

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМИ МОНІТОРИНГУ ТА КЕРУВАННЯ
ПРОЦЕСАМИ В ОПЕРАЦІЙНІЙ СИСТЕМІ WINDOWS. DEVELOPMENT OF
A PROCESS MONITORING AND CONTROL PROGRAM IN WINDOWS OS.

Виконав: студент(ка) 4 курсу, групи КНТ-217
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Ануфрієва О.М.

(прізвище та ініціали)

Керівник Сердюк С.М.

(прізвище та ініціали)

Рецензент Казурова А.Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” _____ 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

Ануфрієвої Олени Миколаївни
 (прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка програми моніторингу та керування процесами в операційній системі Windows. Development of a Process Monitoring and Control Program in Windows OS
 керівник проекту (роботи) доц., к.т.н. Сердюк Сергій Микитович,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від “30” березня 2021 року
 № 103
2. Строк подання студентом проекту (роботи) травня 2021 року
3. Вихідні дані до проекту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Характеристика об'єкта проектування. 2. Матеріали та методи. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора. Додатки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Сердюк С.М., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання 28 лютого 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5–6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8–9 тижні	Розділ 5, Додатки
7	Захист роботи.	10 тиждень	

Студент(ка)

(підпис)

Ануфрієва О.М.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Сердюк С.М.

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
90 с., 3 табл., 27 рис., 2 дод., 21 джерело.

ПРОЦЕС, ПОТІК, WINAPI, ПРІОРИТЕТИ, СТВОРЕННЯ ПРОЦЕСУ,
ЗАВЕРШЕННЯ ПРОЦЕСУ, МОНІТОРИНГ, ОПЕРАЦІЙНА СИСТЕМА.

Об'єкт дослідження – методи і засоби моніторингу та управління процесами в операційній системі (ОС) Windows.

Предмет дослідження – методи розробки програмного забезпечення для моніторингу та управління процесами в сімействі ОС Windows.

Мета роботи – розробка програмного продукту, що забезпечує підвищення ефективності управління процесами в ОС Windows.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування; виконавча частина ОС Windows; мова програмування C++ та інструментарій розробки Qt 5.4.2; персональний комп'ютер (ПК) на базі процесора 1 ГГц з архітектурою x86-64 або IA-32 управлінням ОС Windows 7 або вище.

Результати. Розроблено застосунок, який виконує функції моніторингу та керування такими системними об'єктами ОС Windows як процеси, потоки та модулі.

Висновки. Розроблено програмний продукт для використання досвідченими користувачами ПК, який забезпечує ефективне управління процесами в ОС Windows.

Галузь використання – процеси управління в сімействі ОС Windows.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 90 pages, 27 figures, 3 tables, 2 appendixes, 21 sources.

PROCESS, THREAD, WINAPI, PRIORITIES, CREATE PROCESS, TERMINATE PROCESS, MONITORING, OPERATING SYSTEM.

Object of study is a methods and tools for process monitoring and management in the Windows operating system (OS).

Subject of study are methods for software developing for process monitoring and management in the Windows OS family.

The purpose of the work is development of software product that improves the efficiency of process management in Windows OS.

Materials, methods and technical means: object-oriented programming; Windows executive part; C ++ programming language and Qt 5.4.2 development tools; a personal computer (PC) based on a 1 GHz processor with x86-64 or IA-32 architecture running Windows 7 or higher.

Results. An application has been developed that monitors and manages Windows system objects such as processes, threads, and modules.

Conclusions. A software product for use by experienced PC users, which provides effective process management in Windows OS has been developed

The field of use is the management processes in the Windows OS family.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Характеристика об'єкта проєктування	11
1.1 Огляд внутрішнього устрою процесів.....	11
1.2 Огляд внутрішнього устрою потоків.....	Error! Bookmark not defined.
1.3 Аналіз альтернативних програмних продуктів	23
1.4 Розробка технічного завдання.....	30
1.4.1 Підстава для розробки	30
1.4.2 Мета розробки та її призначення.....	30
1.4.3 Основні вимоги до програмного продукту.....	30
1.4.4 Вимоги до технічних засобів, операційної системи і мови програмування	31
1.4.5 Обмеження на установку і виконання програми	32
1.4.6 Вимоги до програмної документації	32
1.4.7 Умови експлуатації	32
2 Матеріали та методи.....	33
2.1 Обґрунтування вибору мови програмування та середовища розробки...	33
2.2 Огляд обраних для розробки застосунку бібліотек	38
2.3 Розробка структури застосунку	40
3 Основні рішення щодо реалізації компонентів системи	42
3.1 Розробка загального алгоритму роботи програми	42
3.2 Розробка програми застосунку.....	43
4 Керівництво програміста	50
4.1 Призначення і умови застосування застосунку	50
4.2 Характеристики програми	50
4.3 Звернення до програми	51
4.4 Вхідні та вихідні дані	51
5 Керівництво оператора.....	52

5.1 Призначення програми 52

5.2 Умови використання програми.....52

5.3 Виконання програми 52

5.4 Повідомлення оператору 56

Висновки.....57

Перелік джерел посилання 58

Додаток А Текст програми 60

Додаток Б Копії слайдів презентації.....82

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CPU	– Central Processing Unit;
DLL	– Dynamic-Link Library;
PCB	– Process Control Block;
ГГц	– гігагерц;
Гб	– гігабайт;
ОС	– операційна система;
ПК	– персональний комп'ютер.

ВСТУП

З точки зору базової архітектури, операційні системи сімейства Windows є 32-х або 64-х розрядними і мультизадачними (багатопотоковими) системами з багатозадачністю, що витісняється [1]. Всі багатозадачні ОС використовують концепцію процесу і повинні мати у своєму розпорядженні відповідні засоби управління процесами, потоками та іншими одиницями, що здатні незалежно виконуватися в асинхронному режимі. Задачі можуть плануватися і витіснятися відповідно до динамічно обумовлених пріоритетів.

Тому, основним завданням будь-якої ОС є управління процесами. ОС повинна розподіляти між ними ресурси, надавати їм можливість спільно використовувати інформацію, захищати ресурси, що використовуються одним процесом від їх використання іншими процесами, а також забезпечувати можливість синхронної роботи процесів.

Процес сам по собі не має ніякої цінності, адже для виконання роботи він повинен мати хоча б один потік (нитку). Потік – це частина вже самого процесу, що виконує певний перелік дій [2]. Реалізація потоків та процесів відрізняються в різних операційних системах, але загалом потік міститься всередині процесу і різні потоки одного процесу спільно розподіляють деякі ресурси, в той час як різні процеси ресурси не розподіляють.

Для отримання користувачем інформації щодо поточних процесів, в ОС Windows існують стандартні програми-менеджери процесів, а також аналогічні програми (типу Process Viewer), що розроблені третіми фірмами.

Утиліта «Менеджер процесів» дозволяє в режимі реального часу відслідковувати застосунки, що виконуються та запущені у поточний час процеси, оцінювати завантаженість системних ресурсів комп'ютера та використання мережі. З його допомогою можна контролювати продуктивність комп'ютера або завершувати роботу застосунків, які не відповідають. Однак більшість з програм такого типу не може надати повну інформацію про

процеси, а також надати можливості користувачам, в достатній мірі, управляти процесами, підвищуючи цим ефективність управління ОС Windows.

Таким чином, розробка утиліти для управління процесами в сімействі ОС Windows є актуальним завданням.

Метою даної дипломної кваліфікаційної роботи бакалавра є детальне вивчення методів управління процесами в сімействі ОС Windows для реалізації недорогого в розробці та впровадженні, але досить ефективного диспетчера процесів, який би за своїми функціональними можливостями перевершував стандартний, що поставляється разом з ОС Windows.

1 ХАРАКТЕРИСТИКА ОБ'ЄКТА ПРОЄКТУВАННЯ

1.1 Огляд внутрішнього устрою процесів

Менеджер процесів – це комп'ютерна програма, що входить до сімейства систем Microsoft Windows і виконує функції диспетчера завдань, монітору системи та менеджера запуску. Він надає інформацію про продуктивність комп'ютера та поточне програмне забезпечення, що було ініціалізоване користувачем, включаючи назву запущених процесів, завантаження процесора, загальний обсяг віртуальної пам'яті всіх процесів, інформацію щодо вводу-виводу та користувачів, що увійшли в систему, а також службах ОС Windows. Менеджер процесів також може використовуватися для спорідненості (закріплення) процесора, встановлення пріоритетів процесів, запуску та зупинки служб або завершення процесів у разі необхідності.

Програма статичний об'єкт, що представляє собою файл або сукупність файлів з кодами і даними. Для того щоб програма могла бути запущена на виконання, операційна система повинна створити оточення або середовище для виконання завдання, яке включає можливість доступу до різних системних ресурсів (пам'яті, пристроїв введення-виведення, файлів і т. д.). Таке оточення отримало назву процесу [4]. Процес являє собою образ програми, що виконується та включає відображення в пам'яті виконуваного файлу, отриманого в результаті компіляції і зв'язування вихідного коду програми, кодів даних і бібліотек, стека, а також ряду структур даних ядра, необхідних для управління процесом. В цілому, процес можна представити як сукупність даних ядра системи, необхідних для опису образу програми в пам'яті і управління її виконанням.

Зазвичай застосунок складається з одного або декількох процесів. У контексті процесу може виконуватися один або кілька потоків. Потік – це основна одиниця, якій операційна система виділяє процесорний час. Потік може виконувати будь-яку частину коду процесу, включаючи частини, що виконуються в даний момент іншим потоком.

Кожен процес надає ресурси, які необхідні для виконання програми. Процес має віртуальний адресний простір, код, що виконується, відкриті дескриптори для системних об'єктів, контекст безпеки, унікальний ідентифікатор процесу, змінні середовища, клас пріоритету, мінімальний і максимальний розміри робочого набору і принаймні один потік виконання. Кожен процес запускається за допомогою одного потоку, який часто називають основним потоком, але може створювати додаткові потоки з будь-якого зі своїх потоків.

У Windows 2000 введений новий об'єкт ядра – завдання (JobObject). Об'єкт завдання дозволяє управляти групами процесів як єдиним цілим. Об'єкти завдань – це іменовані, колективні об'єкти, або об'єкти що захищаються. Вони керують атрибутами пов'язаних з ними процесів. Операції, що виконуються над об'єктом завдання, впливають на всі процеси, пов'язані з об'єктом завдання.

З поняттям «процес» також пов'язане інше поняття – пул потоків. Пул потоків – це набір робочих потоків, які ефективно виконують асинхронні зворотні виклики від імені програми. Пул потоків в основному використовується для зменшення кількості потоків застосунку і забезпечення управління робочими потоками. Застосунки можуть ставити в чергу робочі елементи, пов'язувати роботу з незавершеними дескрипторами, автоматично ставити в чергу на основі таймера і пов'язувати з операціями вводу-виведення.

Способи реалізації процесів у різних ОС можуть відрізнятися один від одного. Ці відмінності полягають в тому, якими структурами даних користуються процеси, як вони іменуються, якими способами захищені один від одного та які між ними існують відносини [3].

У будь-якій системі поняття «процес» включає наступні складові:

- хоча б один потік, що виконується;
- код, що виконується;
- власний адресний простір, який представляє собою сукупність віртуальних адрес, які може використовувати процес;
- ресурси системи, такі як семафори, файли, і т. ін., які призначені

процесу операційною системою.

Адресний простір кожного процесу захищений від втручання в нього будь-яким іншим процесом. Це забезпечується механізмами віртуальної пам'яті. Операційна система, звичайно, теж захищена від прикладних процесів. Щоб виконати будь-яку процедуру ОС або прочитати що-небудь з її області пам'яті, потік має виконуватися в режимі ядра. Призначені для користувача процеси отримують доступ до функцій ядра за допомогою системних викликів. У призначеному для користувача режимі виконуються не тільки прикладні програми, але й захищені підсистеми Windows [3].

У ОС Windows процес – це звичайний об'єкт, який створюється і знищується менеджером об'єктів. Об'єкт-процес, як і інші об'єкти, містить заголовок, який створює і ініціалізує менеджер об'єктів. Менеджер процесів визначає атрибути, що зберігаються в тілі об'єкта-процесу, а також забезпечує системний сервіс, який відновлює і змінює ці атрибути.

У число атрибутів тіла об'єкта-процесу входять наступні:

- токен доступу, тобто об'єкт, що виконується та містить інформацію про безпеку;
- ідентифікатор процесу, тобто унікальне значення, яке ідентифікує процес в межах операційної системи;
- базовий пріоритет є основою для виконавчого пріоритету потоків процесу;
- час виконання, тобто загальна кількість часу, протягом якого виконуються всі потоки процесу;
- процесорна сумісність, тобто набір процесорів, на яких можуть виконуватися потоки процесу;
- граничні значення квот, тобто максимальна кількість сторінкової та несторінкової системної пам'яті, дискового простору, призначеного для вивантаження сторінок, процесорного часу – які можуть бути використані процесами користувача.

Для опису такого складного динамічного об'єкта як процес, ОС Windows

підтримує набір структур. Однією з головних є так званий блок управління процесом (PCB – Process control block). PCB має наступний склад [5]:

- вміст регістрів процесора;
- стан, в якому знаходиться процес;
- програмний лічильник процесу або адреса команди, яка повинна бути виконана для нього наступною;
- облікові дані (ідентифікаційний номер процесу, який користувач ініціював його роботу, загальний час використання процесора даним процесом і т. д.);
- дані, необхідні для планування використання процесора і управління пам'яттю (пріоритет процесу, розмір і розташування адресного простору і т. д.);
- інформацію про пристрої введення-виведення, пов'язаних з процесом (наприклад, які пристрої закріплені за процесом; таблиця відкритих файлів).

Блок управління процесом для ОС є моделлю процесу. Будь-яка операція, яка здійснена операційною системою над процесом, викликає певні зміни в PCB. Квзіпаралельне виконання процесів передбачає періодичне припинення поточного процесу, а потім його подальше відновлення. Для цього потрібно вміти зберігати частину даних з PCB (так званий контекст процесу), а операцію по збереженню даних одного процесу і відновлення даних іншого називають перемиканням контекстів. Перемикання контексту не має відношення до корисної роботи, що виконується процесами, і час, витрачений на нього, скорочує корисний час роботи процесора.

Блок управління процесом реалізований у вигляді набору пов'язаних структур, головна з яких називається блоком процесу EPROCESS або дескриптором процесу. Ця структура створюється виконуючою системою ОС Windows.

Крім множини атрибутів, які стосуються процесу, блок процесу містить ряд інших пов'язаних з ним структур даних і має покажчики на такі структури.

Блок процесу і основна частина пов'язаних з ним структур даних знаходяться в адресному просторі системи. Єдиним винятком є блок змінних оточення процесу, який знаходиться в адресному просторі процесу (в ньому міститься інформація, яка доступна коду, що виконується в режимі користувача). Крім того, деякі з структур даних процесу, що використовуються для управління пам'яттю, наприклад список робочого набору, діють тільки в контексті поточного процесу, оскільки вони зберігаються в системному просторі, визначеному для процесу.

Будь-яка структура EPROCESS інкапсулюється у вигляді об'єкта процесу за допомогою диспетчера об'єктів виконуючої системи. Оскільки процеси не є іменованими об'єктами, їх не буде видно в такому інструментальному засобі, як WinObj. Але в каталозі \ObjectTypes можна побачити тип об'єкта під назвою «Process». Дескриптор процесу за рахунок використання пов'язаних з процесом API-функцій надає доступ до деяких даних в структурі EPROCESS, а також в деяких пов'язаних з нею структурах. Відповідно, кожен потік також представлений набором структур на чолі з блоком потоку ETHREAD. Блок змінних оточення процесу містить додаткові відомості про об'єкт "процес". Спрощена схема структур даних процесу та потоку представлена на рис. 1.1.

Одними з головних функцій менеджера процесів є функції створення нового процесу та завершення процесу.

Зазвичай процес створюється іншим процесом шляхом виклику функції CreateProcess або ж CreateProcessAsUser і CreateProcessWithLogonW. Створення процесу здійснюється в кілька етапів [6]:

- присвоїти новому процесу унікальний ідентифікатор (занести новий запис в таблицю процесів);
- виділити простір для процесу (виділити адресний простір для всіх елементів образу процесу);
- форматувати управляючий блок процесу;
- помістити процес в список готових або готових призупинених

процесів;

- завантажити частину кодів і даних процесу в оперативну пам'ять.

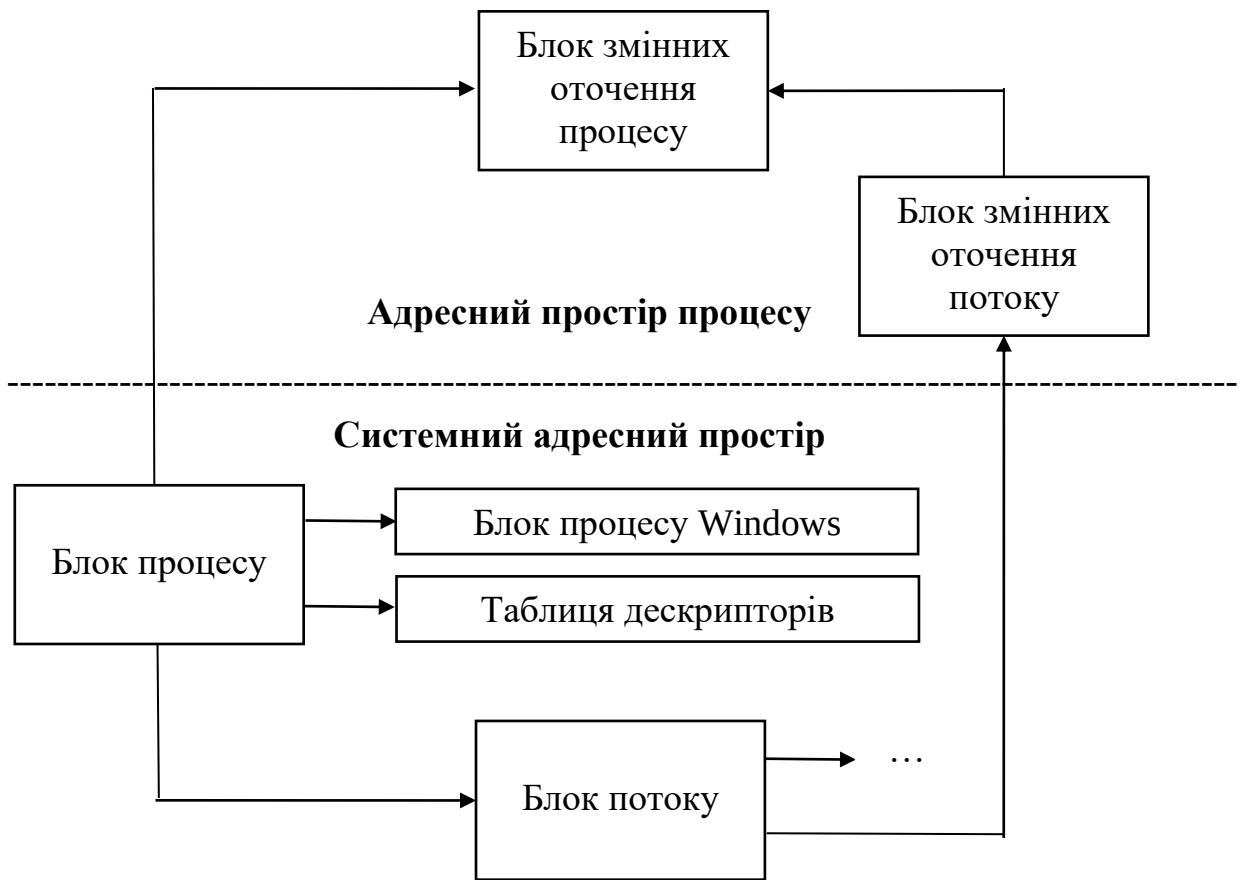


Рисунок 1.1 – Схема структур даних процесу та потоку

Інформація про стан процесора зазвичай ініціалізується нульовими значеннями, за винятком лічильника команд (містить точку входу в програму) і покажчиків системного стека (задають межу стека процесу). Стан процесу зазвичай ініціалізується значенням «готовий» або «готовий і припинений».

Основними причинами створення процесів є:

- запуск задач користувачів і задач в середовищі пакетної обробки;
- надходження запитів від застосунків на виконання деяких функцій;
- породження процесів іншими процесами.

Основними причинами завершення процесів є:

- нормальне завершення;
- перевищення граничного ліміту часу;
- перевищення ліміту відведеної пам'яті;
- помилки при виконанні;
- втручання користувача, адміністратора або ОС;
- завершення батьківського процесу.

Закінчує роботу процесу і всіх його потоків функція `ExitProcess`, в якій в якості параметра вказано код завершення (exit code):

```
VOID ExitProcess(UINT uExitCode);
```

Використання функції `ExitProcess` є найкращим методом завершення процесу. Ця функція забезпечує «чисте» відключення процесу. Таке завершення містить у собі виклик функцій точок входу всіх пов'язаних бібліотек, що динамічно підключаються (DLL) зі значеннями, що вказують, що процес відключається від DLL. Якщо процес закінчується шляхом виклику `TerminateProcess`, бібліотеки DLL, до яких процес підключений, не сповіщаються про завершення процесу. Після того, як всі пов'язані DLL виконали будь-яке значення завершення, ця функція завершує роботу поточного процесу.

Процес можна також завершити за допомогою виклику функції `TerminateProcess`:

```
BOOL TerminateProcess (HANDLE hProcess, UINT fuExitCode);
```

Головна відмінність цієї функції від `ExitProcess` в тому, що її може викликати будь-який потік і завершити будь-який процес. Параметр `hProcess` ідентифікує дескриптор процесу, що завершує роботу. Цей дескриптор повинен мати право доступу `PROCESS_TERMINATE`. А в параметрі `fuExitCode` повертається код завершення процесу. Якщо функція завершується успішно, величина значення, що повертається не дорівнює нулю.

Користуватися функцією слід `TerminateProcess` лише в тому випадку, коли іншим способом завершити процес не вдається. Процес не отримує абсолютно ніяких повідомлень про те, що він завершується, і застосунок не

може ні виконати очистку, ні запобігти своєму несподіваному завершенні. При цьому втрачаються всі дані, які процес не встиг переписати з пам'яті на диск.

