Lightning Talk – Manuel Cameselle

17 de Enero de 2019

GEVENT

greenlets

http://www.gevent.org/intro.html
http://sdiehl.github.io/gevent-tutorial/
https://learn-gevent-socketio.readthedocs.io/en/latest/greenlets.html
https://stackoverflow.com/a/15596277

Gevent es una librería de concurrencia basada en libev.

El patrón primario usado en gevent es el **greenlet**, similar a un thread con significativas diferencias:

POSIX threads (pthreads)	**Green threads (greenlets)**
pthreads can switch between threads pre-emptively at any time	greenlets only switch explicitly or when a performs a I/O blocking operation
Use locks to manage mutex to avoid race conditions.	There will not be any race conditions.

Solo se ejecuta un greenlet al mismo tiempo**.**

Greenlets provide concurrency but not parallelism.

- Concurrency is when code can run independently of other code.
- Parallelism is the execution of concurrent code simultaneously.

- Concurrency is useful for breaking apart problems.
- Parallelism is particularly useful for CPU-heavy stuff.

- Greenlets really shine in network programming where interactions with one socket can occur independently of interactions with other sockets.
- Threading in Python is more expensive and more limited than usual due to the GIL (Global Interpreter Lock).

Projects like gevent expose concurrency without requiring change to an asynchronous API.

Demo 1

```
import gevent
import random

def task(pid):
    gevent.sleep(random.randint(0,2)*0.001)
    print('Task %s done' % pid)

print('Asynchronous:')
threads = [gevent.spawn(task, i) for i in xrange(10)]
gevent.joinall(threads)
```

Demo 2

```
import gevent.monkey
gevent.monkey.patch_socket()

import gevent
import urllib2
import json
from time import time

def fetch(pid):
    response = urllib2.urlopen('http://now.httpbin.org')
    result = response.read()
    json_result = json.loads(result)
    datetime = json_result['now']['epoch']

    print('Process %s: %s' % (pid, datetime))
```

```
def synchronous():
    for i in range(1,10):
        fetch(i)

def asynchronous():
    threads = []
    for i in range(1,10):
        threads.append(gevent.spawn(fetch, i))
    gevent.joinall(threads)

print('Synchronous:')
print(time())
synchronous()
print(time())

print('-')

print('Asynchronous:')
print(time())
asynchronous()
print(time())
```

Demo 3

```
# ¡¡¡ EJEMPLO NO FUNCIONAL solo  didáctico !!!

import socket
from gevent import monkey; monkey.patch_all()
from gevent import socket
from gevent import queue
import gevent
import pymysql

conn_rx_pool = conn_pool(NCONN)
conn_tx = pymysql.Connection()

global hilos_rx_dict
hilos_rx_dict = dict()

hilos = (gevent.spawn(transmitir_tramas),
             gevent.spawn(recibir_tramas),
             )
gevent.wait(hilos)
```

```
class conn_pool():
    def __init__(self, nconn):
        self.pool = queue.Queue()

        for i in range(nconn):
            conn = pymysql.Connection()
            self.pool.put(conn)

    def get_conn(self):
        return self.pool.get()

    def ret_conn(self, conn):
        self.pool.put(conn)


def transmitir_tramas():
    while run: # Bucle infinito del que saldremos cuando
alguien ponga la variable global run a False
        try:
            num = tx_tramas(conn_tx)
            if num == 0:
                gevent.sleep(0.25)
```

```
def recibir_tramas():
                data, address = udp.recvfrom()
                gevent.spawn(paraleliza_trama, data, 
address[0], address[1])


def paraleliza_trama(data, iptxt, puerto):
    datos = decod(data)
    nserie = datos['nserie']

    # comprobamos si ya hay al menos una trama de este 
nserie procesándose
    g = None
    if nserie in hilos_rx_dict:
        g = hilos_rx_dict[nserie]

    # anotamos que somos el último (si llega otra trama 
más de este nserie, se pondrá a la cola "detrás" de ésta)
    hilos_rx_dict[nserie] = gevent.getcurrent()

    if g is not None:
        # hay una trama de este mismo nserie todavía en 
proceso => esperaremos nuestro turno
        g.join()
    try:
        # cogemos una conexión del pool
        # (si no hubiese ninguna libre nos quedaríamos 
aquí “dormidos”)
        db = conn_rx_pool.get_conn()

        procesa_trama(db, iptxt, puerto, datos)

    except:
        pass

    finally:
        if db:
            # devolvemos la conexión al pool
            conn_rx_pool.ret_conn(db)

        # comprobamos si ha llegado alguna trama más de 
este nserie mientras procesábamos ésta
        if hilos_rx_dict[nserie] == gevent.getcurrent():
            # no ya llegado ninguna trama más
            # por tanto somos los "ultimos" y debemos 
"limpiar" el diccionario para que las siguientes no 
esperen
            hilos_rx_dict[nserie] = None
```