КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПАРАЛЕЛЬНОГО ЗАПУСКУ ТА МОНІТОРИНГУ ВИКОНАННЯ СУКУПНОСТІ ЗАДАЧ

Виконав студент 4-го курсу
Владислав ПЕТРІВ _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Тетяна КАРНАУХ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Юрій КРАК _____

(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 43 сторінки, 12 ілюстрацій, 15 джерел посилань, 1 додаток.

МОНІТОРИНГ, ВІКОННИЙ ЗАСТОСУНОК, ОДНОЧАСНЕ ВИКОНАННЯ, PYTHON, TKINTER, SUBPROCESS

Об’єктом роботи є одночасний запуск та відслідковування виконання програм. Предметом роботи є реалізація віконного застосунку за допомогою Python і бібліотек для нього.

Метою роботи є розроблення та програмна реалізація віконного застосунку для керування одночасним запуском та виконанням програм, а також виведення результатів запуску та інформації щодо використаних програмою ресурсів.

Інструменти розроблення: Python 3.8, TKinter, subprocess, threading, psutil.

Результати роботи: ознайомлення з функціоналом та TKinter та аспектами роботи subprocess, розроблено логіку реалізації віконного застосунку та виконано його базовий функціонал.

Розроблене програмне забезпечення може бути додатково змінено та використано навчальними закладами середньої та вищої освіти для полегшення процесу перевірки лабораторних робіт з програмістських дисциплін або олімпіадних завдань з програмування.

ЗМІСТ

С.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ЕОМ – електронно-обчислювальна машина

ОС – операційна система

ПЗ – програмне забезпечення

ЦП – центральний процесор

API – application programming interface, інтерфейс програмування застосунків

CLI – command-line interface, інтерфейс командного рядка

RAM – random access memory, пам’ять з довільним доступом

UI – user interface, користувацький інтерфейс

ВСТУП

Оцінка сучасного стану об’єкта розробки. У сучасному світі, в умовах тотальної багатозадачності та важливості менеджменту обчислювальних ресурсів, актуальною є задача одночасного запуску багатьох процесів та їх моніторингу. Вбудований в систему Windows диспетчер завдань дає можливість відслідковувати використані ресурси, однак він має свої недоліки в деяких моментах, а також не враховує особливості задачі, що розглядається в цій роботі.

Таким чином, хоча класичний диспетчер завдань Windows і є аналогом програми, створеної мною, все-таки він не є вузькоспрямованим на виконання поставлених задач.

Розроблена ж мною програма хоч і має значно менший у порівнянні з вищеописаним аналогом функціонал, забезпечує запуск, моніторинг та аналіз результатів запуску певних процесів, що дає змогу ефективно та зручно вирішувати поставлені задачі. Ще однією її особливістю є те, що за надмірної кількості задач, що запускаються майже одночасно, вона не запускає їх, натомість вона ставить їх у чергу і запускає по мірі відпрацювання або зняття з виконання. Останнє може бути вельми актуальним, бо за надмірної кількості одночасно запущених задач продуктивність може знижуватись [1].

Актуальність роботи. Як було зазначено вище, розроблене ПЗ має аналоги. Однак наявні аналоги у своїй більшості створені для виконання інших функцій та не заточені під поставлені мною цілі. Наприклад, диспетчер завдань Windows призначений більше для загального моніторингу роботи ОС та містить багато інформації, що не стосується дослідження. Таку як фонові процеси, що неможна вилучити, а також усі інші активні процеси, що не входять в список відстеження.

Таким чином, розробка спеціалізованого ПЗ дає можливість більш точно та зручно виконувати одночасний запуск та аналіз обраних програм без потреби

включати до дослідження сторонні процеси та програми, що його не стосуються. Розроблена мною програма дає на виході чисту вузькоспеціалізовану інформацію та статистику, що значно полегшує виконання поставлених задач.

Мета й завдання роботи. Мета кваліфікаційної роботи полягає у створенні віконного застосунку, який може запускати програми одночасно. Також застосунок надає інформацію про використання запущеними програмами часу виконання та використаної RAM у реальному часі або базуючись на історії попередніх викликів. Розроблюваний засіб має працювати на платформі Windows та запускати програми у форматах .exe та .py.

Для досягнення цієї мети поставлено такі завдання.

- Ознайомитись з паралелізмом на основі потоків у Python та аспектами його реалізації.
- Ознайомитись з керуванням підпроцесами за допомогою бібліотеки subprocess.
- Підібрати надійні та зручні засоби та технології для створення користувацького інтерфейсу.
- Розробити застосунок для запуску та моніторингу даних про сукупність програм.

Об’єкт, методи й засоби розроблення. Об'єктом розроблення є застосунок з віконним графічним інтерфейсом користувача, що підтримує можливості запуску та відстеження виконання програм, забезпечення можливості зняття програми з виконання та можливості моніторингу використаних ресурсів комп’ютера. Використовувалася ітераційна та інкрементна модель розроблення програмного забезпечення. В якості основного засобу розроблення було обрано мову програмування Python, версії 3.8, для якої використовувались стандартні та сторонні бібліотеки, а саме: psutil, threading, subprocess, tkinter, customtkinter, matplotlib.

Можливі сфери застосування. Розроблене програмне забезпечення може використовуватись для надання звітів щодо використання пам’яті та часових

затрат обраних програм. Може бути додатково змінено та використано, наприклад для полегшення процесу перевірки лабораторних робіт, які мають вимоги щодо використаних ресурсів. Наприклад, додавши можливість встановити обмеження (по часу та RAM) та функцію автоматичного призупинення виконання програми за умови перевищення встановлених лімітів, моя програма може значно скоротити час перевірки та частково автоматизувати перевірку лабораторних робіт. Також є можливість додати підтримку файлів інших форматів, наприклад .cpp, .java, .js та інші. Також можливо додати більше ресурсів для моніторингу, наприклад використання процесору, мережі та іншого.

1 ПРЕДМЕТНА ОБЛАСТЬ

1.1 Паралельні обчислення

Паралелізм[2] — це комплекс методів, алгоритмів, програм та апаратних засобів, які дозволяють виконувати завдання паралельно, одночасно використовуючи математичні, алгоритмічні, програмні та апаратні ресурси.

Паралельні обчислення — це вид обчислення, при якому одночасно використовуються кілька обчислювальних ресурсів ЕОМ для виконання певних завдань. Схему роботи можна побачити (рис.1.1)

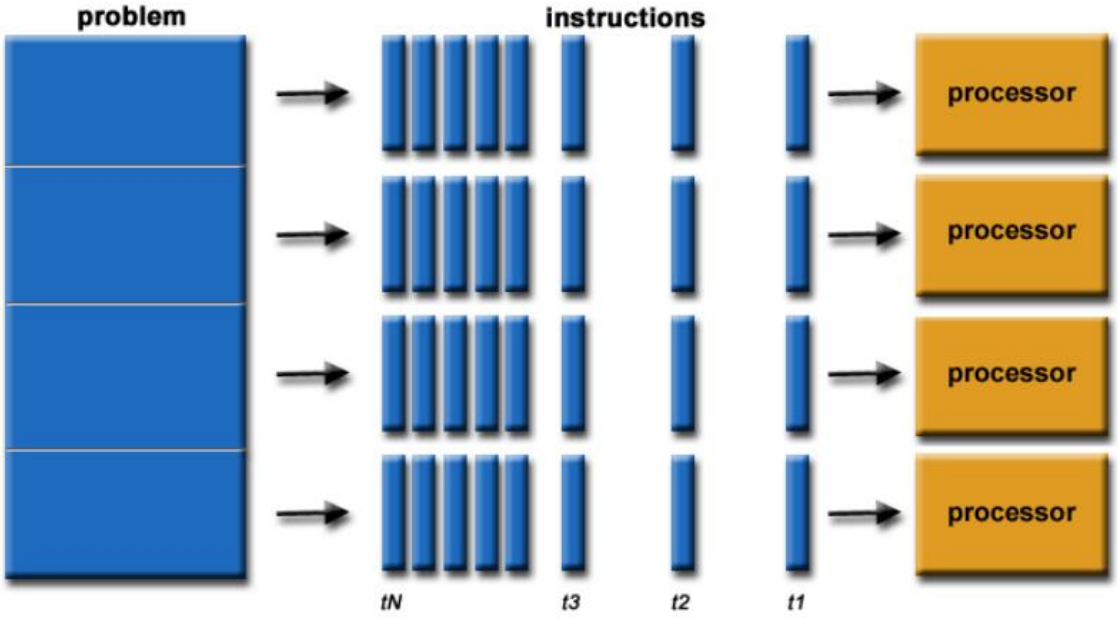

Рисунок 1.1 – Схема паралельних обчислень [3]

Паралельна обробка[4] — це метод обчислювальної техніки, при якому складне завдання розбивається на окремі частини, які виконуються одночасно на кількох центральних процесорах (ЦП). Таким чином, паралельна обробка дозволяє зменшити час виконання завдання шляхом розподілення його на багато паралельних процесів.

Розподіл та призначення окремих частин завдання різним процесорам, як правило, здійснюються комп'ютерними спеціалістами за допомогою програмних інструментів паралельної обробки. Ці інструменти також відповідають за збір і зчитування даних, коли кожен процесор завершує свою частину обчислень. Цей процес може відбуватися через комп'ютерну мережу (тобто ресурсами декількох комп'ютерів що працюють одночасно) або на комп'ютері з двома або більше процесорами.