Процес дійсно не має ні найменшого шансу самому провести очистку, але операційна система вивільняє всі ресурси, що йому належали: повертає собі виділену йому пам'ять; закриває будь-які відкриті файли; зменшує лічильники відповідних об'єктів ядра і руйнує все його User- та GDI-об'єкти.

1.2 Огляд внутрішнього устрою потоків

ОС Windows, яка є багатозадачною операційною системою, ділить доступний процесорний час між процесами або потоками, яким він необхідний. Windows реалізує так звану витискальну багатозадачність. Це такий вид багатозадачності, при якому ОС виділяє відрізок часу процесора кожному потоку, що виконується. Виконання поточного потоку припиняється після закінчення виділеного йому інтервалу часу (кванта), дозволяючи таким чином запускати інший потік. Коли система перемикається з одного потоку на інший, вона зберігає контекст витісненого потоку і відновлює збережений контекст наступного потоку в черзі.

Довжина інтервалу часу, що виділяється для потоку, залежить від операційної системи і процесора. Оскільки кожен часовий інтервал невеликий (мілісекунди), здається, що одночасно виконується кілька потоків. Фактично цей підхід має місце і в багатопроцесорних системах, де множина потоків, що виконуються, розподіляється між доступними процесорами. Однак, слід зауважити, що використання декількох потоків у застосунку, може привести до того, що продуктивність системи може знизитися.

В ОС сімейства Windows потік, впродовж свого існування, може перебувати в одному з шести станів, зображених на рис. 1.2 [3]. Життєвий цикл потоку починається в той момент, коли програма створює новий потік. Запит передається виконавчій системі ОС, менеджер процесів виділяє пам'ять для об'єкта-процесу і звертається до ядра, щоб ініціалізувати об'єкт-процес ядра.

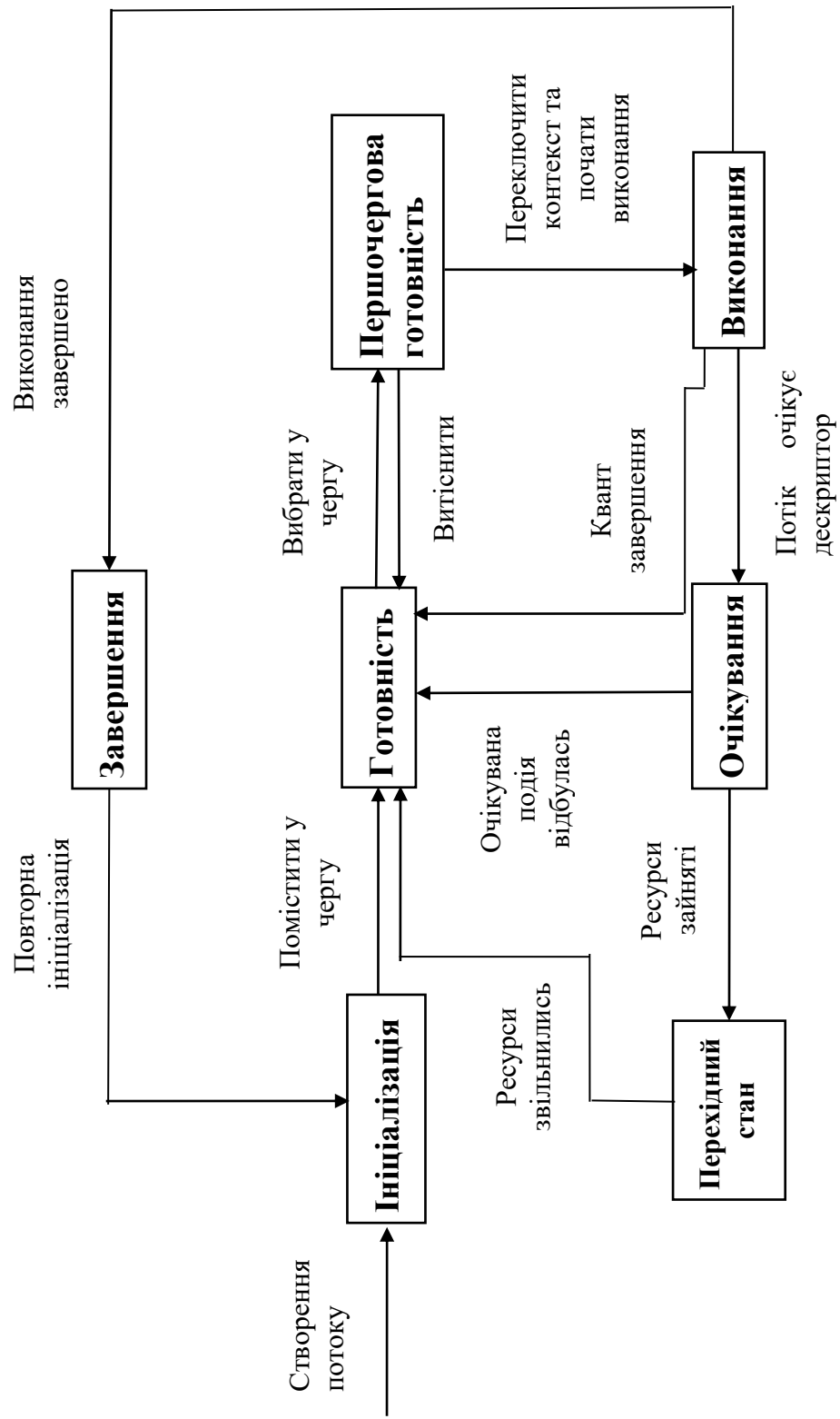


Рисунок 1.2 – Стани потоків

Згідно з рис. 1.2, після ініціалізації процес може знаходитися в одному з наступних станів:

- «Готовність». При пошуку процесу на виконання диспетчер переглядає тільки процеси, що знаходяться в стані готовності, тобто тих, у яких є все необхідне для виконання, крім процесора;

- «Першочергова готовність». Для кожного процесора системи вибирається один процес, який буде виконуватися наступним (це перший процес в черзі). Коли умови дозволяють, відбувається перемикання на контекст цього процесу;

- «Виконання». Після того як відбудеться перемикання контекстів, процес переходить в стан виконання і знаходиться в ньому до тих пір, поки ядро не витіснить його через те, що у системі з'явився процес з більшим пріоритетом, або закінчився квант часу, виділений цьому процесу, або процес завершиться взагалі, або він за власною ініціативою перейде в стан очікування;

- «Очікування». Процес може входити в стан очікування декількома способами: він за своєю ініціативою очікує деякий об'єкт для того, щоб синхронізувати своє виконання; операційна система (наприклад, підсистема вводу-виводу) може очікувати в інтересах процесу; підсистема оточення може безпосередньо змусити процес призупинити себе. Коли очікування процесу дійде до кінця, він повертається в стан готовності;

- «Перехідний стан». Процес входить в перехідний стан, якщо він готовий до виконання, але ресурси, які йому потрібні, зайняті. При звільненні ресурсів процес переходить в стан «Готовність»;

- «Завершення». Коли виконання процесу закінчилося, він входить в стан завершення. Перебуваючи в цьому стані, процес може бути або видалений, або не видалений. Це залежить від алгоритму роботи менеджера об'єктів, відповідно до якого він і вирішує, коли видаляти об'єкт. Якщо виконавча система ОС має покажчик на об'єкт-процес, то він може бути ініціалізований і використаний знову.

З точки зору продуктивності системи, використання потоків має наступні переваги [3]:

- створення нового потоку в уже існуючому процесі займає істотно менше часу, ніж створення нового процесу;
- потік можна завершити швидше, ніж процес;
- перемикання потоків в рамках одного процесу відбувається набагато швидше;
- використання потоків підвищує ефективність обміну інформацією між процесами. У більшості ОС обмін інформацією між процесами відбувається за участю ядра, яке забезпечує механізм здійснення обміну і захист. Обмін інформацією між потоками також може проводитися і без участі ядра.

Первинний потік процесу створюється при виконанні функції `CreateProcess`, проте для створення додаткових потоків необхідно викликати функцію `CreateThread`, яка знаходиться в бібліотеці `Kernel32.dll`:

```
HANDLE CreateThread (
    PSECURITY_ATTRIBUTES lpsa,
    DWORD dwStackSize,
    LPTHREAD_START_ROUTINE lpStartAddr,
    LPVOID lpThreadParm,
    DWORD dwCreationFlags,
    LPDWORD lpThreadId);
```

Розглянемо параметри функції `CreateThread`:

- `lpsa`, покажчик на структуру атрибутів захисту;
- `dwStackSize`, розмір стека нового потоку в байтах. Значенню 0 цього параметра відповідає розмір стека за замовчуванням, рівний розміру стека основного потоку;
- `lpStartAddr`, покажчик на функцію (яка належить контексту процесу), яка повинна виконуватися. Ця функція приймає єдиний аргумент у

вигляді покажчика і повертає 32-бітовий код завершення. Цей аргумент може інтерпретуватися потоком або як змінна типу `DWORD`, або як покажчик;

- `lpThreadParm`, покажчик, який передається потоку в якості аргументу, який зазвичай інтерпретується потоком як покажчик на структуру аргументу;

- `dwCreationFlags`, якщо значення цього параметра встановлено рівним 0, то потік запускається відразу ж після виклику функції `CreateThread`. Установка у цьому параметрі значення `CREATE_SUSPENDED` призведе до запуску потоку в загальмованому стані, з якого потік може бути переведений в стан готовності шляхом виклику функції `ResumeThread`;

- `lpThreadId`, покажчик на змінну типу `DWORD`, яка отримує ідентифікатор нового потоку; в Windows 9x і Windows NT 3.51 значення `NULL` для цього параметра встановлювати не можна.

Будь-який потік процесу може завершити своє виконання самостійно, шляхом виклику функції `ExitThread`:

```
VOID ExitThread (DWORD dwExitCode)
```

Однак більш звичайним способом самостійного завершення потоку є повернення з функції потоку з використанням коду завершення в якості значення, що повертається. По завершенні виконання потоку пам'ять, яку займає її стек, звільняється. У разі якщо потік був створений в бібліотеці `DLL`, буде викликана відповідна точка входу `DllMain` із зазначенням прапора `DLL_THREAD_DETACH` в якості причини цього виклику.

Виконання потоку також може бути завершено іншим потоком за допомогою функції `TerminateThread`. Однак звільнення ресурсів потоку при цьому не відбувається, обробники завершення не виконуються і повідомлення бібліотекам `DLL` не посилаються. Виконання процесу буде завершено після завершення останнього потоку даного процесу.

Для отримання списку процесів, модулів, потоків конкретного процесу необхідно зробити знімок стану системної пам'яті [7]. Списки в системній пам'яті змінюються при запуску і завершенні процесів, створення і руйнуванні

потоків, завантаження і вивантаження модулів, що виконуються, створенні і видаленні куп. Тому використання інформації зі знімка запобігає виникненню можливих неузгодженостей. Інакше будь-які зміни в будь-якому зі списків могли б привести до некоректних результатів або до порушення загального захисту. Наприклад, якщо програма пройде за списком потоків в той момент, коли з'являються нові потоки, а колишні руйнуються, інформація, яка використовується цією програмою, виявиться невірною і призведе до помилки при перегляді списку.

Для вилучення інформації про системну пам'ять ОС Windows надає кілька способів. Найвідомішим є застосування ToolHelp32 API. Центральною функцією в даному API є CreateToolHelp32Snapshot, яка повертає покажчик на об'єкт, який містить знімок системної пам'яті. Після виклику цієї функції слід послідовно застосовувати пари функцій 32First і 32Next з різними префіксами: Process, Thread, Heap і Module.

1.3 Аналіз альтернативних програмних продуктів

В даний час існує ряд утиліт за допомогою яких можна отримати інформацію про процеси і здійснювати управління процесами. Однією з найвідоміших є стандартний диспетчер завдань (Task Manager), що був представлений в його теперішньому вигляді починаючи з Windows NT 4.0.

Його перевагою є тільки те, що він завжди присутній в комплекті ОС Windows, він легко запускається за допомогою комбінації клавіш CTRL-ALT-DEL, що часом є єдиним варіантом, якщо, наприклад, зависла оболонка системи.

Ця корисна утиліта відображає різну системну інформацію, таку як завантаження процесора, розмір файлу підкачки, що використовується, розподіл фізичної пам'яті, обсяг використовуваної пам'яті тощо (див. рис.1.3). Але вона не надає розгорнутої інформації щодо процесів.

Имя	Состояние	29% ЦП	40% Память	0% Диск	0% Сеть	1% GPU	Ядро GPU	Энергопотребл...	Тенденция эн...
Приложения (5)									
Firefox (14)		0,9%	2 829,5 МБ	0,1 МБ/с	0 Мбит/с	0,2%	Графический процессор 0 - 3D	Очень низкое	Низкий
Microsoft Word (3)		24,0%	203,7 МБ	0 МБ/с	0 Мбит/с	0%		Очень высокое	Очень низкое
Диспетчер задач		0,6%	22,9 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Низкий
Ножницы		0,4%	3,5 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	
Проводник		0,6%	38,7 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Очень низкое
Фоновые процессы (56)									
360 Total Security		0%	0,6 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Очень низкое
360 Total Security		0,2%	48,0 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Очень низкое
360 Total Security (32 бита)		0%	17,1 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Очень низкое
Adobe Acrobat Update Service		0%	0,9 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Очень низкое
Adobe CEF Helper		0%	26,6 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Очень низкое
Adobe CEF Helper		0%	6,9 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Очень низкое
Adobe CEF Helper		0%	28,1 МБ	0 МБ/с	0 Мбит/с	0%		Очень низкое	Очень низкое

Рисунок 1.3 – Головне вікно Task Manager

У вікні Applications (Застосунки), що відповідає за призначені для користувача програми, і в Processes (Процеси), що відображає в тому числі і приховані сервіси, по кожному процесу можна дізнатися тільки ім'я файлу, що виконується, в якому контексті він запущений (тобто з якими правами – користувача, системи і так далі), скільки споживає пам'яті і ресурсів процесора.

Незважаючи на те, що у Windows 8 розробники Microsoft значно вдосконалили диспетчер завдань, зробивши його простіше і могутніше функціонально, ця утиліта найбільше підходить тільки для зупинки виконання процесів. За замовчуванням утиліта тепер запускається не в розширеному режимі, а в спрощеному. Якщо програма не відповідає, досить запустити диспетчер завдань, щоб відразу ж побачити завислий процес і завершити його кнопкою «Завершити задачу» (End Task) [8].

До недоліків утиліти Task Manager також слід віднести наступні:

- відсутня можливість подивитися ієрархію запущених процесів та структуру їх зв'язків між собою;
- відсутня розширена інформація про процеси;
- відсутня інформація щодо відкритих дескрипторів і завантажених бібліотеках DLL;

– відсутня можливість завершити деякі завислі процеси, особливо системні.

Саме через це багато розробників почали створювати власні версії цієї утиліти, у яких додається нова функціональність і розширюється стандартна функціональність диспетчера задач, в тому числі можливостей моніторингу та управління процесами.

Розглянемо найбільш поширені, безкоштовні існуючі альтернативні програмні продукти.

Одним з найвідоміших аналогів диспетчера завдань є утиліта Process Explorer (рис.1.4) від компанії Sysinternals. Утиліти цієї компанії Microsoft часто розміщує на своєму офіційному сайті в якості допоміжних для вирішення проблем з Windows. І саме Process Explorer рекомендується в Microsoft як альтернатива диспетчеру задач.

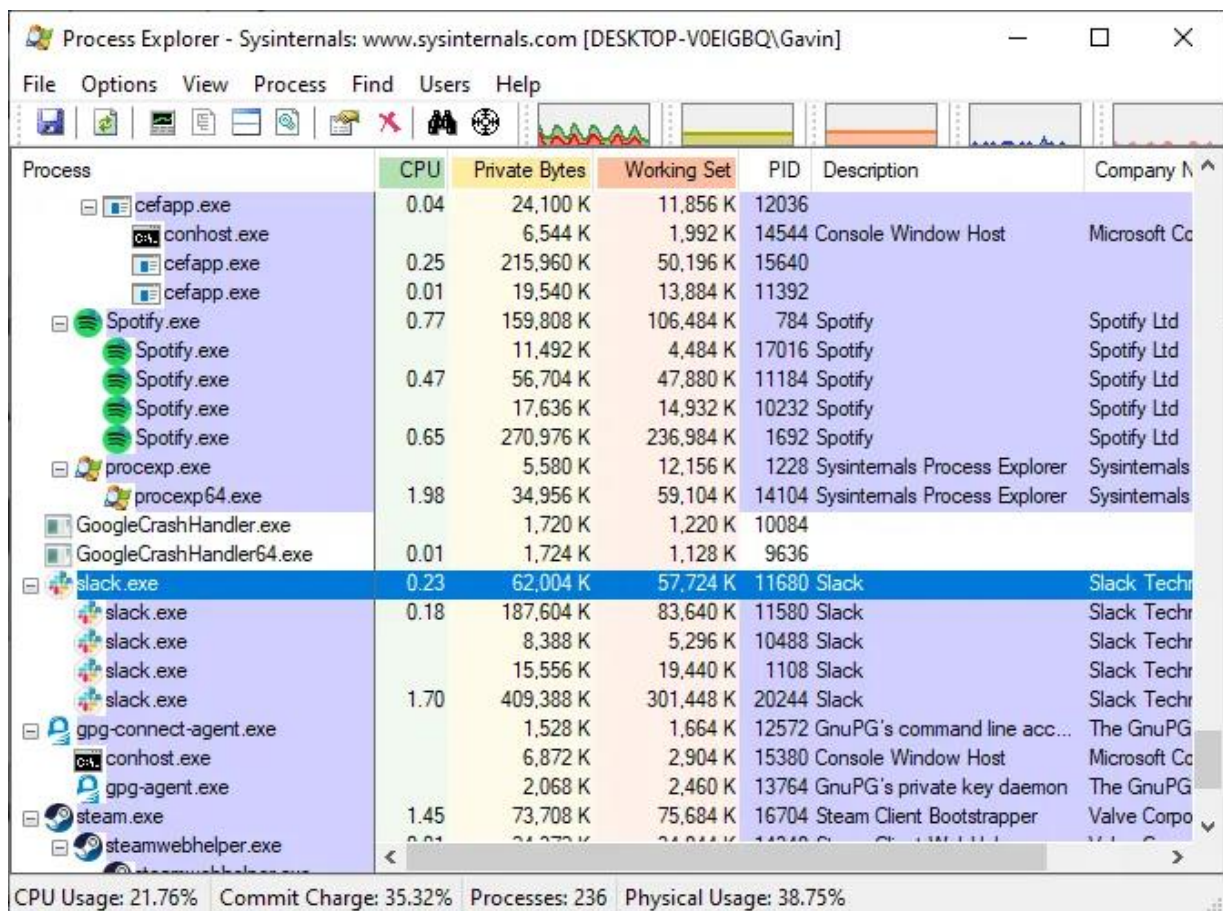


Рисунок 1.4 – Приклад інтерфейсу Process Explorer

Process Explorer в режимі реального часу здійснює постійний моніторинг всіх запущених в системі процесів. Для кожного процесу програма відображає наступну інформацію [9]-[10]:

- використання пам'яті;
- відсоток завантаження процесора;
- власник процесу;
- використовувані протоколи TCP/IP;
- родинні стосунки між процесами у вигляді дерева;
- список бібліотек, що використовуються обраним процесом;
- відкриті файли, що заблоковані конкретними процесами;
- розташування файлу, що виконується.

Крім того, Process Explorer надає кольоровий список процесів для спрощення навігації, дозволяє перезавантажити, призупинити, завершити процеси, встановлювати їх пріоритет, а також шукати DLL, що використовуються процесами.

Process Explorer поширюється у вигляді zip-архіву і не потребує установки.

Ще одним аналогом диспетчера задач є програма Starter (розробник Code Stuff). Дана утиліта в першу чергу розроблена для управління запуском програм при старті ОС Windows. Вона дозволяє редагувати список програм в автозавантаженні, файл win.ini і ключі реєстру. Крім того, програма відображає список запущених процесів і служб. Starter має дуже корисну та зручну функцію перегляду бібліотек, що використовуються процесом (рис. 1.5).

Користувач має змогу зупинити будь-який процес або встановити для нього пріоритет. У списку відображається дескриптор процесу, шлях до виконуваного файлу процесу і обсяг пам'яті, який їм використовується. Передбачена в програмі і можливість збереження таблиці процесів в текстовий або HTML-файл [9].

Недоліком програми є те, що вона більше не підтримується розробниками.

