*	*OpenHardware Monitor*	*CPU-Z*	*ProcessHacker*
Докладна інформація про ЦП	+	+	-	+	-
Докладна інформація про ГП	+	+	-	+	-
Докладна інформація про ОЗП	+	+	-	+	-
Докладна інформація про жорсткий диск	+	+	-	+	-
Докладна інформація про материнську/головну плату	+	+	-	+	-
Докладна інформація про переферію ПК	+	+	-	-	-
Докладна інформація про монітори	+	+	-	-	-
Інформація про мережеві контролери та мережеві з’єднання	+	+	-	-	-/+
Докладна інформація про ОС	+	+	-	-	-
Інформація про процеси ОС	+	+	-	-	+

Продовження таблиці 1.1

Інформація про служби ОС	+	-	-	-	+
Інформація про час роботи системи	+	+	-	-	-
Температура ЦП	+	+	+	-	-
Температура ГП	+	+	+	-	-
Температура жорстких дисків	+	+	+	-	-
Температура материнської/головної плати	+	+	+	-	-
Навантаження на ЦП	+	-	+	-	+
Навантаження на ГП	+	-	+	-	+
Навантаження на ОЗП	+	+	+	-	+
Навантаження на жорсткі диски	+	+	+	-	-
Швидкість оберту вентиляторів кулерів	+	-	+	-	-
Інформація про *BIOS*	+	+	-	-	-

