ПРАКТИЧНА РОБОТА 12
Асинхронне програмування мовою Python
Система оцінювання

№	Тема	К-ть балів
1.	Задача 1	0,6
2.	Задача 2	1
3.	Задача 3	1
4.	Задача 4	1,2
5.	Задача 5	1,2
6.	Задача 6	1
	Всього за практичну роботу	**6**
7.	ІНДЗ-1	1,5
8.	ІНДЗ-2	1,5
9.	ІНДЗ-3	1
	Всього	**10**

1. ***(Створення потоків)*** Напишіть програму, яка створить два потоки, кожен з яких спатиме випадкову кількість секунд, а потім буде виводити повідомлення, яке включатиме ID потоку.

2. ***(Гонитва даних)*** Спробуйте написати програму, яка продемонструє гонитву даних. Нехай на рахунку подружжя буде 100000 грн. Після гучної сварки чоловік і дружина вирішили зняти гроші зі спільного рахунку, проте для сум понад 10000 грн. іншому члену подружжя приходить сповіщення про це. Тому кожен з них вирішив знімати по 10000 грн, щоб інша половинка не знала, що рахунок спустошується.
   Напишіть клас, що описуватиме банківський рахунок - у першу чергу, баланс, операцію знімання та вкладення грошей. Покладіть на цей рахунок 100000 грн. та за допомогою двох потоків зімітуйте знімання по 10000 грн. з випадковим засинанням потоку на період до 1 секунди (Інтернет у подружжя не дуже хороший). Наприкінці виведіть рахунок подружжя. Несинхронізована версія програми може зняти більше грошей, ніж було на рахунку – спробуйте це продемонструвати. Якщо така подія сталась, виправте код з метою усунення гонитви даних.

3. ***(Блокування)*** Напишіть клас, який буде стягувати (fetch) код веб-сторінки та записувати його у файл. Клас повинен забезпечити створення окремого потоку для стягування та запису даних зі списку сайтів. Сформуйте кілька списків веб-сайтів, які будуть стягуватись та сконструюйте відповідні об’єкти екземпляри класу. Наприклад,
   urls1 = ['http://www.google.com', 'http://www.facebook.com']
   urls2 = ['http://www.yahoo.com', 'http://www.youtube.com']

   Приклад виводу може бути наступним

```
write done by Thread-68
URL http://www.facebook.com fetched by Thread-68
write done by Thread-68
URL http://www.google.com fetched by Thread-68
write done by Thread-69
URL http://www.youtube.com fetched by Thread-69
write done by Thread-69
URL http://www.yahoo.com fetched by Thread-69
```

Слід відмітити, що порядок стягування даних не відповідає порядку появи сайтів у списках. Тому слід виконати синхронізацію роботи потоків.
Напишіть багатопоточні версії програми, які будуть використовувати (1) блокування (locks), (2) блокування з повторним входом (rlocks).

4. ***(Семафори)*** Створіть програму, яка буде виконувати «обробку» інформації з сайтів. Один потік займатиметься власне «обробкою» - спатиме протягом випадкової кількості секунд (від 5 до 15), а інший – це потік-демон, який буде перебирати сайти із введеного списку (за Вашим бажанням), проте здобуватиме прив'язаний семафор, щоб обмежити «обробку» лише для двох сайтів одночасно. У процесі роботи програми виводьте інформацію стосовно того, який сайт здобув/звільнив семафор та скільки секунд триватиме «обробка».

5. ***(Condition-об'єкти)*** Уявіть, що пишете відеоплеєр, який відтворюватиме потокове відео. Тут буть задіяні мінімум два потоки: один буде буферизувати відео для подальшого відтворення, а інший – власне відтворювати відеоряд. Основна проблема синхронізації – плеєр не може відтворювати небуферизоване відео.
Створіть програму з двома потоками. Один потік буде «буферизувати» відео – генеруватиме випадкове число від 1 до 5 (відсоток буферизації) та спатиме 1 секунду. Як тільки у буфері накопичилось ще 10% відео, можна розпочати «відтворення» (в іншому потоці, який спить 3 секунди). Виводьте повідомлення щодо всіх подій у програмі: новий відсоток при буферизації, спрацювання умови, запуск та завершення роботи потоків.

