Raport
PYTHON01

Jan Kwinta

21 grudnia 2025

Wstęp

Celem ćwiczenia jest porównanie działania: kodu sekwencyjnego, wieloprocesowego (multiprocessing) oraz wielowątkowego (threading) w dwóch środowiskach: Python z GIL oraz Python skompilowany w wersji free-threaded (bez *Global Interpreter Lock*).

Do raportu dołączony jest kod python01.py, którego zadaniem jest wygenerowanie słownika

```
{ "id": ..., "text": "..."}
```

oraz wykonanie pewnych operacji na nim: policzenia liczby słów w każdym tekście, liczby unikatowych liter oraz score - liczby samogłosek. Dla każdego tekstu trzymamy wyniki w tablicy results.

Przykładowo dla rekordu

```
{'id': i, 'text': 'eqqkvk cpiylsj xzhgxw vavuq
vhvez bbkgtvc tya gsiwjq ansk hurxb'}
```

w tablicy results będzie się znajdowało

```
results[i] = {'word_count': 10, 'unique': 22, 'score': 11}
```

Skrypt i wyniki

W odpowiednio przygotowanym środowisku skrypt był uruchamiany za pomocą komend:

- `python3 -X gil=1 python01.py`
- `python3 -X gil=0 python01.py`

Przykładowe wyniki tych poleceń to odpowiednio:

```
GIL = True
Sekwencyjnie: 2.7484s
Multiprocessing: 0.8269s | Czy poprawny: True
Threading (bez zabezpieczen): 3.1285s | Czy poprawny: True
Threading (z synchronizacja): 3.3266s | Czy poprawny: True
```

oraz

```
GIL = False
Sekwencyjnie: 2.7488s
Multiprocessing: 0.8695s | Czy poprawny: True
Threading (bez zabezpieczen): 1.1311s | Czy poprawny: True
Threading (z synchronizacja): 1.4181s | Czy poprawny: True
```

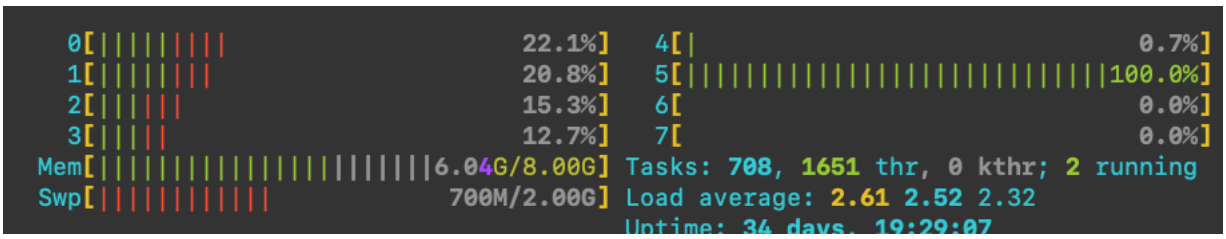
Poprawność obliczeń

W teorii w wynikach wersji wielowątkowej bez żadnych blokad mogą pojawiać się nieścisłości, błędy lub pominięte rekordy. Mimo eksperymentowania z obciążeniem przy liczeniu, liczbą wątków i rekordów ani razu nie udało mi się wykryć takiego błędu, nawet przy stworzeniu idealnego miejsca, w którym może wystąpić *race condition*:

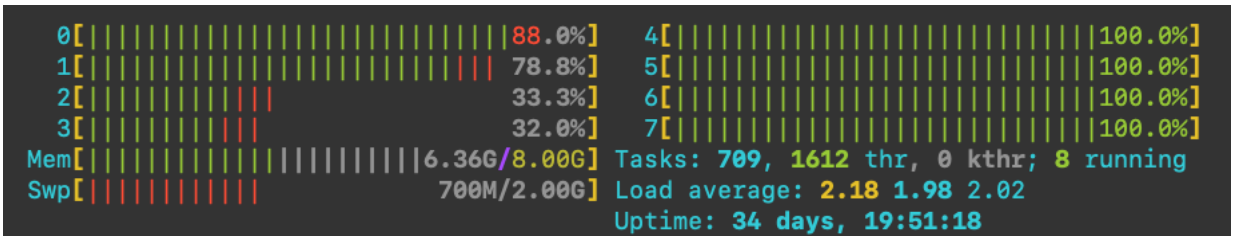
```
def worker():
    while True:
        try:
            record = shared_data_list.pop()
        #...
        rid, res = process_record(record)
        results[rid] = res
#...

for _ in range(num_threads):
    t = threading.Thread(target=worker)
    #...
    t.start()
```

Obciążenie procesora

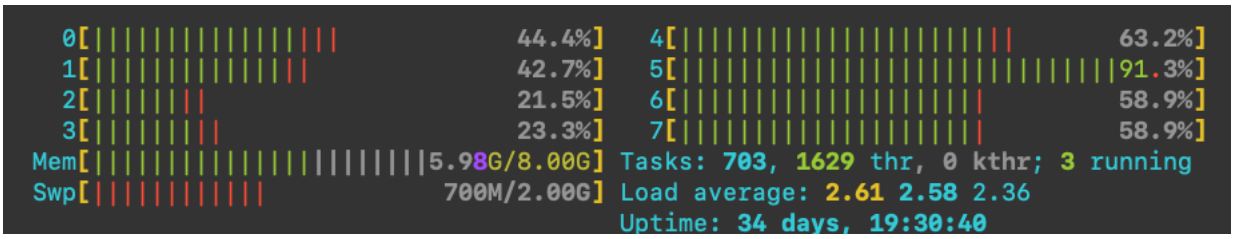
W czasie różnych obliczeń zmieniało się obciążenie poszczególnych rdzeni. Poniżej zrzuty ekranu z htop przy wykonywaniu obliczeń skrytpu dla GIL=0.



Obciążenie CPU przy obliczeniach sekwencyjnych.



Obciążenie CPU przy obliczeniach wieloprocesowych.



Obciążenie CPU przy obliczeniach wielowątkowych.

Interpretacja wyników

Dla obliczeń sekwencyjnych i wieloprocesowych czas dla wersji pythona z blokadą globalną i bez nie zmienił się dużo. Obliczenia wieloprocesowe, które zajmowały wszystkie rdzenie CPU były szybsze niż sekwencyjne.

Obliczenia wielowątkowe z GIL były podobne, a nawet trochę wolniejsze niż sekwencyjne. GIL sprawia, że w każdej jednostce czasu tylko jeden z wątków wykonuje obliczenia. Pozostałe wątki czekają na swoją kolej.

Wyłączenie GIL, możliwe w Pythonie 3.13, sprawia, że wiele wątków może wykonywać kod Pythona w tym samym czasie na różnych rdzeniach. Przyspiesza to znacząco czas w stosunku do wersji sekwencyjnej i zajmuje wszystkie rdzenie procesora - nie jest jednak tak wydajne (ani tak obciążające) jak wersja multiprocess, ponieważ Python musi zarządzać wieloma małymi blokadami (tzw. *fine-grained locking*) wewnątrz obiektów.

Podsumowanie

Różnice pomiędzy podejściem wieloprocesowym a wielowątkowym (bez GIL) można podsumować w tabelce:

	Multiprocessing	**Threading**
Pamięć	Osobna	Współdzielona
Komunikacja	Trudna	Współdzielone obiekty
Zużycie RAM	Wysokie	Niskie
Bezpieczeństwo	Brak *race conditions*	Wymaga użycia blokad