ArduinoComCenter

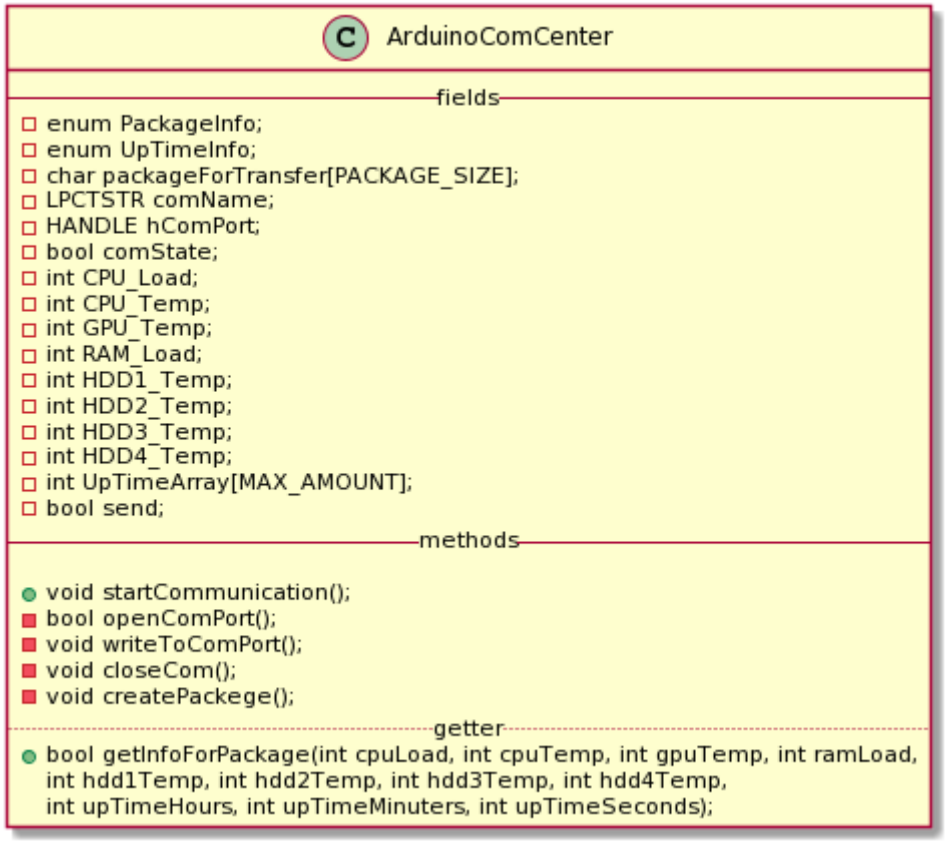

Рис. А.1. *UML*-діаграма класу *ArduinoComCenter*

smartHandle

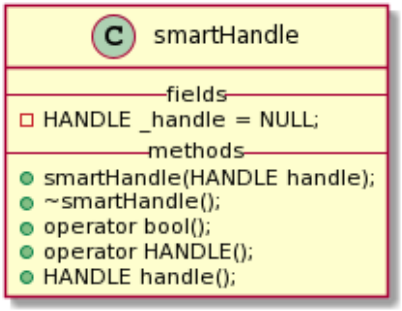

Рис. А.2. *UML*-діаграма класу *smartHandle*

CountingCenter

Рис. А.3. *UML*-діаграма класу *Countingcenter*

CpuUsage

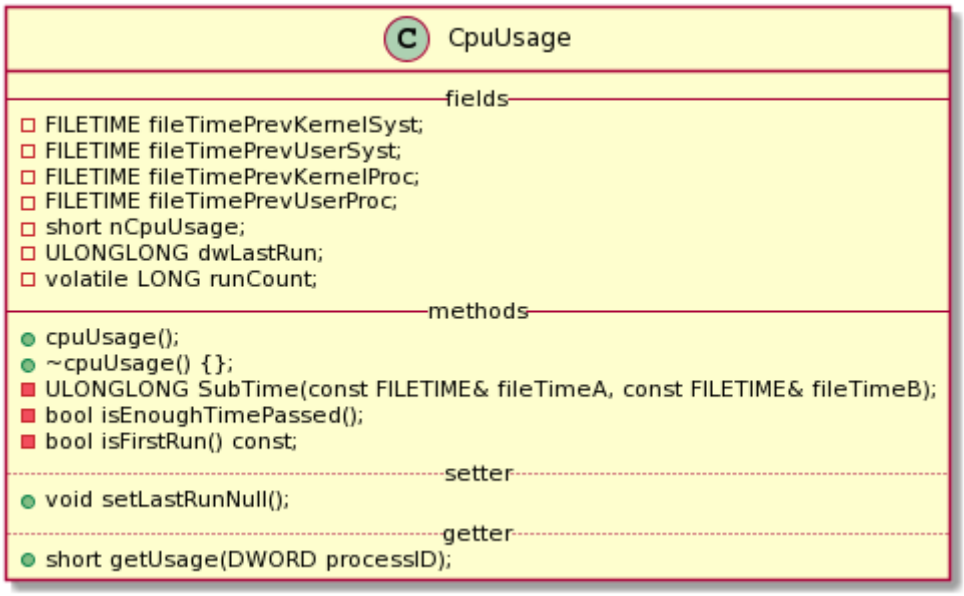

Рис. А.4. *UML*-діаграма класу *CpuUsage*

HardwareInformationCenter

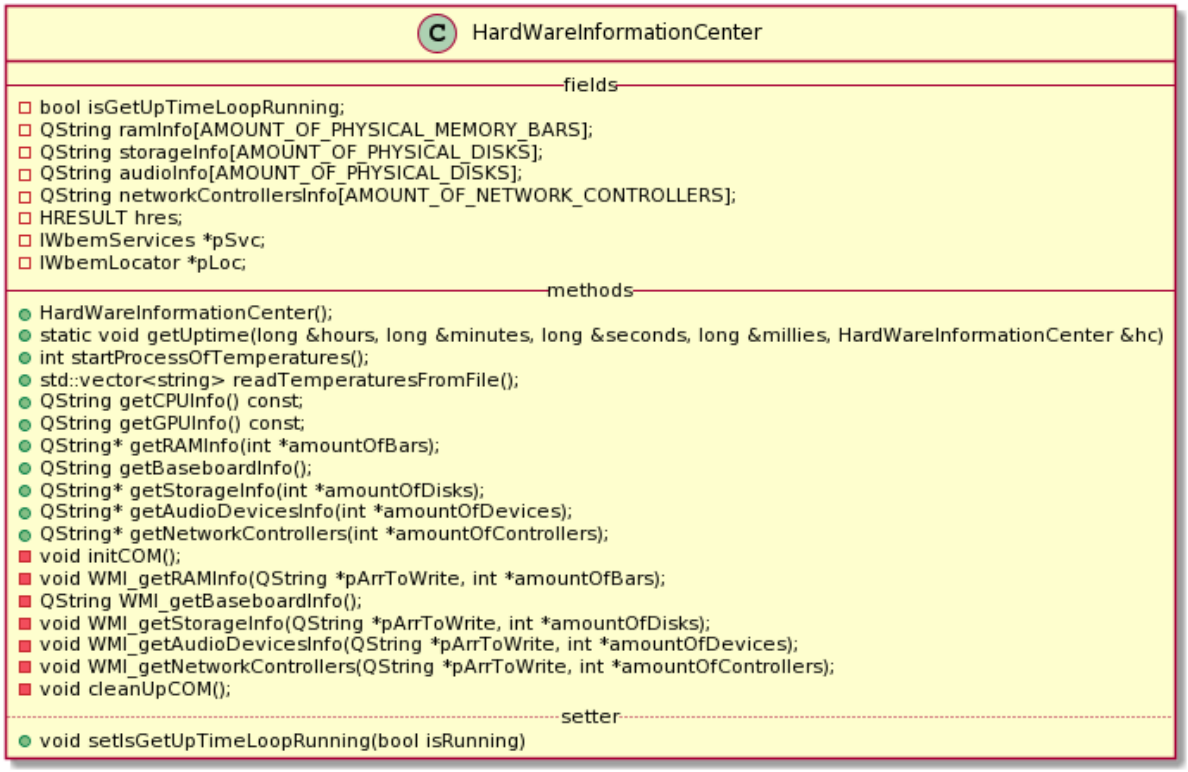

Рис. А.5. *UML*-діаграма класу *HardwareInformationCenter*

MainWindow

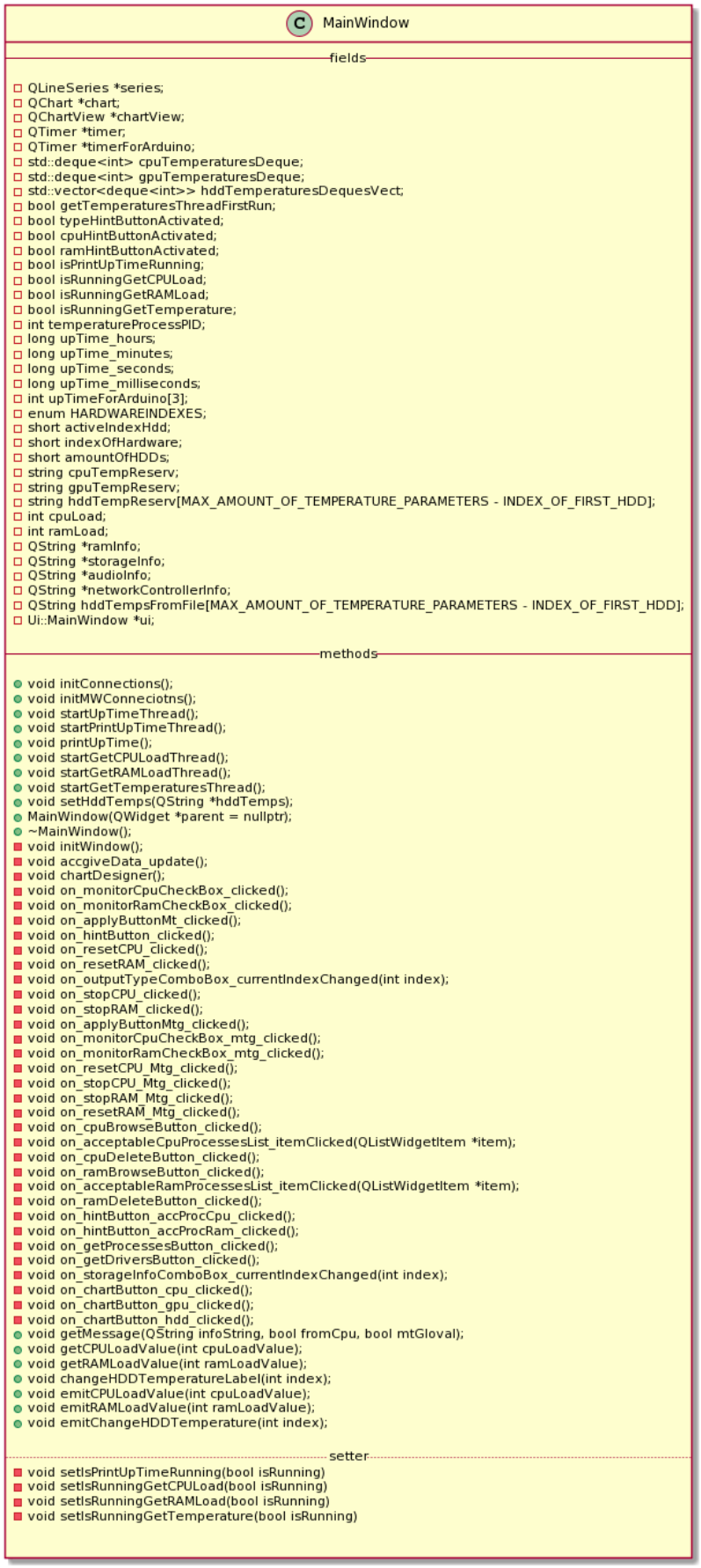

Рис. А.6. *UML*-діаграма класу *MainWindow*

Додаток Б

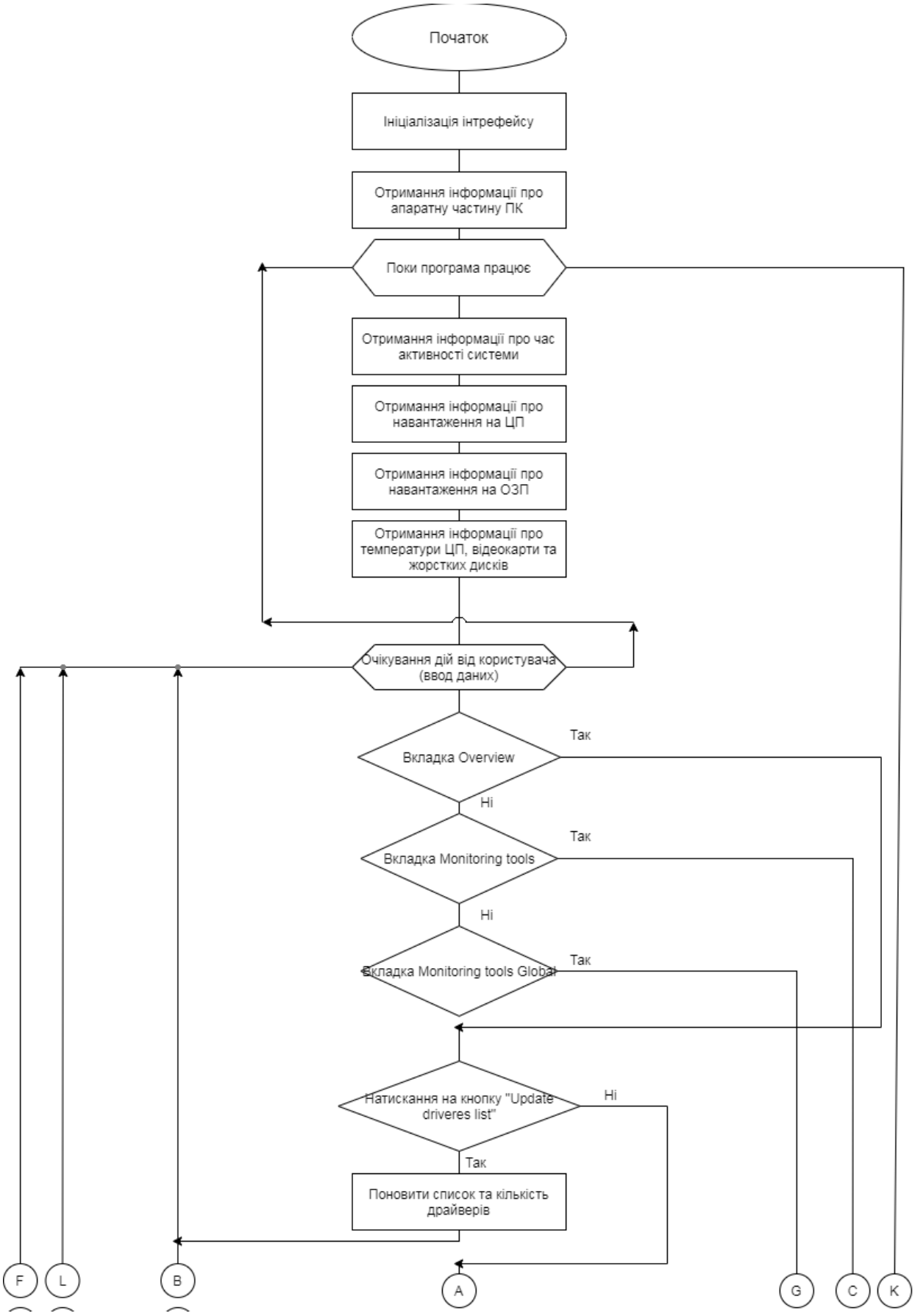

Рис. 1 Блок-схема алгоритму програми

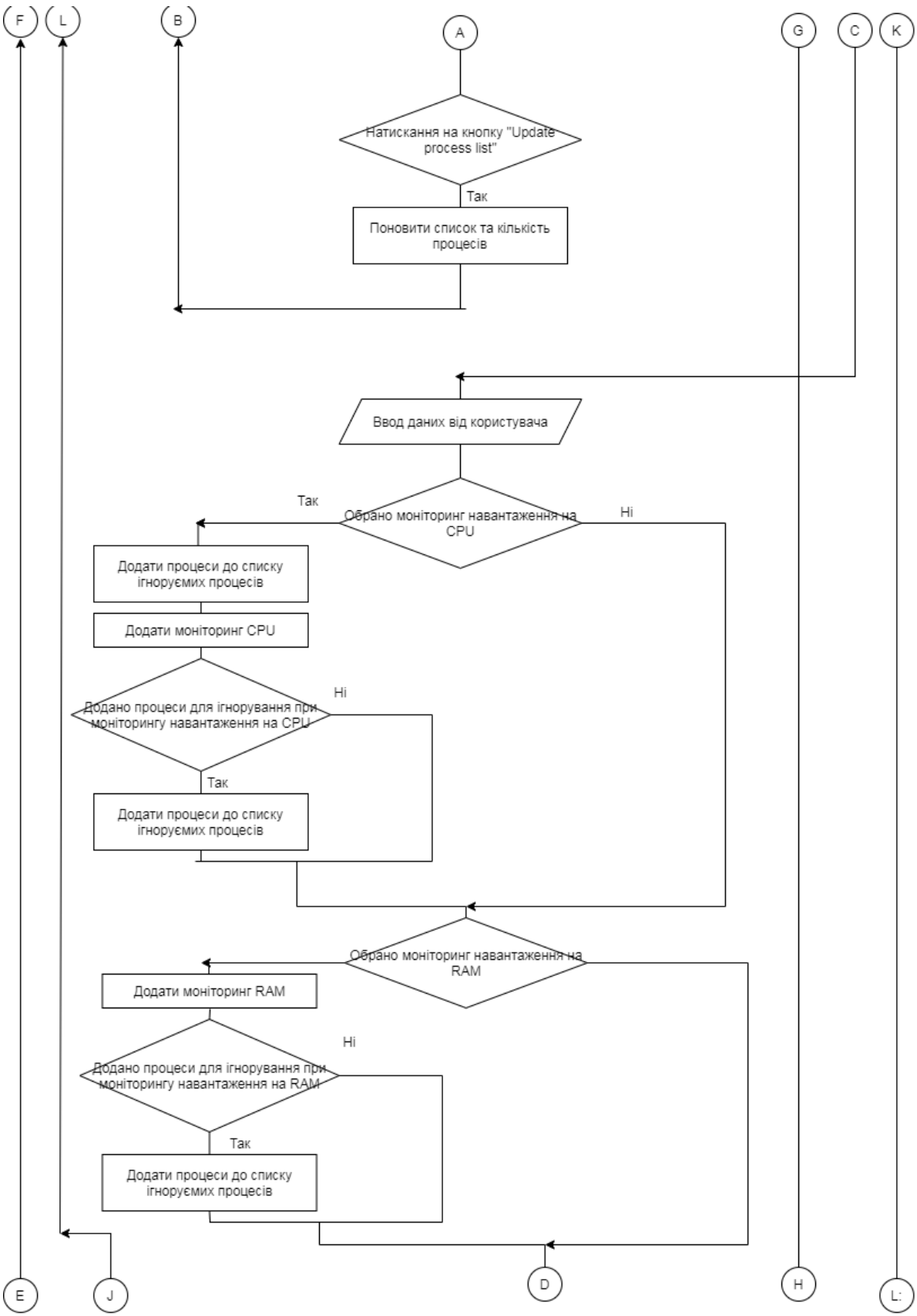

Рис. 2 Блок-схема алгоритму програми продовження

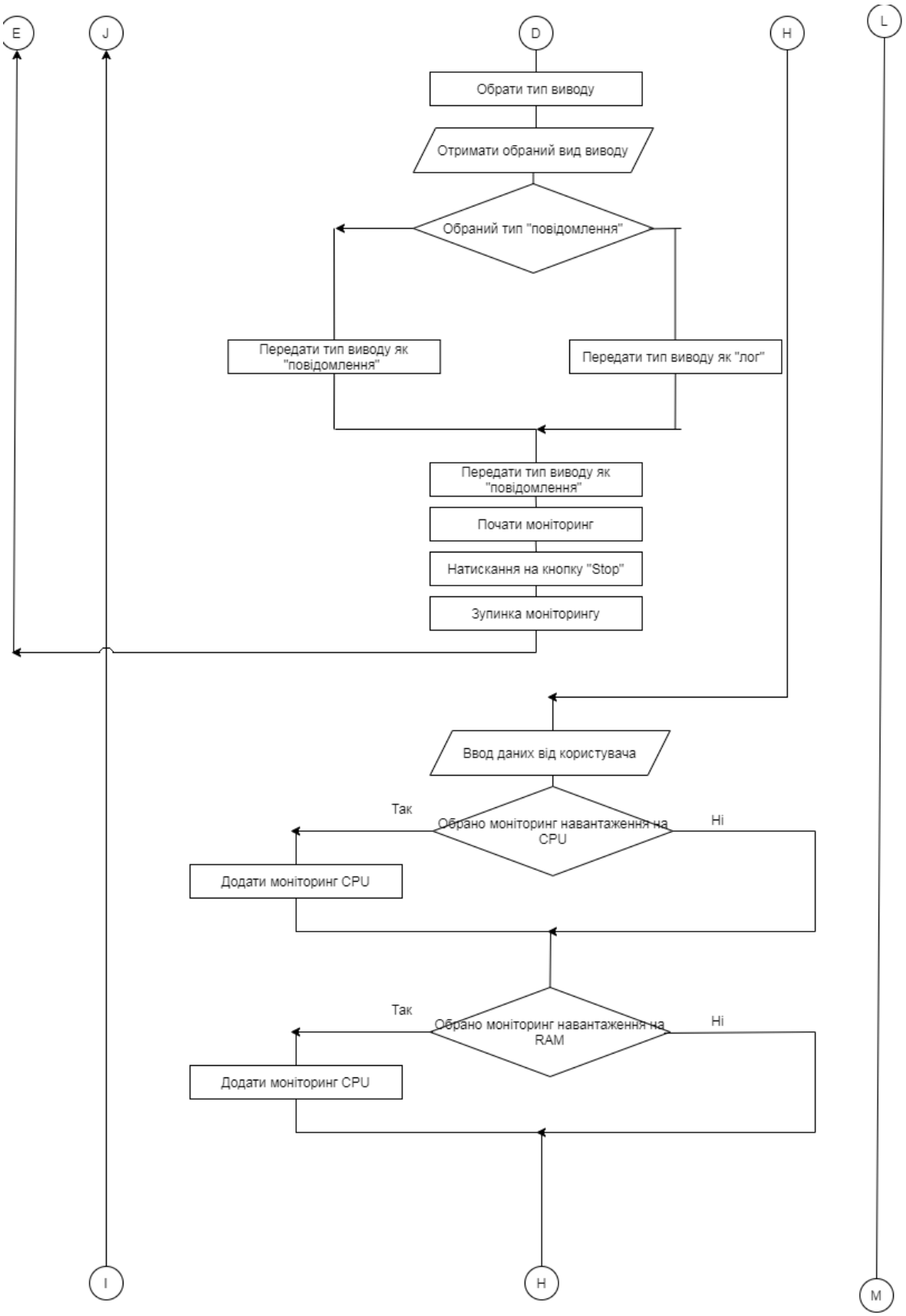

Рис. 3 Блок-схема алгоритму програми продовження

