Многопоточное программирование

№ урока: 6 **Курс:** Python Advanced

Средства обучения: PyCharm

Обзор, цель и назначение урока

Изучить основы многопоточности. Получить опыт работы с модулем threading в Python. Рассмотреть способы синхронизации работы потоков. Разобраться с понятием GIL в Python и ограничений, которые накладываются на эталонную реализацию языка Python- CPython. Рассмотреть примеры работы с модулем concurrent.futures.

Изучив материал данного занятия, учащийся сможет:

- Создавать многопоточные программы.
- Понимать ограничение CPython накладываемые GIL при написании многопоточных программ.
- Использовать модули threading и concurrent.futures.

Содержание урока

1. Основные понятия многопоточности.
2. GIL в Python.
3. Изучение модуля threading: Thread, Lock, RLock, Event, Semaphore, Timer.
4. Изучение библиотеки concurrent. futures.

Резюме

Каждая программа запускается в отдельном процессе. Минимальной единицей программы является поток. Каждый процесс состоит из некоторого количества потоков. Каждая программа состоит как минимум из одного потока, который называется главным потоком.

Многопоточность используется для запуска параллельных вычислений с целью ускорения вычислений или использования всей мощи многоядерной архитектуры.

В эталонной реализации языка Python- *CPython*, которую мы используем в своей практике для создания приложений, существует специальный механизм, называемый GIL (*Global Interpreter Lock*).
Global Interpreter Lock (GIL) — это способ синхронизации потоков, который используется в некоторых интерпретируемых языках программирования, например в Python и Ruby.
Данный механизм накладывает ограничение на многопоточные программы и позволяет только одному потоку выполняться в конкретный момент времени. То есть мы не имеем как таковой многопоточности в явном виде в CPython. Существуют и другие реализации языка Python: IronPython, Jython, и они не имеют механизма GIL. То есть, даже если мы пишем многопоточную программу, создаваемые потоки не будут выполняться параллельно из-за блокировки.

В стандартной библиотеки языка Python имеется модуль threading, который предназначен для создания многопоточных программ и содержит в себе набор классов для данных задач. Модуль также предоставляет специальные механизмы синхронизации процессов и потоков, такие как блокировки, события, семафоры и условные переменные.
Начиная с 3 версии в Python доступен модуль concurrent.futures, который предоставляет удобные интерфейсы для запуска многопоточных и многопроцессных вычислений. Существует два класса для таких задач: ThreadPoolExecutor и ProcessPoolExecutor. Соответственно ThreadPoolExecutore- это класс для создания пула потоков, а ProcessPoolExecutor- для пула процессов.
В рамках библиотеки concurrent.futures имеется специальный класс Future (часто называемого «*фьючером*» или «*футуром*»), который является примером *Promise* (так называемых обещаний). Он позволяет получить результат отложенного вычисления. То есть задача, которая вернет результат в

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

ближайшем будущем, например, после выполнения какого-то набора нагруженных операций или, как пример, обращения к стороннему серверу. Данный класс имеет набор методов, таких как result, done, running, позволяющих проверять текущее состояние Future и получать результат выполнения по завершению вычислений.
Executor-ы напрямую связаны с классом Future. При добавлении задачи на выполнение, соответствующий класс executor-а возвращает объект Future, который связан с передаваемой задачей на исполнение. И через полученный Future, как уже говорилось ранее, можно получить результат выполнения данной задачи из пула.

Пример
Рассмотрим пример программы, которая запускает 2 потока. Один поток пишет десять раз "*0*", другой — десять раз "*1*", причем строго поочереди.

```
import threading

def writer(x, event_for_wait, event_for_set):
    for i in xrange(10):
        event_for_wait.wait() # ждём события
        event_for_wait.clear() # очищаем событие для future
        print x
        event_for_set.set() # устанавливаем событие для соседнего потока

# создание события
e1 = threading.Event()
e2 = threading.Event()

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

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

e1.set() # установка события для первого потока

# слияние потоков (блокирование основного потка, пока t1 и t2 не выполнятся)
t1.join()
t2.join()
```

Закрепление материала

- Что такое потоки и как они связаны с процессами?
- Какое минимальное количество потоков содержит любая программа?
- В чем преимущество многопоточных программ?
- Что такое GIL и в чем особенности данного механизма?
- Какой класс из модуля threading необходимо использовать для создания потока, от которого можно наследоваться или же просто передать функцию для исполнения?
- Какие примитивы синхронизации процессов и потоков вы знаете?
- Какой класс используется для создание пула потоков из модуля concurrent.futures?
- Какой класс используется для создание пула процессов из модуля concurrent.futures?
- Как добавить задачу на исполнение в пулл Executor-а?
- Что такое Future?

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

Самостоятельная деятельность учащегося

Задание 1
Создайте функцию по вычислению факториала числа. Запустите несколько задач, используя ThreadPoolExecutor и замерьте скорость их выполнения, а затем замерьте скорость вычисления, используя тот же самый набор задач на ProcessPoolExecutor. В качестве примеров, используйте крайние значения, начиная от минимальных и заканчивая максимально возможными, чтобы увидеть прирост или потерю производительности.

Задание 2
Создайте три функции, одна из которых читает файл на диске с заданным именем и проверяет наличие строку “Wow! ”. В случае, если файла нет, то засыпает на 5 секунд, а затем снова продолжает поиск по файлу. В случае, если файл есть, то открывает его и ищет строку “Wow!”. При наличии данной строки закрывает файл и генерирует событие, а другая функция ожидает данное событие и в случае его возникновения выполняет удаление этого файла. В случае если строки «Wow!» не было найдено в файле, то засыпать на 5 секунд. Создайте файл руками и проверьте выполнение программы.

Рекомендуемые ресурсы

Официальный сайт Python - threading
https://docs.python.org/3.11/library/threading.html
Официальный 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