Tomasz Wieczorek, s27161

Dokumentacja Scentralizowanego Systemu Obliczeniowego (CCS)

Cel projektu

Scentralizowany System Obliczeniowy (CSS) to aplikacja serwerowa, której celem jest obsługa klientów w sieci lokalnej. Serwer realizuje trzy główne funkcjonalności:

1. **Wykrywanie usługi** – umożliwia klientom odnalezienie serwera w sieci lokalnej za pomocą protokołu UDP.
2. **Obsługa żądań obliczeniowych** – serwer przyjmuje połączenia TCP od klientów, wykonuje operacje matematyczne (dodawanie, odejmowanie, mnożenie, dzielenie) i zwraca wyniki.
3. **Raportowanie statystyk** – serwer zbiera dane o swojej pracy (liczba klientów, liczba operacji, błędy) i wypisuje je na konsolę co 10 sekund.

Opis klas i ich funkcjonalności

1. Main

(*Punkt wejścia do aplikacji)*

Funkcjonalności:

- Sprawdza, czy użytkownik podał poprawny numer portu jako argument.
- Tworzy instancję klasy **CentralizedComputingSystem** i uruchamia serwer.

Przykładowe użycie:

java -jar CCS.jar 12345

Jeśli użytkownik nie poda portu lub poda niepoprawny numer, program wyświetli odpowiedni komunikat i zakończy działanie.

2. CentralizedComputingSystem

(Główna klasa zarządzająca działaniem serwera)

Funkcjonalności:

- Uruchamia serwer UDP do wykrywania usługi.
- Uruchamia serwer TCP do obsługi klientów.
- Zarządza wątkami klientów za pomocą puli wątków.
- Cyclicznie raportuje statystyki.

Metody:

1. **start()**

   Cel: Uruchamia serwer, obsługując zarówno wykrywanie UDP, jak i połączenia TCP od klientów.

   Jak działa:
   - Tworzy nowy wątek dla serwera UDP.
   - Nasłuchuje połączeń TCP w pętli.

- Dla każdego połączenia TCP uruchamia wątek z obsługą klienta za pomocą puli wątków.

Kluczowe cechy:
- Wykorzystuje pulę wątków, aby efektywnie zarządzać wieloma połączeniami.
- Obsługuje klientów równocześnie, co zwiększa wydajność.

2. **shutdown()**
   - **Cel**: Zamyka serwer w sposób kontrolowany.
   - **Jak działa**:
     - Zatrzymuje serwer UDP i zamyka gniazdo TCP.
     - Zamyka pulę wątków i czeka na zakończenie aktywnych zadań.
   - **Kluczowe cechy**:
     - Zapewnia zwolnienie wszystkich zasobów.
     - Unika nagłego przerwania połączeń z klientami.

3. **startStatisticsReportingThread()**

Cel: Uruchamia wątek, który cyklicznie wypisuje statystyki serwera.

Jak działa:
- Używa *ScheduledExecutorService*, aby co 10 sekund wywoływać metodę **printStatistics()** z klasy **StatisticsManager**.

Kluczowe cechy:
- Umożliwia monitorowanie pracy serwera w czasie rzeczywistym.

Wydajność:
- Liczba wątków w puli jest dynamicznie obliczana na podstawie liczby dostępnych procesorów:

```
int processors = Runtime.getRuntime().availableProcessors();
this.clientThreadPool = Executors.newFixedThreadPool(processors * 4);
```

1. Tworzenie nowego wątku dla każdego klienta jest kosztowne. Pula wątków pozwala na ponowne wykorzystanie istniejących wątków, co zmniejsza narzut związany z obsługą wielu klientów.

3. UDPDiscoveryServer

(Odpowiada za wykrywanie serwera w sieci lokalnej)

Funkcjonalności:

- Odbiera wiadomości od klientów o treści „*CCS DISCOVER*”.
- Wysyła odpowiedź „*CCS FOUND*” na adres klienta, co pozwala klientowi poznać adres IP serwera.

Metody:

1. **run()**
    - **Cel**: Nasłuchuje wiadomości UDP i odpowiada na zapytania klientów.
    - **Jak działa**:
        - Odbiera wiadomości za pomocą UDPHandler.
        - Wysyła odpowiedź, jeśli wiadomość zawiera CCS DISCOVER.
    - **Kluczowe cechy**:
        - Działa w osobnym wątku, aby nie blokować głównego serwera.
2. **stopServer()**
    - **Cel**: Zatrzymuje serwer UDP.
    - **Jak działa**:
        - Zamyka gniazdo UDP i ustawia flagę running na false.

4. ClientRequestHandler

(Obsługuje pojedynczego klienta po nawiązaniu połączenia TCP)

Funkcjonalności:

- Odbiera żądania od klienta.
- Wykonuje operacje matematyczne.
- Wysyła wyniki do klienta.
- Aktualizuje statystyki.

