Многопоточность в Python

Медведева Светлана Юрьевна

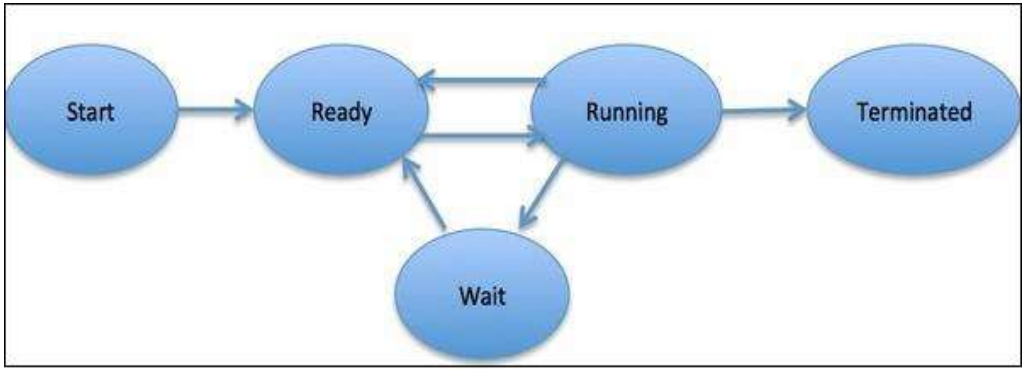
Кафедра Информатики и вычислительной математики МФТИ

Многопоточность в Python

- Поток и процесс. Создание процессов и процессов.
- Передача данных между потоками при помощи pipe и общей памяти.
- GIL.
- Асинхронное выполнение потоков.
- Библиотеки multiprocessing и asyncio.

Поток и процесс

Поток - это наименьшая единица выполнения с независимым набором инструкций.



Плюсы и минусы многопоточности

Повышение скорости вычислений в многопроцессорных или многоядерных системах

Multithreading позволяет программе оставаться отзывчивой, пока один поток ожидает ввода, а другой одновременно запускает графический интерфейс.

Все потоки процесса имеют доступ к его глобальным переменным.

В однопроцессорной системе многопоточность не влияет на скорость вычислений.

Синхронизация необходима, чтобы избежать взаимного исключения при доступе к общим ресурсам процесса.

Многопоточность увеличивает сложность программы, что также затрудняет отладку.

thread module

_thread.start_new_thread (function, args[, kwargs])

Работает в linux, windows

```
import _thread
import time

# Define a function for the thread
def print_time( threadName, delay):
   count = 0
   while count < 5:
      time.sleep(delay)
      count += 1
      print ("%s: %s" % ( threadName, time.ctime(time.time()) ))

# Create two threads as follows
try:
   _thread.start_new_thread( print_time, ("Thread-1", 2, ) )
   _thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
   print ("Error: unable to start thread")

while 1:
   pass
```

The Threading Module

- threading.activeCount(): Эта функция возвращает общее число активных в настоящий момент времени объектов потока в данной программе
- threading.currentThread(): Данная функция возвращает общее число объектов потока в данном текущем потоке, управляемых стороной вызова
- threading.enumerate(): Функция возвращает список всех активных в настоящий момент объектов потока в данной программе

The Threading Module

- run(): Этот метод исполняется когда инициализируется и запускается некий новый поток.
- start(): Данный метод запускает инициализированный вызывающий объект потока, вызывая соответствующий метод run().
- join(): Такой метод ожидает завершения соответствующего вызывающего объекта потока прежде чем продолжить исполнение оставшейся части программы.
- isAlive(): Данный метод возвращает Булево значение, указываюшее исполняется ли в данный момент вызывающий объект потока.
- getName(): Этот метод возвращает собственно название данного вызывающего объекта потока.
- setName(): Этот метод устанавливает соответствующее название данного вызывающего объекта потока.

The Threading Module

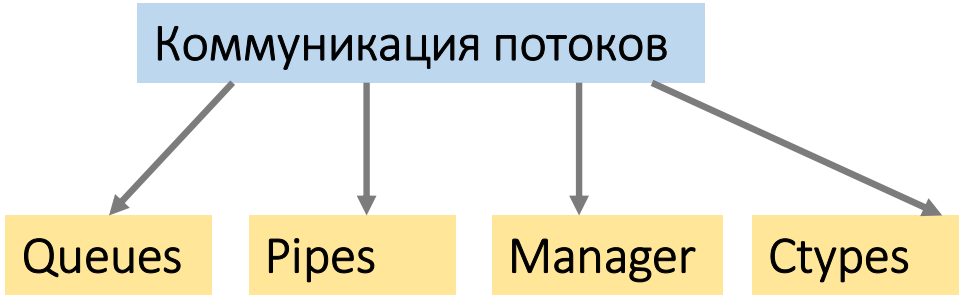
Создание и настройка нового потока:

- В вашей программе определите некий подкласс общего класса threading.Thread.
- Внутри полученного подкласса перепишите установленный по умолчанию метод __init__(self [,args]) для добавления необходимых индивидуальных аргументов в этот класс.
- Внутри этого же подкласса перепишите установленный по умолчанию метод run(self [,args]) для персонализации имеющегося поведения данного класса потоков при инициализации и запуске некоего нового потока

The Threading Module

```
import threading
import time


class MyThread(threading.Thread):
    def __init__(self, name, delay):
        threading.Thread.__init__(self)
        self.name = name
        self.delay = delay

    def run(self):
        print('Starting thread %s.' % self.name)
        thread_count_down(self.name, self.delay)
        print('Finished thread %s.' % self.name)

def thread_count_down(name, delay):
    counter = 5

    while counter:
        time.sleep(delay)
        print('Thread %s counting down: %i...' % (name, counter))
        counter -= 1
```

Передача данных между потоками при помощи pipe и общей памяти



Queue предоставляет механизм взаимодействия потоков между процессами FIFO (первым пришел — первым обслужен).

