Ускорение Flask приложения, потоки

№ урока: 9 **Курс:** Flask

Средства обучения: Python3 и любимая среда разработки, например, PyCharm.

Обзор, цель и назначение урока

Узнаем, как более эффективно использовать ресурсы сервера/процессора, изучим способы ускорения Flask приложения и рассмотрим варианты применения каждого из способов. Также, познакомимся с примитивами синхронизации в Python и поговорим о GIL.

Изучив материал данного занятия, учащийся сможет:

- Использовать потоки и процессы в разработке.
- Правильно применять потоки и процессы.
- Различать CPU-bound и I/O-bound задачи.
- Ускорять Flask приложение при I/O-bound задачах.
- Получать данные из сторонних API с помощью библиотеки requests.

Содержание урока

1. Потоки и процессы
2. Модуль threading
3. Модуль multiprocessing и concurrent.futures
4. Примитивы синхронизации
5. Ускорение Flask приложения, знакомство с библиотекой requests
6. GIL

Резюме

- С появлением многоядерных процессоров стала общеупотребительной практика распространять нагрузку на все доступные ядра. Существует два основных подхода в распределении нагрузки: использование процессов и потоков.
  Использование нескольких процессов фактически означает использование нескольких программ, которые выполняются независимо друг от друга. Программно это решается с помощью системных вызовов exec и fork. Такой подход создает большие неудобства в управлении обмена данными между этими программами.
  В качестве альтернативы существует другой подход – создание многопоточных программ. Обмен данными между потоками существенно упрощается. Но управление такими программами усложняется, и вся ответственность ложится на программиста.
- **Process** (процесс) — выполняющаяся программа. Каждый процесс обладает некоторыми параметрами, характеризующими его состояние, включая объем памяти, список открытых файлов, программный счетчик, который ссылается на очередную выполняемую инструкцию, и стек вызовов. Обычно процесс выполняет инструкцию одну за другой, в единственном потоке управления. В каждый конкретный момент программа делает что-то одно. Процессы изолированы друг от друга.
- **Thread** (поток) — это отдельный поток выполнения. Это означает, что в вашей программе могут работать две и более подпрограммы одновременно. Но разные потоки на самом деле не работают одновременно: компьютер ничего не делает одновременно, на самом деле он просто быстро переключается между потоками. Поток напоминает процесс тем, что он выполняет собственную последовательность инструкций и имеет

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

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

свой стек вызовов. Разница состоит в том, что потоки выполняются в пределах процесса, создавшего их, и они совместно используют данные и системные ресурсы, выделенные процессу.

- Суть работы потоков заключается в том, что они переключаются превентивно (мы никак не можем повлиять на это переключение и никак не узнаем, когда оно произойдет). Это удобно, потому что вам не нужно добавлять явный код, чтобы вызвать переключение задачи. Цена этого удобства состоит в том, что вы должны предположить, что переключение может произойти в любой момент. Соответственно, критические участки должны быть защищены примитивами синхронизации или блокировками. Решением в этом случае является применение блокировки. При этом доступ к заблокированному списку будет иметь только один поток, второй будет ждать, пока блокировка не будет снята. Применение блокировки порождает другую проблему – дедлок **(deadlock)** – мертвая блокировка.
- Пример дедлока: имеется два потока и два списка. Первый поток блокирует первый список, второй поток блокирует второй список. Первый поток изнутри первой блокировки пытается получить доступ к уже заблокированному второму списку, второй поток пытается проделать то же самое с первым списком. Получается неопределенная ситуация с бесконечным ожиданием. Эту ситуации легко описать, на практике все гораздо сложнее.

