ВПРОВАДЖЕННЯ ПАТЕРНУ СПІВПРОГРАМ ДЛЯ ПРОЕКТУВАННЯ ЕФЕКТИВНИХ ВЕБ-СЕРВЕРІВ

Кваліфікаційна робота

Виконав студент КН-4

Цегельник Богдан Володимирович

Керівник Бублик В. В.

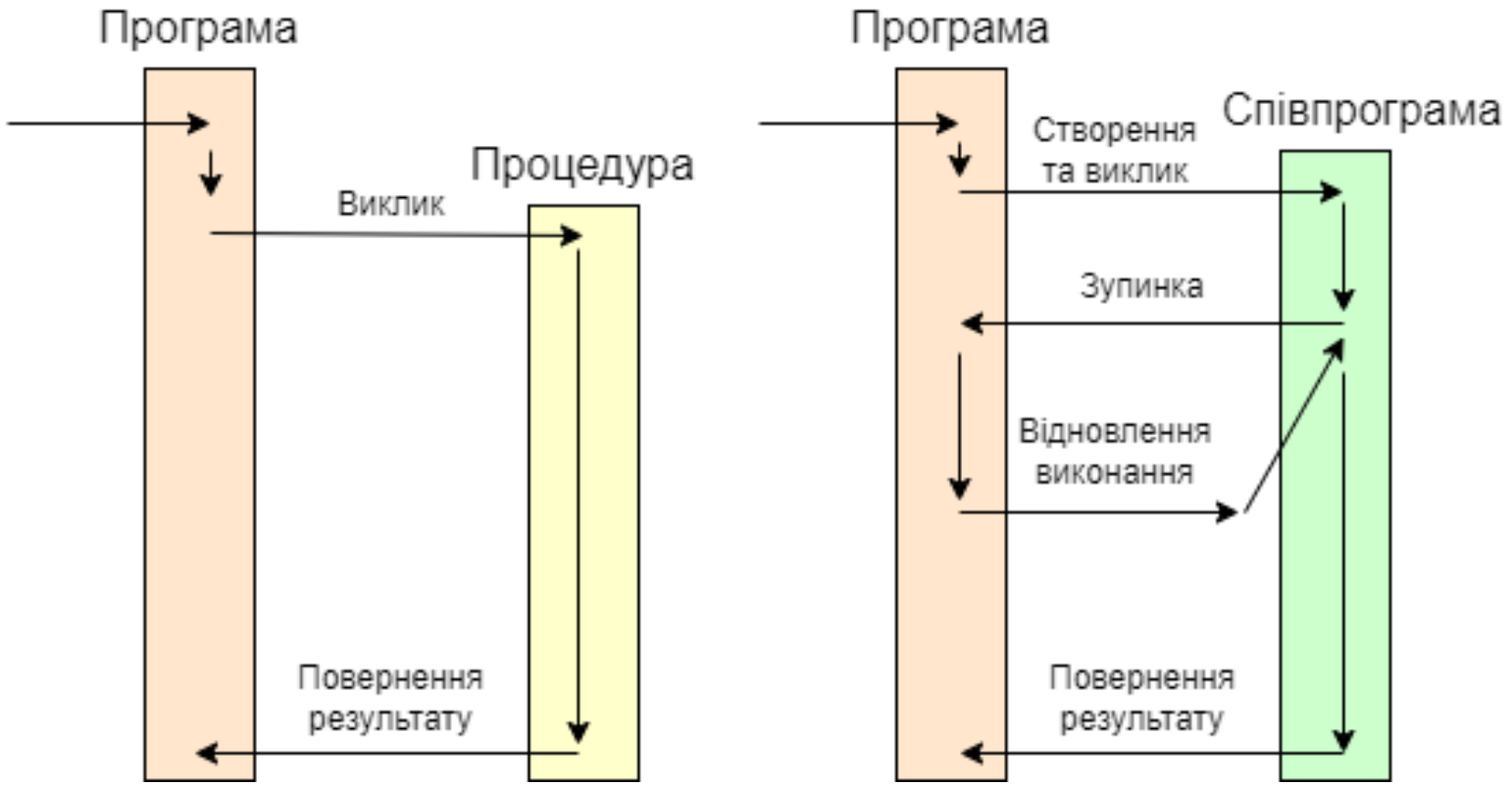
ПОСТАНОВКА ЗАДАЧІ

- Розглянути співпрограми як патерн багатозадачності, їх класифікацію, відмінності від потоків та переваги використання з асинхронними I/O операціями.
- Дослідити співпрограми в стандарті C++20 та ефективність реалізації в бібліотеці Boost Asio в порівнянні з потоками.
- Реалізувати веб-сервери з використанням різних підходів: синхронного, асинхронного із зворотними викликами та асинхронного із співпрограмами.
- Провести порівняльний аналіз ефективності роботи й складності в реалізації кожного підходу.

СТАН ПРОБЛЕМИ

- Популярні мови програмування такі як Python, C++, Java, JavaScript, C#, Kotlin можуть використовувати співпрограми на рівні бібліотеки або мови.
- Підтримка співпрограм на рівні мови в C++ з’явилася не так давно, разом із стандартом мови C++20.

