PROJEKT UDP

Rozproszony System Uśredniający

Milena Tarkowska
S30638

OPIS PROGRAMU:

Program implementuje komunikację sieciową między serwerem (Master) oraz klientami (Slave) w języku Java. Jego podstawowe zadania to odbierane i wysyłanie liczb całkowitych, wyliczanie z nich średniej arytmetycznej oraz obsługa zamknięcia połączenia.

UŻYWANY PROTOKÓŁ:

UDP (User Datagram Protocol)

Jeden z protokołów internetowych, używany w warstwie transportowej sieci. Jest bezpołączeniowy – nie ma inicjalizacji między nadawcą i odbiorcą. Każdy segment UDP jest obsługiwany niezależnie, nie mamy stanu połączenia u nadawcy i odbiorcy, nie ma kontroli przeciążenia – dane są wysyłane bez kontroli szybkości ich przepływu. Należy uważać na możliwość zgubienia, zdublowania lub zmiany kolejności w komunikacji pakietów.

W programie do obsługi komunikacji UDP używane są klasy DatagramSocket i DatagramPacket z pakietu java.net.

URUCHOMIENIE I DZIAŁANIE PROGRAMU:

Uruchamianie

Program uruchamiany jest z terminala za pomocą komendy *java DAS <nr portu> <liczba całkowita>* po wcześniejszej kompilacji (*javac DAS.java*) w katalogu, gdzie znajduje się plik źródłowy. Poprawność argumentów sprawdzana jest pod kątem ich liczby – powinny być dokładnie dwa - oraz czy zmienna *port* jest w zakresie od 1024 do 65535 (do 1023 są porty zarezerwowane, do 65536, który jest końcem zakresu ($2^{16} - 1$)). Jeżeli są niepoprawne, program zwróci błąd, jeżeli port podany w argumencie jest wolny, aplikacja wejdzie w tryb Master, natomiast jeżeli port będzie zajęty – wejdzie w tryb Slave.

TRYB MASTER

Aplikacja w trybie Master działa jako serwer, tworzy gniazdo na zadanym porcie i inicjalizuje listę liczb. Dodaje do listy wejściową liczbę (z argumentu) i czeka w pętli na dane od Slave'ów. Działania programu zależą od wartości przesłanej liczby:

- **0**: Oblicza średnią liczb z listy (pomijając zera) i odsyła wynik jako broadcast do sieci lokalnej na porcie 60000.
- **-1**: Odsyła otrzymaną wartość (-1) jako broadcast do sieci lokalnej i zamyka gniazdo.
- **Liczby różne od -1 i 0**: Dodaje liczbę do zbioru liczb.

TRYB SLAVE

W trybie Slave aplikacja działa jako klient. Tworzy gniazdo na losowym porcie, przygotowuje pakiet z liczbą daną w argumencie i wysyła go do Mastera. Po wysłaniu czeka na odpowiedź od serwera – czy pakiet został otrzymany i kończy pracę.

KLASY I METODY:

Klasa DAS

Główna klasa programu, obsługuje wybór i działanie trybów Master i Slave.

Pola klasowe:

- **Port (int)** – zmienna inicjalizowana przez użytkownika jako 1 argument (arg[0]) w wywołaniu programu. Oznacza port UDP, na którym ma działać program.
- **Number (int)** – zmienna inicjalizowana przez użytkownika jako 2 argument (arg[1]) w wywołaniu programu. Oznacza liczbę całkowitą przekazywaną dalej do programu.

Metody:

- **[public static void] modeMaster** – w argumentach ma podany port i numer jako liczby całkowite Int. Socket otwiera się na danym w argumentach porcie, ma możliwość rozgłaszania komunikatów w sieci lokalnej – jako broadcast do sieci lokalnej na porcie 60000. Tworzy listę liczb (Integer), w której przechowuje liczby wysłane od Slave'ów. W cyklicznej pętli, zależnej od zmiennej logicznej *isWorking*, program w trybie Master czeka na odebranie pakietów od Slave, omija je, jeżeli pochodzą z broadcastu od niego samego, a dalsze działania zależne są od wartości przesłanej do niego liczby:
  - **Dla 0** – wylicza średnią liczb, które zapisał na liście (z pominięciem zer) i odsyła wynik jako broadcast do swojej sieci lokalnej na porcie 60000.
  - **Dla -1** – odsyła otrzymaną wartość (-1) jako broadcast do sieci lokalnej i zamyka swój socket.
  - **Dla liczb różnych od -1 i 0 (default)** – dodaje liczbę do zbioru liczb.
- **[public static void] modeSlave** – w argumentach ma podany port i numer w typie liczby całkowitej. W tym trybie programu otwiera się socket UDP na losowym porcie (slaveSocket.getLocalPort()), podany w argumencie numer zamieniany jest na tablicę bajtów i tak przygotowany pakiet wysyłany jest do localhosta. Po tym socket czeka na odpowiedź od serwera, kończy pracę i jest zamykany.
- **[public static void] sendResponse** – w argumentach ma dany socket (DatagramSocket), response (String), address (InetAddress), port (int). Służy do obsługi mechanizmu wysyłania odpowiedzi z Mastera do Slave/innych urządzeń w sieci.
- **[public static boolean] isNumberInteger** – w argumencie ma zmienną typu String. Sprawdza, czy podana wartość w String, to liczba całkowita.

Klasa Log

Do zapisu logów w trakcie działania programu.

Pola klasowe:

- **PREFIX (public static String)** – w programie definiuje w jakim trybie znajduje się program, pojawia się przed zadanym komunikatem
- **LOG_FILE (public static final String)** – plik tekstowy, w którym zapisywana jest historia komunikatów wysłanych przez program

Metody:

- **[public static void] log** – jako parametr przyjmuje zmienną typu String, która oznacza komunikat wysyłany przez program. Logi pojawiają się na konsoli terminalu oraz zostają zapisane do pliku tekstowego w schemacie: [PREFIX] komunikat.

TESTOWANIE:

Testowanie aplikacji odbywało się poprzez uruchamianie jej z różnymi argumentami, tak aby sprawdzić wszystkie funkcjonalności. Do projektu zostały dołączone pliki .bat z komendami do skompilowania programu, uruchomienia trybu Master i trybu Slave dla liczb 0, -1 i 35 (różnej od 0 i -1).

PROBLEMY W REALIZACJI:

1. Rozgłoszenie komunikatu w sieci lokalnej, co ostatecznie udało się zrealizować dzięki zmiennej broadcastAddress typu InetAddress, do której za pomocą klasy NetworkInterface udało się otrzymać wyliczony adres broadcastowy dla sieci LAN.
2. Obsługa wysyłania odpowiedzi z Mastera do Slave'a w zależności od timeout'u – użycie klauzuli try-catch-finally do poprawnej obsługi zachowania programu.
3. Pojawiające się błędy w argumentach wejściowych np. niepoprawny numer portu – walidacja wprowadzonych przez użytkownika danych za pomocą instrukcji warunkowej IF.
4. Port Slave'a otwierał się na porcie (-1) zamiast losowej wartości – użycie slaveSocket.getLocalPort() zamiast slaveSocket.getPort().