Multiprocessing

December 12, 2020

1 Concurrencia vs Paralelismo

Concurrency signfica que hay 2 o más procesos ejecutandose dentro del mismo periodo de tiempo. Paralelismo signfica que hay 2 o más procesos ejecutandose en el mismo momento. Paralelismo es un caso especial de concurrencia.

1.0.1 Proceso

Es lo que llama un programa que es cargado en memoria. Tiene su propio espacio de memoria. ### Hilo Un hilo es una unidad de ejecución dentro de un proceso. Un proceso puede tener varios hilos ejecutandonse. Los hilos comparten el espacio de memoria del proceso. ### Multithreading Es un técnica dónde existen multiples hilos dentro del proceso, para diferentes tareas, en el mismo espacio de tiempo. Esto da la ilusión de que los hilos se ejecutan en paralelo, pero son gestionados por GIL, que evita que los hilos se ejecuten al mismo tiempo. ### Multiprocessing Es una téncica dónde hay paralelismo. Multiples procesos corren en diferentes cores del CPU, que no comportanten recursos entre ellos. Cada proceso tiene su espacio de memoria.

2 Multiprocessing

Como ya vimos en la clase pasada, hilos en Python no está diseñado para funcionar en computadoras con más de un core, por ello GIL es necesario porque Python por sí solo no es "thread-safe".

Multiprocessing nos permite crear programas que puedan correr concurrentemente (bypaseando GIL) y usar todos los core de la CPU. Multiprocessing es sintácticamente parecido a Threading pero implementados de manera diferente. Multiprocessing le da a cada proceso su propio interprete de Python cada uno con su propio GIL.

[2]:
```
import multiprocessing

def worker(i):
    print(i)

if __name__ == '__main__':
    for i in range(5):
        p = multiprocessing.Process(target=worker, args=(i,))
        p.start()
        p.join()
```

```
0
1
2
3
4
```

Cuando usar multiprocessing: * Cuando se usa mucho consumo de CPU. * Cuando tiene una gran cantidad de uso de red o entrada/salida * Si se está usando UI.

2.1 Process

Se pueden crear procesos heredando de la clase Process, y el proceso comienza a ejecutarse cuando se llama a la función `start()`, parar modificar el funcionamiento del proceso, sobreescribimos `run()` al igual que con `Thread`. Esperamos que finalice con `join()`.

También tenemos los métodos: `is_alive()`, `terminate()`, `kill()`, `close()`.

[3]:
```
from multiprocessing import Process

class Worker(Process):
    def __init__(self, q):
        self.q = q
        super(Worker, self).__init__()
    def run(self):
        self.q.put('local hello')
```

2.2 Intercambio de objetos entre procesos

En `multiprocessing` hay dos tipos de comunicación.

2.2.1 Queue

Es muy parecido a `queue.Queue`.

[5]:
```
from multiprocessing import Process, Queue

def worker(q):
    q.put([42, None, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=worker, args=(q,))
    p.start()
    print(q.get())    # prints "[42, None, 'hello']"
    p.join()
```

```
[42, None, 'hello']
```

2.3 Pipes

La función `Pipe()` devuelve un par de conexión (las dos puntas de una conexión) que nos permite comunicar procesos.

[11]:
```
from multiprocessing import Process, Pipe

def f(conn):
    conn.send([42, None, 'hello', True])
    conn.close()

def f2(conn):
    print(conn.recv())

if __name__ == '__main__':
    parent_conn, child_conn = Pipe()

    p = Process(target=f, args=(child_conn,))

    p2 = Process(target=f2, args=(parent_conn,))

    p.start(); p2.start()
    # print(parent_conn.recv())   # prints "[42, None, 'hello']"
    p.join(); p2.join()
```

```
[42, None, 'hello', True]
```

3 Sincronización entre procesos.

Al igual que Threading tenemos primitivas de sincronización.

[6]:
```
from multiprocessing import Process, Lock

def f(l, i):
    l.acquire()
    try:
        print('hello world', i)
    finally:
        l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
        Process(target=f, args=(lock, num)).start()
```

```
hello world 0
hello world 1
```

```
hello world 2
hello world 3
hello world 4
hello world 5
hello world 6
hello world 7
hello world 8
hello world 9
```

4 Algunas funciones interesantes

- active_children()
- cpu_count()
- current_process()
- parent_process() (python3.8)