Actividad: ¿Son primos? ¡Multiprocesa y comprueba!

Objetivo

Aprender a usar el módulo multiprocessing para determinar simultáneamente si varios números son primos, mejorando el rendimiento cuando la lista es extensa o sus elementos son números grandes.

Paso 1: Función para determinar si un número es primo

1. Abre tu editor y crea el archivo primo_multiproceso.py.
2. Implementa la función clásica en Python para comprobar si un número es primo

```python
def es_primo(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
```

Paso 2: Lista de números a analizar

1. Añade una lista de números, por ejemplo:

```
numeros = [29, 97, 221, 541, 104729, 1234567, 999331, 25, 2, 13]
```

Paso 3: Multiproceso con Pool

1. Importa el módulo **multiprocessing**.
2. Usa la clase **Pool** para mapear la función **es_primo** sobre la lista de números:

```python
from multiprocessing import Pool

if __name__ == '__main__':
    with Pool() as pool:
        resultados = pool.map(es_primo, numeros)
    # Mostrar los resultados
    for numero, resultado in zip(numeros, resultados):
        print(f"{numero} es primo: {resultado}")
```

Paso 4: Ejecuta y analiza

1. Guarda el archivo y ejecuta en la terminal:

```
python primo_multiproceso.py
```

2. Observa cómo los números se evalúan en paralelo y revisa el rendimiento si pruebas con listas más largas o números más grandes.

Paso 5: Variaciones y conclusiones

- Modifica la lista de entrada agregando más números.
- Cambia el tamaño del Pool para experimentar
- Cronometra el tiempo de ejecución para ver la diferencia frente a la versión secuencial (usa el módulo time).

Código completo de referencia

```python
from multiprocessing import Pool

def es_primo(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

numeros = [29, 97, 221, 541, 104729, 1234567, 999331, 25, 2, 13]

if __name__ == '__main__':
    with Pool() as pool:
        resultados = pool.map(es_primo, numeros)
    for numero, resultado in zip(numeros, resultados):
        print(f"{numero} es primo: {resultado}")
```

Conclusión:

La programación multiproceso permite aprovechar varias CPU y analizar números de forma concurrente, reduciendo el tiempo total, especialmente cuando se trata de tareas computacionalmente intensivas.