Event Loops

Computación Tolerante a Fallas

Alumno: Ochoa Díaz Raúl Francisco

Código: 220790776

Profesor(a): Dr. Michel Emanuel López Franco

Sección: D06

Fecha de Entrega: 27 de septiembre de 2022

Escalabilidad y *Event Loops*

Podemos definir al bucle de eventos (*event loop*) como una técnica que nos permite dejar que un proceso se realice en segundo plano y una vez haya terminado, se entregue el resultado al programa.

Una buena analogía para entenderlo es cocinar algo. Supongamos que debemos preparar pollo y arroz; los pasos que debemos seguir serian: conseguir el pollo, conseguir el arroz, preparar el pollo, preparar el arroz, y emplatar. Si tuviésemos que esperar a que se complete un paso antes de pasar al otro, tardaríamos mucho tiempo en cocinar todo. Podemos cocinar ambas cosas al mismo tiempo, siempre vigilando cuando alguna de las dos este listo para detener la cocción. Los procesos se pueden trabajar de forma similar mediante los bucles de evento y las funciones asíncronas.

Para ponerlo a prueba, hice un script en Python el cual hace llamadas a una API de valores de mercado, que es parte de un proyecto en el que estuve trabajando. Primero, lo realice síncronamente.

```python
def request () -> None:
    inicio = time.time ()
    for empresa in empresas:
        respuesta = requests.get (url)
        datos.append (respuesta.json ())
    fin = time.time ()

    tiempoTotal = fin - inicio

    print (f"Tomo {tiempoTotal} segundos hacer la llamada a la API.")
```

Usando el modulo time, obtengo el tiempo actual de la maquina en segundos. Al final, simplemente resto el inicio y el final para obtener el tiempo total.

Realizar 5 consultas toma un total de 2 segundos.

```
Consultando AAPL...
Consultando GOOG...
Consultando TSLA...
Consultando MSFT...
Consultando AAPL...
```

```
Tomo 1.9874966144561768 segundos hacer la llamada a la API.
```

Para hacerlo asíncronamente, debemos importar los módulos *asyncio,* y *aiohttp.* Luego, convertimos la función en asíncrona utilizando la palabra reservada *async.*

```
async def request () → None:
```

Usaremos el modulo *aiohttp* para realizar las consultas en lugar de *requests.* Esto debido a que nos permite trabajar de mejor manera junto a *asyncio.* Debemos crear una sesión, para lo que usamos la clausula *with* con el fin de evitar fallos y poder cerrarla automáticamente.

```
async with aiohttp.ClientSession () as sesion:
```

Con una función síncrona, ponemos en una lista todas las consultas que se van a realizar a la API.

```
def getRequests (sesion) → list:
    peticiones = []

    for empresa in empresas:
        peticiones.append(asyncio.create_task (sesion.get (url.format (empresa, apiKey), ssl=False)))

    return peticiones
```

Una vez tenemos la lista, usamos la función *gather* para ir realizando las consultas. El asterisco antes del identificador sirve para iterar sobre cada posición de memoria de la lista. Se tiene que usar *await* junto a la función para evitar que el programa continue su ejecución sin tener resultados válidos. Al final, se itera sobre las respuestas para convertirlas en JSON, sin olvidar agregar *await.*

```
async with aiohttp.ClientSession () as sesion:
    peticiones = getRequests (sesion)
    respuestas = await asyncio.gather (*peticiones)

    for respuesta in respuestas:
        datos.append (await respuesta.json ())
```

La función anterior se ejecuta dentro de un event loop. Este se crea usando la función *run* de asyncio.

```
asyncio.run (request ())
```

Modifique un poco el mensaje que se imprime al terminar la ejecución para mostrar cuantas consultas se están realizando.

```
print (f"Tomo {tiempoTotal} segundos realizar {len (empresas)} consultas a la API.")
```

Probamos el programa, esta vez con más consultas a la API, para ver cuánto tiempo tarda.

```
Tomo 0.6328489780426025 segundos realizar 15 consultas a la API.
```

Como vemos, estamos realizando el triple de consultas y son obtenidas en menos de la mitad del tiempo. Una mejora de rendimiento considerable. Trabajar de este modo se asemeja a los hilos y demonios, donde se ejecutan procesos independientemente y el programa no espera a que cada uno termine su ejecución para continuar con el siguiente.

Una captura del script completo.

```python
import asyncio, aiohttp, time

apiKey = 
url = "https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol={}&interval=5min&apikey={}"
empresas = ['AAPL', 'GOOG', 'TSLA', 'MSFT', 'AAPL','AAPL', 'GOOG', 'TSLA', 'MSFT', 'AAPL','AAPL', 'GOOG', 'TSLA', 'MSFT', 'AAPL']
datos = []

inicio = time.time ()

def getRequests (sesion) -> list:
    peticiones = []

    for empresa in empresas:
        peticiones.append(asyncio.create_task (sesion.get (url.format (empresa, apiKey), ssl=False)))

    return peticiones

async def request () -> None:
    #Usando una funcion asincrona, no necesitamos esperar a que termine el proceso para avanzar.
    #La clausula with permite cerrar la sesion automaticamente y prevenir que se ejecute el codigo en caso de que no se abra.
    async with aiohttp.ClientSession () as sesion:
        peticiones = getRequests (sesion)
        respuestas = await asyncio.gather (*peticiones)

        for respuesta in respuestas:
            datos.append (await respuesta.json ())

asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
asyncio.run (request ())

fin = time.time ()
tiempoTotal = fin - inicio

print (f"Tomo {tiempoTotal} segundos realizar {len (empresas)} consultas a la API.")
```

Conclusiones

Esta practica fue un poco desafiante, ya que desde que comencé a aprender desarrollo web comprender las funciones asíncronas ha sido de los temas que más me ha costado. Aunque aun no puedo decir que lo entiendo completamente, leer la documentación proveída por el profesor me ayudo bastante. Además, estuve revisando otros textos y videos del internet que fueron bastante útiles.

Quizá pude haber construido un programa con un grado de complejidad más alto, pero creo que esto es suficiente. Espero poder seguir aprendiendo sobre estos temas de programación escalable y descentralizada.

Fuentes de Información

https://realpython.com/async-io-python/#the-asyncio-package-and-asyncawait

https://www.educative.io/blog/scaling-in-python