Терміни паралельні обчислення та паралельна обробка часто використовуються як синоніми, оскільки ці процеси, як правило, відбуваються разом. Однак, відмінність полягає в тому, що паралельна обробка стосується використання багатьох ядр та центральних процесорів, які працюють паралельно в комп'ютері, тоді як паралельні обчислення стосуються оптимізації програмного забезпечення для ефективної роботи в таких умовах.

1.2 Асинхронність

Асинхронне програмування [5] — це підхід до розробки програм, при якому кілька завдань виконуються паралельно та незалежно одне від одного, без необхідності очікувати закінчення кожної операції перед переходом до наступної. На відміну від синхронного програмування, де кожна операція виконується послідовно і блокує роботу програми, асинхронне програмування дозволяє програмі продовжувати виконувати інші завдання, тим самим підвищуючи ефективність і стійкість до відмов.

Асинхронне програмування особливо важливе для сучасних додатків, які мають опрацьовувати великі обсяги даних та одночасно забезпечувати високу продуктивність.

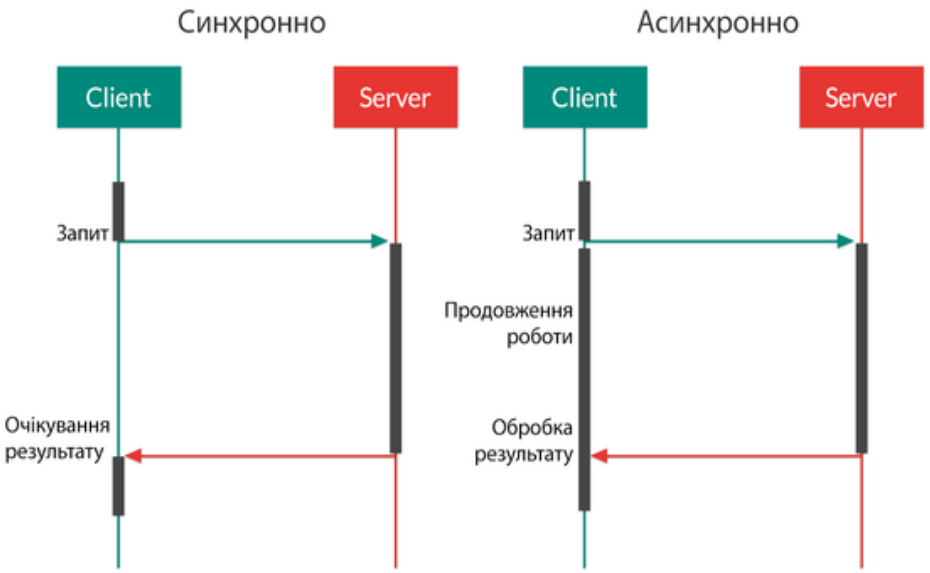

Рисунок 1.2 – Порівняння асинхронної і синхронної системи викликів [6]

У традиційному синхронному програмуванні, якщо одна операція вимагає багато часу, це блокує роботу всієї програми і робить її недоступною для користувачів. У випадку асинхронного програмування, тривалі операції можуть виконуватися у фоновому режимі, без блокування головного потоку програми, що забезпечує швидкість програми.

Наприклад, при завантаженні кількох файлів на сервер, синхронний код змушував би користувача дочекатись повного завантаження першого файлу перед ти як надати можливість завантажити наступний. Асинхронний код навпаки дозволяє почати завантаження наступного файлу без очікування на повне завантаження попереднього, що дозволяє ефективно виконувати завдання швидше.

Крім того, користувач має змогу продовжувати користуватися функціоналом програми без необхідності постійного очікування, що, безперечно, робить користувацький досвід більш дружнім.

1.3 Багатопотоковість

Багатопотоковість (multithreading)[7] — це концепція або властивість системи чи програми, що дозволяє виконувати завдання у кількох потоках одночасно, забезпечуючи цим більш ефективне використання наданих ресурсів.

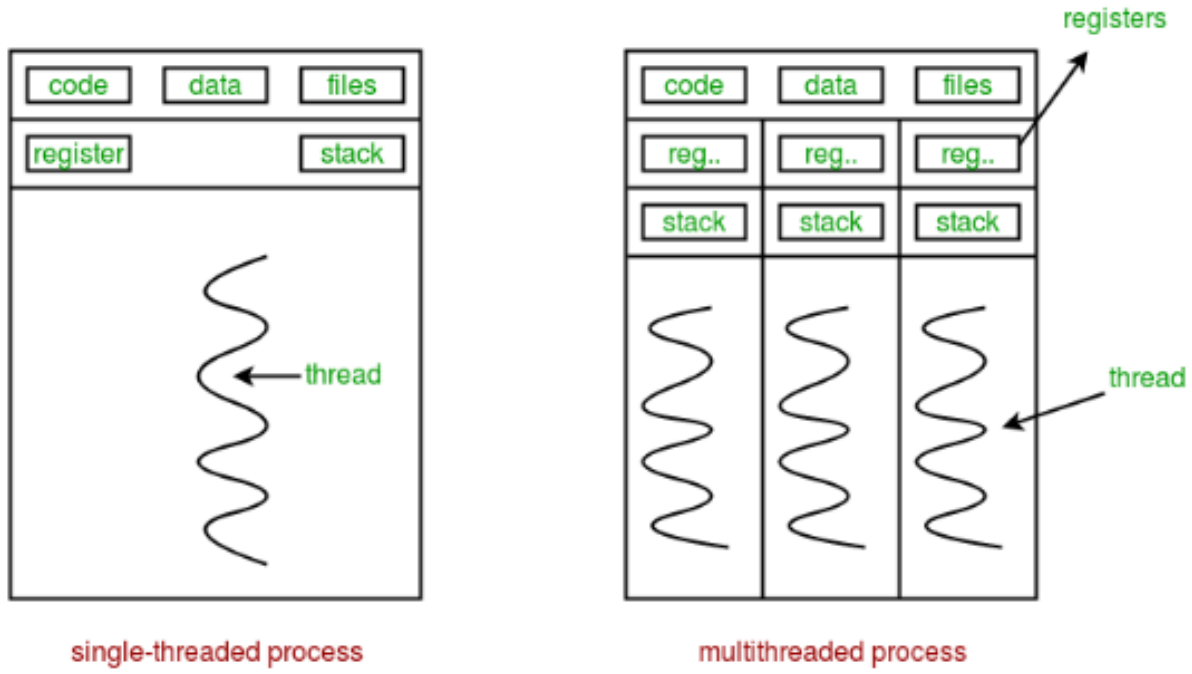

Рисунок 1.3 – Приклад роботи multithreading [8]

Потік — це незалежна послідовність інструкцій, яка виконується паралельно з іншими потоками в межах процесу .

Поняття потоку та процесу є достатньо близькими, однак, у той час як процеси являють собою те, що користувачами сприймається як окремі програми або додатки, потоки, у свою чергу, існують всередині процесів. Кожен процес може складатись з одного або більше потоків. Таким чином, наявність кількох потоків дозволяє розподіляти роботу всередині процесу для паралельного виконання.

Багатопотоковість дозволяє розділити завдання на декілька потоків, які можуть виконуватися паралельно. Кожен потік може мати свої власні

обчислення, стек викликів, змінні та інші атрибути. Ще багатопотоковість дозволяє досягти більшої ефективності та швидкодії програм, оскільки ресурси системи, такі як процесорний час тощо, можуть бути використані більш ефективно за рахунок паралельного виконання різних потоків.

Також, завдяки тому, що кілька потоків працюють всередині одного процесу, маючи при цьому спільний простір даних з головним потоком, забезпечується більш ефективна комунікація між потоками, порівняно з ситуацією, коли вони виконуються як незалежні процеси.

Використання багатопотоковості може бути особливо корисним у завданнях, які включають одночасну обробку різних операцій, взаємодію зі зовнішніми пристроями або роботу з багатопотоковими даними. Крім того, багатопотоковість може покращити відклик програми, забезпечуючи розподіл ресурсів між потоками та уникнення блокування під час виконання тривалих операцій.

Багатопотоковість є важливою концепцією сучасного програмування і дозволяє реалізувати паралельне виконання, підвищити продуктивність програм та покращити користувацький досвід.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Мова програмування Python

Для реалізації поставленою мною задачі я надав перевагу використанню мови програмування Python, версії 3.8. Python – це високорівнева інтерпретована мова програмування з об'єктно-орієнтованим підходом та строгою динамічною типізацією. Гвідо ван Россум розробив її в 1990 році. Python має високорівневі структури даних, властиву динамічну семантику та динамічне зв'язування, що робить її привабливою для швидкої розробки програм та поєднання наявних компонентів [9].

Мова Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні у скомпільованій та вихідній формі на всіх основних платформах. Python підтримує кілька парадигм програмування, таких як об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Безліч користувачів створюють різноманітні фреймворки (наприклад DJango, Flask, Fastapi) та бібліотеки (TensorFlow, opencv, та інші), що постійно збільшує кількість розробників на цій мові, завдяки зручності та простоті освоєння.

Але загалом, у моєму випадку, він був вибраний через наступний перелік переваг.

