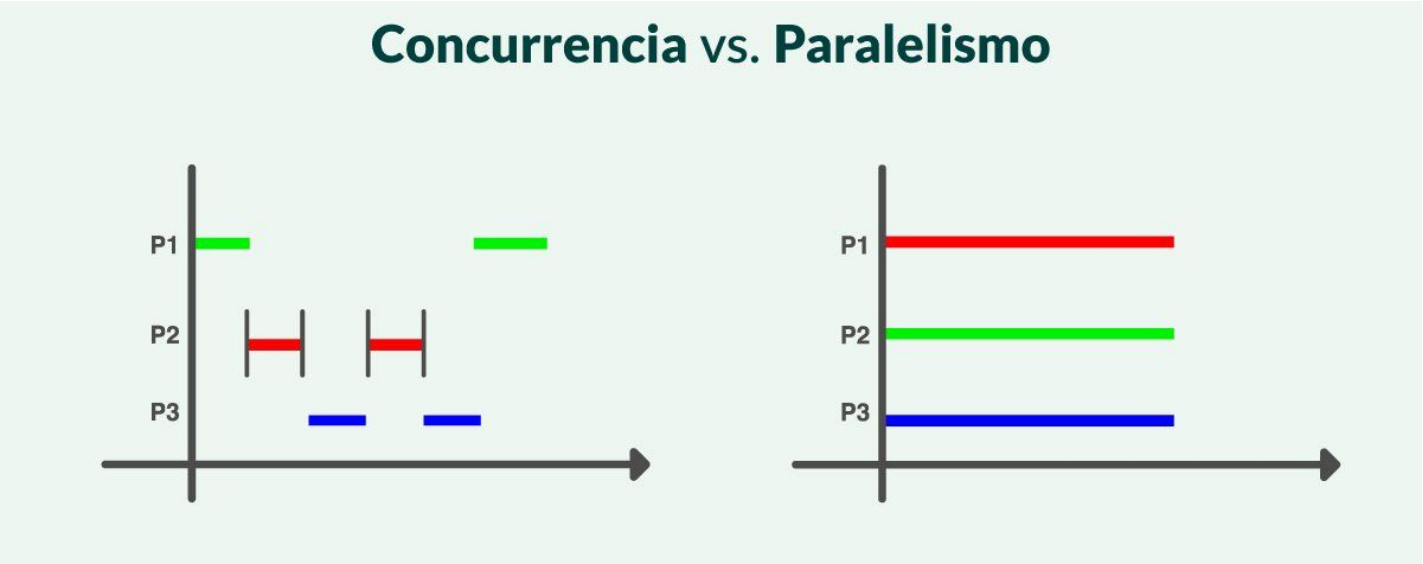
Curso Avanzado de Python

Multiproceso, Multihilo y async

Multithreading vs Multiprocessing vs Async

Hacer cosas en paralelo o muchas cosas a las vez, hay 3 opciones. ¿Cuál uso? ¿Cuál me conviene? Primero hay que entender la diferencia entre concurrencia y paralelismo.



Concurrencia

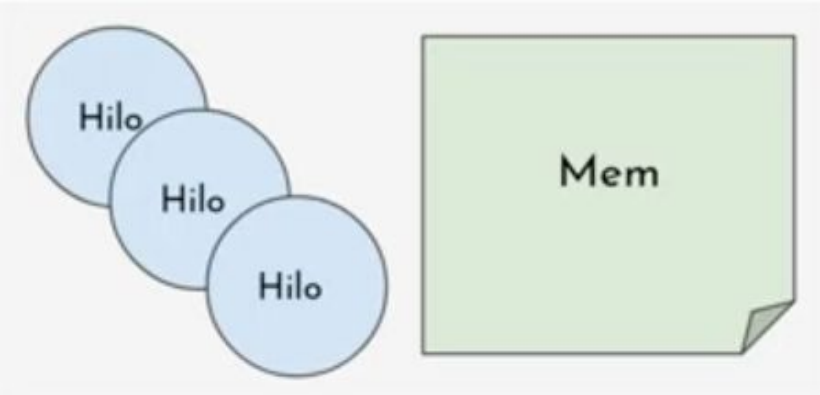
Hacer muchas cosas juntas

Paralelismo

Hacer muchas cosas **al mismo tiempo.**

Es una forma de lograr concurrencia. Pero NO la única.

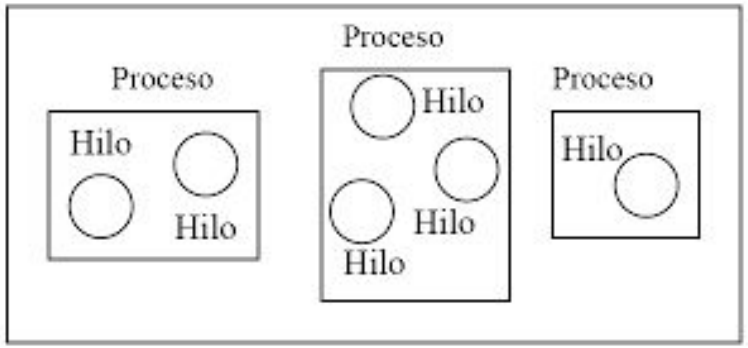
Proceso 1



Proceso 2

¿Por qué necesitamos Multiprocessing?