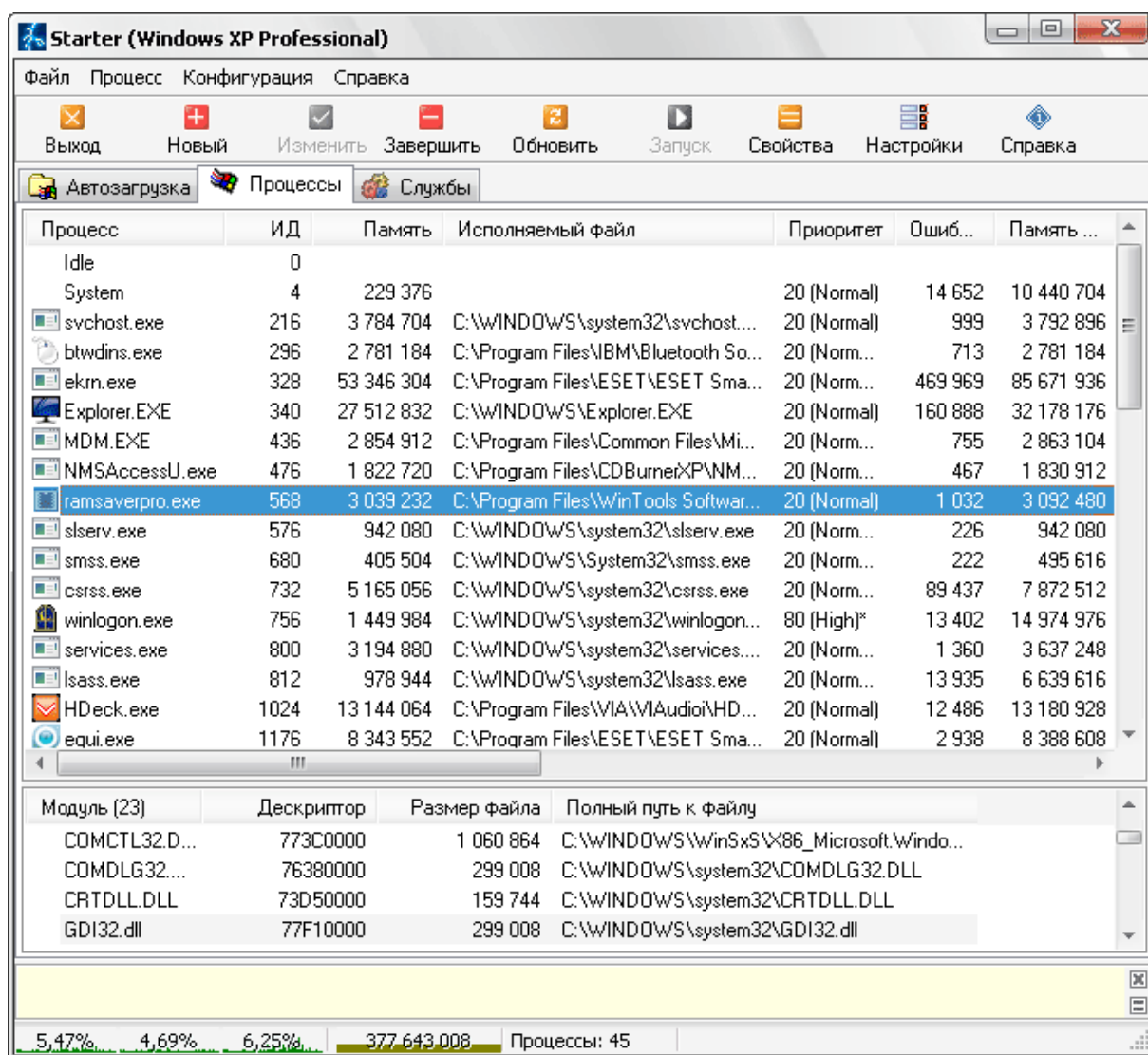


Рисунок 1.5 – Приклад інформації щодо процесів у програмі Starter

Утиліта PROWiSe Manager (розробник Д. Комаров) надає користувачеві детальну інформацію про служби та процеси, які були запущені на комп'ютері. Що стосується можливостей моніторингу та управління процесами, то користувач може отримати інформацію про список модулів, які процес завантажив і дескриптори ресурсів системи, які він відкрив. Також відображається статус запущених програм. Будь-який процес можна завершити або змінити пріоритет для нього. Таким чином функціонал даної утиліти, а також її інтерфейс фактично співпадає з розглянутою вище програмою Starter (рис. 1.5).

Програма PROWiSe Manager надає детальну інформацію про всі служби

і драйвери. За її допомогою можна змінити статус і тип запуску служби або взагалі вимкнути її. Є можливість видалити службу і видалити пов'язаний з нею файл, що виконується [11].

Подібно стандартному диспетчеру завдань (Task Manager), програма відображає загальні показники швидкодії системи, такі як кількість виділеної і доступною віртуальної і фізичної пам'яті, а також використання пам'яті ядра, що вивантажується чи пам'яті, що не вивантажується. Крім того, аналогічно з Task Manager, утиліта PROWiSe Manager надає таку функцію, як список програм в Автозавантаженні. За допомогою такого списку можна легко відключити непотрібні при завантаженні ОС програми.

Утиліта AnVir Task Manager (розробник AnVir Software) надає зручні інструменти для налаштування комп'ютера. Основні функції, які виконує дана утиліта наступні [12]:

- управління автозавантаженням, запущеними процесами, сервісами і драйверами і заміна стандартного диспетчера завдань;
- повна інформація про процеси, включаючи командний рядок, завантажені DLL, зупиняє/повертає процеси і дерево процесів, звільняючи оперативну пам'ять, має журнал запуску і зупинки процесів, вхідний і вихідний трафік, навантаження на диск, іконку в треї, підключення до Інтернету, вікна, потоки, відкриті файли і т.д.;
- виявлення і видалення вірусів і шпигунів (spyware);
- тонка настройка Windows, включаючи установку прихованих налаштувань;
- прискорення завантаження системи і роботи комп'ютера.

До недоліків програми можна віднести те, що інсталятор AnVir Task Manager за замовчуванням встановлює на комп'ютер користувача ряд утиліт непотрібних для функціонування програми, а також змінює пошукову систему браузерів, встановлює непотрібні панелі інструментів і проводить інші небажані дії [13].

Результати аналізу безкоштовних альтернативних програмних продуктів за вісьмома критеріями, необхідними для вирішення задач моніторингу та управління процесами, наведені у табл. 1.1.

Таблиця 1.1 – Результати аналізу альтернативних програмних продуктів

Функції	Task Manager	Process Explorer	Starter	PROWiSe Manager	AnVir Task Manager
Відображення процесів	+	+	+	+	+
Подання у вигляді дерева процесів	—	+	—	—	+
Список використовуємих бібліотек і служб	—	+	+	+	+
Установка пріоритетів	+	+	+	+	+
Завершення/призупинення процесу	+	+	+	+	+
Завершення/призупинення дерева процесів	—	+	—	—	+
Відображення шляху до файлу, що виконується	—	+	+	+	+
Відсоток завантаження процесора	+	+	—	+	+

На основі даних табл. 1.1 можна зробити висновок, що найбільшими функціональними можливостями для реалізації завдань моніторингу та управління процесами мають утиліти Process Explorer та AnVir Task Manager.

Порівняно з AnVir Task Manager утиліта Process Explorer надає більш розширений функціонал щодо діагностичних даних та графіків CPU, GPU, RAM і вводу/виводу в реальному часі.

Результати аналізу альтернативних програмних продуктів послужили основою для розробки технічного завдання програми для моніторингу та керування процесами в операційній системі Windows.

1.4 Розробка технічного завдання

1.4.1 Підстава для розробки

Підставою для розробки служить завдання на дипломну кваліфікаційну роботу бакалавра на тему "Розробка програми моніторингу та керування процесами в операційній системі Windows", затверджене наказом № 103 від 30 березня 2021 р. по НУ «Запорізька політехніка».

1.4.2 Мета розробки та її призначення

Метою розробки є створення програмного продукту, що забезпечує підвищення ефективності управління процесами в ОС Windows та отримання розширеної інформації щодо процесів, потоках і модулях процесу.

1.4.3 Основні вимоги до програмного продукту

На підставі огляду та аналізу проведеного в п.п. 1.1-1.3, можна сформулювати функціональні вимоги до програми, що розроблюється.

Програма повинна забезпечувати:

- зрозумілий та інтуїтивний інтерфейс;
- роботу на ПК під управлінням сімейства ОС Windows;
- отримання списку всіх процесів в системі;
- оновлення списку запущених процесів за запитом користувача;
- показувати розширену інформацію про обраний користувачем процес зі списку;
- можливість взаємодії з процесами та потоками (зміна стану, пріоритету, запуск нових або зупинка вже запущених процесів на вимогу користувача).

Розширена інформація про обраний користувачем процес повинна включати наступні дані:

- для процесу - ім'я, ідентифікатор процесу, ім'я розробника файлу, що ініціював процес, призначення процесу, базовий пріоритет, стан процесу, показники швидкодії, кількість потоків, шлях до файлу процесу, відсоток зайнятості CPU процесом, показники віртуальної пам'яті та системи вводу-виводу;
- для модулів - список модулів, ім'я модулю, ім'я розробника файлу, шлях до файлу модуля (dll);
- для потоків - ідентифікатор потоку, базовий пріоритет, час створення, відсоток зайнятості CPU потоком, стан.

Вхідними даними для програми є керуючі команди користувача.

Вихідні дані для програми: отримана в результаті роботи програми інформація про процеси, потоки та модулі або змінений, в результаті керуючої команди, стан процесу або потоку.

1.4.4 Вимоги до технічних засобів, операційної системи і мови програмування

Для забезпечення роботи застосунку, необхідна наявність операційної системи сімейства Windows 7 або вище на комп'ютерах IBM або сумісних з ними. В якості базової операційної системи слід використовувати Windows 7. Програмне забезпечення повинно бути розроблено з використанням Qt Creator версії 4.5 – крос-платформового інструментарію розробки програмного забезпечення мовою програмування C++.

Це обумовлює наступні системні вимоги:

- 64-розрядний (x64) процесор з тактовою частотою 1 ГГц або вище. Можуть використовуватися процесори сімейств Intel Pentium/Celeron, AMD або інші сумісні процесори;
- не менш 2 Гб оперативної пам'яті;
- не менш 20 Гб (для 64-розрядної системи) вільного місця на жорсткому диску;

- графічний пристрій DirectX 9 з драйвером WDDM 1.0 або більш пізньої версії;
- клавіатура і миша Microsoft Mouse, або сумісний пристрій введення даних.

1.4.5 Обмеження на установку і виконання програми

Програму, що розробляється в даній роботі, можна встановити на будь-якому ПК з ОС сімейства Windows (x64). Основним обмеженням на використання програми є наявність достатньої кількості вільного дискового простору і оперативної пам'яті, які ОС може використовувати в процесі своєї роботи.

1.4.6 Вимоги до програмної документації

Програмна документація повинна включати: керівництво програміста, керівництво оператора та тексти основних програмних модулів.

1.4.7 Умови експлуатації

Програма повинна функціонувати під управлінням ОС сімейства Windows (x64). В якості базової ОС слід використовувати Windows 7.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Обґрунтування вибору мови програмування та середовища розробки

Для створення десктопних Windows-застосунків найбільш часто застосовують такі мови програмування як C, Java, Python, та C++. Згідно даних компанії TIOBE [14], яка спеціалізується на оцінці та відстеженні якості програмного забезпечення, список топ-8 мов програмування у відповідності до індексу TIOBE, схоже не змінився за останні 7 років. Індекс спільноти програмістів TIOBE - показник популярності мов програмування, що оновлюється раз на місяць. Рейтинги базуються на кількості кваліфікованих інженерів в усьому світі, курсів і сторонніх постачальників. Популярні пошукові системи, такі як Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube і Baidu, використовуються для розрахунку рейтингів. Перші позиції в ньому, традиційно займають вищевказані мови програмування (рис. 2.1).

Feb 2021	Feb 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	16.34%	-0.43%
2	1	▼	Java	11.29%	-6.07%
3	3		Python	10.86%	+1.52%
4	4		C++	6.88%	+0.71%
5	5		C#	4.44%	-1.48%
6	6		Visual Basic	4.33%	-1.53%
7	7		JavaScript	2.27%	+0.21%
8	8		PHP	1.75%	-0.27%
9	9		SQL	1.72%	+0.20%
10	12	▲	Assembly language	1.65%	+0.54%

Рисунок 2.1 – Рейтинг мов програмування (у 2021 році)

За винятком мови C, всі мови програмування в списку топ-8 часто випускають нові версії. Наприклад, оновлення мови C# відбувається майже щороку. Оновлення мови JavaScript відбувається ще частіше. Мова C++

змінюється рідше (раз в три роки), але її останній випуск містить введення модулів, що призведе до серйозного зрушення в програмуванні на C++.

Згідно з вимогами технічного завдання до мови програмування (див. п. 1.4.4), для створення застосунку необхідно використовувати C++. Розглянемо можливості C++, яка за даними рис.2.1 займає четверте місце серед мов програмування.

Ця мова була розроблена співробітником Bell Laboratories Бьорном Страуструпом у 1983 році [15]. Мова C++ є розширенням C, тобто забезпечує кардинальну перевагу мови C++, як над своїм попередником мовою C так і над багатьма іншими мовами високого рівня. Мова C++ є однією з досконалих і складних мов програмування на сьогоднішній день.

За допомогою мови C++ можна вирішувати різні завдання, котрі ставляться перед сучасним програмістом: написання системних програм, розробка повноцінних Windows-застосунків, об'єктне моделювання. Завдяки тому, що мова C++ спочатку розроблялася як мова системного програмування, вона надає програмісту широкі можливості для роботи з апаратною частиною. У зв'язку з тим, що мова C++ піддалася повномасштабній обробці, вона забезпечує користувача широким набором засобів для вирішення задач об'єктно-орієнтованого програмування, завдяки чому дозволяє здійснити інтеграцію принципів об'єктно-орієнтованого програмування на апаратний рівень.

C++ повністю підтримує об'єктно-орієнтоване програмування, тобто включає чотири стовпи об'єктно-орієнтованої парадигми: інкапсуляція; успадкування; поліморфізм та абстракція. Щодо стандартних бібліотек C++, то вони складаються з трьох важливих частин:

- бібліотеки основної мови C, що надають всі будівельні блоки, включаючи змінні, типи даних та літерали тощо;
- стандартні бібліотеки C++, що надають багатий набір функцій, які управляють файлами, рядками тощо;

- стандартна бібліотека шаблонів (STL), що надає багатий набір методів маніпулювання структурами даних тощо.

Мова C++ відповідає стандарту ANSI, який підтримують всі основні виробники компіляторів C++. Завдяки цьому код, який програміст написав для компілятора Microsoft, буде компілюватися без помилок, при використанні компіляторів на Mac, UNIX, Windows або Alpha.

Враховуючи вищевикладене можна пояснити, що дає право запропонувати C++ як мову програмування для застосунку, що розробляється [16]:

- C++ є відкритою стандартизованою мовою ISO. Деякий час C++ не мав офіційного стандарту і підтримувався стандартом де-факто, однак з 1998 року C++ стандартизований комітетом ISO;

- C++ є компільованою мовою. C++ компілюється безпосередньо у машинний код, дозволяючи їй бути однією з найшвидших мов у світі, якщо її оптимізувати;

- C++ є статично-типізованою мовою, тобто мовою, де змінна, параметр підпрограми, значення функції що повертається, пов'язується з типом в момент оголошення і тип не може бути змінений пізніше. Останній стандарт C++ підтримує як явне призначення типів, так і вивід типів (можливість компілятора самому здійснити логічний вивід типу значення у виразу). Однак більшість типів перевірки на C++ є статичними;

- C++ пропонує множину варіантів парадигм (забезпечує підтримку процедурних, загальних та об'єктно-орієнтованих парадигм програмування, при цьому можлива також підтримка багатьох інших парадигм);

- C++ є портативною мовою. Як одна з найбільш часто використовуваних мов у світі та як відкрита мова, C++ має широкий спектр компіляторів, які працюють на багатьох різних платформах, які її підтримують. Код, який використовує виключно стандартну бібліотеку C++, буде працювати на багатьох платформах без змін або з незначними змінами;

- C++ є сумісним з мовою C. C++, будучи мовою, яка безпосередньо базується на C, сумісна майже з усіма кодами C. C++ може використовувати бібліотеки C без змін у коді бібліотек;
- C++ має неймовірну підтримку бібліотек. Пошук "library" (бібліотеки) на популярному веб-сайті управління проєктами SourceForge дасть понад 9300 результатів для бібліотек C++ (рис. 2.2).

Search Results

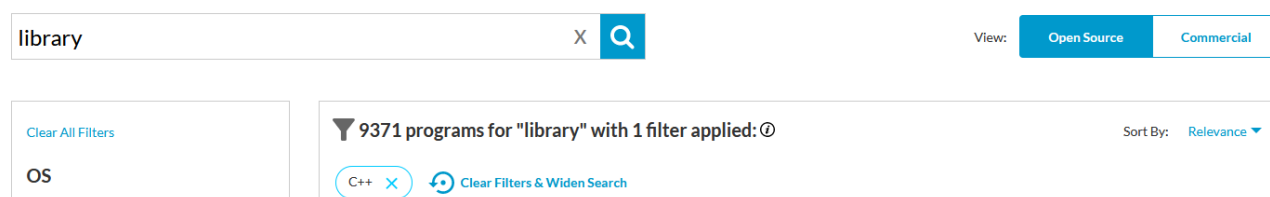


Рисунок 2.2 – Результати пошуку бібліотек для мови C++

Найбільш поширеними середовищами програмування для мови C++ є наступні [17]: Visual Studio; Qt Creator; CLion; Eclipse; Xcode; Code::Blocks та NetBeans. Практично таку ж думку висловили 3240 розробників в результаті опитування, проведеного організацією Standard C++ Foundation (некомерційна організація з Бьорном Страуструпом – розробником мови C++, в якості одного з керівників) (рис. 2.3) [18].

Згідно з даними рис. 2.3, перше місце серед IDE займає Visual Studio (його використовують 55.28% розробників). Друге місце посідає Qt Creator (24.41% розробників). Третє місце у середовища CLion - 14.66%. Крім того, можна відмітити, що багато розробників для роботи використовують текстові редактори з різними розширеннями функцій або без них. У число таких входять: Vim - 37.93% розробників; Visual Studio Code - 25.77% розробників, Emacs (технічно, це текстовий редактор з розширеннями функціоналу до можливостей IDE) - 13.55%, Sublime - 12.78%.

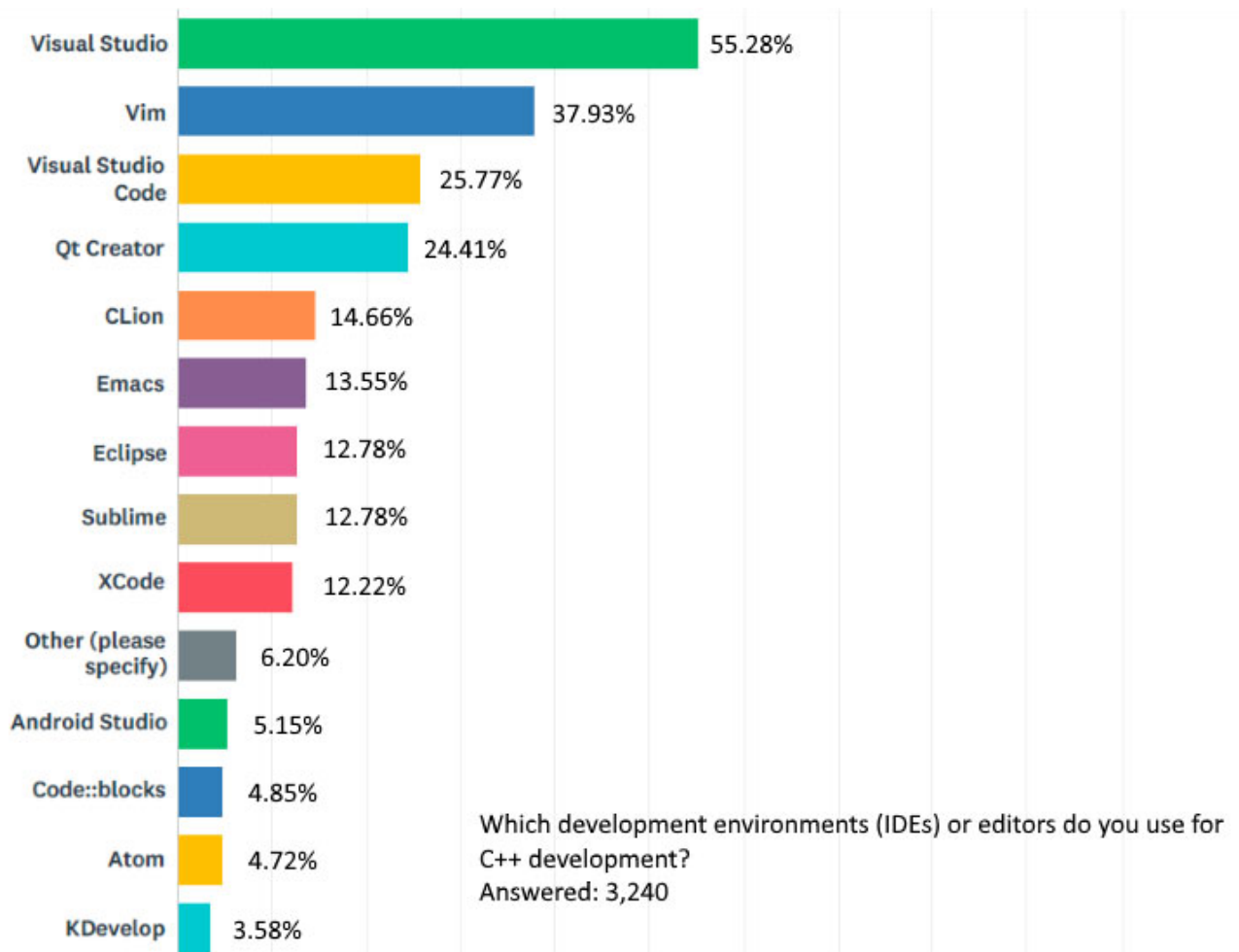


Рисунок 2.3 – Результати опитування, проведеного Standard C++ Foundation

Для розробки у даній роботі програми ProcessViewer, згідно п. п. 1.4.4 технічного завдання потрібно використовувати середовище програмування Qt Creator.

Qt Creator – інтегроване середовище розробки, призначене для створення крос-платформних застосунків з використанням бібліотеки Qt. Підтримується розробка як класичних програм мовою C++, так і використання мови QML, для визначення сценаріїв, в якій використовується JavaScript, а структура та параметри елементів інтерфейсу задаються CSS-подібними блоками [17].

Також Qt Creator дозволяє запускати написане за його допомогою програмне забезпечення в більшості сучасних операційних систем шляхом простої компіляції програми для кожної ОС, без зміни початкового коду.

Середовище включає в себе всі основні класи, які можуть знадобитися при розробці прикладного програмного забезпечення, починаючи від елементів графічного інтерфейсу і закінчуючи класами для роботи з мережею, базами даних і XML. Qt Creator є повністю об'єктно-орієнтованим, легко розширюваним і підтримує техніку компонентного програмування.

Qt дозволяє створювати власні плагіни і розміщувати їх безпосередньо в панелі візуального редактора. Також існує можливість розширення звичної функціональності віджетів, пов'язаної з розміщенням їх на екрані, відображенням, перемальовуванням при зміні розмірів вікна.