а) Простота використання: у Python доволі простий та зрозумілий синтаксис, що полегшує освоєння нових бібліотек та пришвидшує процес написання коду. Також процес встановлення цих бібліотеки максимально простий та зрозумілий за допомогою інструменту pip.

б) Кросплатформеність: мова Python підтримується на багатьох операційних системах, таких як Windows, macOS та Linux. Тобто, відкривається

можливість розробляти кросплатформенні програми та вони будуть працювати на різних платформах без змін.

в) Розширюваність: Python має величезну кількість бібліотек, які полегшують роботу програміста, надаючи готові рішення до безлічі стандартних завдань, допомагаючи фокусуватись лише на високорівневих задачах.

Але і крім переваг, Python має один аспект, який є його великою перевагою та гігантським недоліком у випадку поставленої мною задачі, а саме інтерпретатор.

Через цей фактор, завдяки глобальному блокуванню інтерпретатора (Global Interpreter Lock) [10], лише один потік може виконувати код Python одночасно (навіть якщо певні бібліотеки, орієнтовані на продуктивність, можуть подолати це обмеження). Однак відтворення задачі за допомогою потоків все ще є відповідною моделлю вирішення, якщо його використання призначено для одночасного запуску кількох завдань, пов’язаних із вводом-виводом.

Тобто, «справжньої» багатопотоковості у Python немає, але це не заважає створенню додатку для одночасного, чи паралельного запуску та моніторингу деякого переліку програм.

2.2 Бібліотеки subprocess, threading та psutil

Для реалізації основної логіки, поставленою мною задачі було обрано три бібліотеки: subprocess, threading та psutil.

Призначенням модуля subprocess є керування підпроцесами [11]. Він дозволяє створювати нові процеси, підключатись до їх потоків вводу, виводу та помилок.

Основним же підходом до виклику підпроцесів є використання функції run() або ж використання базового інтерфейсу Popen, що було використано у

моєму випадку.

Модуль threading призначений для побудови високорівневих потокових інтерфейсів [12]. У моему ж прикладному випадку він був використаний безпосередньо для одночасного запуску процесів за допомогою subprocess, надаючи для кожного з підпроцесів окремий «потік». Як було зазначено мною вище, завдяки Global Interpreter Lock, лише один потік може виконувати код, написаний на Python, але це не стає перепоною в одночасному запуску.

Головним же завданням модуля psutil є отримання інформації про запущені процеси та використання ресурсів системи (центральний процесор, пам’ять, диски, мережа) у Python [13]. Це корисно в основному для моніторингу системи, профілювання, обмеження ресурсів процесу та керування запущеними процесами.

Обраний мною він був якраз завдяки можливості відстежування пам’яті специфікованого процесу, у моєму випадку задля відстеження використання ресурсів кожним «потоком» з запущеним підпроцесом.

2.3 Графічний інтерфейс користувача за допомогою бібліотеки tkinter

Для того щоб мій застосунок не був реалізований лише завдяки CLI, я обирав спосіб, яким буду реалізовувати користувацький інтерфейс. Вибір для UI лежав між використанням WEBUI за допомогою використання фреймворку Flask [14] та віконним інтерфейсом використовуючи вбудований набір бібліотек «інтерфейс tk» або ж tkinter [15].

Причини вибору бібліотеки tkinter були такі.

а) Простота використання: tkinter є вбудованою бібліотекою Python, має доволі просту та зрозумілу документацію в порівнянні з Flask.

б) tkinter створений саме для реалізації графічного інтерфейсу користувача, на відміну від Flask, який є повноцінним API.

в) Незалежність від веб-сервера: tkinter працює саме на комп’ютері користувача, не вимагає мережевого з’єднання, що робить його дуже зручним інструментом для створення локальних застосунків.

Крім того, Flask виступає умовним «мостом» між логікою, написаною за допомогою Python, та користувацьким інтерфейсом, реалізованим через веб-сторінки, за допомогою, наприклад, html та css.

З урахуванням того факту, що розроблюване програмне забезпечення у першу чергу буде використовуватись для перевірки лабораторних робіт з програмування, причому в умовах, коли доступу до безоплатного для викладача серверного процесорного часу нема (фактично це означає, що і клієнтська, і серверна частина мають розгортатись на одному комп'ютері), побудова десктопного застосунку є більш прийнятною, оскільки матиме менше залежностей під час розгортання. Останнє теж є суттєвим, бо в бюджетних організаціях не завжди є можливість використовувати сучасні та потужні комп'ютери.

3 АРХІТЕКТУРА РОЗРОБЛЮВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Загальний погляд на розроблювану програму

Розроблюване програмне забезпечення складається з таких високорівневих компонентів.

а) Process – компонента яка повністю відповідає за логіку запуску та контролю за підпроцесами, контроль черги та інше.

б) App – компонента, яка відповідає за графічний інтерфейс користувача та його зв’язок з основною логікою програми.

Запуск та відслідковування задач, а також комунікація з операційною системою, забезпечується вищевказаними модулями, що містять глобальну змінну max_process_count, та класи Process й App, Tasks, Task, TaskInfo, TextInfo у модулях Process та App відповідно.

Глобальна змінна max_process_count необхідна для запобігання надмірної кількості викликів підпроцесів, оскільки при виклику завеликої кількості одночасно, продуктивність одночасного запуску буде тільки падати та програми будуть виконуватись повільніше ніж при меншій кількості.

Клас Process призначений виключно для зберігання всієї інформації про наявні процеси та їх чергу.

Клас App відповідає за створення графічного інтерфейсу користувача та його геометрію.

Клас Tasks відповідає за список задач, які запущені на даний момент, є в черзі, або були закінчені за теперішньої сесії програми.

Об’єкти класу Task є об’єктами, що наповнюють список Tasks, містять можливість викликати клас TaskInfo для детальної інформації про задачу.

Клас TaskInfo виконує роль інтерфейсу для детальної інформації про

процес, містить в собі дані про повний шлях до програми, що запускається, час її виконання, потоки виводу та помилок, графік використання пам’яті.

Клас TextInfo являє собою уособлення потоків виводу та помилок в об’єктах класу TaskInfo.

Для того щоб спростити створення інтерфейсу, мною була створена його базова схема (рис. 3.1):

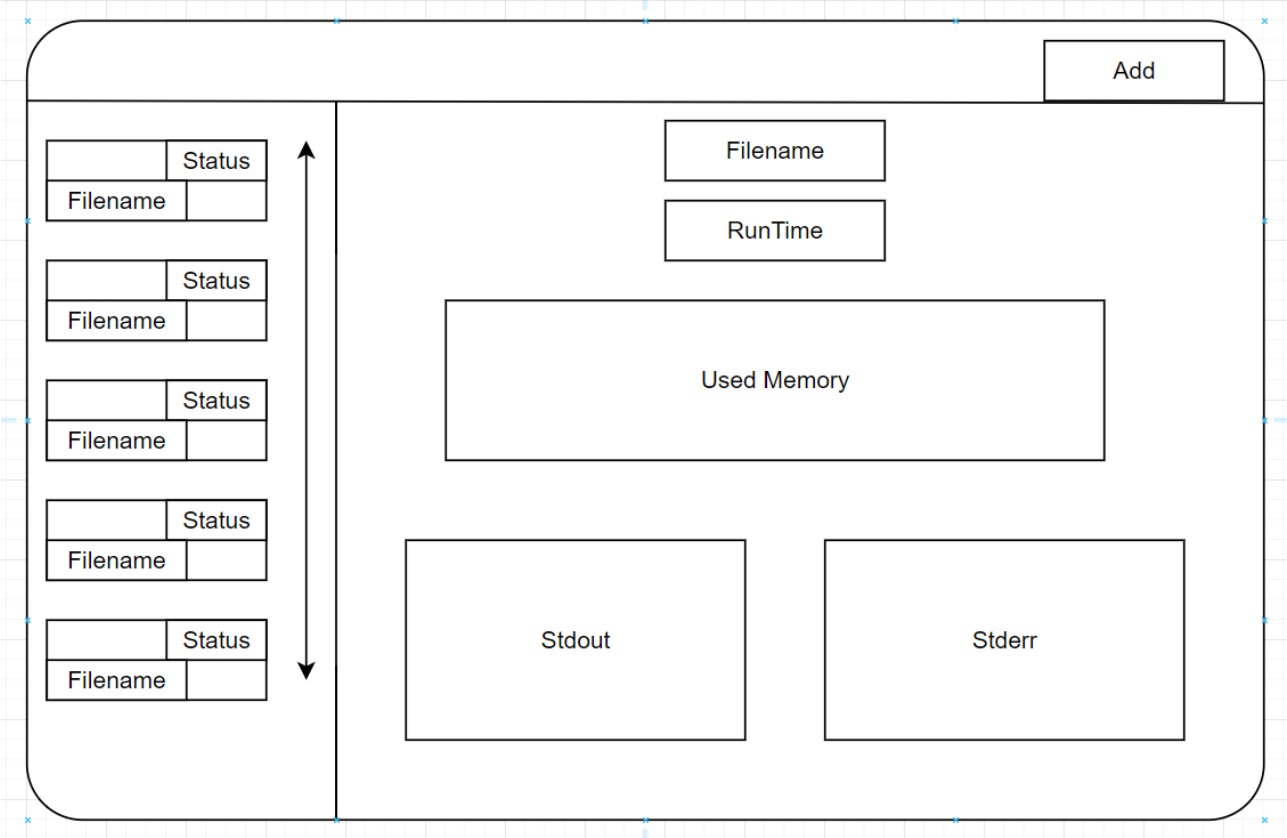

Рисунок 3.1 – Базова схема інтерфейсу

Інтерфейс має складатись з трьох основних речей: списку задач, з деталями про кожну з задач та кнопки для додавання нових завдань. При натисканні на будь-яку з задач у списку будуть виводитись її деталі, а саме повний шлях до файлу, кількість витраченого часу на виконання, графік з використанням пам’яті, потоки виводу та помилок, якщо програма їх повертає. Кнопка відкриватиме провідник ОС для вибору необхідного файлу для запуску.