La diferencia entre hilo y proceso es fácil de reconocer, un hilo ocurre dentro del espacio de memoria de un programa y un proceso es una copia completa del programa, por esta razón, los hilos son rápidos de crear y destruir además de que consumen poca memoria y los procesos son lentos de crear y destruir además de que requieren clonar el espacio de memoria del programa en otro lugar de la RAM, y esto es lento.



1 Core

- 1 proceso-hilo corriendo en cada instante.
- Concurrencia (“un poquito cada uno”) pero no paralelismo
- No puede haber paralelismo, Paralelismo implica N Cores.
- **context switches**: dejar de hacer algo, soltar la data de eso, para

Traerse otro proceso, eso tiene un costo de cpu, no es gratis ese cambio de contexto,

El tiempo en esos context switches.

Si en cualquier momento pausas un programa, para darle paso a otro, puede que pase que me pausen en el medio de algo que es complicado. Ejemplo: lei algo de la base de datos, en el medio algo me cambio algo, (race conditions, locks, etc).

Esperas de I/O

Esperas de operaciones de disco, red, etc.

Estos programas van a pasar un buen tiempo sin necesidad de ejecutar instrucciones. Hago Query a la base de datos, la base de datos tiene que esperar la respuesta de la base de datos y mi programa no puede hacer más nada.

Tiempo donde mi programa no está corriendo instrucciones, sino que está esperando un resultado.

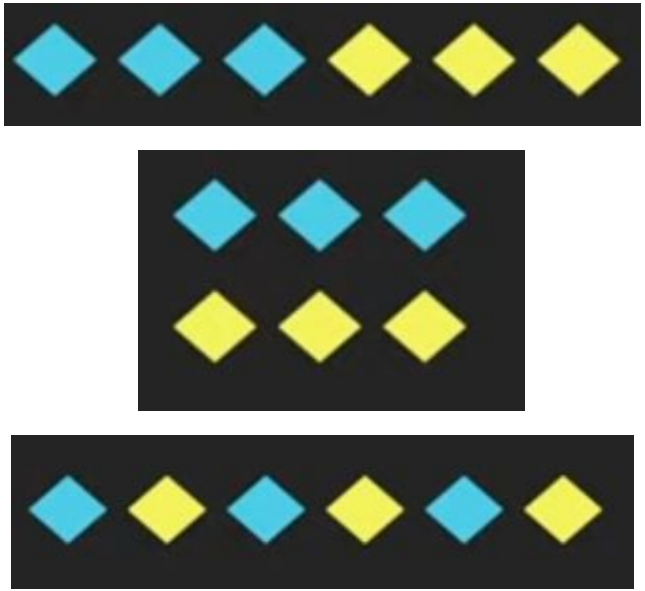
Si un hilo/proceso está **esperando**, ¿tiene sentido darle tiempo de cpu a ese hilo que está esperando algo?

Compartir Datos

Como los procesos tienen su propia RAM y todos los hilos son parte del mismo proceso. Es fácil compartir datos entre hilos de un mismo proceso, pero no es tan fácil compartir datos entre procesos distintos. Si entre word y chrome quieren compartir datos con información que está en la RAM, van a necesitar un mecanismo de área compartida, algo más complicado. Mientras que varios hilos de chrome quieren acceder a la misma data, esa data está en el área donde los hilos pueden acceder.

Programas sin esperas de I/O

2 tareas a realizar, cada una 3 instrucciones que demora 1 segundo de uso de CPU cada una.



En serie, me va a tomar 6 segundos, un solo core

Si tengo 2 cores y puedo correr, procesos o hilos en cada core, puedo reducir el tiempo a la mitad. Con paralelismo reduce los tiempos

Ahora si no tengo paralelismo y quiero hacer concurrencia en 1 core, un ratico cada uno, no importa que intercale un poco cada uno , las hagas en serie, el tiempo va a ser el mismo.

Programas sin esperas de I/O

La concurrencia sin paralelismo no reduce los tiempos.

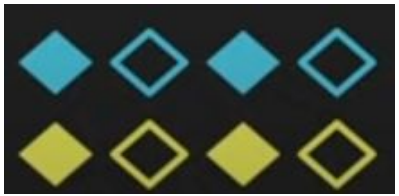
Lo único que ayuda a reducir tiempos es el paralelismo

Programas con esperas de I/O

Es muy distinto la situación en un programa que haga muchas esperas de I/O, como un servidor web, etc.

Concurrencia sin Paralelismo igual ayuda a acelerar.

La azul tiene 2 instrucciones, quiero leer algo de disco, espera que se lea el disco.



En paralelo gano tiempo

Mientras escribo en amarillo, la espera de la azul, el disco puede hacer cosas.

