Іпс-32 Шовкопляс Богдан Вікторович

Завдання: Блокуючий запит, async, Unix-сокет, C++

У якості носіїв компонентів обчислення розглядаються запити, які обробляються через Unix-сокет з використанням блокуючого вводу/виводу.

Ключові компоненти

Класи для обчислень (стратегії):

- **Task (Базовий клас):**
  - Абстрактний клас для визначення завдання обчислення.
  - Містить віртуальний метод `execute(int x)`, який реалізується у похідних класах.
- **Конкретні реалізації:**
  - `SquareTask`: Обчислює квадрат числа: `x * x`.
  - `CubeTask`: Обчислює куб числа: `x * x * x`.
  - `ReverseTask`: Перевертає цифри числа (наприклад, 123 → 321).

Клас UnixSocketServer (Сервер через Unix-сокет):

- **Призначення:** Реалізує сервер для обробки запитів через Unix-сокет із підтримкою асинхронного виконання.
- **Основні методи:**
  - `start()`: Запускає сервер, який прослуховує вказаний шлях Unix-сокета.
  - `handleClient(int clientSocket)`: Обробляє запит клієнта, виконуючи відповідне обчислення.
  - `addTask(const std::string& name, std::shared_ptr<Task> task)`: Додає завдання до списку доступних для виконання.

Команди інтерфейсу

1. help: Виводить список доступних команд.

2. start: Запускає сервер Unix-сокета для прийому запитів.

3. exit: Завершує роботу програми.

Команди клієнта:

Для взаємодії із сервером використовується утиліта `socat` або nc (netcat).

Надіслати запит:
```
echo "<число>" | socat -UNIX-CONNECT:/tmp/unix_socket_example
```
або:
```
echo "<число>" | nc -U /tmp/unix_socket_example
```

Приклад використання

Запустити програму:
```
./unix_socket_server
```

Запустити сервер:
```
> start
Server started. Listening on /tmp/unix_socket_example
```

Відправити запити через клієнт:
У новому терміналі:
```
echo "25" | socat - UNIX-CONNECT:/tmp/unix_socket_example
```
Очікувана відповідь:
```
Square: 625
```
Інший запит:
```
echo "10" | socat - UNIX-CONNECT:/tmp/unix_socket_example
```
Очікувана відповідь:
```
Cube: 1000
```

Завершити роботу програми:
```
> exit
```

Особливості

1. **Блокуючий запит:**
   Сервер чекає на клієнтський запит і блокується, поки не буде отримано дані.
2. **Асинхронна обробка клієнтів:**
   Використання `std::async` дозволяє обробляти кожного клієнта в окремому потоці, не блокуючи інші з'єднання.
3. **Unix-сокет:**
   Забезпечує швидкий та ефективний обмін даними між процесами на одному сервері.

Цей підхід демонструє концепцію блокуючого сервера з асинхронною обробкою запитів та є зручним для локального міжпроцесного зв’язку.