Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

«Лабораторная работа №2»

Автор: Кононов Степан

Факультет: ИКТ

Группа: К33392

Санкт-Петербург 2024

Лабораторая работа №2

Цель работы

Исследовать различия между подходами threading, multiprocessing и async в Python с использованием:

1. Вычислительных задач для нахождения суммы всех чисел от 1 до 1 000 000.
2. Параллельного парсинга веб-страниц с сохранением данных в базу данных.

Задача 1: Вычислительные задачи

Threading

Код программы

```
def calculate_partial_sum(start, end, result, index):
    partial_sum = sum(range(start, end))
    result[index] = partial_sum


def calculate_sum_with_threading(num):
    num_threads = 5
    range_per_thread = num // num_threads
    threads = []
    results = [0] * num_threads

    for i in range(num_threads):
        start = i * range_per_thread + 1
        end = (i + 1) * range_per_thread + 1
        thread = threading.Thread(target=calculate_partial_
sum, args=(start, end, results, i))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()
```

```
    total_sum = sum(results)
    return total_sum


if __name__ == '__main__':
    start_time = time.time()
    total_sum = calculate_sum_with_threading(1000000)
    threading_time = time.time() - start_time

    print(f"Total sum (threading): {total_sum}")
    print(f"Execution time (threading): {threading_time} se
conds")
```

Multiprocessing

Код программы

```
import multiprocessing
import time


def calculate_partial_sum(start, end):
    return sum(range(start, end))


def calculate_sum_with_multiprocessing(num):
    num_processes = 5
    range_per_process = num // num_processes
    with multiprocessing.Pool(processes=num_processes) as p
ool:
        results = []
        for i in range(num_processes):
            start = i * range_per_process + 1
            end = (i + 1) * range_per_process + 1
            results.append(pool.apply_async(calculate_parti
al_sum, (start, end)))
        total_sum = sum(result.get() for result in results)
    return total_sum
```

```
if __name__ == '__main__':
    start_time = time.time()
    total_sum = calculate_sum_with_multiprocessing(1000000)
    multiprocessing_time = time.time() - start_time

    print(f"Total sum (multiprocessing): {total_sum}")
    print(f"Execution time (multiprocessing): {multiprocess
ing_time} seconds")
```

Asyncio

Код программы

```
import asyncio
import time


async def calculate_partial_sum(start, end):
    return sum(range(start, end))


async def calculate_sum_with_asyncio(num):
    num_tasks = 5
    range_per_task = num // num_tasks
    tasks = []

    for i in range(num_tasks):
        start = i * range_per_task + 1
        end = (i + 1) * range_per_task + 1
        tasks.append(calculate_partial_sum(start, end))
    results = await asyncio.gather(*tasks)

    total_sum = sum(results)
    return total_sum
```

```
if __name__ == '__main__':
    start_time = time.time()
    total_sum = asyncio.run(calculate_sum_with_asyncio(1000
000))
    asyncio_time = time.time() - start_time

    print(f"Total sum (asyncio): {total_sum}")
    print(f"Execution time (asyncio): {asyncio_time} second
s")
```

Результаты сравнения времени

для n = 10000000

Approach	Total Sum	Execution Time (s)
Threading	50000000500000000	3.5810675621032715
Multiprocessing	50000000500000000	1.3409934043884277
Asyncio	50000000500000000	3.632997989654541

для n = 100000

Approach	Total Sum	Execution Time (s)
Threading	5000050000	0.003000497817993164
Multiprocessing	5000050000	0.46596431732177734
Asyncio	5000050000	0.003998517990112305