Lo más optimo es combinar ambas cosas!

Paralelismo con varios cores, concurrencia en cada core, para aprovechar los tiempos muertos

Hablemos de python: Multi threading

Módulo threading que permite lanzar hilos y controlarlos

Paralelismo?

No es CPython! (el oficial) tiene algo que se llama la GIL, correr solo 1 hilo por instancia de tiempo.

Si en otros pythons (N cores)

Hablemos de python: Multi threading

Los hilos comparten memoria

Ej: creo una lista en un hilo y agrego items desde otro, sin hacer nada raro.

Pero en cualquier momento puede ocurrer un context switch, porque puede alternar con otro hilo que afecta la lista, y el al volver puede fallar.

Hablemos de python: Multi threading

En casos de que muchas tareas (hilos) o esperas de I/O muy largas, hay más context switch que no sea eficiente. Vuelves al hilo y te dice no puedo, estoy esperando y vuelves a otra a memoria y te dice que también está esperando.

Hablemos de python: Multiprocessing

Simple! El módulo multiprocessing nos permite lanzar procesos y controlarlos.

Paralelismo?

Si hay N cores si!

Hablemos de python: Multiprocessing

Cada proceso tiene su memoria.

Si quiero compartir datos? Python te da algunas herramientas para hacer eso, pero se empieza a complicar.

En casos de muchos procesos o esperas de I/O muy largas, hace que no sea eficiente.

Hablemos de python: Multiprocessing

Con este tengo paralelismo, a diferencia de multi threading, pero con este tengo sobrecarga cada vez que quiera hacer comunicación entre procesos para compartir datos.

Async

Hay un solo proceso, un solo hilo. No hay multiprocessing, no hay threading. Lo que tengo son corrutinas.

Yo voy a tener N Corrutinas dentro de mi hilo que yo quiero que se vayan ejecutando.

Corrutinas: funciones que pueden pausar para dejar que otras corran.

La diferencia es que vamos a decir explicitamente en que lugares esa función se pausa.

Ni en Multiprocessing, ni en Multi threading

Ni en Multithreading, ni en multiprocessing se tenía este control. En cualquiera de estos dos, en cualquier momento de mi código se puede pausar y que le toque a otro.

En Async eso no pasa, solo se pausa donde yo digo en el código donde se va a pausar. Vamos a tener un loop que va diciendo ahora le toca a esta tarea hasta que haga una pausa, ahora le toca a esta otra hasta que haga una pasa. **Es una concurrencia colaborativa**

Async es una Concurrencia o Multitarea Colaborativa

Las corrutinas son las que te dicen ahora te dejo que me pauses para que dejes que el resto se pueda ejecutar. Si una dice yo no pauso nunca, más ninguna puede correr.

Threading es multitarea apropiativa, ya que SO decide.

Async

Es un solo proceso/hilo, está genial porque la memoria es compartida y no tengo que hacer nada raro para manejar elementos compartidos.

Si es un solo hilo SOLO Concurrencia, NO Paralelismo.

Async

Si es un solo hilo SOLO Concurrencia, NO Paralelismo.

Tengo muchas cosas corriendo, pero nunca dos cosas corriendo al mismo tiempo, pero lo bueno es que nosotros decidimos cuando son los cambios de contexto entre corrutinas y hasta cuando.

Ej: Pasa esta corrutina hasta que termine de escribirse tal dato al disco.

¿Entonces por qué no todos usan async y nos olvidamos de las otras cosas?

Ventajas:

- Es más eficiente, sobre todo en la carga de tareas o I/O.
- Es más seguro y controlable.

Desventajas:

- Es más complicado escribir un buen código async.
- No todas las librerías lo soportan.

Comparativa de las 3

Concurrency Type	Switching Decision	Number of Processors
Pre-emptive multitasking (`threading`)	The operating system decides when to switch tasks external to Python.	1
Cooperative multitasking (`asyncio`)	The tasks decide when to give up control.	1
Multiprocessing (`multiprocessing`)	The processes all run at the same time on different processors.	Many

Tienes un programa sin espera de I/O?

No hay nada que pensar:

Necesitas multiprocessing y necesitas N Cores.

Tienes un programa con espera de I/O?

Depende

Si la performance es muy importante.

Nada le gana a Async + Multiprocessing, pagando el costo de la complejidad.

En cada core un proceso, que con async le sacas el jugo al máximo de ese core.

Si el control de los context switches es importante:

Async suele hacerlo más natural, una vez que lo aprendiste a usar.

Si no te importanta la performance y no necesitas el control fino

Entonces Multithreading o Multiprocessing, depende.

Si tienes mucha data compartida, una lista compartida, te vas por Multithreading que es más fácil de usar.

Si no tienes data compartida, te vas por Multiprocessing.

Vamos a hacer un ejemplo de async primero con generadores (simulando la pausa) y luego con la libería async

Con los generadores tengo funciones pausables y eso me Viene bien para explicar async.