Qt завоювала репутацію мультиплатформного набору інструментальних засобів, проте, не дивлячись на це, найчастіше середовище використовується для розробки застосунків на якій-небудь одній платформі. Як приклад програми яка отримала масове поширення і була написана за допомогою Qt, можна навести Adobe Photoshop Album. Крім того, на базі Qt побудовано величезну кількість вузькоспеціалізованого програмного забезпечення. Сюди можна віднести програми, що були розроблені для створення 3D-анімації, цифрової обробки відео-зображень, автоматизації розробки електронних компонентів (мікросхем), для геологічних досліджень, для роботи в галузі медицини та багато інших [18].

За допомогою бібліотек Qt можливо створити простий та гнучкий графічний інтерфейс. Середовище також допоможе розробити програмне забезпечення одразу для багатьох існуючих платформ.

2.2 Огляд обраних для розробки застосунку бібліотек

Базовим класом для всіх об'єктів користувацького інтерфейсу є клас `QWidget`. Віджет – це елементарний об'єкт призначеного для користувача інтерфейсу. Він отримує від віконної системи події миші, клавіатури та інші події та малює своє зображення на екрані. Кожен віджет має прямокутну форму. Крім того всі вони відсортовані в порядку в порядку розміщення

графічних елементів по відношенню до спостерігача. Віджет обмежений своїм батьком і іншими графічними елементами, розташованими перед ним. Віджет, що не вбудований в батьківський віджет, називається вікном. Зазвичай вікно має рамку і смугу із заголовком. Але, якщо використовувати відповідні прапори вікна, можна створити вікно і без такого зовнішнього оформлення. Найбільш поширеними типами вікон в Qt є `QMainWindow` та різноманітні підкласи `QDialog` [19].

Клас `QString` являє собою рядок символів Unicode. Він зберігає рядок 16-бітних значень, що мають тип `QChar`, де кожен `QChar` зберігає символ Unicode 4.0. Unicode – це міжнародний стандарт, що підтримує більшість існуючих на сьогодні письмових систем. Це множина символів ASCII і Latin-1 (стандарт ISO 8859-1), і всі символи ASCII/Latin-1 доступні у відповідних кодових позиціях.

З метою запобігання зайвого виділення пам'яті і непотрібного копіювання інформації клас `QString` неявно застосовує спільне використання даних. Такий підхід допомагає зменшити надлишки пам'яті, які властиві для зберігання 16-бітних символів в порівнянні з 8-бітним представленням символів.

Клас `QString` виконує конвертацію `const char*` в стандарт Unicode використовуючи функцію `fromAscii()`. За замовчуванням, ця функція надає перевагу символам код яких більше 128, тобто таким символам як Latin-1. Але таку поведінку за замовченням можна змінити визвавши функцію `QTextCodec::setCodecForCStrings()`.

Для зберігання сирих байтів і традиційних нультермінальних рядків (такі, що закінчуються на `'\0'`), на додаток до класу `QString` середовище Qt також надає клас `QByteArray`. У більшості випадків `QString` – це необхідний для використання клас. Він використовується у всьому API інтерфейсі Qt, а підтримка стандарту Unicode гарантує, що застосунок можна буде легко перевести на іншу мову, якщо в якийсь момент програміст захоче збільшити їх ринок розповсюдження [20].

Клас QLabel використовується для відображення тексту або рисунка. Ніяких функцій взаємодії з користувачем цей клас не підтримує. Візуальна поведінка мітки може бути задана різними способами. Мітка може бути використана для завдання мнемонічних ключів для отримання фокусу іншими віджетами [21].

Клас QLabel може містити дані наведених в табл. 2.1 типів.

Таблиця 2.1 – Типи даних QLabel

Вміст	Установка
Простий текст	Передати QString в setText()
Форматований текст	Передати QString, що містить форматований текст, в setText()
Анімація	Передати QMovie а setMovie()
Піксельна карта	Передати QPixmap в setPixmap()
Число	Передати int або double в setNum(), яка конвертує число в простий текст

Для налаштування зовнішнього вигляду QLabel існує декілька способів. Наприклад, щоб задати рамку віджету, доступні будь-які налаштування класу QFrame. Крім того, позиціонування вмісту всередині області віджета QLabel може бути здійснене за допомогою функцій setAlignment() і setIndent(). Крім того, перенесення текстового вмісту за словами можна задати за допомогою функції setWordWrap().

2.3 Розробка структури застосунку

При розробці програми будуть використовуватися віджети, які створені на основі концепції проєктування «модель-представлення-контролер» (MVC – Model-View-Controller). Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та

модуль керування таким чином, що модифікація кожного компонента може здійснюватися незалежно. Саме це визначило функціональну направленість модулів програми.

Структура застосунку наведена на рис. 2.4.

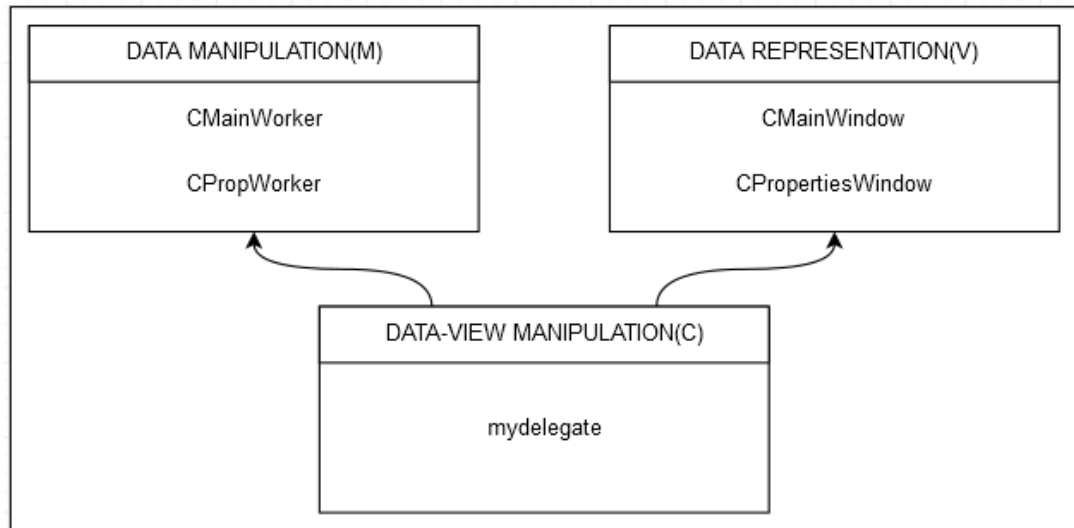


Рисунок 2.4 – Структура застосунку

Отже модулі CMainWorker, CPropWorker відповідають за обробку інформації яка зберігається у моделі даних. Модулі CMainWindow, CPropertiesWindow відповідають за відображення інформації яка зберігається та обробляється попередніми модулями. Модуль mydelegate виступає в ролі контролера, має доступ як до даних моделі, так і до відображення цієї моделі у графічному інтерфейсі користувача, тому здатен впливати як на дані так і на вид.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Розробка загального алгоритму роботи програми

Для забезпечення виконання застосунком функцій, передбачених п. 1.4.3 технічного завдання був розроблений узагальнений алгоритм роботи програми, наведений на рис. 3.1.

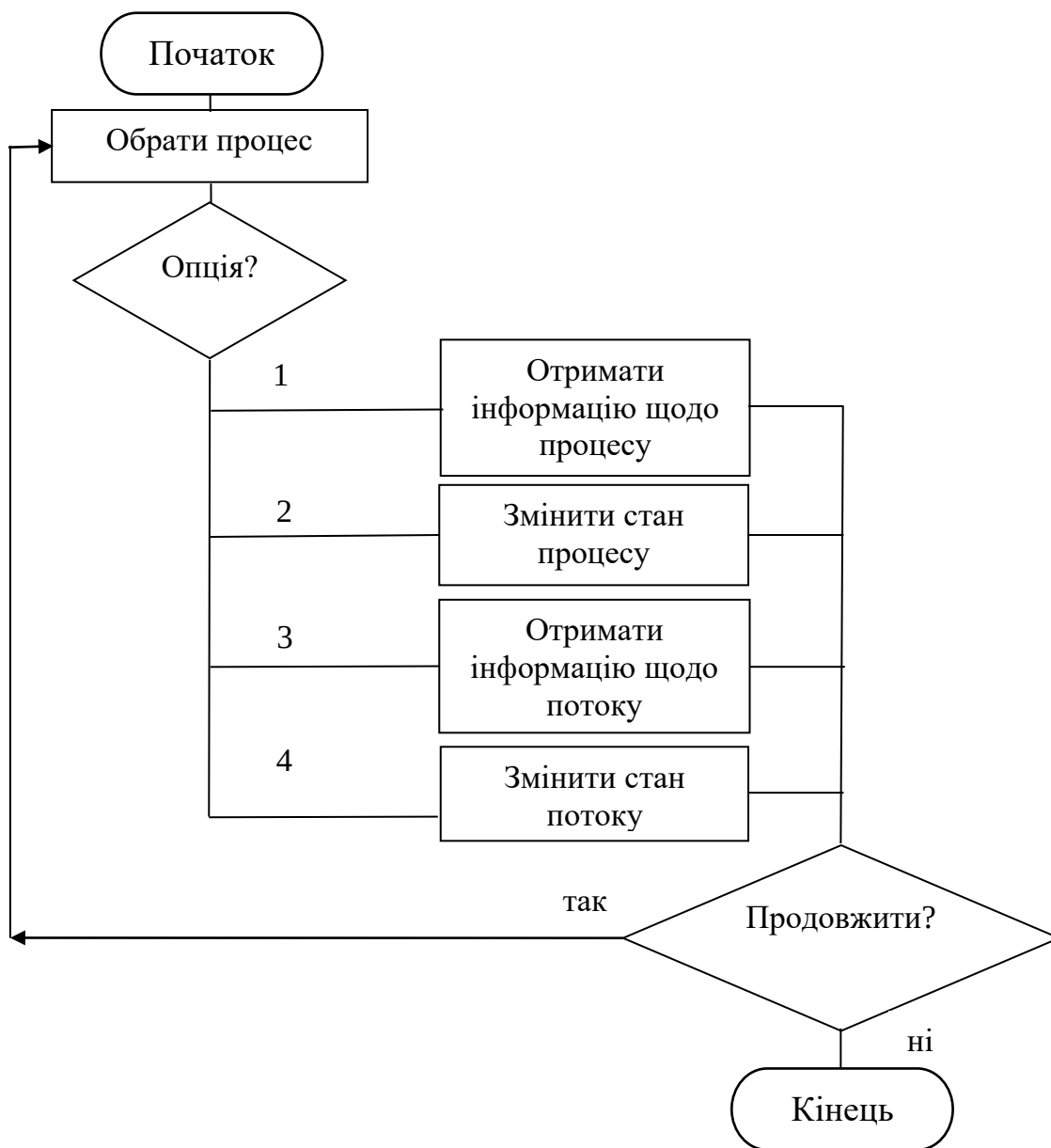


Рисунок 3.1 – Блок-схема алгоритму програми

На початковому етапі користувач отримує список активних процесів у ОС, який вже містить мінімальну інформацію про кожен з процесів. У випадку, коли цієї інформації недостатньо, користувач повинен перейти до властивостей процесу. У властивостях користувач може знайти розширену інформацію як щодо процесу так і таких його складових як потоки та модулів процесу. Якщо користувач досягнув своєї мети, він може або обрати інший процес для дослідження, або вийти з програми.

3.2 Розробка програми застосунку

Розглянемо головні реалізовані класи програми.

Клас CMainWindow (рис. 3.1) є класом головного вікна програми – вікна у якому при старті програми з'являється список процесів у ОС. Цей клас дозволяє користувачу взаємодіяти з процесами – зупиняти, призупиняти (метод `suspendResumeProcess()`), змінювати їх пріоритет (метод `changePriority()`), відкривати вікно властивостей обраного процесу (метод `openProcessProperties(CProcess*)` параметр – показник на обраний процес).

Метод `openContextMenu(QPoint)` відповідає за відкриття контекстного меню у позиції яка передається як параметр.

У конструкторі цього класу, тобто на старті програми відбувається ініціалізація недокументованих функцій `WinApi(NtQuerySystemInformation(), NtQueryInformationProcess(), NtSuspendProcess(), NtResumeProcess())`. Їх опис наданий у розділі «Керівництво програміста» (метод `NtInit()`).

Клас CMainWorker (рис.3.3) є допоміжним класом, класу головного вікна програми. До його функцій входить оновлення інформації на якій базується головне вікно програми (метод `doWork()`). Також у цьому класі відбувається пошук нових процесів та відсіювання вже доданих до моделі (методи `SearchForNewProcesses()` та `ManageCurrentProcesses()`).

```

class CMainWindow : public QWidget //Клас є нащадком класу QWidget
{
    Q_OBJECT//Макрос для використання сигналів та слотів
signals:
    void openProcessProperties(CProcess*);//Перегляд властивостей процесу
    void updateView();//Метод для оновлення списку процесів
public:
    explicit CMainWindow(QWidget *parent = 0);//Конструктор
    ~CMainWindow();//Деструктор
public slots:
    void openContextMenu(QPoint);//Метод для відкриття контекстного меню
    void killProcess();//Метод для закриття процесу
    void suspendResumeProcess();//Метод для зупинки процесу
    void changePriority();//Метод для зміни пріоритету процесу
private:
    void NtInit();//Метод для ініціалізації недокументованих функцій
    Ui::mainwindow *ui;//Показчик на клас форми
    CPropertiesWindow* pw;//Показчик на вікно властивостей
    CMainWorker *m_pMainWorker;//Показчик на клас обробки даних головного вікна
    QThread *m_pThread;//Показчик на потік
    mydelegate delegate;//Показчик на делегат
    QMenu conMenu, *prioritySubMenu;//Показчик на меню
    QAction *suspend_res;//Показчик на дію меню
};

```

Рисунок 3.2 – Код класу CMainWindow

```

class CMainWorker : public QObject//Клас є нащадком класу QObject
{
    Q_OBJECT//Макрос для використання сигналів та слотів
    CProcessDataModel m_pProcessDataModel;//Модель даних для зберігання
    процесів
    bool m_bStop, m_bFirstProcessSelected;
    int selectedIndex;
signals:
    void sendSelectedProcessData(CProcess*);//Метод для передачі об'єкта
    процесу до форми
    void updateProcCount(QString);//Метод для оновлення кількості процесів
    void finished();
    void updateModelView();//Метод для оновлення інтерфейсу
public:
    CMainWorker();//Конструктор
    QString priorityToString(LONG pr);//Метод для перетворення LONG до QString
    void stopThread();//Метод зупинки потоку
    QString GetVerInfo(void* buf, const char *value, LANGANDCODEPAGE*
codePagePtr);//Метод для отримання інформації про версію файла
    int ProcessAlreadyInList(int id);//Перевірка на наявність процесу у моделі
    void SearchForNewProcesses();//Метод для пошуку нових процесів у ОС
    void ManageCurrentProcesses();//Метод для оновлення вже доданих процесів
    void UpdateMainWindowGUI();//Метод для оновлення інтерфейсу
    CProcessDataModel* GetProcessModel();//Метод для отримання моделі даних
public slots:
    void doWork();//Головний метод нового потоку
    void setSelected(const QModelIndex&, const QModelIndex&);
};

```

Рисунок 3.3 – Код класу CMainWorker

Клас CModule (рис. 3.4) зберігає інформацію щодо модулів процесу.

```
class CModule {
public:
    QString name, path, description, companyName; //Ім'я, шлях, опис, назва
    компанії модуля
};
```

Рисунок 3.4 – Код класу CModule

Клас CModuleDataModel (рис. 3.5) реалізує модель даних здатну маніпулювати об'єктами класу CModule та взаємодіяти з віджетами відображення елементів моделі даних.

Клас має можливість додати та видалити модуль з моделі даних (методи AddModule(CModule* module) та RemoveModule(int idx)). Модулі зберігаються у полі m_ModulesList, яке є списком показчиків на CModule.

```
class CModuleDataModel : public QabstractTableModel//Клас є нащадком класу
QAbstractDataModel
{
public:
    CModuleDataModel();//Конструктор
    void AddModule(CModule* module);//Метод для додавання модуля до моделі
    void RemoveModule(int idx);//Метод для видалення модуля з моделі
    int rowCount(const QModelIndex &parent = QModelIndex()) const;//Метод
отримання кількості строк
    int columnCount(const QModelIndex &parent = QModelIndex()) const;//Метод
отримання кількості стовпців
    QVariant data(const QModelIndex &index, int role) const;//Метод отримання
даних чарунки
    QVariant headerData(int section, Qt::Orientation orientation, int role =
Qt::DisplayRole) const;//Метод для вказання назв стовпців
    QList<CModule*> m_ModulesList;//Список модулів
};
```

Рисунок 3.5 – Код класу CModuleDataModel

Клас CProcess (рис. 3.6) зберігає інформацію щодо процесу.

Клас CProcessDataModel (рис. 3.7) реалізує модель даних здатну маніпулювати об'єктами класу CProcess та взаємодіяти з віджетами відображення елементів моделі даних. Цей клас має можливість додати та видалити процес з моделі даних (методи AddProcess(CProcess* process) та

RemoveProcess(int idx)). Процеси зберігаються у полі m_ProcessList, яке є списком показчиків на CProcess.

```
class CProcess
{
public:
    CProcess() : suspended(false) {} // Конструктор
    HANDLE handle;//Хендл процесу
    int id, usage, threads, ram;// Id, ім'я, використання процесору та пам'яті
    QString createTime, imageCreateTime, fileSize, cpu_percents, name,
    fullpath, companyName, fileDescription, productName, productVersion;//Дата
    створення, розмір файлу, шлях до файлу
    QIcon icon;//Рисунок процесу
    LONG priority;//Пріоритет LONG
    QString spriority;//Пріоритет QString
    bool suspended;
    QString privBytes, peakPrivBytes, virtualSize, peakVirtualSize, pageFaults,
    workingSet, peakWorkingSet;//Показчики пам'яті
    QString reads, readBytes, writes, writeBytes, other, otherBytes;//Показчики
    операцій з читання та записом даних
};
```

Рисунок 3.6 – Код класу CProcess

```
class CProcessDataModel : public QabstractTableModel//Клас є нащадком класу
QAbstractDataModel
{
public slots:
public:
    CProcessDataModel();//Конструктор
    void AddProcess(CProcess* process);//Метод для додання процесу до моделі
    void RemoveProcess(int idx);//Видалити процес з моделі
    void UpdateView();//Оновлення списку процесів
    int rowCount(const QModelIndex &parent = QModelIndex()) const;//Кількість
    рядків
    int columnCount(const QModelIndex &parent = QModelIndex())
    const;//Кількість стовпців
    QVariant data(const QModelIndex &index, int role) const;//Дані чарунки
    QVariant headerData(int section, Qt::Orientation orientation, int role =
    Qt::DisplayRole) const;//Задати назви стовпців
    QList<CProcess*> m_ProcessList;//Список процесів
};
```

Рисунок 3.7 – Код класу CProcessDataModel

Клас CPropertiesWindow (рис. 3.8) є класом вікна властивостей обраного користувачем процесу. У ньому з'являються вкладки з загальною інформацією, потоками та модулями процесу. Цей клас дозволяє користувачу взаємодіяти з

потоками – зупиняти, призупиняти (метод `suspendResumeThread()`), змінювати їх пріоритет (метод `changePriority()`).

```
class CPropertiesWindow : public QWidget//Клас є нащадком класу QWidget
{
    Q_OBJECT//Макрос для використання сигналів та слотів
public:
    explicit CPropertiesWindow(QWidget *parent = 0, CProcess* process =
0);//Конструктор
    ~CPropertiesWindow();//Деструктор
public slots:
    void openContextMenu(QPoint);//Відкрити контекстне меню
    void killThread();//Зупинити потік
    void suspendResumeThread();//Призупинити потік
    void changePriority();//Змінити пріоритет потоку
    void setupImageTab(CProcess* p);//Метод для передачі даних до форми
    void updatePerformanceInfo(CProcess* p);//Оновлення показчиків процесу
    void showInExplorer();//Відкрити вікно провідника Windows
    void openExplorerProperties();//Відкрити вікно властивостей файлу
signals:
    void pageChanged(int num);//Сторінка змінилася
    void updateView();//Оновлення інтерфейсу
private:
    CProcess* m_pProc;//Показчик на процес для якого відкрите вікно
    властивостей
    Ui::cpropertieswindow *ui;//Показчик на форму
    CThreadDataModel *m_pThreadDataModel;//Модель даних потоків
    CPropWorker *m_pPropWorker;//Показчик класу для оновлення інформації форми
    QThread *m_pThread;//Показчик на об'єкт нового потоку
    mydelegate delegate;//Делегат
    QMenu conMenu, *prioritySubMenu;//Меню
    QAction *susp_res;//Дія для меню };
```

Рисунок 3.8 – Код класу CPropertiesWindow

Клас CPropWorker (рис. 3.9) є допоміжним класом класу вікна властивостей програми. До його функцій входить оновлення інформації на якій базується вікно властивостей програми (метод `doWork()`). Також у цьому класі відбувається пошук нових потоків та модулів та відсіювання вже доданих до моделі (методи `SearchForNewThreads()`, `GetModulesList()` та `ManageCurrentThreads()`).