3.2 Реалізовані прецеденти

Для кращої візуалізації роботи програми, мною була створена діаграма прецедентів, що описує її основний функціонал (рис. 3.2):

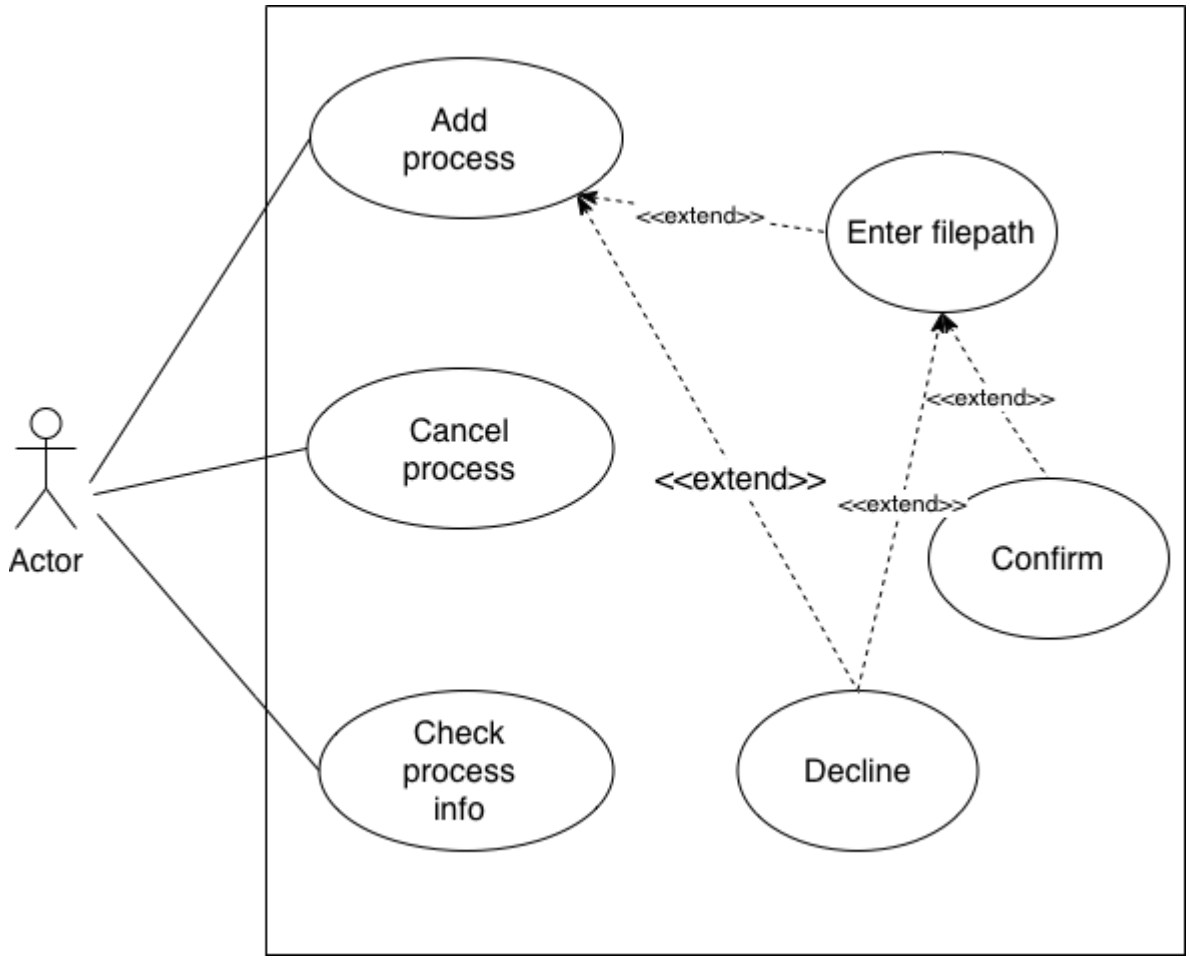

Рисунок 3.2 – Діаграма прецедентів майбутньої програми

Тут можна побачити, що основні можливі дії є такими.

а) Додати процес.

б) Зняти процес з виконання чи з черги на виконання.

в) Подивитись детальну інформацію про процес.

При додаванні процесу необхідно ввести шлях до виконуваного файлу і підтвердити додавання або ж на будь-якому з етапів відхилити додавання нового процесу.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Реалізація основної логіки програми

Для початку потрібно було реалізувати модуль, який буде відповідати за виконання процесів, запускаючи їх в різних потоках. Для цього я створив клас Process, з такими статичними полями:

```
class Process:
    __queue = []
    __process_count = 0
    __lock = Lock()
    change_in_queue_count = None
```

Спочатку при створенні об’єкту класу Process, викликається метод __new__ :

```
    def __new__(cls, *args, **kwargs):
        process = super(Process, cls).__new__(cls)
        if max_process_count > cls.__process_count:
            cls.__process_count += 1
        else:
            cls.__queue.append(process)
            if cls.change_in_queue_count is not None:
                cls.change_in_queue_count(len(cls.__queue))
        return process
```

Цей метод власне і створює об’єкт з параметрами, а потім передає його на ініціалізацію методу __init__.

При створенні об’єкту класу Process, ми перевіряємо кількість активних процесів на даний момент і порівнюємо його з максимально доступною кількістю, яка попередньо задається так:

```
max_process_count = max(1, multiprocessing.cpu_count() - 2)
```

Максимальна кількість одночасних процесів – це кількість ядр процесору мінус два.

Якщо ж кількість активних процесів більша за максимальну, виклик нового процесу додається в чергу, допоки не звільниться місце під новий. У випадку, коли кількість активних менша за максимальну, ми запускаємо новий і він є активним.

Метод __init__ виглядає так:

```
def __init__(self, path: str, task):
        if not path.endswith('.exe') and not path.endswith('.py'):
            raise Exception('File must be .exe or .py')

        self.update_chart = None
        self.stdout_write = None
        self.stderr_write = None
        self.set_run_time = None

        self.__path = path
        self.__task = task
        self.__time_start = 0

        self.__execute_time = 0
        self.__data = dict(stdout=list(), stderr=list())
        self.__ram_use = []

        self._process = None
        self.__is_running = False
```

З основного, він приймає шлях до файлу, що буде запущений та задачу. У ньому присутня перевірка формату файлу (за розширенням), щоб він був .exe або .py, в інших випадках видає помилку. Також в методі __init__ ініціалізуються

поля, які необхідні для наступних методів та визначаються назви функцій для класу Task.

Наступною функцією, яка була реалізована, є функція start(), ось її реалізація:

```
def start(self):
        if self in Process.__queue:
            return

        if self.__path.endswith('.exe'):
            self._process = Popen(self.__path, stdout=PIPE, stderr=PIPE)
        else:
            self._process = Popen(['python', self.__path], stdout=PIPE,
stderr=PIPE)
        self.__is_running = True
        psutil_process = psutil.Process(self._process.pid)
        self.__time_start = time.time()

        Thread(target=self.__reader, args=(self._process.stdout,
'stdout')).start()
        Thread(target=self.__reader, args=(self._process.stderr,
'stderr')).start()
        Thread(target=self.__get_ram_usage, args=(psutil_process,
)).start()
        self.__task.set_status('Running')
```

Ця функція запускає програму за переданим у них шляхом за допомогою бібліотеки subprocess, надає їй статус активної, починає відслідковувати час та використану RAM. Також ініціалізуються потоки для отримання потоків виводу та помилок, разом з відслідковуванням пам’яті.

Оскільки існує функція старту процесу, також створена і функція його призупинення:

```
def stop(self):
        if self._process is None:
            self.__queue.remove(self)
            self.__task.set_status('Canceled')
        else:
            self._process.kill()
```

Вона забирає процес з черги, якщо він не активний чи вбиває його, якщо активний.

Для відслідковування використання пам’яті було створене наступне:

```
def __get_ram_usage(self, psutil_process):
        while self._process.poll() is None:
            self.__ram_use.append(psutil_process.memory_info().rss /
(1024 * 1024))
            if self.update_chart is not None:
                try:
                    self.update_chart()
                except (Exception,):
                    pass
            self.__execute_time = time.time() - self.__time_start
            if self.set_run_time is not None:
                try:
                    self.set_run_time(self.__execute_time)
                except (Exception, ):
                    pass
            time.sleep(1)
```

Дана функція приймає процес та надає інформацію про використану RAM у мегабайтах щосекунди, заодно і оновлює графік RAM для UI.

