Багатопотокове програмування

№ уроку: 6 **Курс:** Python Advanced

Засоби навчання: PyCharm

Огляд, мета та призначення уроку

Вивчити основи багатопотоковості. Отримати досвід роботи з модулем threading в Python. Розглянути способи синхронізації роботи потоків. Розібратися з поняттям GIL у Python та обмежень, які накладаються на еталонну реалізацію мови Python-CPython. Розглянути приклади роботи з модулем concurrent.futures.

Вивчивши матеріал цього заняття, учень зможе:

- Створювати багатопотокові програми.
- Розуміти обмеження CPython, які накладаються GIL під час написання багатопотокових програм.
- Використовувати модулі threading і concurrent.futures.

Зміст уроку

- Основні поняття багатопотоковості.
- GIL у Python.
- Вивчення модуля threading: Thread, Lock, RLock, Event, Semaphore, Timer.
- Вивчення бібліотеки поточного futures.

Резюме

Кожна програма запускається в окремому процесі. Мінімальною одиницею програми є потік. Кожен процес складається з певної кількості потоків. Кожна програма складається щонайменше з одного потоку, який називається головним потоком.

Багатопотоковість використовується для запуску паралельних обчислень для прискорення обчислень або використання всієї потужності багатоядерної архітектури.

В еталонній реалізації мови Python-CPython, яку ми використовуємо у своїй практиці для створення застосунків, наявний спеціальний механізм, що називається GIL (*Global Interpreter Lock*).

Global Interpreter Lock (GIL) – це спосіб синхронізації потоків, який використовується в деяких інтерпретованих мовах програмування, наприклад Python і Ruby.

Цей механізм накладає обмеження на багатопотокові програми та дозволяє лише одному потоку виконуватись у конкретний момент. Тобто ми не маємо як такої багатопотоковості у явному вигляді у CPython. Є й інші реалізації мови Python: IronPython, Jython. Вони не мають механізму GIL. Навіть якщо ми пишемо багатопотокову програму, потоки, які створюються, не будуть виконуватися паралельно через блокування.

У стандартній бібліотеці мови Python є модуль **threading**, який призначений для створення багатопотокових програм і містить набір класів для цих завдань. Модуль також надає спеціальні механізми синхронізації процесів та потоків: блокування, події, семафори й умовні змінні.

Починаючи з 3 версії Python, доступний модуль **concurrent.futures**, який надає зручні інтерфейси для запуску багатопотокових і багатопроцесних обчислень. Є два класи для таких завдань: **ThreadPoolExecutor** та **ProcessPoolExecutor**. **ThreadPoolExecutore** – це клас для створення пулу потоків, а **ProcessPoolExecutor** – для пулу процесів.

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

У межах бібліотеки concurrent.futures є спеціальний клас **Future** (його часто називають «ф'ючером» або «футуром»), який є прикладом *Promise* (так званих обіцянок). Він дає змогу отримати результат відстроченого обчислення. Тобто завдання, яке поверне результат у найближчому майбутньому, наприклад, після виконання якогось набору навантажених операцій або звернення до стороннього сервера. Цей клас має набір методів, як-от result, done, running, що дають змогу перевіряти поточний стан Future та отримувати результат виконання після завершення обчислень.

Executor-и безпосередньо пов'язані з класом Future. Під час додавання завдання на виконання відповідний клас executor-а повертає об'єкт Future, який пов'язаний із завданням, що передається на виконання. Через отриманий Future, як уже зазначалося раніше, можна отримати результат виконання цього завдання з пулу.

Приклад

Розглянемо приклад програми, яка запускає 2 потоки. Один потік пише десять разів «0», інший – десять разів «1», причому суворо по черзі.

```
import threading

def writer(x, event_for_wait, event_for_set):
 for i in xrange(10):
 event_for_wait.wait() # чекаємо подію
 event_for_wait.clear() # очищуємо подію для future
 print x
 event_for_set.set() # встановлюємо подію для сусіднього потоку

# створення події
e1 = threading.Event()
e2 = threading.Event()

# створення потоків
t1 = threading.Thread(target=writer, args=(0, e1, e2))
t2 = threading.Thread(target=writer, args=(1, e2, e1))

# запуск потоків
t1.start()
t2.start()

e1.set() # встановлення події для першого потоку

# злиття потоків (блокування основного потоку, доки t1 і t2 не виконаються)
t1.join()
t2.join()
```

Закріплення матеріалу

- Що таке потоки та як вони пов'язані з процесами?
- Яку мінімальну кількість потоків містить будь-яка програма?
- У чому перевага багатопотокових програм?
- Що таке GIL та які особливості цього механізму?
- Який клас з модуля threading необхідно використовувати для створення потоку, від якого можна успадковуватися або просто передати функцію для виконання?
- Які примітиви синхронізації процесів і потоків ви знаєте?
- Який клас використовується для створення пулу потоків із модуля concurrent.futures?
- Який клас використовується для створення пулу процесів із модуля concurrent.futures?
- Як додати завдання на виконання в пул Executor?

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- Що таке Future?

Самостійна діяльність учня

Завдання 1
Створіть функцію для обчислення факторіала числа. Запустіть декілька завдань, використовуючи ThreadPoolExecutor, і заміряйте швидкість їхнього виконання, а потім заміряйте швидкість обчислення, використовуючи той же набір завдань на ProcessPoolExecutor. Як приклади використовуйте останні значення, від мінімальних і до максимально можливих, щоб побачити приріст або втрату продуктивності.

Завдання 2
Створіть три функції, одна з яких читає файл на диску із заданим ім'ям та перевіряє наявність рядка «Wow!». Якщо файлу немає, то засипає на 5 секунд, а потім знову продовжує пошук по файлу. Якщо файл є, то відкриває його і шукає рядок «Wow!». За наявності цього рядка закриває файл і генерує подію, а інша функція чекає на цю подію і у разі її виникнення виконує видалення цього файлу. Якщо рядки «Wow!» не було знайдено у файлі, то засипати на 5 секунд. Створіть файл руками та перевірте виконання програми.

Рекомендовані ресурси

Офіційний сайт Python – threading
https://docs.python.org/3.11/library/threading.html
Офіційний сайт Python – concurrent.features
https://docs.python.org/3/library/concurrent.futures.html

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com