Задача 2: Параллельный парсинг веб-страниц

Код программы

```
urls = ["https://www.example.com", "https://www.python.or
g", "https://www.github.com"]
```

```
def parse_and_save_threading(url):
    open_session = SessionLocal()
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'html.parser')
        title = soup.title.string if soup.title else "No ti
tle"
        page = Page(url=url, title=title)
        open_session.add(page)
        open_session.commit()
        print(f"Threading - URL: {url}, Title: {title}")
    except sqlalchemy.exc.IntegrityError:
        open_session.rollback()
        print(f"Threading - URL: {url} already exists in da
tabase")
    finally:
        open_session.close()


def threading_main(urls):
    threads = []
    for url in urls:
        thread = threading.Thread(target=parse_and_save_thr
eading, args=(url,))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()


def parse_and_save_multiprocessing(url):
    open_session = SessionLocal()
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'html.parser')
        title = soup.title.string if soup.title else "No ti
tle"
```

```
        page = Page(url=url, title=title)
        open_session.add(page)
        open_session.commit()
        print(f"Multiprocessing - URL: {url}, Title: {titl
e}")
    except sqlalchemy.exc.IntegrityError:
        open_session.rollback()
        print(f"Multiprocessing - URL: {url} already exists
in database")
    finally:
        open_session.close()


def multiprocessing_main(urls):
    processes = []
    for url in urls:
        process = multiprocessing.Process(target=parse_and_
save_multiprocessing, args=(url,))
        processes.append(process)
        process.start()
    for process in processes:
        process.join()


async def fetch(session, url):
    async with session.get(url) as response:
        return await response.text()


async def parse_and_save_async(url):
    open_session = SessionLocal()
    async with aiohttp.ClientSession() as aio_session:
        try:
            html = await fetch(aio_session, url)
            soup = BeautifulSoup(html, 'html.parser')
            title = soup.title.string if soup.title else "N
o title"
            page = Page(url=url, title=title)
```

```
            open_session.add(page)
            open_session.commit()
            print(f"Async - URL: {url}, Title: {title}")
        except sqlalchemy.exc.IntegrityError:
            open_session.rollback()
            print(f"Async - URL: {url} already exists in da
tabase")
        finally:
            open_session.close()


async def async_main(urls):
    tasks = [parse_and_save_async(url) for url in urls]
    await asyncio.gather(*tasks)


def run_async(urls):
    asyncio.run(async_main(urls))


def measure_performance():
    results = []

    start_time = time()
    threading_main(urls)
    threading_time = time() - start_time
    results.append(("Threading", threading_time))

    start_time = time()
    multiprocessing_main(urls)
    multiprocessing_time = time() - start_time
    results.append(("Multiprocessing", multiprocessing_tim
e))

    start_time = time()
    run_async(urls)
    async_time = time() - start_time
    results.append(("Async", async_time))
```

```
    table = PrettyTable()
    table.field_names = ["Method", "Time (s)"]
    for method, t in results:
        table.add_row([method, t])

    print(table)


if __name__ == "__main__":
    measure_performance()
```

Результаты сравнения времени

```
Threading - URL: https://www.python.org, Title: Welcome to Python.org
Threading - URL: https://www.example.com, Title: Example Domain
Threading - URL: https://www.github.com, Title: GitHub: Let's build from here · GitHub
Multiprocessing - URL: https://www.python.org, Title: Welcome to Python.org
Multiprocessing - URL: https://www.github.com, Title: GitHub: Let's build from here · GitHub
Multiprocessing - URL: https://www.example.com, Title: Example Domain
Async - URL: https://www.python.org, Title: Welcome to Python.org
Async - URL: https://www.github.com, Title: GitHub: Let's build from here · GitHub
Async - URL: https://www.example.com, Title: Example Domain
+-----------------+--------------------+
|      Method     |      Time (s)      |
+-----------------+--------------------+
|    Threading    | 0.7416963577270508 |
| Multiprocessing | 1.8923213481903076 |
|      Async      | 0.6190145015716553 |
+-----------------+--------------------+
```

Заключение

При выполнении параллельного парсинга веб-страниц асинхронный подход (async) снова показал наилучшие результаты по производительности. Это связано с тем, что при работе с сетевыми операциями Asyncio использует асинхронный ввод-вывод, что значительно ускоряет работу.

Рекомендации

Для задач, требующих значительноы асинхронных ввод-вывод (таких как сетевые операции), рекомендуется использовать Asyncio. Task-ориентированные задачи с большим числом параллельно выполняемых операций лучше решать посредством Multiprocessing.