Клас CThread (рис. 3.10) зберігає інформацію щодо потоку.

```

class CPropWorker : public QObject//Клас є нащадком класу QObject
{
    Q_OBJECT//Макрос для використання сигналів та слотів
    CThreadDataModel m_pThreadsDataModel;//Модель потоків
    CModuleDataModel m_pModulesDataModel; //Модель модулів
    bool m_bStop, m_bModulesFound, m_bImageInfoSet;
    int m_pageNum;
    ULARGE_INTEGER lastCPU, lastSysCPU, lastUserCPU;
    CProcess* m_pProcess;//Показчик на процес
signals:
    void finished();//Сигнал про зупинку потоку
    void updateThreadsCount(QString);//Оновити кількість процесів
    void sendImageTabData(CProcess*);//Оновити дані про процес
    void sendPerfomanceData(CProcess*);//Відправити формі дані
    void updateModelView();//Оновити інтерфейс
public:
    CPropWorker(CProcess* p);//Конструктор
    QString priorityToString(DWORD priorityClass);//Пріоритет LONG у QString
    int ThreadAlreadyInList(int id);//Потік вже у моделі
    void SearchForNewThreads();//Пошук нових потоків
    void ManageCurrentThreads();//Оновити список потоків
    void GetModulesList();//Взяти список модулів
    void UpdateProcessInfo();//Оновити інформацію щодо процесу
    void UpdatePropWindowGUI();//Оновити інтерфейс
    CThreadDataModel* GetThreadModel();//Показчик на модель потоків
    CModuleDataModel* GetModuleModel();//Показчик на модель модулів
    void stopThread();
    bool getThreadCPU(CThread*);//Вирахувати процент використання процесора
    потоком
public slots:
    void setActivePage(int num);//Встановити активною вкладку вікна
    властивостей
    void doWork();//Головний метод потоку
};

```

Рисунок 3.9– Код класу CPropWorker

```

class CThread
{
public:
    CThread() : suspended(false) {}
    int id;//ID потоку
    HANDLE threadHandle;//Хендл потоку
    LONG priority;//Пріоритет LONG
    QString spriority, cpu_percents, startTime;//Пріоритет QString, процент
    використання процесору, час старту
    bool suspended;
    ULARGE_INTEGER lastCPU, lastSysCPU, lastUserCPU;
};

```

Рисунок 3.10– Код класу CThread

Клас `mydelegate` (рис. 3.11) реалізує контролер з моделі MVC. Його метод `paint(QPainter *painter, const QStyleOptionViewItem &option, const QModelIndex &index) const` має доступ як до даних моделі так і до відображення.

```
class mydelegate : public QStyledItemDelegate//Клас є нащадком класу
QStyledItemDelegate
{
public:
    mydelegate();//Конструктор
    void paint(QPainter *painter, const QStyleOptionViewItem &option, const
QModelIndex &index) const;//Метод малювання чарунки віджету
};
```

Рисунок 3.11– Код класу `mydelegate`

Клас `CThreadDataModel` (рис. 3.12) реалізує модель даних здатну маніпулювати об'єктами класу `CThread` та взаємодіяти з віджетами відображення елементів моделі даних.

Клас має можливість додати та видалити потік з моделі даних (методи `AddThread(CThread* thread)` та `RemoveThread(int idx)`). Потіки зберігаються у полі `m_ThreadsList`, яке є списком показчиків на `CThread`.

```
class CThreadDataModel : public QAbstractTableModel//Клас є нащадком класу
QAbstractTableModel
{public:
    CThreadDataModel();//Конструктор
    void UpdateView();//Оновити інтерфейс
    void AddThread(CThread* thread);//Додати потік до моделі
    void RemoveThread(int idx);//Видалити потік з моделі
    int rowCount(const QModelIndex &parent = QModelIndex()) const;//Кількість
рядків
    int columnCount(const QModelIndex &parent = QModelIndex())
const;//Кількість стовпців
    QVariant data(const QModelIndex &index, int role) const;//Дані чарунки
    QVariant headerData(int section, Qt::Orientation orientation, int role =
Qt::DisplayRole) const;//Дані заголовку
    QList<CThread*> m_ThreadsList;//Список потоків процесу
};
```

Рисунок 3.12– Код класу `CThreadDataModel`

Код програми наведено у Додатку А.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення і умови застосування застосунку

Головним призначенням програми «Process Viewer» є надання користувачу зручного графічного інтерфейсу для моніторингу та управління такими об'єктами ОС як процеси, потоки та модулі.

Застосунок розроблено на мові C++ у інтегрованому середовищі розробки Qt Creator версії 4.5.

Для забезпечення роботи програми під управлінням ОС сімейства Windows (7 або вище) необхідно забезпечити виконання наступних системних вимог:

- 64-розрядний (x64) процесор з тактовою частотою 1 ГГц або вище. Можуть використовуватися процесори сімейств Intel Pentium/Celeron, AMD або інші сумісні процесори;
- не менш 2 Гб оперативної пам'яті;
- не менш 20 Гб (для 64-розрядної системи) вільного місця на жорсткому диску;
- графічний пристрій DirectX 9 з драйвером WDDM 1.0 або більш пізньої версії;
- клавіатура і миша Microsoft Mouse, або сумісний пристрій введення даних.

Основні функціональні можливості розробленого програмного забезпечення наведені у п. 1.4.3.

4.2 Характеристики програми

Програма застосунку складається з класів та методів наведених у п. 3.2.

Застосунок використовує недокументовані функції WinApi, наведені у табл. 4.1.

Таблиця 4.1 – Основні функції WinApi, які використовуються у програмі

Функція	Аргументи	Опис
NtQuerySystemInfo	1. ULONG SystemInformationClass 2. PVOID SystemInformation 3. ULONG SystemInformationLength 4. PULONG ReturnLength	Отримати інформацію про систему в цілому, включаючи список процесів, та їх властивості
NtQueryInformationProcess	1. HANDLE ProcessHandle 2. ULONG ProcessInformationClass 3. PVOID ProcessInformation 4. ULONG ProcessInformationLength 5. PULONG ReturnLength	Отримати інформацію для окремого процесу в системі
NtSuspendProcess	HANDLE ProcHandle	Зупинити процес
NtResumeProcess	HANDLE ProcHandle	Продовжити процес

4.3 Звернення до програми

Завантаження й запуск застосунку відбувається шляхом ініціалізації користувачем відповідного .exe-файлу. Програма завантажується ОС до оперативної пам'яті, та починає своє виконання з точки входу (OEP).

4.4 Вхідні та вихідні дані

Вхідними даними програми застосунку є керуючі команди користувача. Вихідними даними для програми застосунку є: інформація про процеси, потоки та модулі, що отримана в результаті роботи програми або змінений, в результаті керуючої команди, стан процесу або потоку.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Програма «ProcessViewer» призначена для надання користувачу інформації про процеси у ОС, їх потоки та модулі, яка оновлюється у реальному часі. Вона реалізує зручний інтерфейс для відображення інформації і взаємодії користувача з об'єктами ОС.

Даний застосунок надає можливість користувачу виконувати функції, що наведені у п. 1.4.3.

5.2 Умови використання програми

Умови використання застосунку зазначені у технічному завданні (див п.п. 1.4.4 та 1.4 5).

5.3 Виконання програми

Для запуску програми потрібно відкрити файл «ProcessViewer» з розширенням .exe. Результатом запуску буде вікно, що містить список активних процесів у системі (рис.5.1). Цей список автоматично оновлюється показуючи тільки активні процеси ОС.

Щоб виконати над процесом управляючі дії (призупинити, зупинити, змінити пріоритет процесу) треба натиснути на потрібному процесі правою кнопкою миші та обрати відповідний варіант команди (рис. 5.2).

Для отримання інформації щодо властивостей обраного процесу, треба відкрити вікно властивостей процесу та обрати відповідний пункт у контекстному меню або відкрити це вікно подвійним натисканням лівою кнопкою миші на потрібному процесі.

ProcessViewer

Count : 33

ID	Name	State	Priority	Company name	Description
3848	sihost.exe	R	Normal	Microsoft Corporation	Shell Infrastructure Host
2824	explorer.exe	R	Normal	Microsoft Corporation	Windows Explorer
4136	RuntimeBroker.exe	R	Normal	Microsoft Corporation	Runtime Broker
4656	ShellExperienceHost.exe	R	Normal	Microsoft Corporation	Windows Shell Experience H...
4764	SearchUI.exe	R	Normal	Microsoft Corporation	Search and Cortana applicat...
4184	dllhost.exe	R	High	Microsoft Corporation	COM Surrogate
5268	ProfilerU.exe	R	Normal	Saitek	Saitek SST Profile Launcher
5296	SaiMfd.exe	R	Low	Saitek	Saitek MFD File System Driver
5324	egui.exe	R	Below normal	ESET	ESET Main GUI
5356	schedhlp.exe	R	Normal	Acronis	Acronis Scheduler Helper
5364	LCore.exe	R	Normal	Logitech Inc.	Logitech Gaming Framework
5552	TibMounterMonitor.exe	R	Normal	Acronis International GmbH	Acronis TIB Mounter Monitor
2200	taskhostw.exe	R	Normal	Microsoft Corporation	Host Process for Windows T...
1396	MSIAfterburner.exe	R	Below normal	-	MSIAfterburner

Рисунок 5.1 – Список активних процесів

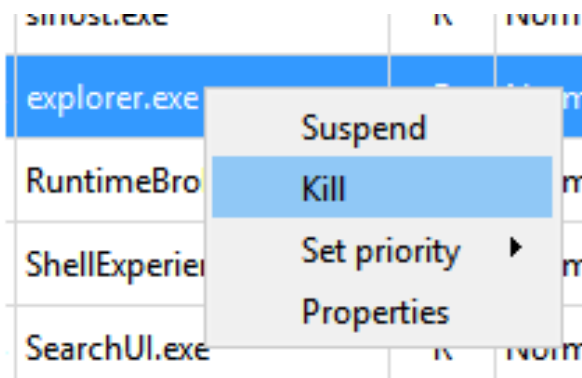


Рисунок 5.2 – Контекстне меню управління процесом

Для безпосереднього перегляду інформації про виконуваний файл процесу, а також про відсоток використання процесору та показники віртуальної пам'яті, треба перейти до вкладки “Information” (рис.5.3).

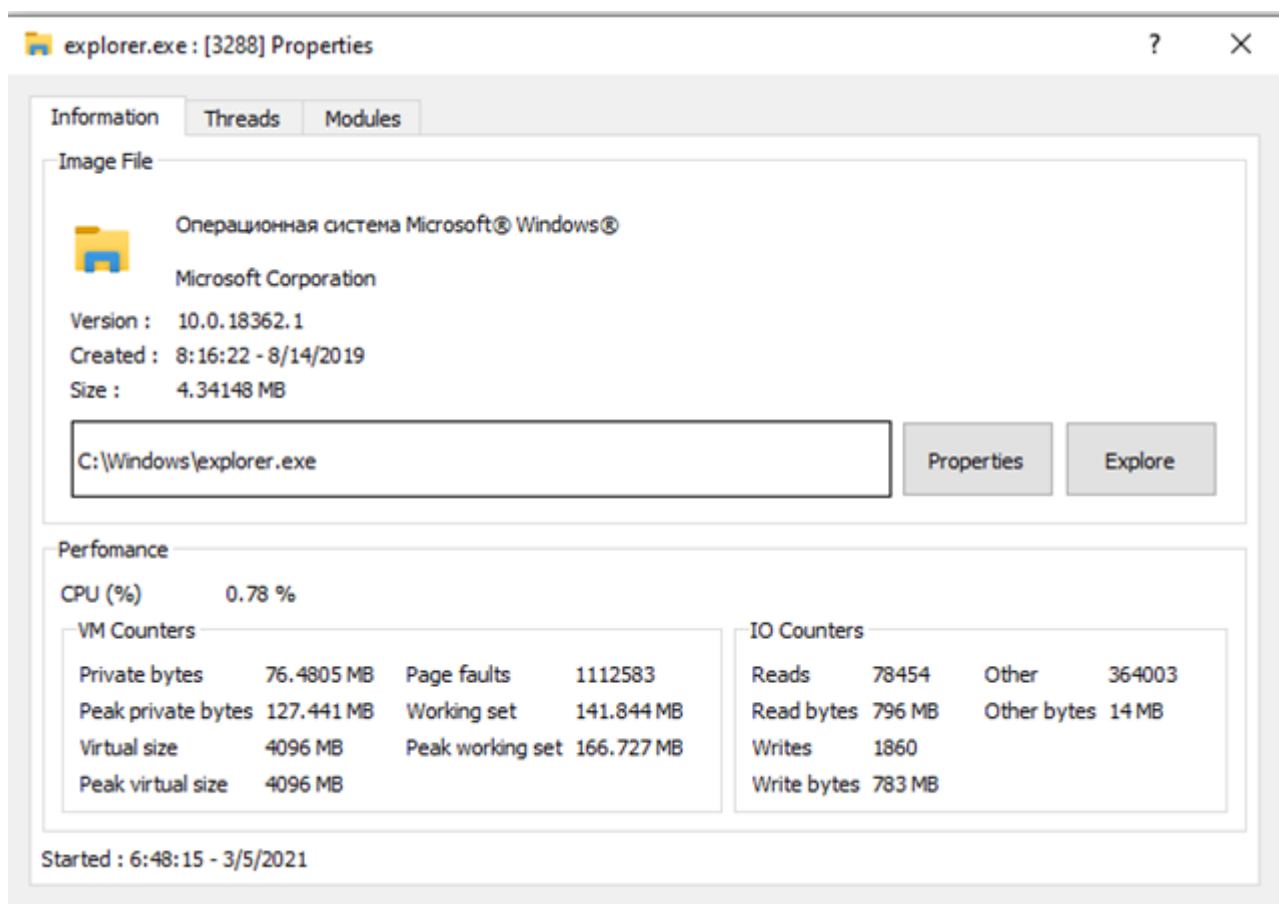


Рисунок 5.3 – Вкладка “Information” вікна властивостей процесу

Для перегляду списку потоків у процесі та інформації щодо потоків, треба перейти до вкладки “Threads” (рис. 5.4).

Щоб призупинити, зупинити, змінити пріоритет потоку треба натиснути на потрібному потоку правою кнопкою миші та у контекстному меню керування потоком (яке аналогічне контекстному меню керування процесом, наведеному на рис. 5.2) обрати відповідний варіант.

Для перегляду списку модулів у процесі та інформації щодо модулів, треба перейти до вкладки “Modules” (рис. 5.5).

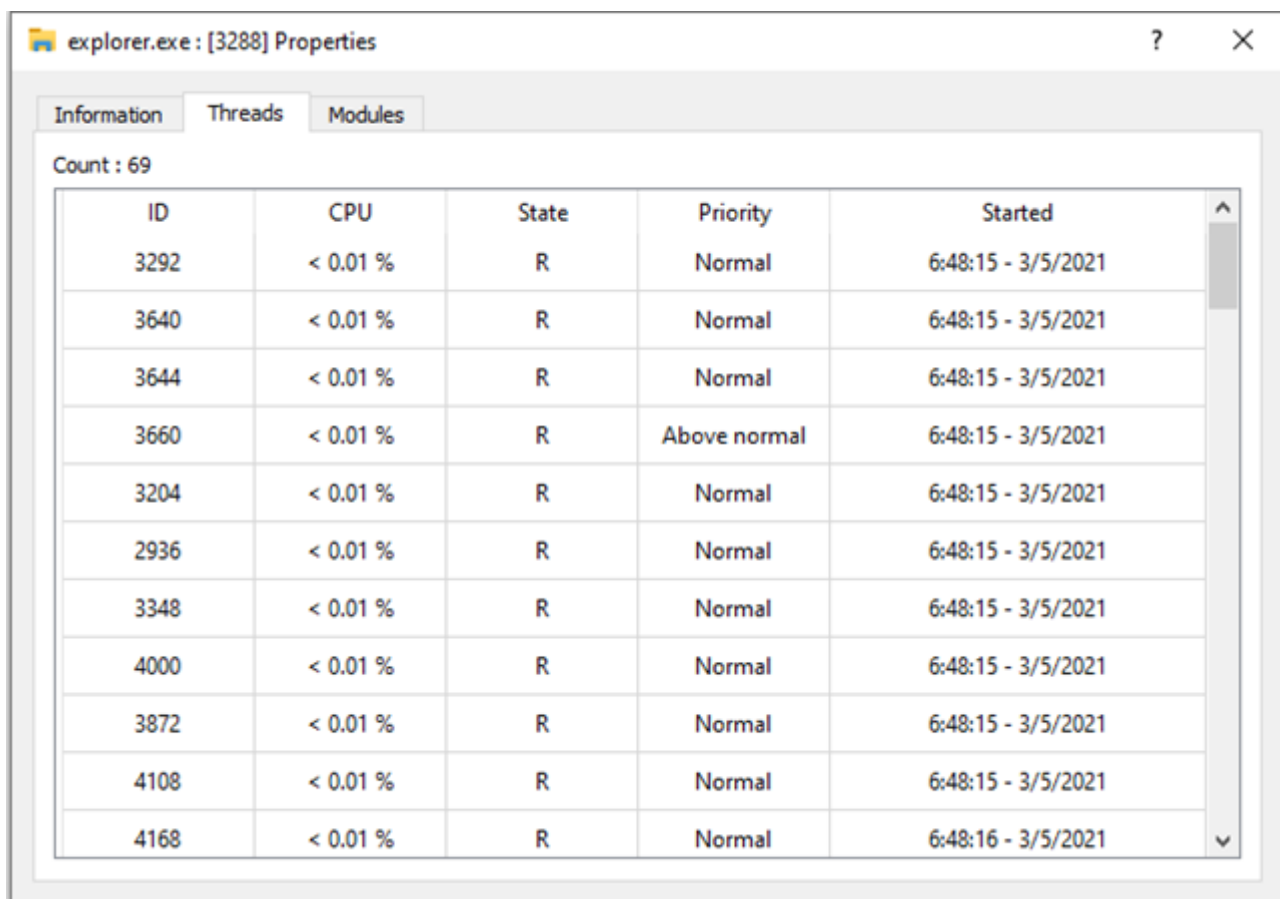


Рисунок 5.4— Вкладка “Threads” вікна властивостей процесу

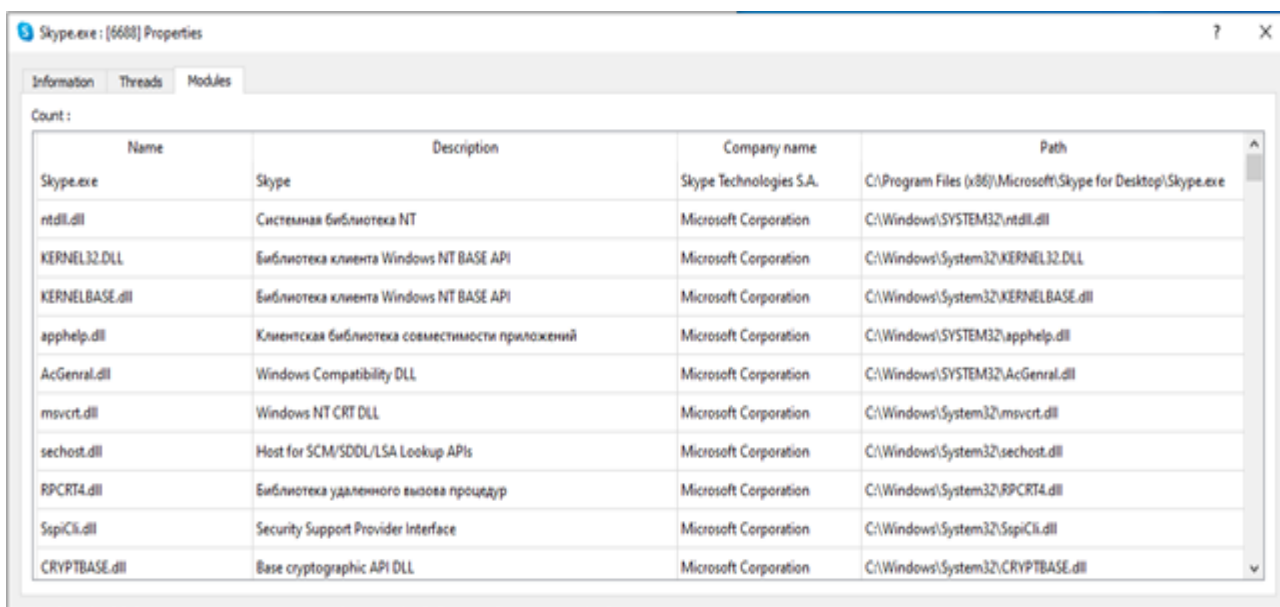


Рисунок 5.5 – Вкладка “Modules” вікна властивостей процесу

5.4 Повідомлення оператору

Якщо натиснути на кнопку “?” у вікні властивостей процесу, користувач отримає інформаційне повідомлення щодо розробленої програми (рис.5.6).

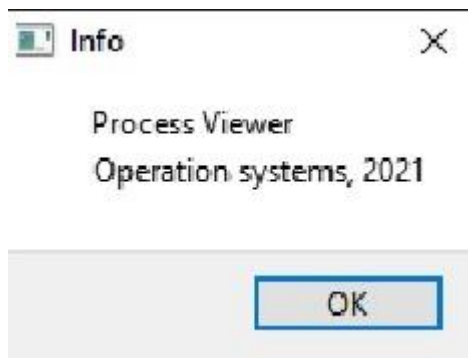


Рисунок 5.6 – Інформація щодо програми

ВИСНОВКИ

Під час розробки даної дипломної кваліфікаційної роботи бакалавра було проведено огляд внутрішнього устрою як процесів ОС Windows, так і потоків процесів.

Базуючись на результатах проведених оглядів, а також на результатах аналізу альтернативних програмних продуктів, було розроблено технічне завдання на розробку програми моніторингу та керування процесами в операційній системі Windows. Мовою програмування застосунку було обрано мову C++, адже саме вона входить до найбільш часто застосовуваних мов для створення десктопних Windows-застосунків.

У якості інтегрованого середовища розробки було обрано Qt Creator – одне з найбільш поширених середовищ програмування для мови C++.

В результаті виконання роботи, було створено програмний продукт "ProcessViewer", що забезпечує підвищення ефективності управління процесами в ОС Windows та отримання розширеної інформації щодо процесів, потоків і модулів процесу.

Розроблено комплект необхідної програмної документації, що включає в себе, опис програми, керівництво програміста та керівництво оператора.

