کار با ورودی/خروجی (I/O) غیرهمزمان با `asyncio`

در برنامه‌نویسی غیرهمزمان، عملیات ورودی/خروجی (I/O) می‌تواند به طور قابل توجهی زمان‌بر باشد، به ویژه زمانی که با منابع خارجی مانند فایل‌ها، پایگاه‌داده‌ها، یا ارتباطات شبکه‌ای سروکار داریم. در این موارد، استفاده از `asyncio` به شما کمک می‌کند که این عملیات را بدون بلوکه کردن برنامه انجام دهید.

1. خواندن و نوشتن به فایل‌ها به‌صورت غیرهمزمان

اگرچه پایتون به طور مستقیم از خواندن و نوشتن غیرهمزمان به فایل‌ها پشتیبانی نمی‌کند، می‌توانید از روش‌هایی برای انجام عملیات غیرهمزمان با استفاده از `asyncio` و کتابخانه‌هایی مانند `aiofiles` برای کار با فایل‌ها استفاده کنید.

نصب کتابخانه `aiofiles`:

```
pip install aiofiles
```

مثال برای خواندن و نوشتن به فایل‌ها به‌صورت غیرهمزمان:

```
import asyncio
import aiofiles

async def read_file():
    async with aiofiles.open('example.txt', mode='r') as f:
        contents = await f.read()
        print(contents)

async def write_file():
    async with aiofiles.open('example.txt', mode='w') as f:
        await f.write("Hello, Asyncio!")

async def main():
    await asyncio.gather(read_file(), write_file())  # خواندن و نوشتن به صورت غیرهمزمان

asyncio.run(main())
```

در اینجا:

- `aiofiles.open()` به شما این امکان را می‌دهد که فایل‌ها را به صورت غیرهمزمان باز کنید.
- `await f.read()` و `await f.write()` عملیات خواندن و نوشتن را به صورت غیرهمزمان انجام می‌دهند.

2. برقراری ارتباط شبکه‌ای به‌صورت غیرهمزمان با `asyncio`

با استفاده از `asyncio`، می‌توان به راحتی ارتباطات شبکه‌ای را به صورت غیرهمزمان انجام داد، به ویژه زمانی که با پروتکل‌های I/O مانند HTTP یا TCP سروکار داریم.

مثال برای ایجاد یک سرور TCP غیرهمزمان:

```
import asyncio

async def handle_client(reader, writer):
    data = await reader.read(100)
    message = data.decode()
```

```
    addr = writer.get_extra_info('peername')

    print(f"Received {message} from {addr}")

    response = 'Hello, client!'
    writer.write(response.encode())
    await writer.drain()

    print("Closing the connection")
    writer.close()

async def main():
    server = await asyncio.start_server(
        handle_client, '127.0.0.1', 8888)
    addr = server.sockets[0].getsockname()
    print(f'Serving on {addr}')

    async with server:
        await server.serve_forever()

asyncio.run(main())
```

در اینجا:

- `asyncio.start_server()` به شما امکان می‌دهد که یک سرور TCP غیرهمزمان راه‌اندازی کنید.
- متد `reader.read()` برای دریافت داده‌ها به صورت غیرهمزمان و `writer.write()` برای ارسال پاسخ استفاده می‌شود.

3. ارسال درخواست‌های HTTP غیرهمزمان با `aiohttp`

کتابخانه `aiohttp` برای ارسال درخواست‌های HTTP به صورت غیرهمزمان بسیار مفید است. این کتابخانه به شما این امکان را می‌دهد که درخواست‌های HTTP را به صورت غیرهمزمان ارسال کنید، که در زمان‌هایی که تعداد زیادی درخواست همزمان دارید، عملکرد بهتری نسبت به استفاده از `requests` (که همزمان است) خواهد داشت.

نصب کتابخانه `aiohttp`:

```
pip install aiohttp
```

مثال برای ارسال درخواست HTTP غیرهمزمان با `aiohttp`:

```
import asyncio
import aiohttp

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

async def main():
    urls = ['https://example.com', 'https://example.org', 'https://example.net']
    tasks = [fetch(url) for url in urls]
    results = await asyncio.gather(*tasks)
    for result in results:
        print(result[:100])  # چاپ اولین 100 کاراکتر از هر صفحه
```

```
asyncio.run(main())
```

در اینجا:

- `aiohttp.ClientSession()` برای برقراری ارتباط HTTP به صورت غیرهمزمان استفاده می‌شود.
- `await session.get(url)` برای ارسال درخواست HTTP به صورت غیرهمزمان به کار می‌رود.
- `asyncio.gather()` برای اجرای هم‌زمان چندین درخواست HTTP استفاده می‌شود.

4. اجرای عملیات I/O موازی بدون بلوکه کردن برنامه

با استفاده از `asyncio`, می‌توانید چندین عملیات I/O را به صورت موازی و غیرهمزمان اجرا کنید بدون اینکه یک عملیات، برنامه را بلوکه کند. این امر موجب بهبود کارایی و سرعت برنامه در شرایطی می‌شود که نیاز به انجام چندین عملیات ورودی/خروجی به طور همزمان دارید.

مثال برای انجام عملیات موازی با `asyncio.gather()`:

```
import asyncio

async def task1():
    await asyncio.sleep(2)
    return "Task 1 Complete"

async def task2():
    await asyncio.sleep(1)
    return "Task 2 Complete"

async def main():
    results = await asyncio.gather(task1(), task2())
    print(results)

asyncio.run(main())
```

در اینجا:

- با استفاده از `asyncio.gather()`, دو تابع `task1()` و `task2()` به صورت غیرهمزمان و موازی اجرا می‌شوند.
- زمان اجرای کلی برنامه فقط برابر با بیشترین زمان از دو عملیات است (در اینجا 2 ثانیه).

نتیجه‌گیری

با استفاده از `asyncio` و کتابخانه‌های مرتبط مانند `aiofiles`, `aiohttp`, می‌توانید عملیات ورودی/خروجی را به صورت غیرهمزمان و موازی انجام دهید. این باعث می‌شود که برنامه شما بدون بلوکه شدن و بهینه‌تر اجرا شود، به ویژه در شرایطی که نیاز به انجام چندین عملیات ورودی/خروجی به صورت همزمان دارید.