

Exercices — Threading vs Multiprocessing

Exercice : threading ou multiprocessing ?

Pour chaque situation, indiquez si vous utiliseriez **threading**, **multiprocessing**, ou **aucun des deux** (séquentiel suffit). Justifiez en une phrase.

1. Vous développez un script qui télécharge 50 fichiers PDF depuis un serveur distant.
2. Votre programme doit appliquer un filtre de flou sur 500 images haute résolution.
3. Vous écrivez un chatbot qui écoute les messages de 10 utilisateurs simultanés sur un serveur socket.
4. Votre script lit un fichier CSV de 20 lignes et affiche le résultat dans la console.
5. Vous devez entraîner 4 modèles de machine learning indépendants sur des jeux de données différents.
6. Votre application web doit envoyer un email de confirmation et une notification Slack à chaque inscription.
7. Vous écrivez un script qui calcule les nombres premiers jusqu'à 10 millions, en découpant la plage en 4 sous-plages.
8. Votre programme doit surveiller 3 dossiers en même temps et réagir quand un nouveau fichier apparaît.

Exercice 2 — Téléchargeur vs compresseur : mesurer le bon outil

Cet exercice vous fait écrire du vrai code pour les deux cas — IO-bound et CPU-bound — et comparer les résultats en termes de temps.

Partie A — IO-bound : téléchargement concurrent

Vous avez une liste de 8 URLs publiques à interroger (utilisez `requests.get(url).status_code`). Simulez un délai réseau en ajoutant `time.sleep(1)` dans la fonction de téléchargement.

1. Écrivez une version **séquentielle** qui traite les 8 URLs l'une après l'autre et mesure le temps total.
2. Écrivez une version **avec threading** (un thread par URL) qui les traite en parallèle et mesure le temps total.
3. Affichez pour chaque URL : son statut et quel thread l'a traitée.

Résultat attendu : la version threading doit être ~8x plus rapide.

```
URLS = [  
    "https://httpbin.org/get",  
    "https://httpbin.org/ip",  
    "https://httpbin.org/user-agent",  
    "https://httpbin.org/headers",  
    "https://httpbin.org/uuid",  
    "https://httpbin.org/base64/SwYgeW91IGNhbiByZWFKIHRoaXMsIGJyYXZvICE=",  
    "https://httpbin.org/bytes/100",  
]
```

```
] "https://httpbin.org/delay/0",
```

Partie B — CPU-bound : calcul de hachage en parallèle

Vous devez calculer le hash SHA-256 d'un grand nombre de chaînes (simulez un travail lourd).

```
import hashlib

def calculer_hash(donnee):
    for _ in range(500_000): # travail CPU intentionnellement lourd
        hashlib.sha256(donnee.encode()).hexdigest()
    return hashlib.sha256(donnee.encode()).hexdigest()

DONNEES = ["alpha", "beta", "gamma", "delta"]
```

1. Écrivez une version **séquentielle** qui calcule les 4 hashes l'un après l'autre.
2. Écrivez une version **avec threading** (un thread par hash).
3. Écrivez une version **avec multiprocessing** (un process par hash).
4. Comparez les trois durées. Expliquez pourquoi le threading n'aide pas ici.

Ce que vous devez observer : séquentiel \approx threading, multiprocessing \approx séquentiel / nombre de cœurs.