Застосунок "ProcessViewer" тестувався на ПК під керування ОС Windows 7 та Windows 10. Валідація програми проводилася шляхом порівняння результатів роботи програми з інформацією щодо процесів, потоків та модулів, яка була отримана за допомогою таких програмних аналогів як Task Manager та Process Explorer. Порівняння показало, що застосунок може експлуатуватись у якості програми для моніторингу та керування процесами в ОС сімейства Windows (x64).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гордеев, А.В. Операционные системы: учебник для вузов / А. В. Гордеев. – СПб.: Питер, 2004. – 416 с.
2. Процессы и потоки [Электрон. ресурс]. – Режим доступа: <https://proginfo.ru/processes-and-threads/>.
3. Концепции Windows NT [Электрон. ресурс]. – Режим доступа: http://citforum.ck.ua/operating_systems/sos/glava_39.shtml.
4. Робачевский, А.М. Операционная система UNIX / А.М. Робачевский. – СПб.: БХВ-Петербург, 2002. – 528 с.
5. Коньков, К. А. Основы организации операционных систем Microsoft Windows / К. А. Коньков. – М.: Интуит, 2005. – 536 с.
6. Столингс, В. Операционные системы / В. Столингс. – М.: Вильямс, 2002. – 848с.
7. Харт, Т., Джонсон, М. Системное программирование в среде Windows / Т. Харт, М. Джонсон. – М.: Вильямс, 2005. – 592 с.
8. Обзор новых функций Диспетчера задач в Windows 8 [Электрон. ресурс]. – Режим доступа: <http://www.winblog.ru/windows8/1147767810-30111101.html>.
9. Обзор аналогов Диспетчера задач [Электрон. ресурс]. – Режим доступа: <https://www.pkgid.ru/articles/obzor-analogov-dispatchera-zadach.html>.
10. 5 мощных альтернатив для диспетчера задач Windows [Электрон. ресурс]. – Режим доступа: <https://www.istocks.club/5-мощных-альтернатив-для-диспетчера-за/2020-05-13/>.
11. PROWiSe Manager для Windows [Электрон. ресурс]. – Режим доступа: <https://www.softportal.com/software-4296-prowise-manager.html>.
12. AnVir Task Manager [Электрон. ресурс]. – Режим доступа: <https://www.anvir.net/>.
13. AnVir Task Manager 9.3.3 [Электрон. ресурс]. – Режим доступа: https://biblprog.org.ua/ru/anvir_task_manager/.

14. TIOBE Index for February 2021 [Electronic resource]. – Access mode: <https://www.tiobe.com/tiobe-index/>.

15. C++ Overview [Electronic resource]. – Access mode: https://www.tutorialspoint.com/cplusplus/cpp_overview.htm.

16. A Brief Description C++ [Electronic resource]. – Access mode: <https://www.cplusplus.com/info/description/>.

17. Выпуск интегрированной среды разработки Qt Creator 4.12 [Электрон. ресурс]. – Режим доступа: <https://www.opennet.ru/opennews/art.shtml?num=52817>.

18. Разработка графического интерфейса с помощью библиотеки Qt3 [Электрон. ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/qt3_prog/qt3.html.

19. Подробное описание класса QWidget [Электрон. ресурс]. – Режим доступа: <http://doc.crossplatform.ru/qt/4.7.x/qwidget.html#details>.

20. Описание класса QString [Электрон. ресурс]. – Режим доступа: <http://qtdocs.narod.ru/4.1.0/doc/html/qstring.html>.

21. Описание класса QLabel [Электрон. ресурс]. – Режим доступа: <http://qtdocs.narod.ru/4.1.0/doc/html/qlabel.html>.

ДОДАТОК А
Текст програми

A.1 Код файла main.cpp

```
#include "cmainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    new CMainWindow;
    return a.exec();
}
```

A.2 Код файла mydelegate.cpp

```
#include "mydelegate.h"
mydelegate::mydelegate() {}
void mydelegate::paint(QPainter *painter, const QStyleOptionViewItem &option,
const QModelIndex &index) const
{
    QStyleOptionViewItem itemOption(option);
    if (itemOption.state & QStyle::State_HasFocus) itemOption.state =
itemOption.state ^ QStyle::State_HasFocus;
    QStyledItemDelegate::paint(painter, itemOption, index);
}
```

A.3 Код файла cthreaddatamodel.cpp

```
#include "cthreaddatamodel.h"
CThreadDataModel::CThreadDataModel()
{
}
void CThreadDataModel::UpdateView()
{
    emit layoutChanged();
}
void CThreadDataModel::AddThread(CThread *thread)
{
    beginInsertRows(QModelIndex(), rowCount(), rowCount());
    m_ThreadsList << thread;
    endInsertRows();
}
void CThreadDataModel::RemoveThread(int idx)
{
    beginRemoveRows(QModelIndex(), rowCount(), rowCount());
    CloseHandle(m_ThreadsList[idx]->threadHandle);
    m_ThreadsList.removeAt(idx);
    endRemoveRows();
}
int CThreadDataModel::rowCount(const QModelIndex &parent) const
{
    return m_ThreadsList.size();
}
```

```

}

int CThreadDataModel::columnCount(const QModelIndex &parent) const
{
    return 5;
}

QVariant CThreadDataModel::data(const QModelIndex &index, int role) const
{
    CThread* t = m_ThreadsList[index.row()];
    if(role == Qt::DisplayRole)
    {
        switch(index.column())
        {
            case 0 : return t->id;
            case 1 : return t->cpu_percents;
            case 2 : return t->suspended ? "S" : "R";
            case 3 : return t->spriority;
            case 4 : return t->startTime;
        }
    }
    else if(role == Qt::TextAlignmentRole)
    {
        return Qt::AlignCenter;
    }

    return QVariant();
}

QVariant CThreadDataModel::headerData(int section, Qt::Orientation orientation,
int role) const
{
    if (role == Qt::DisplayRole)
    {
        if (orientation == Qt::Horizontal)
        {
            switch (section)
            {
                case 0 : return QString("ID");
                case 1 : return QString("CPU");
                case 2 : return QString("State");
                case 3 : return QString("Priority");
                case 4 : return QString("Started");
            }
        }
    }
    return QVariant();
}

```

A.4 Код файла cpropworker.cpp

```

#include "cproworker.h"
CPropWorker::CPropWorker(CProcess *p) : m_pProcess(p), m_bModulesFound(false),
m_bImageInfoSet(false)
{

```

```

        connect(this, &CPropWorker::updateModelView, &m_pThreadsDataModel,
&CThreadDataModel::UpdateView);
    }
    int CPropWorker::ThreadAlreadyInList(int id)
    {
        if(m_pThreadsDataModel.m_ThreadsList.size() > 0)
        {
            for(int i = 0; i < m_pThreadsDataModel.m_ThreadsList.size(); ++i)
            {
                if(m_pThreadsDataModel.m_ThreadsList[i]->id == id) return i;
            }
        }
        return -1;
    }
    void CPropWorker::SearchForNewThreads()
    {
        ULONG cbBuffer = 131072;
        PVOID pBuffer = NULL;
        NTSTATUS Status = STATUS_INFO_LENGTH_MISMATCH;
        HANDLE hHeap = GetProcessHeap();
        pBuffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, cbBuffer);
        if (pBuffer == NULL) return;
        Status = pNtQuerySysInfo(5, pBuffer, cbBuffer, &cbBuffer);
        if (Status == STATUS_INFO_LENGTH_MISMATCH) {
            while(Status == STATUS_INFO_LENGTH_MISMATCH)
            {
                HeapFree(hHeap, NULL, pBuffer);
                cbBuffer *= 2;
                pBuffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, cbBuffer);
                if (pBuffer == NULL) return;
                Status = pNtQuerySysInfo(5, pBuffer, cbBuffer, &cbBuffer);
            }
        }
        if (!NT_SUCCESS(Status)) {
            HeapFree(hHeap, NULL, pBuffer);
        }
        else {
            PSYSTEM_PROCESSES infoP = NULL;
            infoP = (PSYSTEM_PROCESSES)pBuffer;

            while (infoP)
            {
                if(infoP->ProcessId == m_pProcess->id)
                {
                    for(int i = 0; i < infoP->ThreadCount; ++i)
                    {
                        int checkIndex = ThreadAlreadyInList(infoP-
>Threads[i].ClientId.UniqueThread);
                        if(checkIndex != -1)
                        {
                            getThreadCPU(m_pThreadsDataModel.m_ThreadsList[checkIndex]);
                            m_pThreadsDataModel.m_ThreadsList[checkIndex]-
>spriority =
priorityToString(GetThreadPriority(m_pThreadsDataModel.m_ThreadsList[checkIndex]
->threadHandle));
                            continue;
                        }
                    }
                }
                infoP = infoP->Next;
            }
        }
    }

```

```

        }
        HANDLE threadHandle = OpenThread(THREAD_QUERY_INFORMATION |
        THREAD_SUSPEND_RESUME | THREAD_TERMINATE | THREAD_SET_INFORMATION, FALSE,
        infoP->Threads[i].ClientId.UniqueThread);
        if(!threadHandle) continue;
        CThread* thread = new CThread;
        thread->id = infoP->Threads[i].ClientId.UniqueThread;
        thread->threadHandle = threadHandle;
        getThreadCPU(thread);
        FILETIME ftime, fexit, fsys, fuser;
        GetThreadTimes(thread->threadHandle, &ftime, &fexit, &fsys,
        &fuser);

        SYSTEMTIME sThreadTime;
        FileTimeToSystemTime(&ftime, &sThreadTime);
        thread->startTime = QString::number(sThreadTime.wHour) +
        ":" + QString::number(sThreadTime.wMinute) + ":" +
        QString::number(sThreadTime.wSecond) + " - " +
        QString::number(sThreadTime.wMonth) + "/" + QString::number(sThreadTime.wDay) +
        "/" + QString::number(sThreadTime.wYear);
        thread->spriority =
        priorityToString(GetThreadPriority(thread->threadHandle));

        m_pThreadsDataModel.AddThread(thread);
    }
    break;
}
else
{
    if (!infoP->NextEntryDelta) break;
    infoP = (PSYSTEM_PROCESSES)(((LPBYTE)infoP) + infoP->
    NextEntryDelta);
}
}
if (pBuffer) HeapFree(GetProcessHeap(), NULL, pBuffer);
}
emit updateModelView();
emit updateThreadsCount("Count : " +
QString::number(m_pThreadsDataModel.m_ThreadsList.size()));
}
QString CPropWorker::priorityToString(DWORD priorityClass)
{
    switch(priorityClass)
    {
        case THREAD_PRIORITY_IDLE: return "Idle";
        case THREAD_PRIORITY_LOWEST: return "Low";
        case THREAD_PRIORITY_BELOW_NORMAL: return "Below normal";
        case THREAD_PRIORITY_NORMAL: return "Normal";
        case THREAD_PRIORITY_ABOVE_NORMAL: return "Above normal";
        case THREAD_PRIORITY_HIGHEST: return "High";
        case THREAD_PRIORITY_TIME_CRITICAL: return "High";
    }
    return QString::number(priorityClass);
}
void CPropWorker::ManageCurrentThreads()
{
    for(int i = 0; i < m_pThreadsDataModel.m_ThreadsList.size(); ++i)

```

```

{
    DWORD retVal = 0;
    if(GetExitCodeThread(m_pThreadsDataModel.m_ThreadsList[i]-
>threadHandle, &retVal) && retVal == STILL_ACTIVE)
    {
        int a = 5;
    }
    else
    {
        m_pThreadsDataModel.RemoveThread(i);
    }
}
}
void CPropWorker::GetModulesList()
{
    MODULEENTRY32 me32;
    HANDLE hModuleSnap;
    hModuleSnap = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE, m_pProcess->id);
    if( hModuleSnap == INVALID_HANDLE_VALUE )
    {
        return;
    }
    me32.dwSize = sizeof( MODULEENTRY32 );
    if(!Module32First(hModuleSnap, &me32))
    {
        CloseHandle(hModuleSnap);
        return;
    }
    do
    {
        CModule* module = new CModule;
        module->name = QString::fromWCharArray(me32.szModule);
        module->path =   QString::fromWCharArray(me32.szExePath);
        DWORD dwLen, dwUseless;
        LPTSTR lpVI;
        QString companyName = QString("-"),
                fileDescription = QString("-");
        UINT verMajor;
        //wchar_t path[MAX_PATH];
        //GetModuleFileNameExW(process->handle, 0, path, MAX_PATH);
        dwLen = GetFileVersionInfoSizeW(me32.szExePath, &dwUseless);
        if (dwLen!=0)
        {
            lpVI = (LPTSTR) GlobalAlloc(GPTR, dwLen);
            if (lpVI)
            {
                DWORD dwBufSize;
                VS_FIXEDFILEINFO* lpFFI;
                BOOL bRet = FALSE;
                WORD* langInfo;
                UINT cbLang;
                TCHAR tszVerStrName[128];
                LPVOID lpt;
                UINT cbBufSize;
                GetFileVersionInfoW(me32.szExePath, NULL, dwLen, lpVI);

```

```

        VerQueryValue(lpVI, L"\\VarFileInfo\\Translation",
(LPVOID*)&langInfo, &cbLang);
        wsprintf(tszVerStrName, L"\\StringFileInfo\\%04x%04x\\%s",
langInfo[0], langInfo[1], L"CompanyName");
        if (VerQueryValue(lpVI, tszVerStrName, &lpt, &cbBufSize))
companyName = QString::fromUtf16((ushort*)lpt);
        wsprintf(tszVerStrName, L"\\StringFileInfo\\%04x%04x\\%s",
langInfo[0], langInfo[1], L"FileDescription");
        if (VerQueryValue(lpVI, tszVerStrName, &lpt, &cbBufSize))
fileDescription = QString::fromUtf16((ushort*)lpt);
        GlobalFree((HGLOBAL)lpVI);
    }
}
module->companyName = companyName;
module->description = fileDescription;
m_pModulesDataModel.AddModule(module);
}while(Module32Next(hModuleSnap, &mme32));
m_bModulesFound = true;
}
void CPropWorker::UpdateProcessInfo()
{
    SYSTEM_INFO sysInfo;
    FILETIME ftime, fexit, fsys, fuser;
    ULARGE_INTEGER now, sys, user;
    double percent;
    GetSystemInfo(&sysInfo);
    GetSystemTimeAsFileTime(&ftime);
    memcpy(&now, &ftime, sizeof(FILETIME));
    GetProcessTimes(m_pProcess->handle, &ftime, &fexit, &fsys, &fuser);
    memcpy(&sys, &fsys, sizeof(FILETIME));
    memcpy(&user, &fuser, sizeof(FILETIME));
    percent = (sys.QuadPart - lastSysCPU.QuadPart) + (user.QuadPart -
lastUserCPU.QuadPart);
    percent /= (now.QuadPart - lastCPU.QuadPart);
    percent /= sysInfo.dwNumberOfProcessors;
    lastCPU = now;
    lastUserCPU = user;
    lastSysCPU = sys;
    m_pProcess->cpu_percents = (percent * 100) == 0 ? "< 0.01 %" :
QString::number(percent * 100, 'g', 2) + " %";
    VM_COUNTERS vmc;
    pNtQueryInformationProcess(m_pProcess->handle, 3, &vmc, sizeof(VM_COUNTERS),
0);
    IO_COUNTERS ioc;
    pNtQueryInformationProcess(m_pProcess->handle, 2, &ioc, sizeof(IO_COUNTERS),
0);
    m_pProcess->privBytes = QString::number(vmc.PagefileUsage/(1024.0*1024.0))
+ " MB";
    m_pProcess->peakPrivBytes =
QString::number(vmc.PeakPagefileUsage/(1024.0*1024.0)) + " MB";
    m_pProcess->virtualSize = QString::number(vmc.VirtualSize/(1024.0*1024.0))
+ " MB";
    m_pProcess->peakVirtualSize =
QString::number(vmc.PeakVirtualSize/(1024.0*1024.0)) + " MB";
    m_pProcess->pageFaults = QString::number(vmc.PageFaultCount);
}

```

```

        m_pProcess->workingSet =
QString::number(vmc.WorkingSetSize/(1024.0*1024.0)) + " MB";
        m_pProcess->peakWorkingSet =
QString::number(vmc.PeakWorkingSetSize/(1024.0*1024.0)) + " MB";
        m_pProcess->reads = QString::number(ioc.ReadOperationCount);
        m_pProcess->readBytes = QString::number(ioc.ReadTransferCount /
(1024*1024)) + " MB";
        m_pProcess->writes = QString::number(ioc.WriteOperationCount);
        m_pProcess->writeBytes = QString::number(ioc.WriteTransferCount /
(1024*1024)) + " MB";
        m_pProcess->other = QString::number(ioc.OtherOperationCount);
        m_pProcess->otherBytes = QString::number(ioc.OtherTransferCount /
(1024*1024)) + " MB";
        emit sendPerformanceData(m_pProcess);
    }
void CPropWorker::UpdatePropWindowGUI()
{
    if(m_pageNum == 0 && !m_bImageInfoSet)
    {
        FILETIME ftime, fexit, fsys, fuser;
        GetProcessTimes(m_pProcess->handle, &ftime, &fexit, &fsys, &fuser);
        SYSTEMTIME sProcessTime;
        FileTimeToSystemTime(&ftime, &sProcessTime);
        m_pProcess->createTime = QString::number(sProcessTime.wHour) + ":" +
QString::number(sProcessTime.wMinute) + ":" +
QString::number(sProcessTime.wSecond) + " - " +
QString::number(sProcessTime.wMonth) + "/" + QString::number(sProcessTime.wDay)
+ "/" + QString::number(sProcessTime.wYear);
        WIN32_FILE_ATTRIBUTE_DATA fileData;
        GetFileAttributesExW((const wchar_t*)m_pProcess->fullpath.utf16(),
GetFileExInfoStandard, &fileData);
        SYSTEMTIME imageTimes;
        FileTimeToSystemTime(&fileData.ftCreationTime, &imageTimes);
        m_pProcess->imageCreateTime = QString::number(imageTimes.wHour) + ":" +
QString::number(imageTimes.wMinute) + ":" + QString::number(imageTimes.wSecond)
+ " - " + QString::number(imageTimes.wMonth) + "/" +
QString::number(imageTimes.wDay) + "/" + QString::number(imageTimes.wYear);
        m_pProcess->fileSize = QString::number((fileData.nFileSizeHigh *
MAXDWORD + fileData.nFileSizeLow) / (1024.0*1024.0)) + " MB";
        emit sendImageTabData(m_pProcess);
        m_bImageInfoSet = true;
    }
    if(m_pageNum == 0)
    {
        UpdateProcessInfo();
    }
    if(m_pageNum == 1)
    {
        if(m_pThreadsDataModel.m_ThreadsList.size() > 0)
ManageCurrentThreads();
        SearchForNewThreads();
    }
    if(m_pageNum == 2)
    {
        if(!m_bModulesFound) GetModulesList();
    }
}

```

```

}
CThreadDataModel *CPropWorker::GetThreadModel()
{
    return &m_pThreadsDataModel;
}
CModuleDataModel *CPropWorker::GetModuleModel()
{
    return &m_pModulesDataModel;
}
void CPropWorker::stopThread()
{
    m_bStop = true;
}
bool CPropWorker::getThreadCPU(CThread* thread)
{
    SYSTEM_INFO sysInfo;
    FILETIME ftime, fexit, fsys, fuser;
    ULARGE_INTEGER now, sys, user;
    double percent;
    GetSystemInfo(&sysInfo);
    GetSystemTimeAsFileTime(&ftime);
    memcpy(&now, &ftime, sizeof(FILETIME));
    GetThreadTimes(thread->threadHandle, &ftime, &fexit, &fsys, &fuser);
    memcpy(&sys, &fsys, sizeof(FILETIME));
    memcpy(&user, &fuser, sizeof(FILETIME));
    percent = (sys.QuadPart - thread->lastSysCPU.QuadPart) + (user.QuadPart -
thread->lastUserCPU.QuadPart);
    percent /= (now.QuadPart - thread->lastCPU.QuadPart);
    percent /= sysInfo.dwNumberOfProcessors;
    thread->lastCPU = now;
    thread->lastUserCPU = user;
    thread->lastSysCPU = sys;
    thread->cpu_percents = (percent * 100) == 0 ? "< 0.01 %" :
QString::number(percent * 100, 'g', 2) + " %";
    return true;
}
void CPropWorker::setActivePage(int num)
{
    m_pageNum = num;
}
void CPropWorker::doWork()
{
    m_pageNum = 0;
    m_bStop = false;
    while(!m_bStop)
    {
        UpdatePropWindowGUI();
        QEventLoop loop;
        QTimer::singleShot(1000, &loop, &QEventLoop::quit);
        loop.exec();
    }
    emit finished();
}

```

A.5 Код файла cpropertieswindow.cpp

```

#include "cpropertieswindow.h"
#include "ui_cpropertieswindow.h"
#include <QDir>
#include <QProcess>
CPropertiesWindow::CPropertiesWindow(QWidget *parent, CProcess *process) :
    QWidget(parent),
    ui(new Ui::cpropertieswindow)
{
    ui->setupUi(this);
    m_pProc = process;
    m_pPropWorker = new CPropWorker(process);
    m_pThread = new QThread;
    m_pPropWorker->moveToThread(m_pThread);
    susp_res = conMenu.addAction("Suspend");
    connect(susp_res, &QAction::triggered, this,
    &CPropertiesWindow::suspendResumeThread);
    connect(conMenu.addAction("Kill"), &QAction::triggered, this,
    &CPropertiesWindow::killThread);
    prioritySubMenu = conMenu.addMenu("Set priority");
    connect(prioritySubMenu->addAction("High"), &QAction::triggered, this,
    &CPropertiesWindow::changePriority);
    connect(prioritySubMenu->addAction("Above normal"), &QAction::triggered,
    this, &CPropertiesWindow::changePriority);
    connect(prioritySubMenu->addAction("Normal"), &QAction::triggered, this,
    &CPropertiesWindow::changePriority);
    connect(prioritySubMenu->addAction("Below normal"), &QAction::triggered,
    this, &CPropertiesWindow::changePriority);
    connect(prioritySubMenu->addAction("Low"), &QAction::triggered, this,
    &CPropertiesWindow::changePriority);
    connect(ui->tableViewThreads, &QTableView::customContextMenuRequested,
    this, &CPropertiesWindow::openContextMenu);
    connect(m_pThread, &QThread::started, m_pPropWorker, &CPropWorker::doWork);
    connect(m_pPropWorker, &CPropWorker::finished, m_pThread, &QThread::quit);
    connect(m_pPropWorker, &CPropWorker::finished, m_pPropWorker,
    &CPropWorker::deleteLater);
    connect(m_pThread, &QThread::finished, m_pThread, &QThread::deleteLater);
    connect(ui->tabWidget, &QTabWidget::currentChanged, m_pPropWorker,
    &CPropWorker::setActivePage);
    connect(m_pPropWorker, &CPropWorker::sendPerfomanceData, this,
    &CPropertiesWindow::updatePerfomanceInfo);
    connect(m_pPropWorker, &CPropWorker::sendImageTabData, this,
    &CPropertiesWindow::setupImageTab);
    connect(ui->explorer_btn, &QPushButton::clicked, this,
    &CPropertiesWindow::showInExplorer);
    connect(ui->properties_btn, &QPushButton::clicked, this,
    &CPropertiesWindow::openExplorerProperties);
    connect(m_pPropWorker, &CPropWorker::updateThreadsCount, ui->thredsCount,
    &QLabel::setText);
    ui->tableViewThreads->setContextMenuPolicy(Qt::CustomContextMenu);
    ui->tableViewThreads->horizontalHeader()->setStretchLastSection(true);
    ui->tableViewThreads->setSelectionBehavior(QAbstractItemView::SelectRows);
    ui->tableViewThreads->setItemDelegate(&delegate);