У функції start() при ініціалізації потоків було використану функцію читача або ж слухача:

```
    def __reader(self, pipe, data):
            try:
```

```
            with pipe:
                for line in iter(pipe.readline, b''):
                    try:
                        (self.stdout_write if data == 'stdout' else
self.stderr_write)(line.decode())
                    except (Exception,):
                        pass
                    self.__data[data].append(line.decode())
        finally:
            with Process.__lock:
                if self.__is_running:
                                        Process.__process_count   =
max(Process.__process_count - 1, 0)
                    self.__is_running = False
                    self.__task.set_status('Stopped')
                             self.__execute_time  =  time.time()  -
self.__time_start
                    if self.set_run_time is not None:
                        try:
                            self.set_run_time(self.__execute_time)
                        except (Exception,):
                            pass
                    if Process.__queue:
                        Process.__queue.pop(0).start()
                       if Process.change_in_queue_count is not None:
                            Process.change_in_queue_count(len(Proce
ss.__queue))
```

Цей фрагмент коду намагається читати дані з каналу, а саме потоки виводу та помилок, додає їх в лист та викликає функцію яка виводить все на екран, щоб відображати дані в реальному часі.

Якщо ж він вже не може читати інформацію, то закриває потік, звітує про те, що програма завершилась, та робить додаткові перевірки.

Також для зручності до класу було додано такі чотири атрибути-властивості, або ж «properties»:

```
@property
def stdout(self):
    return self.__data['stdout']

@property
def stderr(self):
    return self.__data['stderr']

@property
def execute_time(self):
    return self.__execute_time

@property
def ram_use(self):
    return self.__ram_use
```

Це було зроблено, щоб мати доступ до важливої інформації про процес, не порушуючи принципу інкапсуляції.

4.2 Створення графічного інтерфейсу користувача

Почну з головного класу, який відповідає за вивід вікна з програмою на екран – клас App, при ініціалізації об’єкта якого можна буде викликати метод бібліотеки tkinter mainloop():

```
class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()
        self.title('App')
```

```
        self.geometry('800x600')
        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=2)
        self.grid_rowconfigure(1, weight=1)
                    customtkinter.CTkButton(self,    text='Add    +',
command=self.add_task).grid(row=0,   column=1,   padx=10,   pady=10,
sticky='ne')
   text_info  =  customtkinter.CTkLabel(self,  text='In  queue  0',
fg_color='white', corner_radius=6)
          text_info.grid(row=0,  column=0,  padx=(10,  0),  pady=10,
sticky='ew')

        Process.change_in_queue_count       =       lambda       x:
text_info.configure(text=f'In queue {x}')
```

Він наслідується від класу customtkinter.CTk, що є надбудовою для звичайного tkinter. У ньому визначається геометрія вікна, його назва, створюється поле для відображення черги та створюється кнопка, яка буде використовуватись для додавання нових задач.

Також кольорова тема визначається наступним чином:

```
customtkinter.set_appearance_mode('Light')
customtkinter.set_default_color_theme('blue')
```

На даному вікні має бути лист з повзунком, в якому буде відображатись інформація про запущені, закінчені чи заплановані задачі. Для цього і був реалізований клас Tasks:

```
class Tasks(customtkinter.CTkScrollableFrame):
    def __init__(self, master, title):
        super().__init__(master, label_text=title)
        self.grid_columnconfigure(0, weight=1)
        self.tasks = []
```

Щоб наповнювати цей список задачами, потрібно реалізувати клас задачі:

```
class Task(customtkinter.CTkFrame):
    __task_info = None

    def __init__(self, master, path):
        super().__init__(master)
        self.__path = path
        self.grid_columnconfigure(0, weight=1)
        customtkinter.CTkLabel(self,
text=os.path.basename(path)).grid(row=0, sticky='we')

        self.status = customtkinter.CTkLabel(self, text='Status: In
Queue')
        self.status.grid(row=1, sticky='we')

        self.__process = Process(path, self)

        buttons_frame = customtkinter.CTkFrame(self)
        buttons_frame.grid_columnconfigure(0, weight=1)
        buttons_frame.grid_columnconfigure(1, weight=1)
        customtkinter.CTkButton(buttons_frame, text='Info',
command=self.show_data).grid(row=0, column=0, sticky='nsew')
        customtkinter.CTkButton(buttons_frame, text='Cancel',
command=self.__process.stop, fg_color='#E53C3C', hover_color='#992E2E') \
            .grid(row=0, column=1, padx=(10, 0), sticky='nsew')

        buttons_frame.grid(row=2, sticky='we')

        self.__process.start()

    def set_status(self, test):
        self.status.configure(text='Status: ' + test)
```

У ньому маємо великий __init__ метод, в якому ініціалізується задача для виводу в список на екран, вона відображає статус активності, назву запущеного

файлу та кнопки «Cancel» та «Info», які зніматимуть процес з виконання та надаватимуть інформацію про процес відповідно.

Також необхідно було реалізувати клас, який відображає інформацію про процес та додати метод його виклику в класі Task, а саме TaskInfo:

```
class TaskInfo(customtkinter.CTkFrame):
    def __init__(self, master, path, process):
        super().__init__(master)
        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure(2, weight=1)
        self.grid_rowconfigure(3, weight=1)
        self._process = process
        customtkinter.CTkLabel(self, text=path, fg_color='white',
corner_radius=6).grid(row=0, column=0, padx=10, pady=10, sticky='ew')

        run_time = customtkinter.CTkLabel(self, text=f'Run time :
{process.execute_time:.3f} s', fg_color='white', corner_radius=6)
        run_time.grid(row=1, column=0, padx=10, pady=(0, 10),
sticky='ew')

        process.set_run_time = lambda x: run_time.configure(text=f'Run
time : {x:.3f} s')

        fig = Figure(figsize=(6, 4), dpi=100, facecolor='white')
        self._ax = fig.add_subplot()
        self._chart_canvas = FigureCanvasTkAgg(fig, self)
        self.show_chart()
        self._chart_canvas.get_tk_widget().grid(row=2, column=0, padx=10,
pady=(0, 10), sticky='nsew')
```

у даному вікні відображатиметься інформація про повний шлях до виконуваного файлу, час його виконання та графік використання пам’яті з полями про інформацію з потоків stdout, stderr. Приблизно таким чином це

виглядає (рис. 4.1):

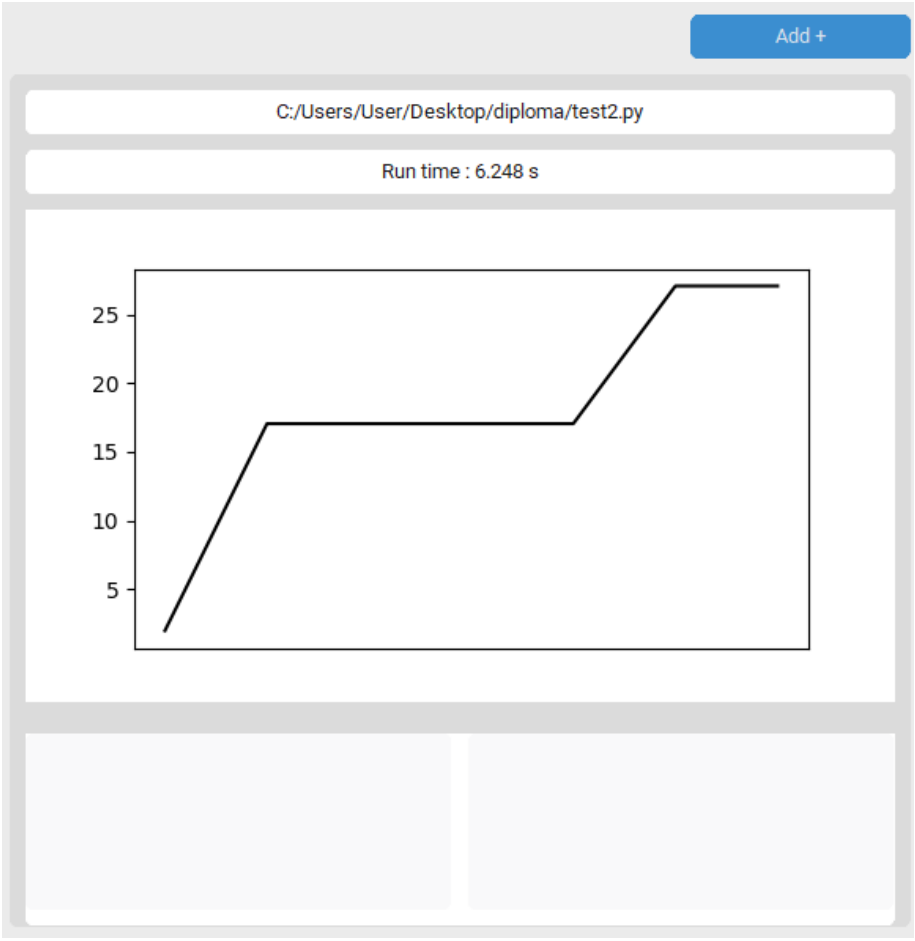

Рисунок 4.1 – Інтерфейс детальної інформації про запущену програму

Щоб відображати в реальному часі інформацію про потоки виводу та помилок, був реалізований клас TextInfo:

```
class TextInfo(customtkinter.CTkFrame):
    def __init__(self, master, process):
        super().__init__(master, fg_color='white')
        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=1)

        stdout = customtkinter.CTkTextbox(self)
        stderr = customtkinter.CTkTextbox(self)
        stdout.grid(row=0, column=0, padx=0, pady=(0, 10), sticky='nsew')
```

```
        stderr.grid(row=0, column=1, padx=(10, 0), pady=(0, 10),
sticky='nsew')

        for i in process.stdout:
            stdout.insert('end', i)

        for i in process.stderr:
            stderr.insert('end', i)

        process.stdout_write = lambda x: stdout.insert('end', x)
        process.stderr_write = lambda x: stderr.insert('end', x)
```

4.3 Зв’язування графічного інтерфейсу користувача з основною логікою

Після написання коду для створення основної логіки та візуальної частини, його необхідно зв’язати між собою. Почнемо зі зворотнього, по мірі наростання.