ПОНЯТТЯ СПІВПРОГРАМИ

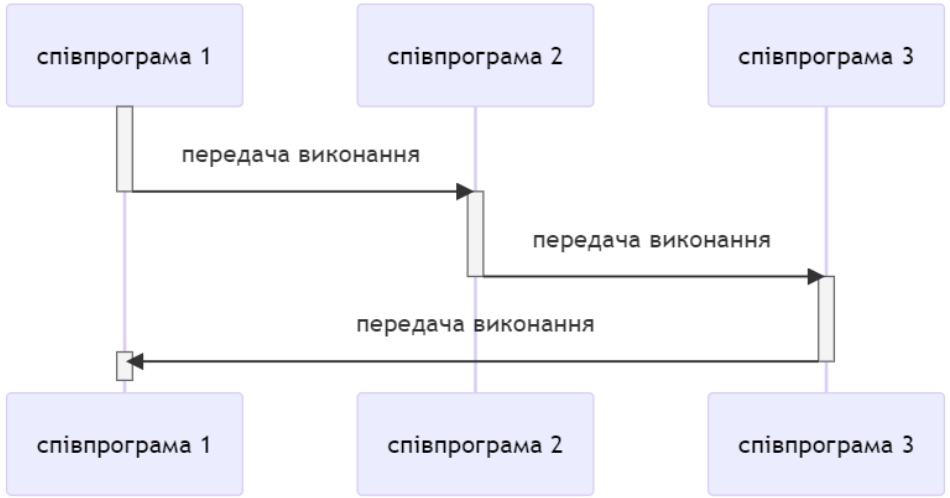


КЛАСИФІКАЦІЯ СПІВПРОГРАМ

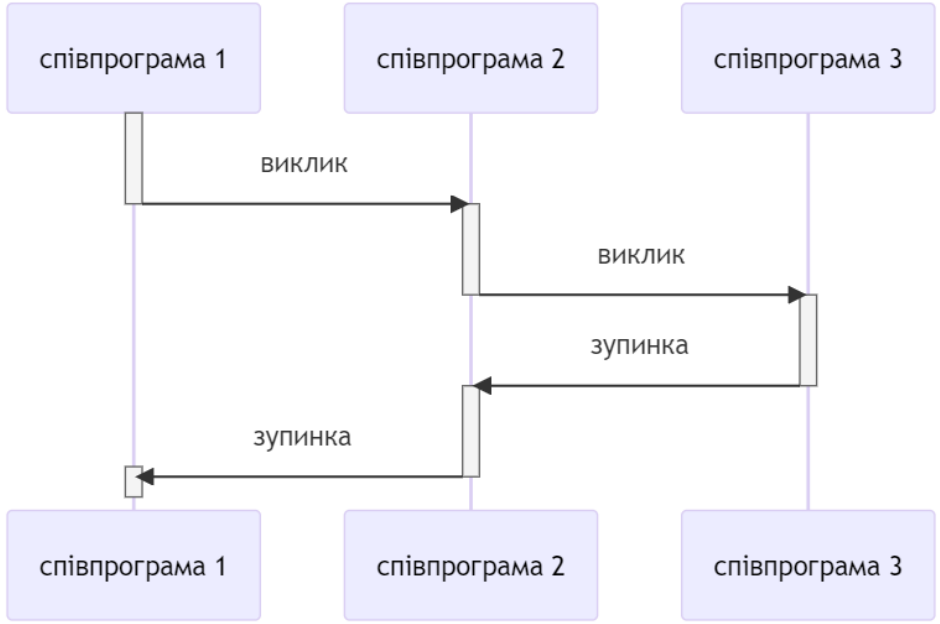
- Механізм передачі управління
- Збереження стеку
- Об’єкт, яким можна маніпулювати, чи мовна конструкція.