  Вариантом решения проблемы дедлоков является политика определения очередности блокировок. Например, в предыдущем примере мы должны определить, что блокировка первого списка идет всегда первой, а уже потом идет блокировка второго списка. Другая проблема с блокировками в том, что несколько потоков могут одновременно ждать доступа к уже заблокированному ресурсу и при этом ничего не делать. Каждая питоновская программа всегда имеет главный управляющий поток.
- **Lock** — мьютекс, простейшая блокировка. Это механизм синхронизации, имеющий 2 состояния: закрыто и открыто. Для изменения состояния блокировки используются методы acquire() и release(). Если блокировка находится в состоянии закрыто, любая попытка приобрести ее будет заблокирована до момента, пока она не будет освобождена. Порядок, в котором потоки смогут продолжить работу, заранее не определен.
- **Семафор** — это механизм синхронизации, основанный на счетчике, который уменьшается при каждом вызове метода acquire() и увеличивается при каждом вызове метода realease(). Если счетчик семафора достигает нуля, метод acquire() приостанавливает работу потока, пока какой-либо другой поток не вызовет метод release().
- Мьютекс отличается от семафора тем, что только владеющий им поток может его освободить, т.е. перевести в отмеченное состояние.
- Питоновская реализация многопоточности ограниченная. Интерпретатор питона использует внутренний глобальный блокировщик (GIL), который позволяет выполняться только одному потоку. Это сводит на нет преимущества многоядерной архитектуры процессоров. Для многопоточных приложений, которые используются в основном для дисковых операций чтения/записи, это не имеет особого значения, а для приложений, которые делят процессорное время между потоками, это является серьезным ограничением.
- **GIL (Global Interpreter Lock)** — мьютекс, который гарантирует, что в каждый момент времени только один поток имеет доступ к внутреннему состоянию интерпретатора. Глобальный блокировщик следит за тем, чтобы активен был всегда только один поток.
- Поскольку потоки выполняется на одном процессоре, они хорошо подходят для ускорения некоторых задач, но не для всех. Задачи, которые требуют значительных вычислений ЦП и тратят мало времени на ожидание внешних событий, очевидно не будут выполняться быстрее, используя многопоточность. Вместо этого следует использовать многопроцессорность (multiprocessing).

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

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- **Race conditions** (Гонка за ресурсами) — попытка изменения некоторых данных одновременно из нескольких потоков, может привести к их повреждению и к нарушению целостности состояния программы.
- **Baby making problem** — есть задачи, которые по своей сути последовательны. Например, вне зависимости от того, сколько будет worker'ов (потоков или процессов) ускорить рождение ребенка не получится. Но, если задача стоит в том, чтобы постричь газон, то добавление worker'а может ускорить процесс.
- **Виды конкурентности**:
    - **Pre-emptive multitasking (threading)** - операционная система решает, когда переключать контекст задачи.
    - **Cooperative multitasking (asyncio)** - задачи сами решают, когда отдать контроль операционной системе.
    - **Multiprocessing (multiprocessing)** - все процессы выполняются одновременно на разных процессорах.
- CPU Bound задачи — операции, задействующие центральный процессор. Как правило, это вычисления: работа с матрицами, изображениями, анализ больших массивов данных, вычисление последовательности Фибоначчи или майнинг биткоинов.
- Самым простым и эффективным способом решения CPU-Intensive задачи, является использование идиомы **Fork-Join** - задачу (например, входные данные) нужно разбить на определенное число подзадач, которые можно выполнить параллельно. Каждая подзадача должна быть независимой и не обращаться к разделяемым переменным/памяти. Затем, нужно собрать промежуточные результаты и объединить их.
- I/O Bound задачи — задачи, использующие ввод/вывод: работу с диском или сетью. К таковым относятся веб-сервера или часть веб-приложения, которая принимает запросы от клиентов.

Закрепление материала

- Что такое процесс?
- Что такое поток?
- В чем минусы использования потоков?
- Как работают потоки?
- Что такое гонка за ресурсами?
- Что такое мьютекс?
- Чем мьютекс отличается от семафора?
- Что такое GIL?

Дополнительное задание

Задание

Реализуйте поиск фильмов с помощью любого публичного API фильмов, например, The Movie DB, используя requests и concurrent.futures библиотеки. Для этого создайте отдельный route и view для обработки этого запроса.

Самостоятельная деятельность учащегося

Задание 1
Выучите основные понятия, рассмотренные на уроке.

Задание 2
Изучите материалы из рекомендуемых ресурсов.

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

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Задание 3
Рассмотрите проблему Производитель-Потребитель и постарайтесь реализовать ее решение с помощью модуля Queue. Программа не может запрашивать сообщения, когда захочет. Она должна прослушивать и принимать сообщения по мере их поступления. Сообщения не будут поступать в обычном темпе, а будут приходить очередями. Эта часть программы называется продюсером (producer). С другой стороны, когда у вас есть сообщение, вам нужно записать его в базу данных. Доступ к базе данных медленный, но достаточно быстрый, чтобы соответствовать среднему темпу сообщений. Скорости доступа недостаточно для того чтобы не отставать, когда приходит поток сообщений. Эта часть является потребителем (consumer). Между ними нужно использовать очередь сообщений или просто очередь для обмена данными.

Рекомендуемые ресурсы

Raymond Hettinger, Keynote on Concurrency:
https://www.youtube.com/watch?v=9zinZmE3Ogk

Введение в потоки в Python:
https://webdevblog.ru/vvedenie-v-potoki-v-python/

PyCon 2015 - Python's Infamous GIL by Larry Hastings:
https://www.youtube.com/watch?v=KVKufdTphKs

An Intro to Threading in Python:
https://realpython.com/intro-to-python-threading/

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

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com