Zaawansowane techniki programowania w języku Python

Lista zadań – Multithreading

Zadanie 1 - Rozgrzewka

Napisz program, który uruchamia 9 wątków, z których każdy drukuje, z sekundową przerwą, odpowiednio cyfrę 1, 2, ..., 9.

Zadanie 2 – Stabilność obiektu

Napisz program, który dla danego obiektu przechowującego liczbę całkowitą, uruchomione zostaną wątki ingerujące w wartość liczby (dodawanie/odejmowanie/mnożenie/dzielenie). Wprowadź odpowiednią synchronizację, aby uzyskać stabilność obiektu.

Zadanie 3 – Kwadrat liczby

Napisz program, który wczyta od użytkownika jedną linię zawierającą liczby oddzielone spacją. Następnie w różnych wątkach podniesie je do kwadratu i wypisze wszystkie na ekranie w jednej linii oddzielone znakiem spacji.

Zadanie wykonaj w dwóch wariantach:

1. do utworzenia dodatkowych wątków wykorzystaj moduł `threading` i klasę `Thread`,
2. wykorzystaj moduł `concurrent.futures`, klasę `ThreadPoolExecutor` i funkcję `map`.

Który z wariantów preferujesz, dlaczego?

Zadanie 4 – Liczby pierwsze

Napisz program, który dla podanych przez użytkownika liczb, sprawdzi w różnych wątkach czy poszczególne liczby są pierwsze.

Notatki:

.....

.....

Zadanie 5 – Lista liczb

Napisz program, w którym utworzysz cztery wątki. Każdy wątek powinien odliczać od `10*i` do `10*i+9`, gdzie `i` to indywidualny jednocyfrowy numer, który przydzielisz wątkowi. Kolejne liczby powinny być dodawane do wspólnej dla wszystkich wątków listy. Zadbaj o to, by liczby na liście były przez jeden wątek zapisane spójnie, tzn. zapisane w kolejności od `10*i` do `10*i+9` – nie powinny być między sobą *pomieszane* w wyniku losowego działania wątków. Różna może być tylko kolejność sekwencji liczb wynikająca z kolejności wykonywania wątków. Po zakończeniu działania wszystkich wątków wypisz zawartość listy na ekranie.

Przykładowy wynik działania aplikacji dla wątków o identyfikatorach 0, 1, 2, 3 (kolorami oddzielono sekwencje liczb wygenerowane przez poszczególne wątki):

```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 30, 31, 32, 33, 34, 35, 36, 37, 38, 0,
1, 2, 3, 4, 5, 6, 7, 8, 20, 21, 22, 23, 24, 25, 26, 27, 28]
```

Wskazówki:

- do utworzenia dodatkowego wątku wykorzystaj moduł `threading` i klasę `Thread`;
- do synchronizacji wykorzystaj klasę `Lock`;

Zadanie 6 – Pobieranie

Napisz program, który pobierze 5 stron internetowych (źródło strony) podanych przez użytkownika. W pierwszej kolejności program powinien zapytać o adresy URL stron do pobrania. Następnie powinien zostać utworzony dodatkowy wątek odpowiedzialny za pobranie wszystkich stron. Dopóki wątek dodatkowy pracuje, w wątku głównym aplikacji co sekundę powinna być drukowana na ekranie informacja o oczekiwaniu na pobranie wszystkich stron. Na zakończenie program powinien wypisać liczbę znaków w kodzie źródłowym strony.

Przykładowy wynik działania aplikacji:

```
Podaj URL 1: https://python.org/
Podaj URL 2: https://docs.python.org/
Podaj URL 3: https://wp.pl/
```

Notatki:

.....

.....

```
Podaj URL 4: https://interia.pl/
Podaj URL 5: https://onet.pl/
Rozpoczynanie pobierania...
Oczekiwanie na zakończenie pobierania...
Oczekiwanie na zakończenie pobierania...
Oczekiwanie na zakończenie pobierania...
Oczekiwanie na zakończenie pobierania...
Wszystko pobrano.
Liczba znaków dla https://python.org/: 50065
Liczba znaków dla https://docs.python.org/: 9969
Liczba znaków dla https://wp.pl/: 486809
Liczba znaków dla https://interia.pl/: 250340
Liczba znaków dla https://onet.pl/: 885960
```

Wskazówki:

- do utworzenia dodatkowego wątku wykorzystaj moduł `threading` i klasę `Thread`;
- możesz wykorzystać metodę `is_alive()` do sprawdzenia, czy wątek nadal pracuje;
- jeśli potrzebujesz synchronizacji, to wykorzystaj klasę `Lock`;
- do pobierania źródła strony wykorzystaj wbudowaną bibliotekę `urllib`, w szczególności sprawdź sekcję przykładów w oficjalnej dokumentacji: https://docs.python.org/3/library/urllib.request.html#examples
- do wstrzymania działania wątku wykorzystaj funkcję `sleep()` z modułu time.

Zadanie 7 – Suma kontrolna

Napisz program, w którym wygenerujesz 200 identyfikatorów UUID4 zapisanych w postaci napisu szesnastkowego. Następnie, za pomocą wielowątkowości, policzysz sumę MD5 każdego z napisów. Po zakończeniu obliczeń wypisz na ekranie parę: identyfikator - suma kontrolna dla każdego identyfikatora.

Przykładowy wynik działania aplikacji:

```
ef0a2ffdf9304cd1bc7a49eaf5117df2 - 3384f58fc2bb4b7ede60db9f06130f8b
<część tekstu wycięta ze względu na długi wynik>
```

Notatki:

.....

.....

```
e0be94164ff745328a3b7f93ed14aa41 - b2b79997ed1fb254b574933b99141448
d5d02e4794ed42fc8126ade4fa78256a - 9a39af4ebd38d721210f58df3f016885
```

Wskazówki:

- identyfikatory UUID w postaci napisów możesz przechowywać w liście;
- do wygenerowania identyfikatorów UUID4 w postaci napisu szesnastkowego użyj modułu `uuid`, funkcji `uuid4()` i pola `hex` z obiektu wynikowego typu `UUID;`
- do obliczenia sumy MD5 użyj modułu `hashlib`, funkcji `md5()` i `hexdigest();`
- funkcja `md5()` w przyjmuje argument typu `bytes`, do zamiany napisu (typ `str`) na ciąg bajtów (typ `bytes`) użyj metody `encode()`, którą wywołasz na obiekcie napisu;
- możesz skorzystać z generatora `zip()` do wypisywania par uuid – suma kontrolna.

Notatki:

.....

.....