```

```

        ui->tableViewThreads->setSelectionMode(QAbstractItemView::SingleSelection);
        ui->tableViewThreads->setColumnWidth(1, 100);
        ui->tableViewThreads->setModel(m_pPropWorker->GetThreadModel());
        ui->tableViewModules->horizontalHeader()->setStretchLastSection(true);
        ui->tableViewModules->horizontalHeader()-
>setSectionResizeMode(QHeaderView::ResizeToContents);
        ui->tableViewModules->setSelectionBehavior(QAbstractItemView::SelectRows);
        ui->tableViewModules->setItemDelegate(&delegate);
        ui->tableViewModules->setModel(m_pPropWorker->GetModuleModel());
        setWindowIcon(process->icon);
        setWindowTitle(process->name + " : [" + QString::number(process->id) + "]"
Properties");
        setAttribute(Qt::WA_DeleteOnClose);
        setWindowFlags(Qt::Dialog);
        setWindowModality(Qt::WindowModal);
        m_pThread->start();
        show();
    }
CPropertiesWindow::~CPropertiesWindow()
{
    m_pPropWorker->stopThread();
    delete ui;
}
void CPropertiesWindow::openContextMenu(QPoint pos)
{
    conMenu.popup(ui->tableViewThreads->viewport()->mapToGlobal(pos));
}
void CPropertiesWindow::killThread()
{
    QModelIndexList mList = ui->tableViewThreads->selectionModel()-
>selectedRows();
    TerminateThread(m_pPropWorker->GetThreadModel()-
>m_ThreadsList[mList.at(0).row()]->threadHandle, STATUS_SUCCESS);
}
void CPropertiesWindow::suspendResumeThread()
{
    QModelIndexList mList = ui->tableViewThreads->selectionModel()-
>selectedRows();
    if(m_pPropWorker->GetThreadModel()->m_ThreadsList[mList.at(0).row()]-
>suspended)
    {
        ResumeThread(m_pPropWorker->GetThreadModel()-
>m_ThreadsList[mList.at(0).row()]->threadHandle);
        m_pPropWorker->GetThreadModel()->m_ThreadsList[mList.at(0).row()]-
>suspended = false;
        susp_res->setText("Suspend");
    }
    else
    {
        SuspendThread(m_pPropWorker->GetThreadModel()-
>m_ThreadsList[mList.at(0).row()]->threadHandle);
        m_pPropWorker->GetThreadModel()->m_ThreadsList[mList.at(0).row()]-
>suspended = true;
        susp_res->setText("Resume");
    }
}
}

```

```

void CPropertiesWindow::changePriority()
{
    QAction *act = qobject_cast<QAction *>(sender());
    LONG threadPriority;
    if(!act->text().compare("High")) threadPriority = THREAD_PRIORITY_HIGHEST;
    if(!act->text().compare("Above normal")) threadPriority =
THREAD_PRIORITY_ABOVE_NORMAL;
    if(!act->text().compare("Normal")) threadPriority =
THREAD_PRIORITY_NORMAL;
    if(!act->text().compare("Below normal")) threadPriority =
THREAD_PRIORITY_BELOW_NORMAL;
    if(!act->text().compare("Low")) threadPriority = THREAD_PRIORITY_LOWEST;
    QModelIndexList mList = ui->tableViewThreads->selectionModel()-
>selectedRows();
    SetThreadPriority(m_pPropWorker->GetThreadModel()-
>m_ThreadsList[mList.at(0).row()]->threadHandle, threadPriority);
}
void CPropertiesWindow::setupImageTab(CProcess *p)
{
    ui->tabWidget->currentWidget()->findChild<QLabel*>("icon_label")-
>setPixmap(p->icon.pixmap(50, 50));
    ui->tabWidget->currentWidget()->findChild<QLabel*>("product_label")-
>setText(p->productName);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("company_label")-
>setText(p->companyName);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("version_label")-
>setText(p->productVersion);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("path_label")-
>setText(p->fullpath);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("started_label")-
>setText("Started : " + p->createTime);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("created_lbl")-
>setText(p->imageCreateTime);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("size_lbl")->setText(p-
>fileSize);
}
void CPropertiesWindow::updatePerformanceInfo(CProcess *p)
{
    ui->tabWidget->currentWidget()->findChild<QLabel*>("cpu_per_val")-
>setText(p->cpu_percents);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("priv_bytes")-
>setText(p->privBytes);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("peak_priv_bytes")-
>setText(p->peakPrivBytes);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("virt_size")->setText(p-
>virtualSize);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("peak_virt_size")-
>setText(p->peakVirtualSize);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("page_faults")-
>setText(p->pageFaults);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("work_set")->setText(p-
>workingSet);
    ui->tabWidget->currentWidget()->findChild<QLabel*>("peak_ws")->setText(p-
>peakWorkingSet);
}

```

```

        ui->tabWidget->currentWidget()->findChild<QLabel*>("reads_lbl")->setText(p-
>reads);
        ui->tabWidget->currentWidget()->findChild<QLabel*>("readB_lbl")->setText(p-
>readBytes);
        ui->tabWidget->currentWidget()->findChild<QLabel*>("writes_lbl")->
>setText(p->writes);
        ui->tabWidget->currentWidget()->findChild<QLabel*>("writesB_lbl")->
>setText(p->writeBytes);
        ui->tabWidget->currentWidget()->findChild<QLabel*>("other_lbl")->setText(p-
>other);
        ui->tabWidget->currentWidget()->findChild<QLabel*>("otherB_lbl")->
>setText(p->otherBytes);
    }
void CPropertiesWindow::showInExplorer()
{
    QStringList args;
    args << "/select," << QDir::toNativeSeparators(ui->path_label->text());
    QProcess *process = new QProcess(this);
    process->start("explorer.exe", args);
}
void CPropertiesWindow::openExplorerProperties()
{
    SHELLEXECUTEINFO ShellExecuteInfo;
    ShellExecuteInfo.cbSize = sizeof(ShellExecuteInfo);
    ShellExecuteInfo.fMask = SEE_MASK_NOCLOSEPROCESS | SEE_MASK_INVOKEIDLIST |
SEE_MASK_FLAG_NO_UI;
    ShellExecuteInfo.hwnd = 0;
    ShellExecuteInfo.lpVerb = L"properties";
    ShellExecuteInfo.lpFile = (const wchar_t*)m_pProc->fullpath.utf16();
    ShellExecuteInfo.lpParameters = NULL;
    ShellExecuteInfo.lpDirectory = NULL;
    ShellExecuteInfo.nShow = 0;
    ShellExecuteInfo.hInstApp = 0;
    ShellExecuteInfo.lpIDList = 0;
    ShellExecuteEx(&ShellExecuteInfo);
}

```

A.6 Код файла cprocessdatamodel.cpp

```

#include "cprocessdatamodel.h"

CProcessDataModel::CProcessDataModel() {}
void CProcessDataModel::AddProcess(CProcess *process)
{
    beginInsertRows(QModelIndex(), rowCount(), rowCount());
    m_ProcessList << process; endInsertRows();
}
void CProcessDataModel::RemoveProcess(int idx)
{
    beginRemoveRows(QModelIndex(), idx, idx);
    CloseHandle(m_ProcessList[idx]->handle);
    m_ProcessList.removeAt(idx); endRemoveRows();
}
void CProcessDataModel::UpdateView()

```

```

{
    emit layoutChanged();
}
int CProcessDataModel::rowCount(const QModelIndex &parent) const
{
    return m_ProcessList.size();
}
int CProcessDataModel::columnCount(const QModelIndex &parent) const
{
    return 6;
}
QVariant CProcessDataModel::data(const QModelIndex &index, int role) const
{
    CProcess* p = m_ProcessList[index.row()];
    if(role == Qt::DisplayRole)
    {
        switch(index.column())
        {
            case 0 : return p->id;
            case 1 : return p->name;
            case 2 : return p->suspended ? "S" : "R";
            case 3 : return p->spriority;
            case 4 : return p->companyName;
            case 5 : return p->fileDescription;
        }
    }
    else if(role == Qt::DecorationRole && index.column() == 0)
    {
        if(p->icon.isNull()) return 0; return p->icon;
    }
    else if(index.column() == 2 && role == Qt::TextAlignmentRole)
    {
        return Qt::AlignCenter;
    }
    return QVariant();
}
QVariant CProcessDataModel::headerData(int section, Qt::Orientation
orientation, int role) const
{
    if (role == Qt::DisplayRole)
    {
        if (orientation == Qt::Horizontal)
        {
            switch (section)
            {
                case 0 : return QString("ID");
                case 1 : return QString("Name");
                case 2 : return QString("State");
                case 3 : return QString("Priority");
                case 4 : return QString("Company name");
                case 5 : return QString("Description");
            }
        }
    }
    return QVariant();
}

```

A.7 Код файла cmoduledatamodel.cpp

```
#include "cmoduledatamodel.h"
CModuleDataModel::CModuleDataModel()
{
}
void CModuleDataModel::AddModule(CModule *module)
{
    beginInsertRows(QModelIndex(), rowCount(), rowCount());
    m_ModulesList << module;
    endInsertRows();
}
void CModuleDataModel::RemoveModule(int idx)
{
    beginRemoveRows(QModelIndex(), rowCount(), rowCount());
    //CloseHandle(m_ThreadsList[idx]->threadHandle);
    m_ModulesList.removeAt(idx);
    endRemoveRows();
}
int CModuleDataModel::rowCount(const QModelIndex &parent) const
{
    return m_ModulesList.size();
}
int CModuleDataModel::columnCount(const QModelIndex &parent) const
{
    return 4;
}
QVariant CModuleDataModel::data(const QModelIndex &index, int role) const
{
    CModule* m = m_ModulesList[index.row()];
    if(role == Qt::DisplayRole)
    {
        switch(index.column())
        {
            case 0 : return m->name;
            case 1 : return m->description;
            case 2 : return m->companyName;
            case 3 : return m->path;
        }
    }
    return QVariant();
}
QVariant CModuleDataModel::headerData(int section, Qt::Orientation orientation,
int role) const
{
    if (role == Qt::DisplayRole)
    {
        if (orientation == Qt::Horizontal)
        {
            switch (section)
            {
                case 0 : return QString("Name");
                case 1 : return QString("Description");
                case 2 : return QString("Company name");
            }
        }
    }
}
```

```

        case 3 : return QString("Path");
    }
}
}
return QVariant();
}

```

A.7 Код файла cmainworker.cpp

```

#include "cmainworker.h"
CMainWorker::CMainWorker() : m_bFirstProcessSelected(false), selectedIndex(0)
{
    connect(this, &CMainWorker::updateModelView, &m_pProcessDataModel,
&CProcessDataModel::UpdateView);
}
QString CMainWorker::priorityToString(LONG pr)
{
    switch(pr)
    {
        case 4: return "Low";
        case 6: return "Below normal";
        case 8: return "Normal";
        case 10: return "Above normal";
        case 13: return "High";
        case 24: return "High";
    }
    return QString::number(pr);
}
void CMainWorker::stopThread()
{
    m_bStop = true;
}
int CMainWorker::ProcessAlreadyInList(int id)
{
    if(m_pProcessDataModel.m_ProcessList.size() > 0)
    {
        for(int i = 0; i < m_pProcessDataModel.m_ProcessList.size(); ++i)
        {
            if(m_pProcessDataModel.m_ProcessList[i]->id == id) return i;
        }
    }
    return -1;
}
void CMainWorker::SearchForNewProcesses()
{
    ULONG cbBuffer = 131072;
    PVOID pBuffer = NULL;
    NTSTATUS Status = STATUS_INFO_LENGTH_MISMATCH;
    HANDLE hHeap = GetProcessHeap();

    pBuffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, cbBuffer);
    if (pBuffer == NULL) return;
    Status = pNtQuerySysInfo(5, pBuffer, cbBuffer, &cbBuffer);
}

```

```

if (Status == STATUS_INFO_LENGTH_MISMATCH) {
    while(Status == STATUS_INFO_LENGTH_MISMATCH)
    {
        HeapFree(hHeap, NULL, pBuffer);
        cbBuffer *= 2;
        pBuffer = HeapAlloc(hHeap, HEAP_ZERO_MEMORY, cbBuffer);
        if (pBuffer == NULL) return;
        Status = pNtQuerySysInfo(5, pBuffer, cbBuffer, &cbBuffer);
    }
}
if (!NT_SUCCESS(Status)) {
    HeapFree(hHeap, NULL, pBuffer);
    return;
}
else {
    PSYSTEM_PROCESSES infoP = NULL;
    infoP = (PSYSTEM_PROCESSES)pBuffer;

    while (infoP)
    {
        int checkIndex = ProcessAlreadyInList(infoP->ProcessId);
        if(checkIndex != -1)
        {
            LONG oldPrior = m_pProcessDataModel.m_ProcessList[checkIndex]-
>priority;
            m_pProcessDataModel.m_ProcessList[checkIndex]->priority =
infoP->BasePriority;
            m_pProcessDataModel.m_ProcessList[checkIndex]->spriority =
priorityToString(infoP->BasePriority);
            if(m_pProcessDataModel.m_ProcessList[checkIndex]-
>priority!=oldPrior) emit updateModelView();

            if(infoP->NextEntryDelta)
            {
                infoP = (PSYSTEM_PROCESSES)(((LPBYTE)infoP) + infoP-
>NextEntryDelta);
                continue;
            }
            else break;
        }
        char pName[256];
        memset(pName, 0, sizeof(pName));
        WideCharToMultiByte(CP_ACP, 0, infoP->ProcessName.Buffer, infoP-
>ProcessName.Length, pName, sizeof(pName), NULL, NULL);
        HANDLE procHandle = OpenProcess(PROCESS_QUERY_INFORMATION |
PROCESS_VM_READ | PROCESS_SUSPEND_RESUME | PROCESS_TERMINATE |
PROCESS_SET_INFORMATION, FALSE, (DWORD)infoP->ProcessId);
        if(!procHandle)
        {
            infoP = (PSYSTEM_PROCESSES)(((LPBYTE)infoP) + infoP-
>NextEntryDelta);
            continue;
        }
        CProcess* process = new CProcess;
        process->id = infoP->ProcessId;
        process->name = QString(pName);
    }
}

```

```

process->handle = procHandle;

DWORD dwLen, dwUseless;
LPTSTR lpVI;
QString companyName = QString("-"),
        fileDescription = QString("-"),
        prodName = QString("-"),
        prodVersion = QString("-");
UINT verMajor;
wchar_t path[MAX_PATH];
GetModuleFileNameExW(process->handle, 0, path, MAX_PATH);
dwLen = GetFileVersionInfoSizeW(path, &dwUseless);
if (dwLen!=0)
{
    lpVI = (LPTSTR) GlobalAlloc(GPTR, dwLen);
    if (lpVI)
    {
        DWORD dwBufSize;
        VS_FIXEDFILEINFO* lpFFI;
        WORD* langInfo;
        UINT cbLang;
        TCHAR tszVerStrName[128];
        LPVOID lpt;
        UINT cbBufSize;
        GetFileVersionInfoW((LPTSTR)path, NULL, dwLen, lpVI);
        if (VerQueryValue(lpVI, L"\\", (LPVOID *) &lpFFI, (UINT *)
&dwBufSize)) verMajor = HIWORD(lpFFI->dwFileVersionMS);
        VerQueryValue(lpVI, L"\\VarFileInfo\\Translation",
(LPVOID*)&langInfo, &cbLang);
        wsprintf(tszVerStrName, L"\\StringFileInfo\\%04x%04x\\%",
langInfo[0], langInfo[1], L"CompanyName");
        if (VerQueryValue(lpVI, tszVerStrName, &lpt, &cbBufSize))
companyName = QString::fromUtf16((ushort*)lpt);
        wsprintf(tszVerStrName, L"\\StringFileInfo\\%04x%04x\\%",
langInfo[0], langInfo[1], L"FileDescription");
        if (VerQueryValue(lpVI, tszVerStrName, &lpt, &cbBufSize))
fileDescription = QString::fromUtf16((ushort*)lpt);
        wsprintf(tszVerStrName, L"\\StringFileInfo\\%04x%04x\\%",
langInfo[0], langInfo[1], L"ProductName");
        if (VerQueryValue(lpVI, tszVerStrName, &lpt, &cbBufSize))
prodName = QString::fromUtf16((ushort*)lpt);
        wsprintf(tszVerStrName, L"\\StringFileInfo\\%04x%04x\\%",
langInfo[0], langInfo[1], L"ProductVersion");
        if (VerQueryValue(lpVI, tszVerStrName, &lpt, &cbBufSize))
prodVersion = QString::fromUtf16((ushort*)lpt);

        GlobalFree((HGLOBAL)lpVI);
    }
}
process->companyName = companyName;
process->fileDescription = fileDescription;
process->productName = prodName;
process->productVersion = prodVersion;
process->fullpath = QString::fromUtf16((ushort*)path);
process->priority = infoP->BasePriority;
process->spriority = priorityToString(infoP->BasePriority);

```

```

        process->icon.addPixmap(QtWin::fromHICON(ExtractIcon(0, path, 0)));
        if(process->icon.isNull()) process->icon =
QIcon(":/icons/rsc/file.png");
        m_pProcessDataModel.AddProcess(process);
        if (!infoP->NextEntryDelta) break;
        infoP = (PSYSTEM_PROCESSES)(((LPBYTE)infoP) + infoP-
>NextEntryDelta);
    }
    if (pBuffer) HeapFree(GetProcessHeap(), NULL, pBuffer);
}
    emit updateProcCount("Count : " +
QString::number(m_pProcessDataModel.m_ProcessList.size()));
}
void CMainWorker::ManageCurrentProcesses()
{
    for(int i = 0; i < m_pProcessDataModel.m_ProcessList.size(); ++i)
    {
        DWORD retVal = 0;
        if(GetExitCodeProcess(m_pProcessDataModel.m_ProcessList[i]->handle,
&retVal) && retVal == STILL_ACTIVE)
        {
        }
        else
        {
            m_pProcessDataModel.RemoveProcess(i);
        }
    }
}
void CMainWorker::UpdateMainWindowGUI()
{
    if(m_pProcessDataModel.m_ProcessList.size() > 0) ManageCurrentProcesses();
    SearchForNewProcesses();
}
CProcessDataModel *CMainWorker::GetProcessModel()
{
    return &m_pProcessDataModel;
}
void CMainWorker::doWork()
{
    m_bStop = false;
    while(!m_bStop)
    {
        UpdateMainWindowGUI();
        QEventLoop loop;
        QTimer::singleShot(1000, &loop, &QEventLoop::quit);
        loop.exec();
    }
    emit finished();
}
void CMainWorker::setSelected(const QModelIndex& cur, const QModelIndex& prev)
{
    selectedIndex = cur.row();
}

```

A.8 Код файла cmainwindow.cpp

```

#include "cmainwindow.h"
#include "ui_cmainwindow.h"
#include "cprocessdatamodel.h"
#include <QMessageBox>
tNtQuerySysInfo pNtQuerySysInfo;
tNtSuspendProcess pNtSuspendProcess;
tNtResumeProcess pNtResumeProcess;
tNtQueryInformationProcess pNtQueryInformationProcess;
CMainWindow::CMainWindow(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::cmainwindow)
{
    ui->setupUi(this);
    NtInit();
    ui->tableView->setContextMenuPolicy(Qt::CustomContextMenu);
    ui->tableView->horizontalHeader()->setStretchLastSection(true);
    ui->tableView->horizontalHeader()-
>setSectionResizeMode(QHeaderView::ResizeToContents);
    ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
    ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);
    susp_res = conMenu.addAction("Suspend");
    connect(susp_res, &QAction::triggered, this,
&CMainWindow::suspendResumeProcess);
    connect(conMenu.addAction("Kill"), &QAction::triggered, this,
&CMainWindow::killProcess);
    prioritySubMenu = conMenu.addMenu("Set priority");
    connect(prioritySubMenu->addAction("High"), &QAction::triggered, this,
&CMainWindow::changePriority);
    connect(prioritySubMenu->addAction("Above normal"), &QAction::triggered,
this, &CMainWindow::changePriority);
    connect(prioritySubMenu->addAction("Normal"), &QAction::triggered, this,
&CMainWindow::changePriority);
    connect(prioritySubMenu->addAction("Below normal"), &QAction::triggered,
this, &CMainWindow::changePriority);
    connect(prioritySubMenu->addAction("Low"), &QAction::triggered, this,
&CMainWindow::changePriority);
    connect(conMenu.addAction("Properties"), &QAction::triggered, this, [=]{
        QModelIndexList mList = ui->tableView->selectionModel()-
>selectedRows();
        pw = new CPropertiesWindow(this, m_pMainWorker->GetProcessModel()-
>m_ProcessList[mList.at(0).row()]);
    });
    connect(ui->tableView, &QTableView::customContextMenuRequested, this,
&CMainWindow::openContextMenu);
    m_pMainWorker = new CMainWorker;
    m_pThread = new QThread;
    m_pMainWorker->moveToThread(m_pThread);
    connect(m_pThread, &QThread::started, m_pMainWorker, &CMainWorker::doWork);
    connect(m_pMainWorker, &CMainWorker::finished, m_pThread, &QThread::quit);