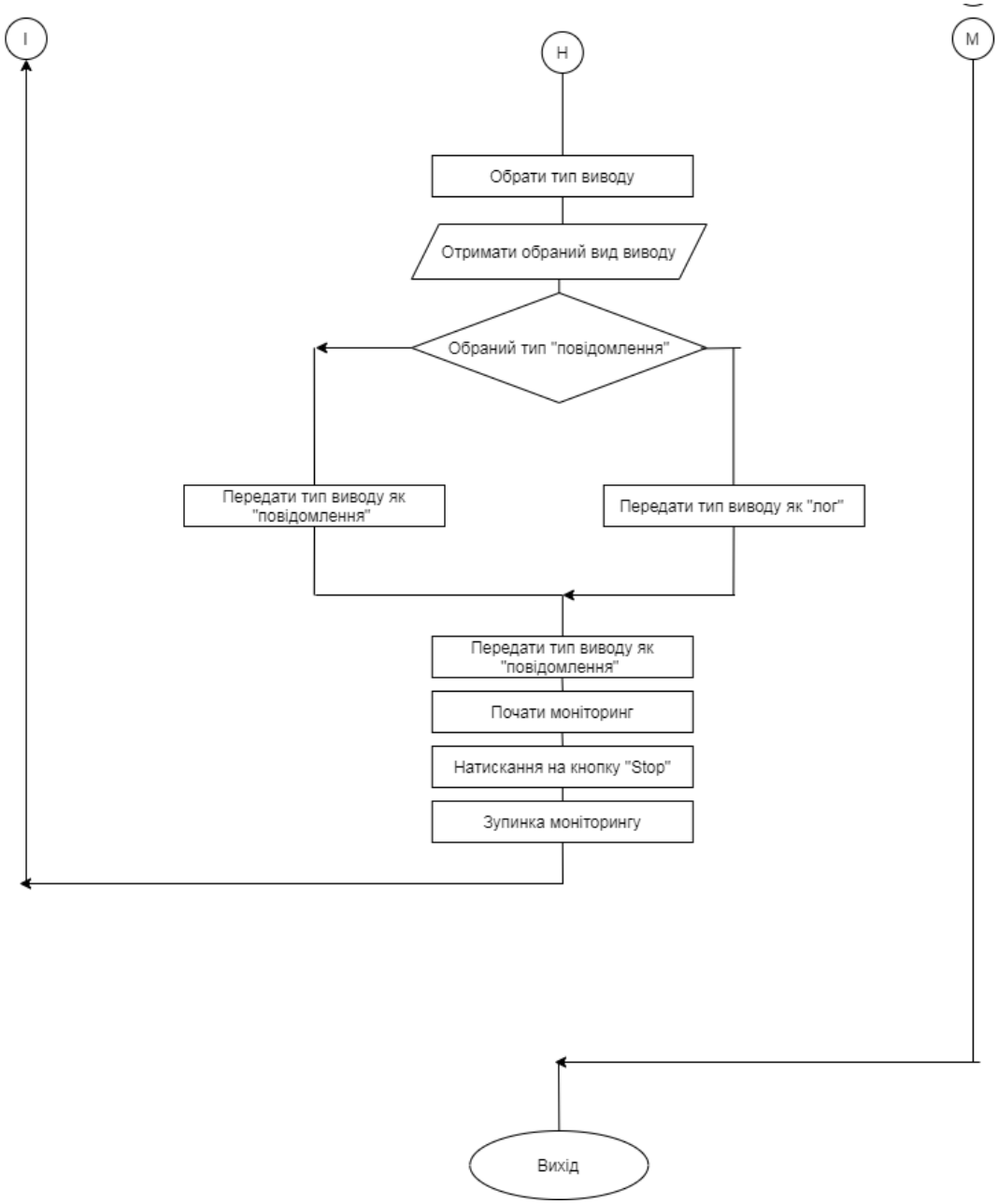

Рис. 4 Блок-схема алгоритму програми продовження

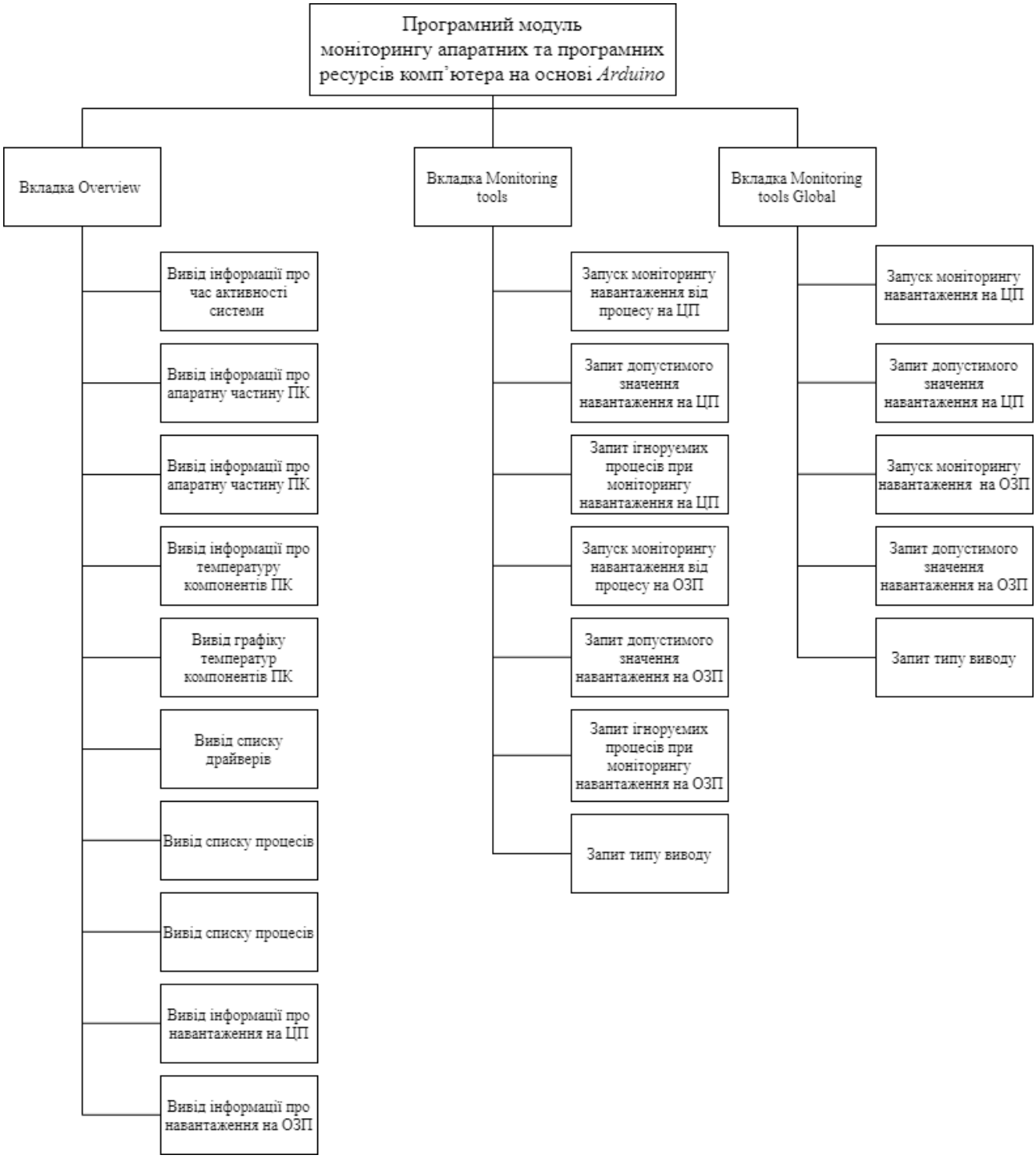

Рис. 5 Структурна схема програми

Додаток В

Лістинг коду програмного модуля

```
bool CountingCenter::processCPUProcLoadMonitoring(DWORD processID, const WCHAR* exeFile){
    qDebug() << "processCPUProcLoadMonitoring";
    int currentLoad = 0;

    QString infoString;

    auto timer = std::chrono::system_clock::now();
    std::time_t end_time = std::chrono::system_clock::to_time_t(timer);

    if(!stopFromUiCpuProcess){  //if user didnt stop the process
        currentLoad = usage.getUsage(processID);

       if (currentLoad > userAcceptableCpuLoad) {
          /*forming text for message*/
          infoString.append(ctime(&end_time));
          infoString.append( "PROCESS: ");
          QString procNameBuff;
          infoString.append(procNameBuff.fromWCharArray(exeFile));
