Vlákna, Paralelní programování, Asynchronní metody, Concurrent design patterns

Vlákna

Vlákna je základní jednotka, kterou operační systém může spustit a naplánovat její vykonávání. Představuje nezávislý tok instrukcí, který běží v rámci procesu a sdílí s ním stejný paměťový prostor. Každá aplikace má minimálně jedno hlavní vlákno, ve kterém probíhá základní vykonávání programu. Pro zvýšení výkonu můžeme vytvořit další vlákna, které poběží současně s hlavním vláknem. Vlákna běží na jádrech procesoru - pokud procesor má více jader, mohou vlákna běžet skutečně paralelně, jinak se střídají. Sdílejí stejný paměťový prostor (proměnné, objekty). Mají vlastní zásobník.

Procesy

Narozdíl od vláken to jsou samostatné programy s vlastní pamětí. Jsou vzájemně izolované. Často jsou systémově náročnější a probíhá složitější komunikace mezi procesy

```python
import threading
import time

def vzestupne():
    for i in range(1, 10):
        print(f"Vzestupně: {i}")
        time.sleep(0.5)  # Krátká pauza pro lepší čitelnost výstupu

def sestupne():
    for i in range(9, 0, -1):
        print(f"Sestupně: {i}")
        time.sleep(0.5)  # Krátká pauza pro lepší čitelnost výstupu

# Vytvoření vláken
vlakno1 = threading.Thread(target=vzestupne)
vlakno2 = threading.Thread(target=sestupne)

# Spuštění vláken
vlakno1.start()
vlakno2.start()

# Počkání na dokončení vláken
vlakno1.join()
vlakno2.join()

print("Hotovo!")
```

Paralelní programování
Paralelní programování je způsob, jak psát programy paralelně. Je opakem sekvenčního, které instrukce zpracovává postupně jednu po druhé. Paralelizace se využívá v případě, že máme nějaký algoritmus na řešení šíleného a náročného problému. My jej rozdělíme na několik různých částí, které každé budou najeno počítač a pak společně problém vyřeší rychleji. Díky tomu nějaké části kódu nemusejí čekat na něco, co jím stejně nepředá data.

Typické problémy paralelního programování

1. Race condition - nastává, když výsledek závisí na pořadí operací různých vláken - dvoje vlákna současně aktualizují stejnou proměnnou, řeší se zámky

2. Deadlock - situace, kdy vlákno A čeká na zdroj držený vláknem B, zatímco vlákno B čeká na zdroj držený vláknem A, řeší se správným pořadím zamykání a časovými limity

3. Starvation - vlákno nemůže získat potřebné zdroje, protože jiná vlákna je neustále zabírají, řeší se prioritami

Asynchronní metody

Asynchronní metody jsou takové metody, které neblokují svým spuštěním chod aplikace. Místo čekání na dokončení operace, program pokračuje v činnosti a k výsledku se vrátí později. Příkladem může být třeba spojení s databázovým serverem – Pokud máme slabé internetové spojení a bude komunikace a spojení trvat dlouho, třeba tři vteřiny, asynchronní metoda se bude s databázovým serverem spojovat tři vteřiny, ale program nebude tři vteřiny čekat, ale rovnou bude pokračovat v běhu a načítání různých dalších komponent.

<u>Hlavní koncepty</u>

Asynchronní operace - operace, které mohou běžet nezávisle na hlavním toku programu

Callback - funkce, která se zavolá po dokočení asynchronní operace

Promise/Future - objekt reprezentující budoucí výsledek asyncrhonní operace

Event loop - řídí provádění asynchronních operací

V pythonu se používá klíčové slovo *async* pro označení asynchronních funkcí a *await* pro čekání na výsledek

Concurrent design patterns

Concurrent design patterns (návrhové vzory pro souběžnost) jsou osvědčené postupy pro řešení typických problémů při vývoji vícevláknových a asynchronních aplikací - program běží na několika vláknech najednou a řeší souběžně určité operace.

Nejznámější concurrent návrhové vzory

Thread Pool

Návrhový vzor, který spravuje skupinu pracovních vláken. Místo vytvoření nového vlákna pro každou úlohu, jsou úlohy přiřazování existujícím vláknům z poolu. Jsou seřazené ve frontě a hlavní výhodou je dosažení konkurence - stav, ve kterém se výpočty a části programů mohou řešit mimo normální stanovený průběh

Producer-Consumer

Odděluje generování dat od jejich zpracování. Výrobce vytváří data a ukládá je do sdílené datové struktury (fronty), zatím co spotřebitel je odebírá a zpracovává

Read-Write Lock

Read-Write Lock je synchronizační mechanismus, který rozlišuje mezi čtením a zápisem dat. Umožňuje více vláknům současně číst sdílená data, zatímco zápis je vždy exkluzivní (pouze jedno vlákno). Tím se zvyšuje výkon v aplikacích, kde převažuje čtení nad zápisem, jako jsou databáze nebo cache. Základní princip spočívá v tom, že čtení dat je bezpečné i když probíhá paralelně, ale zápis musí být izolovaný, aby nedošlo k poškození dat.