01.1-Lab Python concurrency

November 30, 2022

1 Lab Processamento Paralelo

1.1 Introdução

Execute os programas propostos e responde as questões

2 Programas I/O Bound

2.1 Versão sincrona

[ ]:
```python
# io_bound/synchronous.py
import requests
import time

def get_session():
    return requests.Session()

def download_site(url):
    session = get_session()
    with session.get(url) as response:
        indicator = "J" if "jython" in url else "R"
        print(indicator, sep='', end='', flush=True)

def download_all_sites(sites):
    for url in sites:
        download_site(url)

    print()

if __name__ == '__main__':
    sites = [
        "https://www.jython.org",
        "http://olympus.realpython.org/dice",
    ] * 80

    print("Starting downloads")
    start = time.time()
    download_all_sites(sites)
```

```
    duration = time.time() - start
    print(f"Downloaded {len(sites)} sites in {duration} seconds")
```

2.1.1 Perguntas:

1. Descreva o funcionamento do programa:
2. Será que cada vez que executa o programa o tempo varia?

2.2 Versão asincrona - Uso de thread

[ ]:
```
# io_bound/threaded.py
import concurrent.futures
import requests
import threading
import time

thread_local = threading.local()

def get_session():
    if not hasattr(thread_local, "session"):
        thread_local.session = requests.Session()

    return thread_local.session

def download_site(url):
    session = get_session()
    with session.get(url) as response:
        indicator = "J" if "jython" in url else "R"
        print(indicator, sep='', end='', flush=True)

def download_all_sites(sites):
    with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
        executor.map(download_site, sites)

if __name__ == '__main__':
    sites = [
        "https://www.jython.org",
        "http://olympus.realpython.org/dice",
    ] * 80

    print("Starting downloads")
    start = time.time()
    download_all_sites(sites)
    duration = time.time() - start
    print(f"\nDownloaded {len(sites)} sites in {duration} seconds")
```

2.2.1 Perguntas:

1. Relativamente a versão sincrona como variou o tempo?.
2. **Resposta** é significativamente mais rápido do que antes - quase dez vezes.
3. Analise o padrão de J e R obtido na saída, relativamente a versão anterior.
4. **Resposta:** Uma coisa a notar aqui são os padrões do J e R. No programa síncrono, era sempre J depois R, J depois R. Neste programa, não é, e isso é porque as threads ficam à espera durante diferentes períodos de tempo.

2.3 Condições de Corrida

Execute o programa no qual race é chamdo com vários valores de entrada.

[ ]:
```python
# io_bound/race.py
from concurrent.futures import ThreadPoolExecutor
counter = 0

def change_counter(amount):

    global counter
    for _ in range(1000):
        counter += amount

def race(num_threads):
    global counter
    counter = 0
    data = [-1 if x %2 else 1 for x in range(1000)]

    with ThreadPoolExecutor(max_workers=num_threads) as executor:
        executor.map(change_counter, data)

    print(counter)

if __name__ == "__main__":
    race(1)
    race(1)
    race(1)
    race(2)
    race(2)
    race(2)
```

2.3.1 Perguntas:

1.Explique o resultados obtidos quando é chamado race(0)

1.Explique o resultados obtidos quando é chamado race(2)

[ ]:
```
```

3 Programas CPU Bound

Execute cada um dos seguintes programas e responda as seguintes questões:

1. Descreva Porque estes programas são considerados programas CPU Bound e não I/O Bound.
2. Se executar varias vezes o mesmo programa, verifica alguma alteração no resultado obtido em termos de tempo de execução?. Qual é a sua explicação.
3. Ordene o tempo de execução de cada dos programas.
4. Tente aumentar o número de max_workers na versão assincrona. O que verifica em termos de desempenho.
5. Apresente uma explicação para a diferença de tempo obtidos.

3.1 Versão sincrona

[ ]:
```python
# cpu_bound/synchronous.py
import time

def calculate(limit):
    return sum(i * i for i in range(limit))

def find_sums(numbers):
    for number in numbers:
        calculate(number)

if __name__ == '__main__':
    numbers = [5_000_000 + x for x in range(20)]

    print("Starting calculation")
    start = time.time()
    find_sums(numbers)
    duration = time.time() - start
    print(f"Duration {duration} seconds")
```

3.2 Versão Assincrona

[ ]:
```python
# cpu_bound/threaded.py
import concurrent.futures
import time

def calculate(limit):
    return sum(i * i for i in range(limit))

def find_sums(numbers):
    with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
        executor.map(calculate, numbers)
```

```
if __name__ == '__main__':
    numbers = [5_000_000 + x for x in range(20)]

    print("Starting calculation")
    start = time.time()
    find_sums(numbers)
    duration = time.time() - start
    print(f"Duration {duration} seconds")
```

3.3 Versão com Processos

[ ]:
```
# cpu_bound/multi.py
import multiprocessing
import time

def calculate(limit):
    return sum(i * i for i in range(limit))

def find_sums(numbers):
    with multiprocessing.Pool() as pool:
        pool.map(calculate, numbers)

if __name__ == '__main__':
    numbers = [5_000_000 + x for x in range(20)]

    print("Starting calculation")
    start = time.time()
    find_sums(numbers)
    duration = time.time() - start
    print(f"Duration {duration} seconds")
```

[ ]:
```

```