          infoString.append(" LOAD: ");
          QString currentLoadString = QString::number(currentLoad);
          infoString.append(currentLoadString);

          if(outputTypeIsLogMt == 0){
             emit emitMessage(infoString, true, false);      //emits signa to mainwindow, where message box being formed
             return false;
          }
          else{
```

```
            qDebug() << "log";
            if (fileCpuMt->open(QFile::Append |QFile::Text))
            {
               qDebug() << "i";
               fileCpuMt->write(infoString.toUtf8() + "\n");
               fileCpuMt->close();
            }
         }

      }
      return true;
   }
   else{      //if user stoped the process
      QString stoppingMsg = "Monitoring was stopped by a user at ";
      stoppingMsg.append(ctime(&end_time));
      emit emitMessage(stoppingMsg, true, false);
      if(outputTypeIsLogMt == 1){
         if (fileCpuMt->open(QFile::Append |QFile::Text))
         {
            fileCpuMt->write(stoppingMsg.toUtf8() + "\n");
            fileCpuMt->close();
         }
      }
      return false;
   }
}

void CountingCenter::monitoringCpuStart(){
   CreateThread(NULL, 0, StaticThreadStart_CPU, (void*) this, 0, NULL);
                                                   }
//**RAM MONITORING**
```

```
bool CountingCenter::processRAMProcLoadMonitoring(DWORD processID, const
WCHAR* exeFile){
    QString infoString;

    auto timer = std::chrono::system_clock::now();
    std::time_t end_time = std::chrono::system_clock::to_time_t(timer);

    if(!stopFromUiRamProcess){
        HANDLE h = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processID);
        if (h == NULL) {
            std::cout << "i dont know why" << std::endl;