МЕХАНІЗМ ПЕРЕДАЧІ УПРАВЛІННЯ

Симетричний

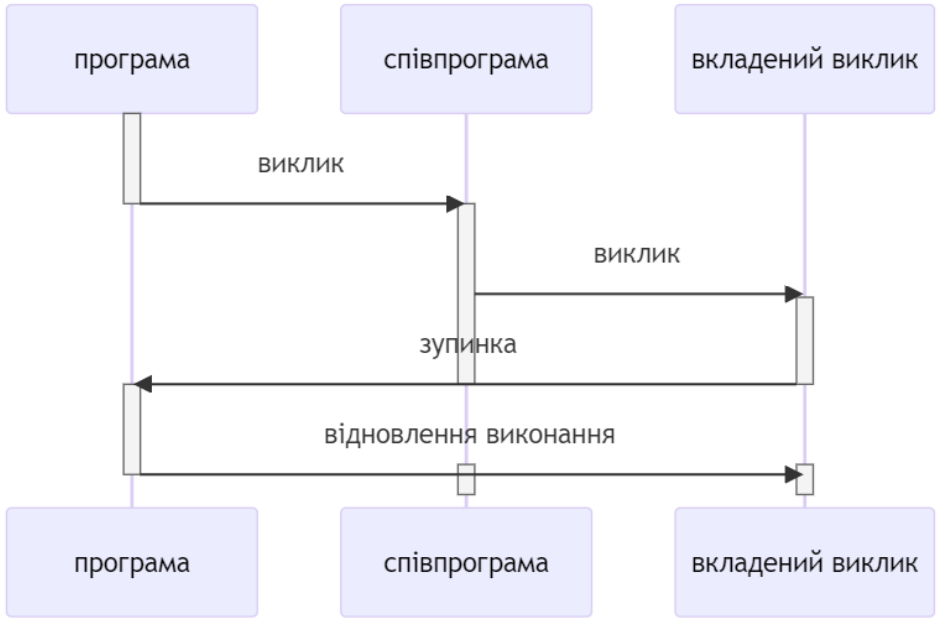


Асиметричний

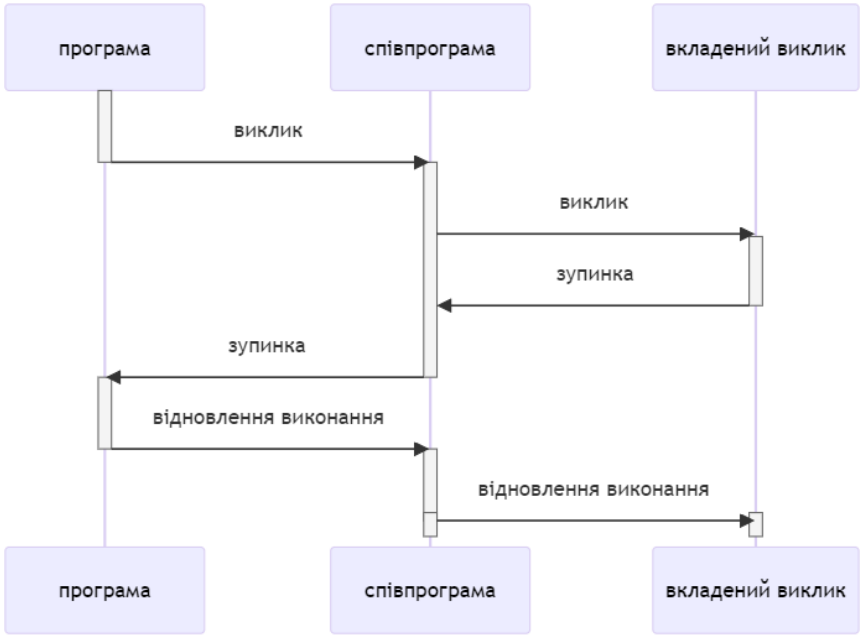


ЗБЕРЕЖЕННЯ СТЕКУ

Із збереженням



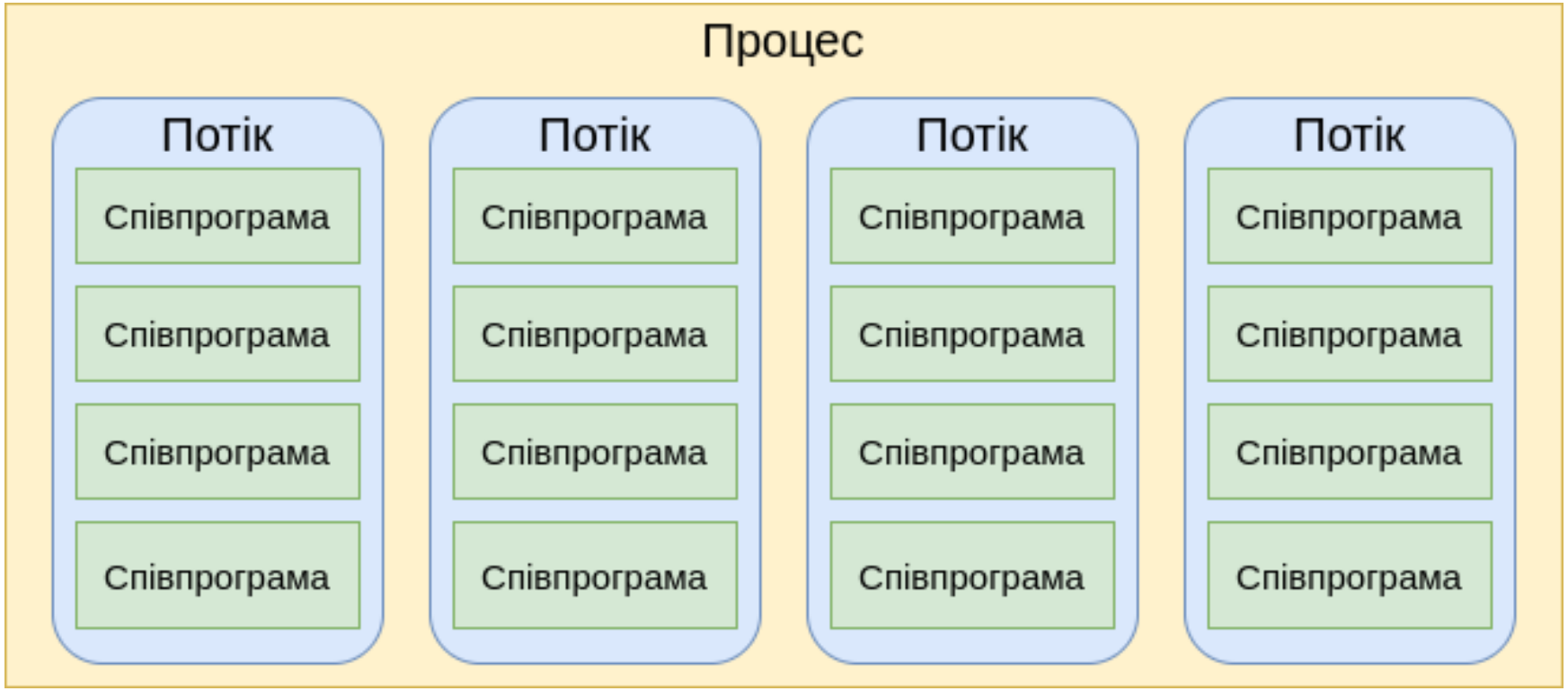
Безстекові



ВІДМІННОСТІ СПІВПРОГРАМ ВІД ПОТОКІВ

Потоки	Співрограми
Плануються операційною системою	Плануванням займається розробник
Дозволяють паралельні обчислення	Надають багатозадачність (не паралельність)
Потребують створення окремого стеку й регістрів	Потребують збереження стеку залежно від виду

ПОТОКИ ДЛЯ ПАРАЛЕЛЬНОСТІ, СПІВПРОГРАМИ ДЛЯ БАГАТОЗАДАЧНОСТІ



ПОРІВНЯННЯ КОДУ СИНХРОННИХ ТА АСИНХРОННИХ ОПЕРАЦІЙ

Синхронні блокуючі операції

```
try
{
    socket sock = accept(sock);
    //...
    sock.read(buffer)
    //...
    connection conn = connect_database();
    //...
    conn.database_query(query);
    //...
    conn.disconnect_database();
    //...
    sock.write(buffer);
    //...
    sock.close();
}
catch (exception e)
{
    //...обробка помилок
}
```

Асинхронні із застосуванням зворотних викликів