Додамо до класу __init__ методу класу TaskInfo наступне:

```
process.update_chart = self.show_chart
text_frame = TextInfo(self, process)
text_frame.grid(row=3, column=0, padx=10, pady=(10, 0), sticky="nsew")
```

Для виведення графіку та інформації про потоки вводу і помилок спочатку створімо метод для відображення графіку:

```
def show_chart(self):
   self._ax.clear()
   self._ax.plot(range(len(self._process.ram_use)),
                 self._process.ram_use, color='black')
   self._ax.set_facecolor('white')
   self._ax.set_xticks([])
```

```
   self._ax.spines[:].set_color('black')
   self._ax.tick_params(axis='y', colors='black')
   self._chart_canvas.draw()
```

Наразі відображення детальної інформації працює коректно. Наступним кроком буде надати можливість відображати це за натисканням кнопки «Info» на об’єкті Task:

```
def show_data(self):
    if self.__task_info is not None:
       self.__task_info.destroy()
   self.__task_info = TaskInfo(self.master.master.master.master,
                               self.__path, self.__process)
    self.__task_info.grid(row=1, column=1, padx=10, pady=(0, 10),
sticky='nsew')
```

Цей метод власне і буде викликати відображення детальної інформації.

Далі йде створення методу, що додаватиме задачу в список і в подальшому викликатиметься кнопкою на головному інтерфейсі, він доданий у класі Tasks:

```
def add(self, path):
    task = Task(self, path=path)
    task.grid(row=len(self.tasks), column=0, padx=10, pady=(10, 0),
              sticky='we')
    self.tasks.append(task)
   return task
```

Останнім кроком підв’язки логіки до графічної частини буде додавання події на натискання кнопки «Add+» на головному меню, це код доданий у клас App:

```
def add_task(self):
   if path := fd.askopenfilename(title='Executable file', initialdir='/',
                         filetypes=(('Executable file', '*.exe;*.py'),)):
       self.tasks.add(path)
```

За допомогою цього можна вибрати файл за допомогою провідника Windows з специфікацію, якого формату мають бути файли.

5 ЕКСПЛУАТАЦІЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Перевірка працездатності розробленого програмного забезпечення

Запуск програми відбувається за допомогою наступного коду:

```
from app import App

app = App()
app.mainloop()
```

Отримаємо наступний зовнішній вигляд нашого віконного застосунку (рис. 5.1):

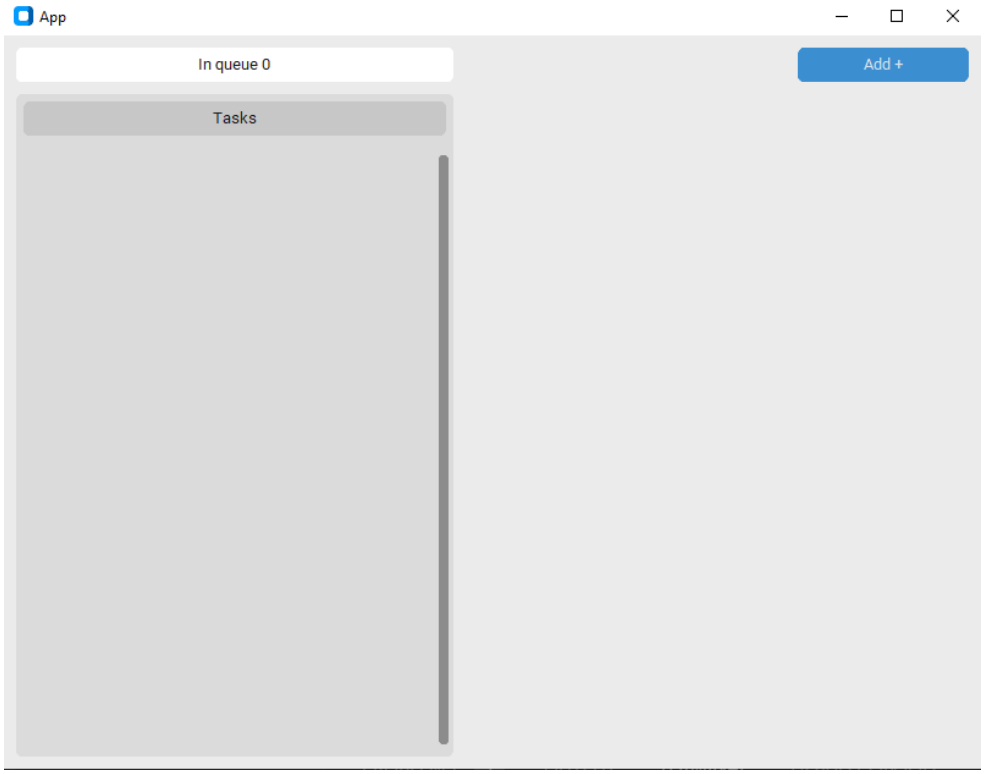

Рисунок 5.1 – Головний екран готового застосунку

При натисканні на кнопку «Add+», на екрані з’явиться провідник, для вибору файлу для запуску, я відкрив папку System32, щоб відкрити MSPaint (рис. 5.2).

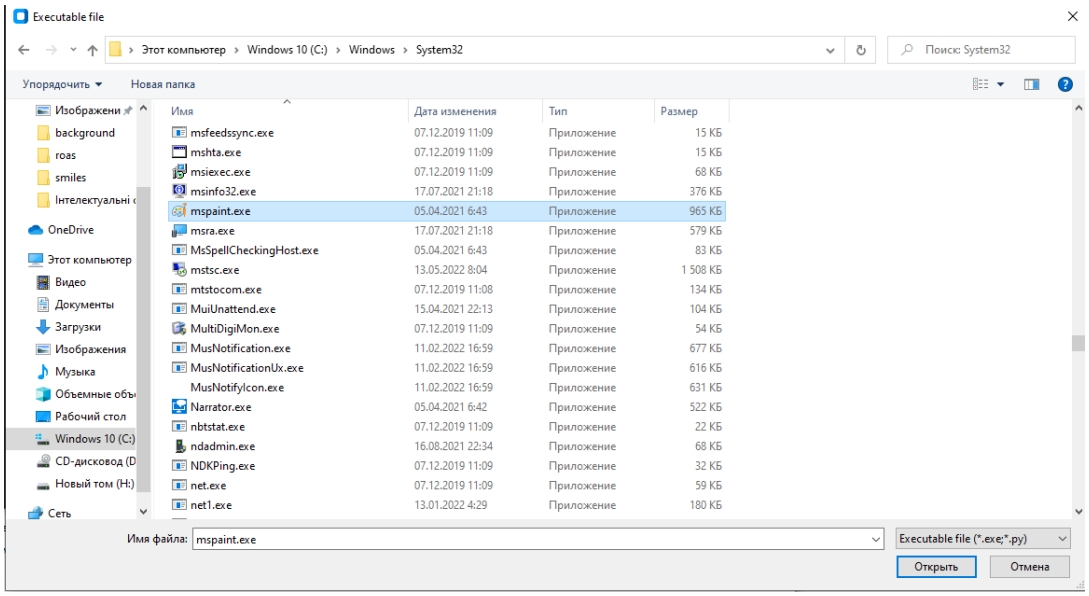

Рисунок 5.2 – Вибір файлу для запуску

Програма успішно запустилась і відображається в інтерфейсі мого додатку разом з детальною інформацію про процес (рис. 5.3):

