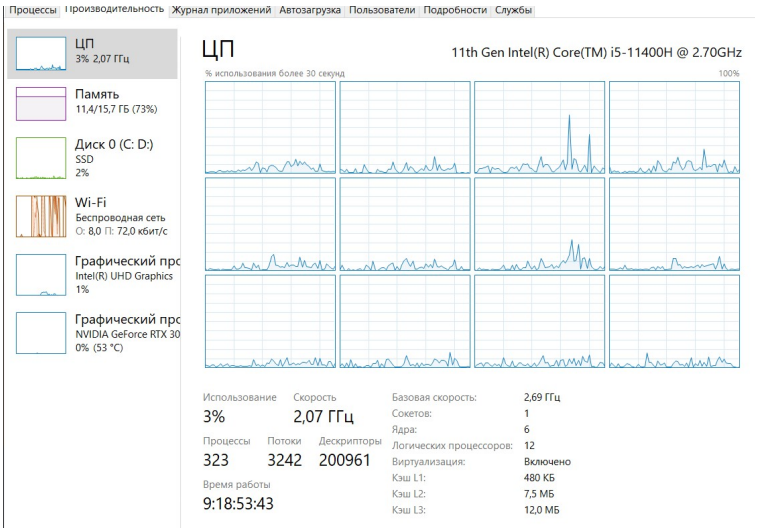
Васильева Елизавета Валерьевна АТ-03
Технологии программирования. 13 задание task_01

```python
import threading
import time
import multiprocessing
import requests

# список url
urls = ['https://www.example.com'] * 10

def fetch_url(url):
    response = requests.get(url)
    return response.text

def sequence():
    start_time = time.perf_counter()  # время старта
    for url in urls:
        fetch_url(url)  # выполнение функции fetch_url для каждого url из urls
    end_time = time.perf_counter()  # время окончания
    print(f'sequence time: {end_time - start_time:0.2f} seconds\n')

def threads():
    start_time = time.perf_counter()  # время старта
    thread_list = []
    for url in urls:
        thread = threading.Thread(target=fetch_url, args=(url,))
        thread_list.append(thread)
        thread.start()

    for thread in thread_list:
        thread.join()  # ожидание окончания выполнения всех потоков
    end_time = time.perf_counter()  # время окончания
    print(f'threads time: {end_time - start_time:0.2f} seconds\n')
```

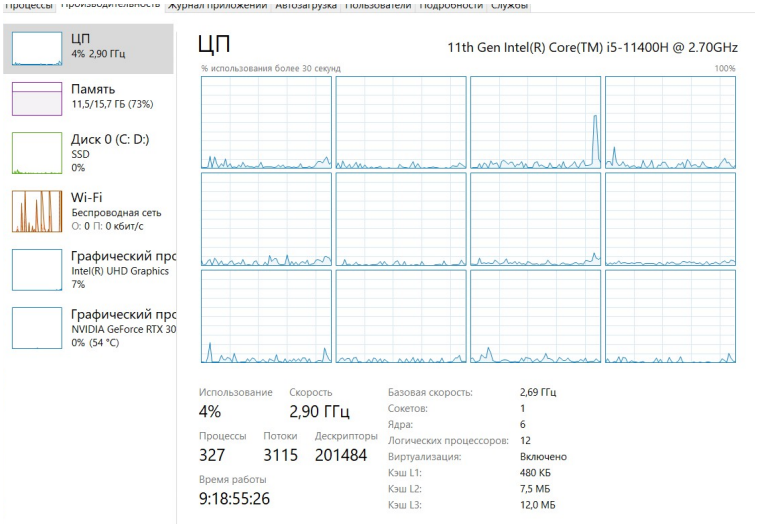
```python

def processes():
    start_time = time.perf_counter()  # время старта
    with multiprocessing.Pool() as pool:
        pool.map(fetch_url, urls)  # выполнение с помощью процессов
    end_time = time.perf_counter()  # время окончания
    print(f'processes time: {end_time - start_time:0.2f} seconds\n')

if __name__ == '__main__':
    sequence()
    threads()
    processes()

```

```
sequence time: 5.75 seconds

threads time: 0.59 seconds

processes time: 1.77 seconds
```

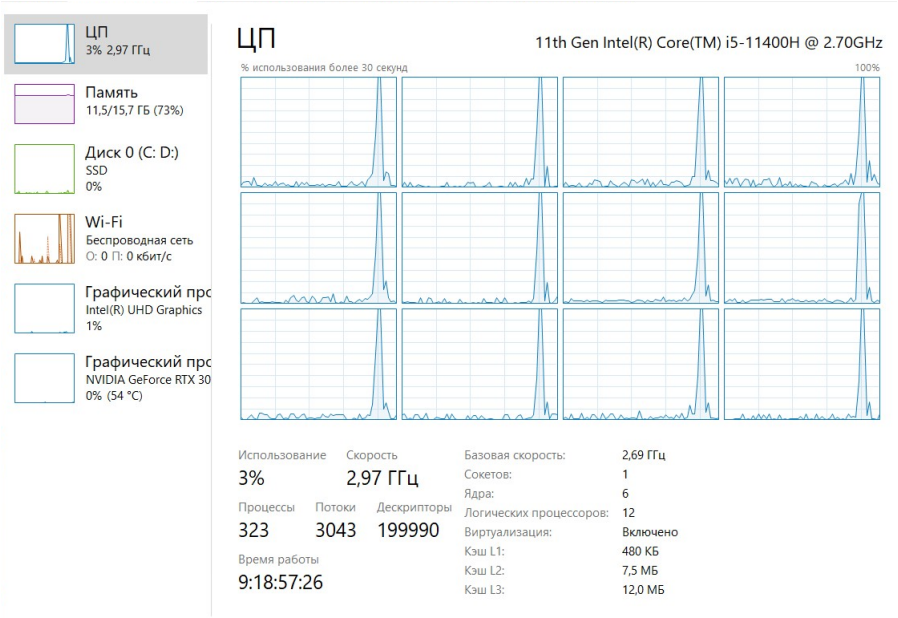
1)последовательно



2) на потоках



3) через процессы



sequence time: 5.75 seconds
Последовательный подход - самый медленный, так как каждый запрос выполняется по очереди.

threads time: 0.59 seconds
Быстрее, так как потоки позволяют одновременно выполнять несколько задач, либо сократить время на ожидание выполнения ресурсоемких задач.

processes time: 1.77 seconds

Многопроцессный подход, хотя и может быть более эффективным для задач, требующих интенсивных вычислений, так как распределяет нагрузку между ядрами процессора, тем самым ускоряя выполнение.
Однако межпроцессное взаимодействие зачастую требует дополнительных ресурсов, и его использование не всегда бывает эффективным.

Таким образом, в данной задаче многопоточный подход оказался самым быстрым из-за его способности эффективно использовать время ожидания сетевых запросов.