```
async_accept((socket sock, error err){
    //...потреба обробки помилки
    //...
    sock.asycn_read(buffer, (error err){
        //...потреба обробки помилки
        //...
        async_connect_database((connection conn, error err){
            //...потреба обробки помилки
            //...
            conn.async_database_query(query, (query_result res, error err){
                //...потреба обробки помилки
                //...
                conn.disconnect_database();
                async_write(buffer, result, (error err){
                    //...потреба обробки помилки
                    //...
                    sock.close();
                });
            });
        });
    });
});
```

ПОРІВНЯННЯ КОДУ СИНХРОННИХ ТА АСИНХРОННИХ ОПЕРАЦІЙ

Синхронні блокуючі операції

```
try
{
    socket sock = accept(sock);
    //...
    sock.read(buffer)
    //...
    connection conn = connect_database();
    //...
    conn.database_query(query);
    //...
    conn.disconnect_database();
    //...
    sock.write(buffer);
    //...
    sock.close();
}
catch (exception e)
{
    //...обробка помилок
}
```

Асинхронні із застосуванням зворотних викликів

```
//глобальний/член класу buffer;
//глобальний/член класу socket;
//глобальний/член класу db_conn;

// program
{
    async_accept(async_accept_callback);
}

void async_accept_callback(socket sock, error err) {
    //...потреба обробки помилки
    //...
    sock.async_read(buffer, handleRead);
}

void handleRead(error err) {
    //...потреба обробки помилки
    //...
    async_connect_database(handleConnectDatabase);
}

void handleConnectDatabase(connection db_conn, error err) {
    //...потреба обробки помилки
    //...
    db_conn.async_database_query(query, handleDatabaseQuery);
}

void handleDatabaseQuery(query_result res, error err) {
    //...потреба обробки помилки
    //...
    db_conn.disconnect_database();
    async_write(buffer, result, handleWrite);
}

void handleWrite(error err) {
    //...потреба обробки помилки
    //...
    sock.close();
}
```