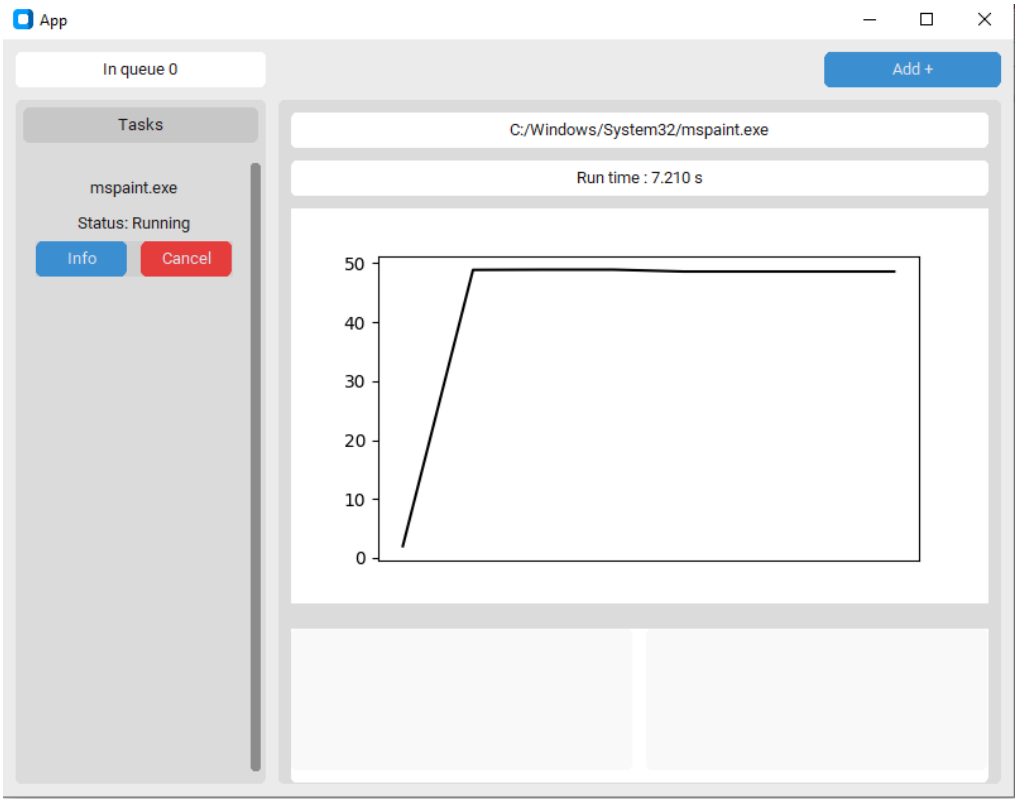

Рисунок 5.3 – Результат запуску першої програми

Отриманий графік показує в реальному часі яку кількість RAM використовує процес.

Наступним кроком тестування буде запуск більшої кількості програм. Мій процесор має 8 ядр, то ж ліміт буде на 6 одночасних завдань, згідно з обмеженнями у коді. Запустивши одночасно 7 програм, 6 активні, а одна в черзі, як і очікувалось (рис. 5.4).

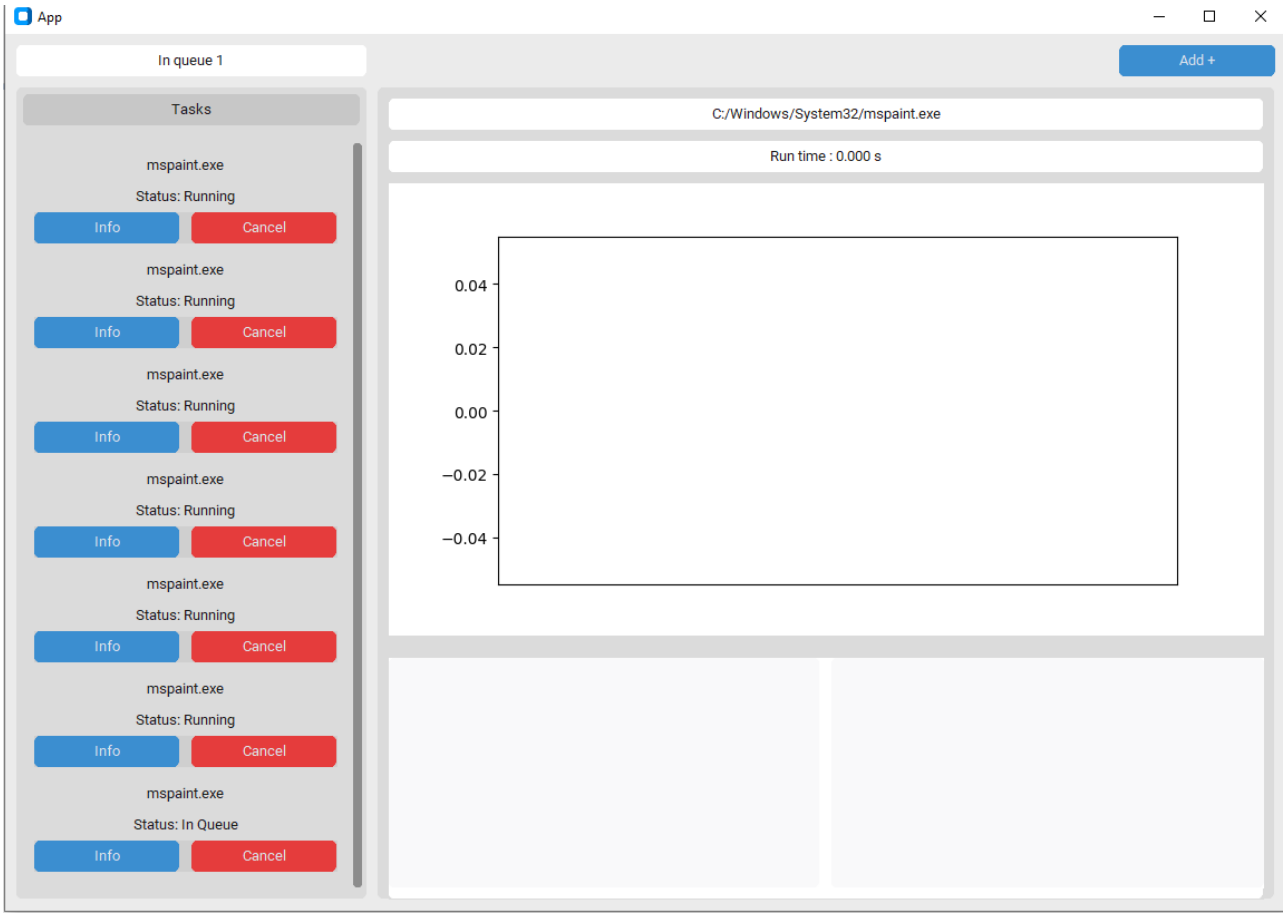

Рисунок 5.4 – Запуск низки програм

Якщо зняти будь яку програму з виконання, то перша з черги має стати активною, що і відбулось (рис. 5.5).

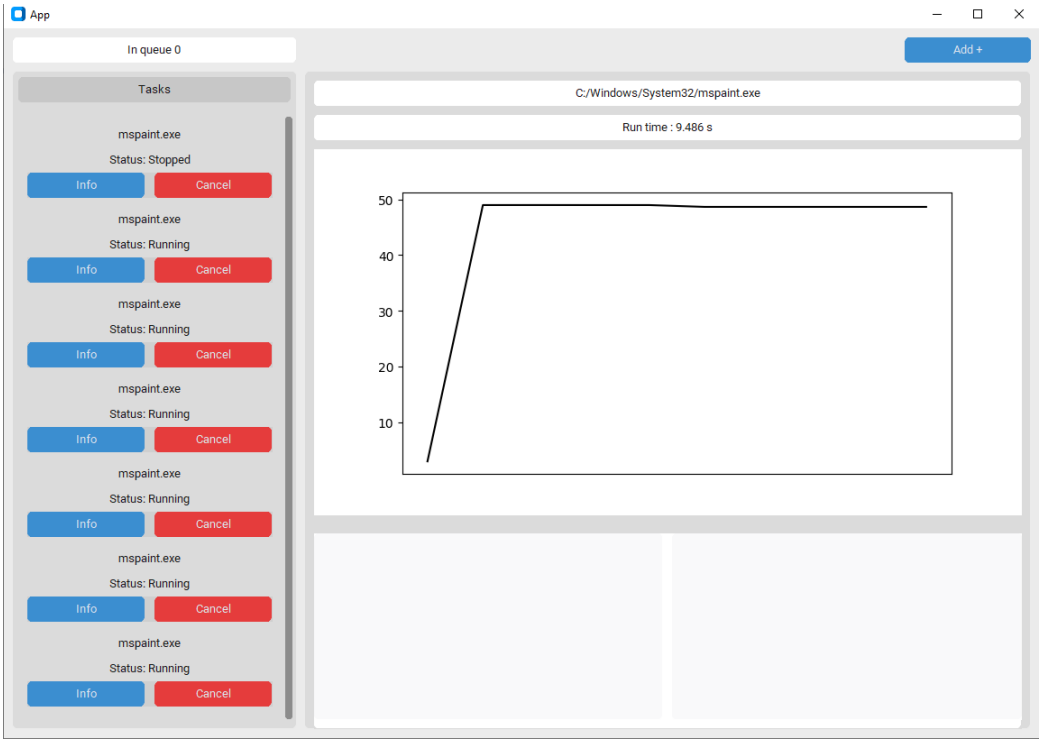

Рисунок 5.5 – Результат зняття процесу з черги

Останнім етапом тестування буде спроба виконання програм на Python, з перехопленням потоків виводу та помилок. Для цього була написана програма, яка вказана в додатку А. Результат її роботи наведено на рис. 5.6.