Pipe используется для связи между процессами в многопроцессорных программах.

Менеджер — это класс многопроцессорных модулей, который обеспечивает способ координации общей информации между всеми его пользователями.

Объекты *Array* и *Value* из **Ctypes** используются для хранения данных в карте общей памяти.

GIL

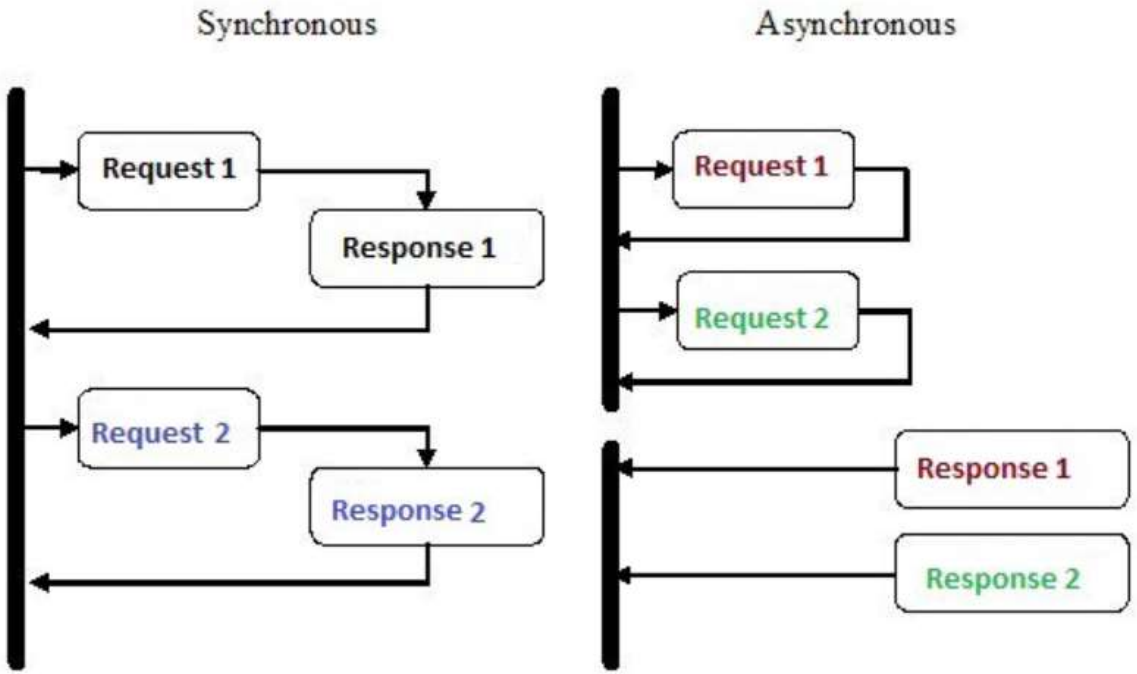
Python Global Interpreter Lock (GIL) — это своеобразная блокировка, позволяющая только одному потоку управлять интерпретатором Python.

```
import sys
a = []
b = a
sys.getrefcount(a)
```

Проблема: в многопоточном приложении сразу несколько потоков могут увеличивать или уменьшать значения этого счётчика ссылок. Это может привести к тому, что память очистится неправильно и удалится тот объект, на который ещё существует ссылка.

Асинхронное выполнение потоков

Асинхронное программирование – это вид параллельного программирования, в котором какая-либо единица работы может выполняться отдельно от основного потока выполнения приложения.



Библиотеки multiprocessing и asyncio

Asyncio – модуль асинхронного программирования, который был представлен в Python 3.4. Он предназначен для использования корутин и future для упрощения написания асинхронного кода и делает его почти таким же читаемым, как синхронный код, из-за отсутствия callback-ов.

Модуль **multiprocessin**g был добавлен в Python версии 2.6. Изначально он был определен в PEP 371 Джесси Ноллером и Ричардом Одкерком. Модуль multiprocessing позволяет вам создавать процессы таким же образом, как при создании потоков при помощи модуля threading. Суть в том, что, в связи с тем, что мы теперь создаем процессы, вы можете обойти **GIL (Global Interpreter Lock)** и воспользоваться возможностью использования нескольких процессоров на компьютере.

Итоги

- Поток и процесс. Создание процессов и процессов.
- Передача данных между потоками при помощи pipe и общей памяти.
- GIL.
- Асинхронное выполнение потоков.
- Библиотеки multiprocessing и asyncio.

Спасибо за внимание!