ПОРІВНЯННЯ КОДУ СИНХРОННИХ ТА АСИНХРОННИХ ОПЕРАЦІЙ

Синхронні блокуючі операції

```
try
{
    socket sock = accept(sock);
    //...
    sock.read(buffer)
    //...
    connection conn = connect_database();
    //...
    conn.database_query(query);
    //...
    conn.disconnect_database();
    //...
    sock.write(buffer);
    //...
    sock.close();
}
catch (exception e)
{
    //...обробка помилок
}
```

Асинхронні із застосуванням співпрограм

```
try
{
    socket sock = co_await async_accept();
    //...
    co_await sock.async_read(buffer)
    //...
    connection conn = co_await async_connect_database();
    //...
    co_await conn.async_database_query(query);
    //...
    conn.disconnect_database();
    //...
    co_await sock.async_write(buffer);
    //...
    sock.close();
}
catch (exception e)
{
    //...обробка помилок
}
```

СПІВПРОГРАМИ В C++20

- Об'єкти, асиметричні, безстекові.
- Стандарт надає лише каркас для написання логіки поведінки співпрограм.
- У роботі використовується готова реалізація співпрограм з бібліотеки Boost Asio.

ПОРІВНЯННЯ ЕФЕКТИВНОСТІ СПІВПРОГРАМ BOOST ASIO ТА ПОТОКУ ДЛЯ I/O ЗАДАЧ НА ДЕКІЛЬКОХ ЯДРАХ

Кількість задач	Час виконання співпрограм (мс)	Час виконання потоків (мс)
10	16071	15824
100	16162	17833
1000	21059	58209
5000	72828	248817
10000	149082	495625
20000	311070	980393

Таблиця 2.1 – Порівняння потоків та співпрограм asio::awaitable в багатоядерному середовищі

ПОРІВНЯННЯ ЕФЕКТИВНОСТІ СПІВПРОГРАМ BOOST ASIO ТА ПОТОКУ ДЛЯ I/O ЗАДАЧ НА ОДНОМУ ЯДРІ

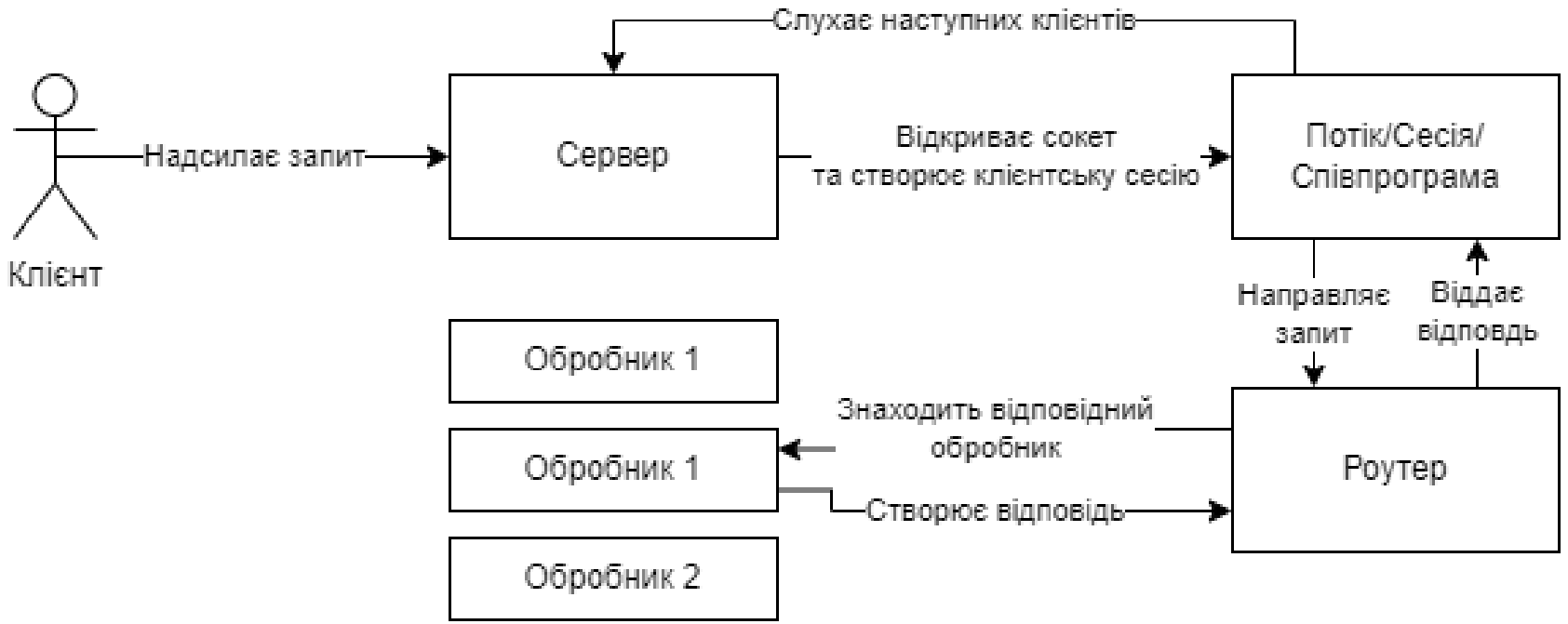
Кількість задач	Час виконання співпрограм (мс)	Час виконання потоків (мс)
10	15346	15933
100	15986	18314
1000	20394	56272
5000	84628	240125
10000	172028	473724
20000	356456	944296