            std::cout << "error = " << GetLastError() << std::endl;
        }
        else{
            //system("pause");

            if (GetProcessMemoryInfo(h, (PROCESS_MEMORY_COUNTERS *)&pmc,
sizeof(pmc))) {
                ramSaveArr[0] = pmc.WorkingSetSize/1024; //kb
                ramSaveArr[1] = pmc.PrivateUsage/1024;   //kb
            }
            else{
                qDebug() << GetLastError();
            }
            if(monitoringRamType){  //type = working set
                if(ramSaveArr[0] >= userAcceptableRamLoad){
                    infoString.append(ctime(&end_time));
                    infoString.append( "PROCESS: ");
                    QString procNameBuff;
                    infoString.append(procNameBuff.fromWCharArray(exeFile));
                    infoString.append(" CURRENT WORKING SET: ");
```

```
        QString currentLoadString = QString::number(ramSaveArr[0]);
        infoString.append(currentLoadString);

        if(outputTypeIsLogMt == 0){
            emit emitMessage(infoString, false, false);
            return false;
        }
        else{
           qDebug() << "log";
           if (fileRamMt->open(QFile::Append |QFile::Text))
           {
              qDebug() << "i";
              fileRamMt->write(infoString.toUtf8() + "\n");
              fileRamMt->close();
           }
        }
        CloseHandle(h);
    }
}
else{   //type = private set
    if(ramSaveArr[1] >= userAcceptableRamLoad){
        infoString.append(ctime(&end_time));
        infoString.append( "PROCESS: ");
        QString procNameBuff;
        infoString.append(procNameBuff.fromWCharArray(exeFile));
        infoString.append(" CURRENT PRIVATE SET: ");
        QString currentLoadString = QString::number(ramSaveArr[1]);
        infoString.append(currentLoadString);

        if(outputTypeIsLogMt == 0){
           emit emitMessage(infoString, false, false);
```

```
                return false;
            }
            else{
                qDebug() << "log";
                if (fileRamMt->open(QFile::Append |QFile::Text))
                {
                    qDebug() << "i";
                    fileRamMt->write(infoString.toUtf8() + "\n");
                    fileRamMt->close();
                }
            }
            CloseHandle(h);
        }
    }
}
CloseHandle(h);
return true;
}
else{
QString stoppingMsg = "Monitoring was stopped by a user at ";
stoppingMsg.append(ctime(&end_time));

emit emitMessage(stoppingMsg, false, false);
if(outputTypeIsLogMt == 1){
    if (fileRamMt->open(QFile::Append |QFile::Text))
    {
        fileRamMt->write(stoppingMsg.toUtf8() + "\n");
        fileRamMt->close();
    }
}
return false;
```