

Universidad de Guadalajara
Sistema de Educación Media Superior
Centro Universitario de Ciencias Exactas e Ingenierías



An Introduction to Scaling Distributed Python Applications

Materia: Computación Tolerante a Fallas

D06 2023 B

Alumno: Esquivel Barbosa Diego Humberto

Código: 211401635

Carrera: INCO

Fecha: 10/09/2023

Introducción

La computación tolerante a fallos se centra en el diseño, desarrollo y despliegue de sistemas que tienen la capacidad de resistir, mitigar y recuperarse de fallos y errores, manteniendo su funcionalidad esencial en situaciones adversas. Estos sistemas no solo aspiran a evitar la interrupción total, sino que también buscan ofrecer una operación confiable en escenarios donde componentes individuales pueden experimentar problemas. Ya sea en aplicaciones críticas para la seguridad, sistemas médicos, redes de comunicación o dispositivos de consumo, la tolerancia a fallos se ha convertido en un aspecto esencial para garantizar la integridad y la confianza en la tecnología que utilizamos diariamente.

Hilos (Threads): Los hilos son unidades de ejecución más pequeñas que un proceso. Un proceso puede contener múltiples hilos, y estos comparten el mismo espacio de memoria. Los hilos permiten que las aplicaciones realicen múltiples tareas de manera aparentemente simultánea, lo que mejora la capacidad de respuesta y el rendimiento. Los hilos son ideales para tareas que pueden ejecutarse en paralelo, como la gestión de solicitudes de servidores web o la actualización de interfaces de usuario.

Procesos: Un proceso, por otro lado, es una instancia independiente de un programa en ejecución. Cada proceso tiene su propio espacio de memoria y recursos asignados. Los procesos pueden ejecutarse en paralelo y, a menudo, se utilizan para separar tareas que deben ser independientes o para aprovechar al máximo los múltiples núcleos de las CPU modernas. Los procesos también proporcionan un alto grado de aislamiento entre las aplicaciones, lo que mejora la seguridad y la estabilidad del sistema.

Demonios (Daemons): Los demonios son procesos en segundo plano que se ejecutan sin una interfaz de usuario visible. Por lo general, se utilizan en sistemas operativos Unix y Linux para realizar tareas de mantenimiento, como la administración de servicios o la gestión de registros. Los demonios son especialmente útiles para garantizar que ciertas funciones se ejecuten de manera continua sin intervención del usuario.

Concurrencia: La concurrencia se refiere a la capacidad de un sistema para realizar múltiples tareas de manera aparentemente simultánea. Tanto los hilos como los procesos son herramientas clave para lograr la concurrencia. La concurrencia es esencial en aplicaciones modernas, como servidores web, aplicaciones de bases de datos y videojuegos, donde múltiples tareas deben ejecutarse al mismo tiempo para brindar una experiencia fluida al usuario.

Código

```
import threading
import time

# Función que simula un proceso que toma un tiempo en ejecutarse
def proceso(id, duracion):
    print(f"Proceso {id} iniciado.")
    time.sleep(duracion)
    print(f"Proceso {id} completado.")

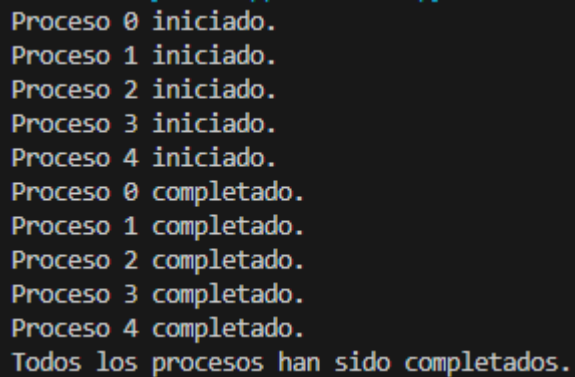
# Lista de datos que se procesarán en hilos
datos = [2, 3, 5, 7, 11]

# Crear una lista para almacenar los hilos
hilos = []

# Crear y empezar un hilo para cada dato
for i, dato in enumerate(datos):
    hilo = threading.Thread(target=proceso, args=(i, dato))
    hilos.append(hilo)
    hilo.start()

# Esperar a que todos los hilos terminen
for hilo in hilos:
    hilo.join()

print("Todos los procesos han sido completados.")
```



```
Proceso 0 iniciado.
Proceso 1 iniciado.
Proceso 2 iniciado.
Proceso 3 iniciado.
Proceso 4 iniciado.
Proceso 0 completado.
Proceso 1 completado.
Proceso 2 completado.
Proceso 3 completado.
Proceso 4 completado.
Todos los procesos han sido completados.
```

Explicación

Importación de módulos: Importamos el módulo `threading` que proporciona funcionalidad para trabajar con hilos en Python.

Función `procesar_datos`: Creamos una función que simula el procesamiento de datos. En este ejemplo, multiplica cada elemento de la lista de datos por 2 y simula un tiempo de procesamiento de 2 segundos. Esto representa una tarea de procesamiento ficticia.

Bloque `if __name__ == '__main__':`: Esta es una construcción común en Python para asegurarse de que el código en su interior solo se ejecute si el script se ejecuta directamente (no cuando se importa como módulo).

Datos de muestra: Creamos una lista llamada `datos_a_procesar` que contiene números del 0 al 999999. Esto representa los datos que queremos procesar.

División de datos: Dividimos los datos en partes iguales para que cada hilo maneje una porción. El número de hilos se define como 4 en este ejemplo.

Creación de hilos: Creamos hilos para procesar cada parte de los datos. Cada hilo ejecuta la función `procesar_datos` con su propia porción de datos y un identificador único.

Inicio de hilos: Iniciamos todos los hilos para que comiencen a ejecutar sus tareas en paralelo.

Espera a que los hilos terminen: Utilizamos el método `join()` en cada hilo para esperar a que todos los hilos terminen su procesamiento antes de continuar.

Combinación de resultados: Combinamos los resultados de cada hilo en una lista final llamada `resultado_final`.

Impresión de resultados: Imprimimos algunos de los resultados procesados para verificar que el programa funcione correctamente.

Conclusión

Este programa en Python ofrece una ilustración práctica del uso de hilos para lograr la concurrencia en tareas que pueden ejecutarse simultáneamente. En este escenario, se simuló el procesamiento de una lista de datos, donde cada elemento representa una tarea con una duración diferente. La utilización de hilos permite que estas tareas se ejecuten en paralelo, lo que resulta en una mejora significativa en la eficiencia y el rendimiento del programa.

El programa resalta la importancia de la administración de hilos, abordando aspectos como la creación, el inicio y la espera de hilos, para garantizar que todos los procesos se completen antes de continuar. A través de esta implementación, se puede comprender cómo Python simplifica la programación concurrente mediante la biblioteca `threading`.

Bibliografia

- Laprie, J. C. (1985). Dependability: Basic Concepts and Terminology. Springer. DOI: 10.1007/978-0-387-35031-4
- Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, 1(1), 11-33. DOI: 10.1109/TDSC.2004.2
- Bondavalli, A., Di Giandomenico, F., & Trivedi, K. S. (2002). Modeling and Assessment of Fault Tolerance Approaches. IEEE Transactions on Computers, 51(5), 548-560. DOI: 10.1109/TC.2002.1004593
- Chandra, S., Hadzilacos, V., & Toueg, S. (1996). The Weakest Failure Detector for Solving Consensus. Journal of the ACM, 43(4), 685-722. DOI: 10.1145/234533.234549