Таблиця 2.2 – Порівняння потоків та співпрограм asio::awaitable в одноядерному середовищі

РОЗГЛЯНУТІ ПІДХОДИ ДО ПРОЕКТУВАННЯ ВЕБ-СЕРВЕРА

- **Синхронний** – для кожного клієнта створюється окремий потік виконання, використовуються синхронні блокуючі операції
- **Асинхронний з використанням зворотних викликів** – для кожного клієнта створюється об’єкт сесії, щоб зберігати з’єднання та інші дані між викликами. Використовуються асинхронні неблокуючі операції.
- **Асинхронний з використанням співпрограм** – для кожного клієнта створюється об’єкт співпрограми. Використовуються асинхронні неблокуючі операції.

СПІЛЬНА АРХІТЕКТУРА



ТЕСТУВАННЯ СЕРВЕРІВ З ПОСТІЙНИМ З’ЄДНАНН

	100	1000	5000	10000
Синхронний	14891	14388	13853	13645
Зворотні виклики	9662	9206	8920	8884
Співпрограми	10814	10141	9858	9838

Таблиця 3.1 – Тестування шляху без додаткових I/O операцій (Connection: keep-alive) запитів/секунду

	100	1000	5000	10000
Синхронний	3210	11804	11440	10935
Зворотні виклики	3195	8319	8059	7920
Співпрограми	3205	9413	9105	9057

Таблиця 3.2 – тестування шляху з додатковими I/O операціями (Connection: keep-alive) запитів/секунду

ТЕСТУВАННЯ СЕРВЕРІВ БЕЗ ПІДТРИМКИ З’ЄДНАНЬ

	100	1000	5000	10000
Синхронний	5354	5255	5296	5200
Зворотні виклики	5491	5526	5569	5457
Співпрограми	5917	6000	5964	5865

Таблиця 3.3 – Тестування шляху без додаткових I/O операцій (Connection: close) запитів/секунду

	100	1000	5000	10000
Синхронний	2771	4737	4663	4534
Зворотні виклики	3167	5075	4960	4669
Співпрограми	3178	5467	5307	5068

Таблиця 3.4 – тестування шляху з додатковими I/O операціями (Connection: close) запитів/секунду

ВИСНОВКИ

- З’ясовано, що:
  - Співпрограми дозволяють одночасно, проте не паралельно виконувати декілька задач.
  - Спрощують програмного код та можуть замінити зворотні виклики для асинхронних операціях.
  - Сервер з використанням співпрограм працює ефективніше інших реалізацій для великої кількості одночасних клієнтів.

ДЯКУЮ ЗА УВАГУ