Metody:

1. **handleConnection()**

    - **Cel**: Obsługuje komunikację z pojedynczym klientem.

    - **Jak działa**:
        - Odbiera żądania od klienta, przetwarza je i zwraca wyniki.
        - Obsługuje błędy i aktualizuje statystyki.

    - **Kluczowe cechy**:
        - Synchronizuje operacje wejścia/wyjścia za pomocą blokady (Lock).
        - Zapewnia bezpieczną obsługę klienta w środowisku wielowątkowym.

2. **processCommand(String command)**
    - **Cel**: Przetwarza polecenie klienta i wykonuje odpowiednią operację matematyczną.

    - **Jak działa**:
        - Rozdziela polecenie na operację i argumenty.
        - Weryfikuje poprawność danych wejściowych.
        - Wykonuje operację za pomocą MathOperationFactory.
        - Aktualizuje statystyki i obsługuje błędy.

    - **Kluczowe cechy**:
        - Obsługuje operacje: ADD, SUB, MUL, DIV.
        - Zwraca komunikaty o błędach w przypadku nieprawidłowych danych.

3. **readMessage() i sendMessage(String message)**
    - **Cel**: Odpowiada za odbieranie i wysyłanie wiadomości przez gniazdo TCP.
    - **Jak działa**:
        - Synchronizuje operacje I/O za pomocą blokady (Lock).

- **Kluczowe cechy**:
  - Zapewnia bezpieczeństwo wątkowe podczas komunikacji.

4. **closeConnection()**
   - **Cel**: Zamyka połączenie z klientem i zwalnia zasoby.

   - **Jak działa**:
     - Zamyka strumienie wejścia/wyjścia oraz gniazdo TCP.

   - **Kluczowe cechy**:
     - Zapewnia poprawne zakończenie pracy z klientem.

5. MathOperationFactory

(Odpowiada za wykonywanie operacji matematycznych)

Funkcjonalności:

- Mapuje nazwy operacji (ADD, SUB, MUL, DIV) na metody klasy **MathOperations**.
- Wykonuje operacje na liczbach całkowitych.

Przykład użycia:

```
int result = operationFactory.performOperation("ADD", 5, 3);
System.out.println(result); // Wynik: 8
```

6. MathOperations

Implementuje podstawowe operacje matematyczne:

- Dodawanie (add).
- Odejmowanie (subtract).
- Mnożenie (multiply).
- Dzielenie (divide).

7. StatisticsManager

(Zarządza statystykami serwera)

Przechowywane dane:

- Liczba podłączonych klientów.
- Liczba wykonanych operacji (całkowita i z ostatnich 10 sekund).
- Liczba błędnych operacji.
- Suma wyników.

Metody:

1. **recordClientConnection()** – Zwiększa licznik podłączonych klientów.
2. **recordOperation(String operation, int result)** – Aktualizuje statystyki dla wykonanej operacji.
3. **recordErrorOperation()** – Zwiększa licznik błędnych operacji.
4. **printStatistics()** – Wypisuje statystyki serwera na konsolę.

8. UDPHandler

Obsługuje komunikację UDP odpowiedzialną za:

- Odbieranie i wysyłanie wiadomości.
- Zamknięcie połączenia po zakończeniu pracy.

Metody:

1. **getMessage()**
   - **Cel**: Odbiera wiadomość UDP.
   - **Jak działa**: Tworzy bufor o rozmiarze 1024 bajtów, odbiera pakiet datagramu, a następnie konwertuje odebrane dane na łańcuch znaków. Zwraca obiekt Message zawierający odebraną wiadomość.

2. **sendMessage(InetAddress address, int port, Message message)**
   - **Cel**: Wysyła wiadomość UDP do określonego adresu i portu.
   - **Jak działa**: Konwertuje treść wiadomości na tablicę bajtów, tworzy pakiet datagramu i wysyła go za pomocą gniazda UDP.

Kluczowe cechy

- Klasa UDPHandler upraszcza proces komunikacji UDP, umożliwiając łatwe odbieranie i wysyłanie wiadomości.
- Obsługuje zarówno odbieranie, jak i wysyłanie wiadomości, przez co jest bardzo praktyczna.

9. Message

Klasa Message jest używana do reprezentowania wiadomości, które są przesyłane za pomocą gniazd UDP. Zawiera informacje o długości wiadomości, treści wiadomości oraz opcjonalnie o pakiecie datagramu, z którego wiadomość została odebrana.

Kluczowe cechy:

- Klasa Message jest używana do enkapsulacji danych wiadomości, co ułatwia ich przesyłanie i przetwarzanie.
- Umożliwia łatwe zarządzanie informacjami o wiadomości, co jest istotne w kontekście komunikacji sieciowej.