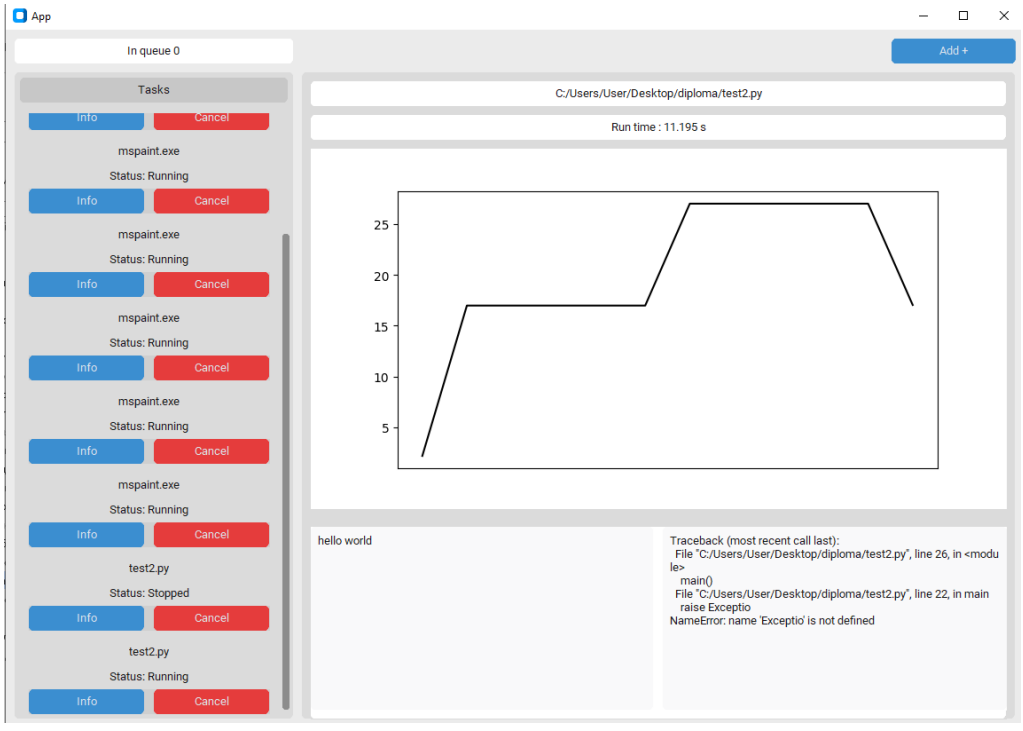

Рисунок 5.6 – Результат виклику файлу з програмою мовою Python

Можна побачити стрибки використання пам’яті під час ініціалізації масиву та сортування, так само як і під час перехоплення потоків stdout та stderr.

5.2 Програмно-апаратні вимоги розгортання

Для коректної роботи програми на комп’ютері має бути наступне:

а) Python – версія 3.8.

б) Комп’ютер з більше ніж 2 ядрами процесору.

А також такі коректно встановлені бібліотеки (зазначено потрібні версії):

```
customtkinter==5.1.3
matplotlib==3.7.1
psutil==5.9.5
```

Загально відомий факт, що Python поки доволі мінливий, особливо в частині, що підтримує паралелізм. Тому перед використанням вищих версій Python необхідно додатково перевірити працездатність коду програми в умовах роботи з іншою версією інтерпретатора та його бібліотек. За потреби, адаптувати код до вищих версій, використовуючи документацію на використовувані версії. Зазвичай, розробники завжди зазначають, що саме змінюється в новій версії.

5.3 Інструкція користувача

а) Запуск програми проходить через файл main.py.

б) Для додавання нової програми на запуск потрібно натиснути кнопку «Add+», після чого обрати її через провідник на вашому комп’ютері.

в) Програми для запуску можуть бути тільки двох форматів .exe та .py.

г) Одночасно можна запустити кількість програм, що дорівнюють кількості ядр вашого процесору мінус два.

д) Для перегляду детальної інформації про хід виконання обраної програми, необхідно перейти в розділ «Info» натиснувши відповідну кнопку під обраним процесом у списку.

е) Для зняття процесу з виконання чи з черги потрібно натиснути кнопку «Cancel».

ВИСНОВКИ

У цій роботі ресурсами мови програмування Python та її сторонніх бібліотек було розроблено застосунок для операційної системи Windows, який забезпечує одночасний запуск декількох задач та відслідковує їх виконання.

Застосунок реалізовано успішно, працездатність перевірено, виконуючи одночасний запуск декількох програм в операційній системі Windows.

Серед основних переваг розробленого застосунку можна зазначити такі.

а) Підвищена продуктивність. Завдяки паралельному виконанню програм, збільшується загальна продуктивність системи.

б) Оптимальне використання ресурсів. Застосунок ефективно розподіляє завдання, що дозволяє оптимально використовувати ресурси системи.

До недоліків можна віднести основні аспекти будь-яких паралельних процесів.

а) Синхронізація та конфлікти. Необхідно ретельно управляти синхронізацією та уникати конфліктів між програмами, які виконуються паралельно.

б) Залежність від ресурсів. Для оптимального функціонування застосунку потрібні системи з багатьма ядрами процесору.

в) Особливості реалізації в мові Python. У мові програмування Python немає як такої багатопотоковості завдяки Global Interpreter Lock, що дозволяє запускати Python код лише у одному потоці, але з можливістю симуляції багатопотоковості.

Застосунок дозволяє ефективно використовувати ресурси багатоядерних процесорів в операційній системі Windows.

Порівняно з послідовним виконанням програм, паралельне виконання забезпечує значне збільшення продуктивності та зменшення часу виконання

завдань.

Розроблена програма також може бути в подальшому покращена та використана для більш спеціалізованих потреб. Наведу в приклад декілька можливих перспектив:

а) Розширення функціональності. Додати можливість налаштування пріоритетів виконання програм та додаткових параметрів. Також, при додаванні можливості встановлення обмежень на використану процесом пам’ять та/або час програму можна використовувати для перевірки ефективності роботи лабораторних робіт.

б) Підтримка інших операційних систем. Розробити версії застосунку для інших операційних систем, таких як macOS або Linux.

в) Інтеграція з іншими інструментами. Розглянути можливість інтеграції з іншими інструментами або бібліотеками, що сприяють паралельному обчисленню.

У цілому, розроблений застосунок дозволяє ефективно використовувати ресурси багатоядерних процесорів, забезпечуючи паралельне виконання декількох програм. Цей проект має потенціал для подальшого розвитку та може знайти застосування в областях, де потрібна висока продуктивність та оптимальне використання ресурсів системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Multitasking - Win32 apps. Microsoft Learn: Build skills that open doors in your career. Режим доступу до ресурсу: https://learn.microsoft.com/en-us/windows/win32/procthread/multitasking
2. Almasi, G.S. and A. Gottlieb (1989). Highly Parallel Computing. Benjamin-Cummings publishers, Redwood City, CA
3. Технології розподілених систем та паралельних обчислень – Режим доступу до лекцій: https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/16315/1/%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%97%20%D1%80%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB%D0%B5%D0%BD%D0%B8%D1%85%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%20%D1%82%D0%B0%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%B8%D1%85%20%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D1%8C.%20%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%97.pdf
4. What is Parallel Computing? Definition and FAQs | HEAVY.AI. The Fastest Analytics and Location Intelligence Platform | HEAVY.AI. Режим доступу до ресурсу: https://www.heavy.ai/technical-glossary/parallel-computing
5. Асинхронне програмування: види, класифікація, принципи програмування, концепція, значення і застосування. What – маємо відповідь. Режим доступу до ресурсу: https://what.com.ua/asinhronne-programyvannia-vid/
6. Асинхронне програмування: що це таке?. FoxmindEd. Режим доступу до ресурсу: https://foxminded.ua/asynkhronne-prohramuvannia/
7. Matan Rabi. Sync, Async, multi-threading and multi-processing programming techniques. Режим доступу до статті: https://www.linkedin.com/pulse/sync-

async-multi-threading-multi-processing-programming-matan-rabi/

8. Multithreading in Python | Set 1 - GeeksforGeeks. GeeksforGeeks. Режим доступу до ресурсу: https://www.geeksforgeeks.org/multithreading-python-set-1/
9. Guido van Rossum, Python Reference Manual, release 2.4.4, 18 October 2006.
10. Real Python. What Is the Python Global Interpreter Lock (GIL)? – Real Python. Python Tutorials – Real Python. Режим доступу до ресурсу: https://realpython.com/python-gil/
11. subprocess. Python documentation. Режим доступу до ресурсу: https://docs.python.org/uk/3/library/subprocess.html
12. threading. Python documentation. Режим доступу до ресурсу: https://docs.python.org/uk/3/library/threading.html
13. psutil documentation – psutil 5.9.5 documentation. psutil documentation – psutil 5.9.5 documentation. Режим доступу до ресурсу: https://psutil.readthedocs.io/en/latest/
14. Welcome to Flask – Flask Documentation (2.3.x). Welcome to Flask – Flask Documentation (2.3.x). URL: https://flask.palletsprojects.com/en/2.3.x/
15. tkinter. Python interface to Tcl/Tk. Python documentation. Режим доступу до ресурсу: https://docs.python.org/3/library/tkinter.html

ДОДАТОК А
ПРОГРАМА ДЛЯ ТЕСТУВАННЯ ЗАСТОСУНКУ

```
import random
import time

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):

        for j in range(0, n-i-1):

            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

array = [random.randint(1, 100) for _ in range(10000)]
def main():

    time.sleep(5)
    data = [0] * (10 * 1024 * 1024 // 8)
    bubble_sort(array)

    print("hello world")

    raise Exceptio


if __name__ == "__main__":
    main()
```