```

```

        connect(m_pMainWorker, &CMainWorker::finished, m_pMainWorker,
&CMainWorker::deleteLater);
        connect(m_pThread, &QThread::finished, m_pThread, &QThread::deleteLater);
        connect(m_pMainWorker, &CMainWorker::updateProcCount, ui->label,
&QLabel::setText);
        connect(this, &CMainWindow::updateView, m_pMainWorker,
&CMainWorker::updateModelView);
        connect(ui->info_btn, &QPushButton::clicked, [=]
        {
            QMessageBox msg;
            msg.setWindowTitle("Info");
            msg.setText("Process Viewer\nRood' N. KNT-423\nOperational systems,
2016");
            msg.exec();
        });
        ui->tableView->setItemDelegate(&delegate);
        ui->tableView->setModel(m_pMainWorker->GetProcessModel());
        QItemSelectionModel *sm = ui->tableView->selectionModel();
        connect(sm, &QItemSelectionModel::currentRowChanged, m_pMainWorker,
&CMainWorker::setSelected);
        connect(ui->tableView, &QTableView::doubleClicked, [=](QModelIndex idx)
        {
            pw = new CPropertiesWindow(this, m_pMainWorker->GetProcessModel()-
>m_ProcessList[idx.row()]);
        });
        m_pThread->start();
        setAttribute(Qt::WA_DeleteOnClose);
        setWindowIcon(QIcon(":/icons/rsc/cogwheel.png"));
        show();
    }
    CMainWindow::~CMainWindow()
    {
        m_pMainWorker->stopThread();
        delete ui;
    }
    void CMainWindow::openContextMenu(QPoint pos)
    {
        conMenu.popup(ui->tableView->viewport()->mapToGlobal(pos));
    }
    void CMainWindow::killProcess()
    {
        QModelIndexList mList = ui->tableView->selectionModel()->selectedRows();
        TerminateProcess(m_pMainWorker->GetProcessModel()-
>m_ProcessList[mList.at(0).row()]->handle, STATUS_SUCCESS);
    }
    void CMainWindow::suspendResumeProcess()
    {
        QModelIndexList mList = ui->tableView->selectionModel()->selectedRows();
        if(m_pMainWorker->GetProcessModel()->m_ProcessList[mList.at(0).row()]-
>suspended)
        {
            pNtResumeProcess(m_pMainWorker->GetProcessModel()-
>m_ProcessList[mList.at(0).row()]->handle);
            m_pMainWorker->GetProcessModel()->m_ProcessList[mList.at(0).row()]-
>suspended = false;
            susp_res->setText("Suspend");
        }
    }

```

```

    }
    else
    {
        pNtSuspendProcess(m_pMainWorker->GetProcessModel()-
>m_ProcessList[mList.at(0).row()]->handle);
        m_pMainWorker->GetProcessModel()->m_ProcessList[mList.at(0).row()]-
>suspended = true;
        susp_res->setText("Resume");
    }
    emit updateView();
}
void CMainWindow::changePriority()
{
    QAction *act = qobject_cast<QAction *>(sender());
    DWORD priorityClass;
    if(!act->text().compare("High")) priorityClass = HIGH_PRIORITY_CLASS;
    if(!act->text().compare("Above normal")) priorityClass =
ABOVE_NORMAL_PRIORITY_CLASS;
    if(!act->text().compare("Normal")) priorityClass = NORMAL_PRIORITY_CLASS;
    if(!act->text().compare("Below normal")) priorityClass =
BELOW_NORMAL_PRIORITY_CLASS;
    if(!act->text().compare("Low")) priorityClass = IDLE_PRIORITY_CLASS;
    QModelIndexList mList = ui->tableView->selectionModel()->selectedRows();
    SetPriorityClass(m_pMainWorker->GetProcessModel()-
>m_ProcessList[mList.at(0).row()]->handle, priorityClass);
}
void CMainWindow::NtInit()
{
    HMODULE hNtdll = GetModuleHandleW(L"ntdll.dll");
    pNtQuerySysInfo = (tNtQuerySysInfo)GetProcAddress(hNtdll,
"NtQuerySystemInformation");
    pNtSuspendProcess = (tNtSuspendProcess)GetProcAddress(hNtdll,
"NtSuspendProcess");
    pNtResumeProcess = (tNtResumeProcess)GetProcAddress(hNtdll,
"NtResumeProcess");
    pNtQueryInformationProcess =
(tNtQueryInformationProcess)GetProcAddress(hNtdll,
"NtQueryInformationProcess");
}

```

ДОДАТОК Б
Копії слайдів презентації

ДИПЛОМНА РОБОТА

РОЗРОБКА ПРОГРАМИ МОНІТОРИНГУ ТА КЕРУВАННЯ ПРОЦЕСАМИ В ОПЕРАЦІЙНІЙ СИСТЕМІ WINDOWS

Розробила: ст. групи КНТ-217
Керівник: доцент, к.т.н.

Ануфрієва О.С.
Сердюк С.М.

Рисунок Б.1 – Слайд №1

Мета роботи

розробка програмного продукту, що забезпечує підвищення ефективності управління процесами в операційній системі (ОС) Windows

Об'єкт дослідження – методи і засоби моніторингу та управління процесами ОС Windows

Предмет дослідження – методи розробки програмного забезпечення для моніторингу та управління процесами в сімействі ОС Windows



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.2 – Слайд №2

Задачі, що вирішуються

- огляд внутрішнього устрою процесів та потоків;
- аналіз альтернативних програмних продуктів;
- розробка технічного завдання;
- вибір і обґрунтування мови програмування та середовища розробки;
- розробка загального алгоритму роботи програми;
- програмна реалізація застосунку;
- розробка необхідної документації.



НУ "ЗП", Кафедра програмних засобів

3

Рисунок Б.3 – Слайд №3

Стани потоків



НУ "ЗП", Кафедра програмних засобів

4

Рисунок Б.4 – Слайд №4

Результати аналізу альтернативних програмних продуктів

Функції	Task Manager	Process Explorer	Startr	PROWiSe Manager	AnVir Task Manager
Відображення процесів	+	+	+	+	+
Подання у вигляді дерева процесів	–	+	–	–	+
Список використовуваних бібліотек і служб	–	+	+	+	+
Установка пріоритетів	+	+	+	+	+
Завершення/призупинення процесу	+	+	+	+	+
Завершення/призупинення дерева процесів	–	+	–	–	+
Відображення шляху до файлу, що виконується	–	+	+	+	+
Відсоток завантаження процесора	+	+	–	+	+



НУ "ЗП", Кафедра програмних засобів

5

Рисунок Б.5 – Слайд №5

Рейтинг мов програмування (лютий 2021 р.)

Feb 2021	Feb 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	16.34%	-0.43%
2	1	▼	Java	11.29%	-6.07%
3	3		Python	10.86%	+1.52%
4	4		C++	6.88%	+0.71%
5	5		C#	4.44%	-1.48%
6	6		Visual Basic	4.33%	-1.53%
7	7		JavaScript	2.27%	+0.21%
8	8		PHP	1.75%	-0.27%
9	9		SQL	1.72%	+0.20%
10	12	▲	Assembly language	1.65%	+0.54%

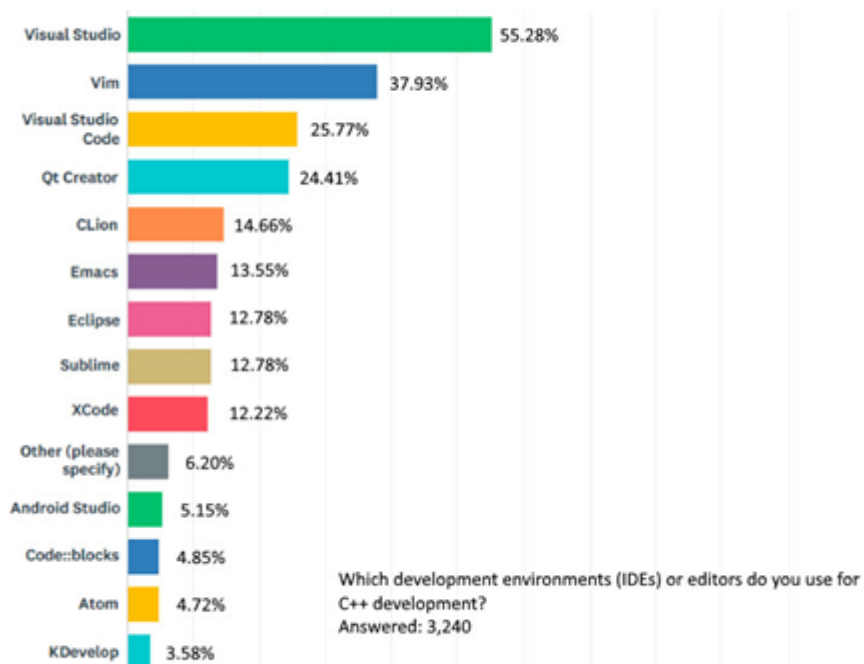


НУ "ЗП", Кафедра програмних засобів

6

Рисунок Б.6 – Слайд №6

Рейтинг середовищ розробки

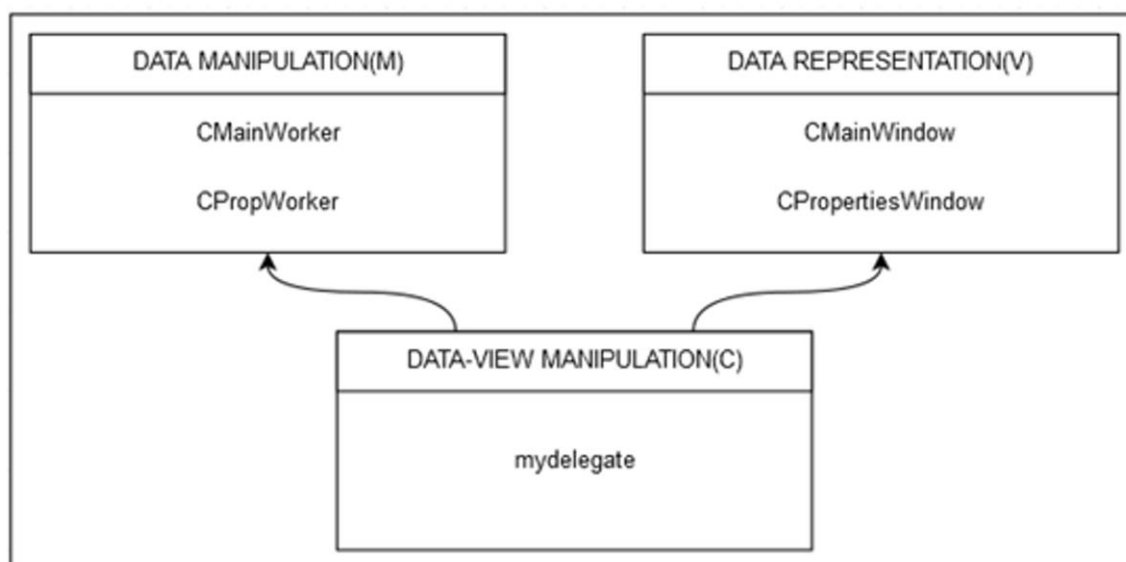


НУ "ЗП", Кафедра програмних засобів

7

Рисунок Б.7 – Слайд №7

Структура застосунку

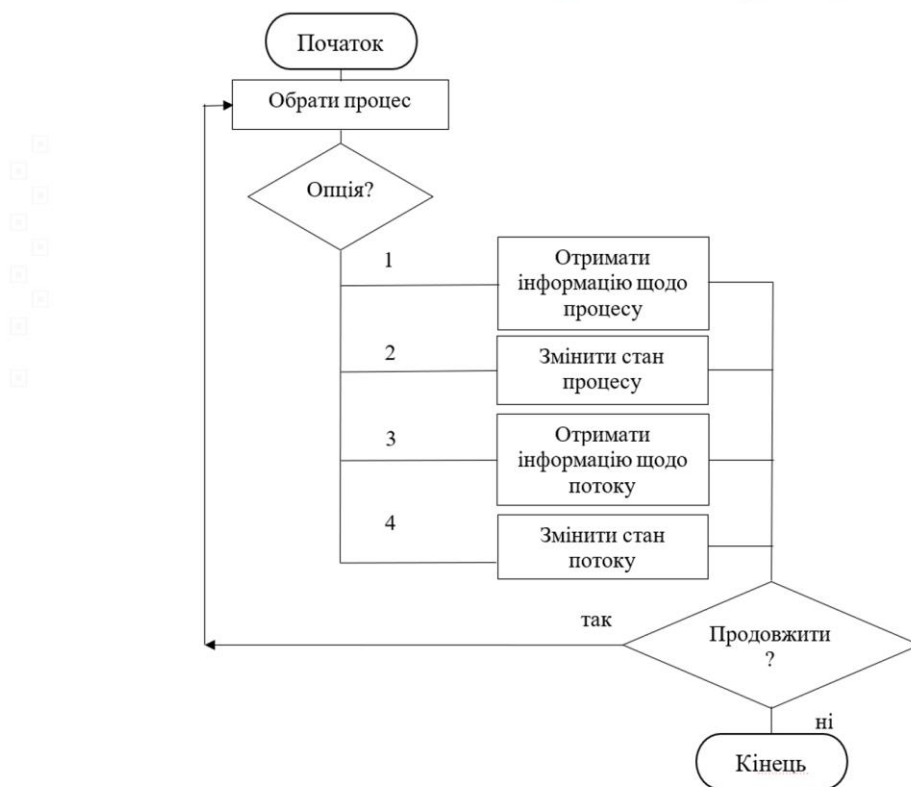


НУ "ЗП", Кафедра програмних засобів

8

Рисунок Б.8 – Слайд №8

Блок-схема алгоритму програми



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.9 – Слайд №9

9

Приклад розробленого класу

```

class CThread
{
public:
    CThread() : suspended(false) {}
    int id;//ID потоку
    HANDLE threadHandle;//Хендл потоку
    LONG priority;//Пріоритет LONG
    QString spriority, cpu_percents, startTime;//Пріоритет Qstring,
    процент використання процесору, час старту
    bool suspended;
    ULARGE_INTEGER lastCPU, lastSysCPU, lastUserCPU;
};
  
```

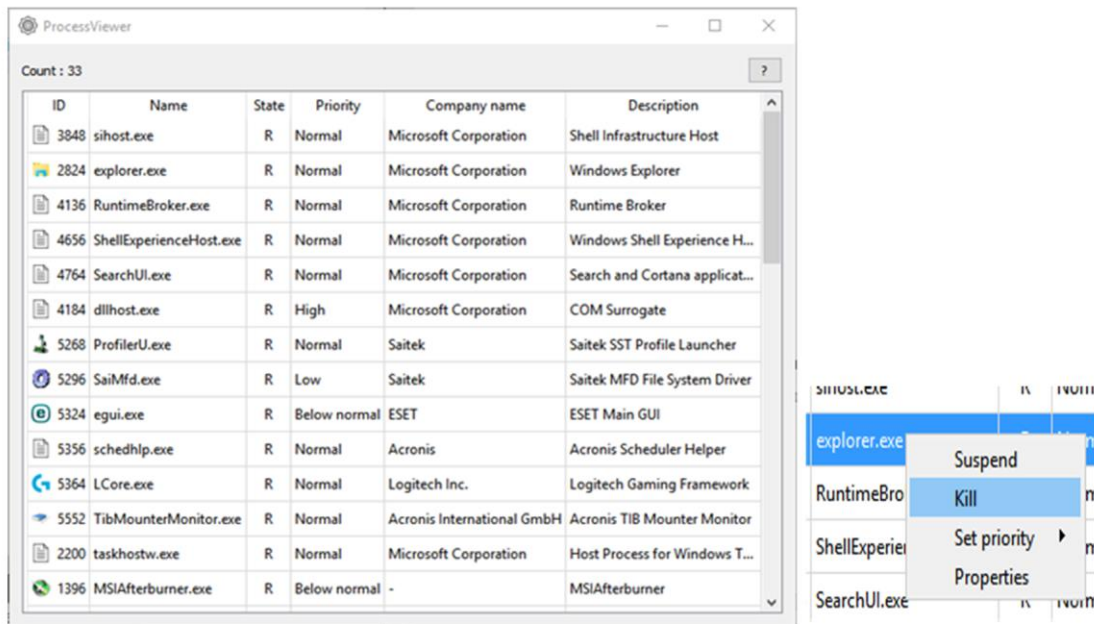


НУ "ЗП", Кафедра програмних засобів

Рисунок Б.10 – Слайд №10

10

Вікно активних процесів

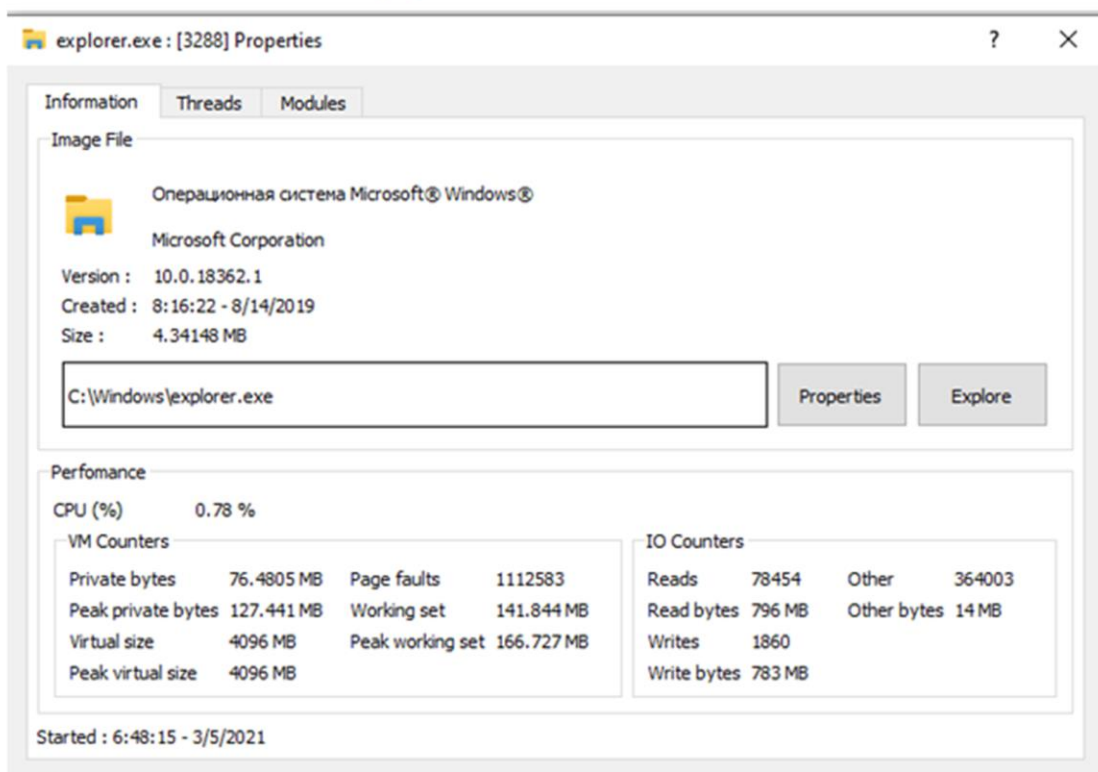


НУ "ЗП", Кафедра програмних засобів

11

Рисунок Б.11 – Слайд №11

Вкладка "Information"



НУ "ЗП", Кафедра програмних засобів

12

Рисунок Б.12 – Слайд №12

Вкладка “Threads”

explorer.exe : [3288] Properties

Information Threads Modules

Count : 69

ID	CPU	State	Priority	Started
3292	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
3640	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
3644	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
3660	< 0.01 %	R	Above normal	6:48:15 - 3/5/2021
3204	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
2936	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
3348	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
4000	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
3872	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
4108	< 0.01 %	R	Normal	6:48:15 - 3/5/2021
4168	< 0.01 %	R	Normal	6:48:16 - 3/5/2021



НУ “ЗП”, Кафедра програмних засобів

13

Рисунок Б.13 – Слайд №13

Вкладка “Modules”

Skype.exe : [6688] Properties

Information Threads Modules

Count :

Name	Description	Company name	Path
Skype.exe	Skype	Skype Technologies S.A.	C:\Program Files (x86)\Microsoft\Skype for Desktop\Skype.exe
ntdll.dll	Системная библиотека NT	Microsoft Corporation	C:\Windows\SYSTEM32\ntdll.dll
KERNEL32.DLL	Библиотека клиента Windows NT BASE API	Microsoft Corporation	C:\Windows\System32\KERNEL32.DLL
KERNELBASE.dll	Библиотека клиента Windows NT BASE API	Microsoft Corporation	C:\Windows\System32\KERNELBASE.dll
apphelp.dll	Клиентская библиотека совместимости приложений	Microsoft Corporation	C:\Windows\SYSTEM32\apphelp.dll
AcGenral.dll	Windows Compatibility DLL	Microsoft Corporation	C:\Windows\SYSTEM32\AcGenral.dll
msvcrt.dll	Windows NT CRT DLL	Microsoft Corporation	C:\Windows\System32\msvcrt.dll
sechost.dll	Host for SCM/SDDL/LSA Lookup APIs	Microsoft Corporation	C:\Windows\System32\sechost.dll
RPCRT4.dll	Библиотека удаленного вызова процедур	Microsoft Corporation	C:\Windows\System32\RPCRT4.dll
SspiCli.dll	Security Support Provider Interface	Microsoft Corporation	C:\Windows\System32\SspiCli.dll
CRYPTBASE.dll	Base cryptographic API DLL	Microsoft Corporation	C:\Windows\System32\CRYPTBASE.dll



НУ “ЗП”, Кафедра програмних засобів

14

Рисунок Б.14 – Слайд №14

Висновки

- проведено огляд внутрішнього устрою процесів та потоків;
- проведено аналіз альтернативних програмних продуктів;
- розроблено технічне завдання;
- обрані засоби розробки;
- розроблено загальний алгоритм роботи програми;
- розроблено програмне забезпечення;
- розроблено необхідну документацію.



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.15 – Слайд №15

15

**Доклад закінчено,
дякую за увагу!**



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.16 – Слайд №16

16