6. ***(Екзекутори та пули потоків)*** Перед Вами поставлено задачу перевірити досяжність хостів у мережі шляхом пінгування їх IP-адрес. Пропонується функція ping(), яка повертає True, якщо хост буде досяжним (reachable) та працюючим (alive) або False, як трапилась помилка доступу (неправильна назва хосту, недосяжність по мережі, таймаут пінгу тощо).

```
import subprocess

def ping(hostname):
    p = subprocess.Popen(["ping", "-c", "3", "-w", "1", hostname],
                         stdout=subprocess.DEVNULL,
                         stderr=subprocess.DEVNULL)
    return p.wait() == 0
```

Наша мета полягає в тому, щоб масштабувати задачу пінгування, знаючи, що кожен пінг займає деякий час. Тому прискорення за рахунок зменшення таймауту або кількості пакетів для передачі для даної задачі не доречне.

Спробуйте пропінгувати діапазон адрес 192.168.2.*, де * - значення від 1 до 255. У послідовній версії програми це може зайняти близько 10 хвилин (в середньому 5 секунд на пінгування однієї адреси).

Уявіть, скільки часу триватиме пінгування всіх IPv4-адрес (4 294 967 296 штук)! Оскільки окрема така задача не завантажує інтенсивно процесор, багатопроцесна система зможе вирішити загальну проблему ефективніше. Основним недоліком є значна залежність від інтенсивності вводу-виводу, тому потрібно задіяти потоки (добре себе проявляють при задачах, пов'язаних з блокуючим вводом-виводом та малим навантаженням процесора).

За допомогою бібліотеки **`concurrent.futures`** можна управляти пулом потоків та планувати виконання задач (task-ів). Спочатку запустіть послідовну версію програми з перебором IP-адрес та їх пінгуванням. Знайдіть час роботи програми. Потім запустіть багатопоточну версію програми з кількістю потоків 4, 8, 16, 32 та 64. Порівняйте часи роботи усіх версій програми на своєму комп'ютері, занесіть порівняльну таблицю у звіт та зробіть висновки щодо доцільності багатопоточної версії програми.

ІНДЗ

1. ***(Клієнт-сервер)*** Створіть клієнт-серверний додаток на основі Python-модуля socket.
2. ***(Корутини та асинхронні запити)*** Напишіть простого пошукового павука, чия задача полягатиме у формуванні списку посилань, що є на веб-сторінці. Список веб-сторінок, для яких відбуватиметься пошук, має знаходитись в окремому txt-файлі, а перелік усіх знайдених посилань записуватись в інший txt-файл. У цілому в роботі може бути 4 основних корутини:
   - отримання HTML-розмітки (GET-запит);
   - парсинг (розбір) отриманого рядка HTML-розмітки: пошук атрибутів href та збереження посилань у випадку їх знаходження;
   - асинхронний запис знайдених після парсингу посилань у відповідний файл: як тільки розібрано рядок HTML-коду, відбувається асинхронний дозапис файлу новою інформацією;
   - координуюча корутина, яка запускає клієнтську сесію та збирає асинхронні таски (обробляє посилання на початкові веб-сторінки, адреси яких завантажуються з файлу). В наступному уривку коду вона називається crawler_write, проте можна використовувати довільну назву.

   Для запуску роботи можна використовувати нижче наведений уривок коду (crawler_write – назва відповідної корутини):

```
import pathlib
import sys

assert sys.version_info >= (3, 7), "Скрипт потребує версію Python 3.7+."
here = pathlib.Path(__file__).parent

with open(here.joinpath("urls.txt")) as infile:
    urls = set(map(str.strip, infile))

outpath = here.joinpath("found.txt")
with open(outpath, "w") as outfile:
    outfile.write("Джерело\tРозпарсене посилання\n")

asyncio.run(crawler_write(file=outpath, urls=urls))
```

Виконувати запити до веб-сторінок та файловий ввід-вивід потрібно асинхронно. У цьому можуть допомогти корутини та бібліотеки, що працюють з async/await, зокрема:

- aiohttp: асинхронний клієнт-серверний HTTP-фреймворк.
- aiofiles: асинхронний файловий ввід-вивід.

Окремо зверніть увагу на обробку виключень та ведення логу роботи програми. Рекомендується використовувати модуль logging для відстеження дій протягом сесії та виключень з їх подальшим налагоджуванням.

3. ***(Реактивне програмування)*** Запустіть та детально опишіть реалізацію та роботу RSS-агрегатора.