SKJ projekt
Serhii Kovalenko s17187, 25c

I. Ogólny opis rozwiązania:

Głownym celem projektu jest realizacja połączenia UDP, a później TCP pomiędy serwerem i klientami (których może być dowolna liczba) i komunikacji między nimi. Dla implementacji rozwiązania tego problemu zostały stworzone programy o nazwach **Server** i **Klient**, które pełnią rolę serwera i klienta odpowiednio. Program jest napisany w jezyku Java8.

II. Szczególowy opis rozwiązania:

W parametrach programu serwera musimy podać listę numerów portów, które potem odczutyjemy i, oczywiście, sprawdzamy, czy przypadkiem nie podaliśmy port o numerze poniżej 1024 lub w ogóle zamiast numeru portu wpisaliśmy jakiegos dowolnego stringa. Jeżeli tak rzeczywiście jest, to kończymy pracę z odpowiednim komunikatem błędu, natomiast jeśli wszystko jest OK, to tworzymy nowy serwer i jako parametr podajemy listę portów UDP. W konstrukorze klasu **Server** na podanych numerach portów tworzymy **DatagramSocket**, które będą otrzymywać i wysyłać paikiety do klientów. Oczywiście, sprawdzamy, czy przypadkiem ten port nie jest zajęty przez jakiś inny proces, a jeżeli jest, podnosimy błąd i kończymy pracę. W innym przypadku nowo stworzony socket dodajemy do listy **sockets**.

Zeby umozliwić jednoczesną pracę z wieloma klientami, wszystkie sokety na tej liście muszą przyjmować komunikaty od klientów w odrębnych wątkach. W tym celu tworzymy nową klasę o nazwie **ReceiveThread**, która dziedziczy po **Thread** i implementuje interfejs **Runnable**. W konstruktorze klasy **Server** dla każdego **DatagramSocket** na liście **sockets** tworzymy nowy **ReceiveTheread**, któremu jako parametr podajemy **DatagramSocket**, i natychmiast dodajemy nowo stworzony wątek do listy **receiveThreads**, a potem uruhamiamy wszystkie wątki na tej liście. W klasie **ReceiveThread** w medozie **run()** została wywołana metoda **receiveData(socket)**, która przyjmuje dane od klientów. W tej metodzie tworzymy **DatagramPacket**, który otrzymamy od klienta. Zeby otzrymać pakiet, który wysłał klient,

musimy skorzystać z metody **receive(DatagramPacket)**, która wywoluje **DatagramSocket**. Otrzumujemy adres IP i port klienta przy pomocy metod **getAddress()** i **getPort()** klasy **DatagramPacket** żeby wiedzieć, do kogo potem wysyłac odpowiedź. Serwer wysyła odpowiedź tylko wtedy, gdy klient wyśle pakiet o odpowiedniej treści, który będzie sygnałem tego że on czeka na port TCP. W moim przypadku klient musi wysłać "getTCPConnection" żeby serwer wysłał mu port TCP.

Po otrzymaniu danych od klienta serwer sprawdza, czy jest to pakiet o treści "getTCPConnection", i jeżeli tak jest muśimy zrealizować pomiędzy serwerem a klientem połączenie TCP. Ponieważ, serwer jednocześnie pracuje z wieloma klientami i może tak zdarzyć, że na jeden port, na którym on nasłuchuje pakiety od klientów, jednocześnie przyjdzie prośba o połączeniu TCP od dwóch lub więcej klientów, więc, całe te połączenie musimy uruhomić w odrębnym wątku, żeby to w żadny sposób nie przeszkadzało otrzymaniu pakietów lub nawiązaniu połączenia przez innych klientów. W tym celu, jeżeli otrzymaliśmy pakiet o treści "getTCPConnection", tworzymy nowy wątek i w metodzie **run()** nawiązujemy połączenie. Najpierw musimy ustalić port TCP i sprawdzić, czy nie jest on przypadkiem zajęty, a jeżeli jest, to zwiększyć o jeden i sprawdzić ponownie. W klasie **ReceiveThread** mamy pole **static port = 6999**, które jest polem pomocniczym w ustaleniu portów TCP. Natomiast w naszym wątku mamy pole **int TCP_Port = 0**, z którym będziemy pracować. Dopóki **TCP_Port == 0**, probujemy przypisać **TCP_Port = port** i stworzyć **SerwerSocket socket**, który jest niezbędny do komunikacji przy polączeniu TCP. Jeżeli taki port już jest zajęty, zwiekszamy zmienną **port** o jeden i probujemy ponownie. Ponieważ, zmienna **port** jest statyczna, jej znaczenie zmienia sie dla wszystkich innych obiektów klasy **ReceiveThread**. Po ustawieniu portu TCP wysyłamy jego do klienta przy pomocy metody **send(DatagramPacket)** klasy **DatagramSocket** i czekamy, dopóki klient się podłączy i wyśle jakiś dowolny string. Stworzymy **Socket TCPSocket** przy pomocy metody **accept()** klasy **SerwerSocket**, który jest porzebny do komunikacji. Przyjmujemy dane od klienta przy pomocy metody **getInputStream()** klasy **Socket**, którą podajemy jako parametr w nowo stworzony **Scanner scanner**, a wysyłać dane bedziemy przy pomocy **PrintWriter writer**, któremu podajemy w parametrach metodę **getOutputStream()**. Wreście, kiedy otrzymamy dane (**scanner.nextLine()**),

wyślimy ich z powrotem do klienta dodając na początku "Server TCP: ". Po wyslaniu odpowiedzi do klienta kończymy połączenie TCP, zamykając ServerSocket i Socket.

Po zakończeniu działania tego wątku TCP (lub po sprawdzeniu zawartości pakietu, który był wysłany przez użytkownika), uruhamiamy ponownie metodę **receiveData(DatagramSocket)**, czyli jest to metoda rukuręcyjna.

Teraz zajmiejmy się programem klienta. W swoich parametrach on przyjmuje adres ip serwera, a potem listę portów, na ktorę on będzie “pukał”. Tak samo, jak i w programie serwera, sprawdzamy poprawność argumentów. W przypadku adresu ip może powstać wyjątek **UnknownHostException**, a dla numerów portów nie możemy podać liczby ujemnej lub coś, co w ogóle nie jest liczbą. Po sprawdzeniu tworzymy nowego klienta i jako parametry podajemy adres ip serwera i listę portów UDP.

W konstruktorze klienta tworzymy **DatagramSocket client socket** i ustawiamy timer oczekiwania odpowiedzi przy pomocy metody **setSoTimeout(1000)** klasy **DatagramSocket**. Teraz po wysłaniu pakietu na jakiś z portów srewera czekamy 1 sekundę na odpowiedź, i jeżeli odpowiedzi nie ma, możemy kontynuować pracę (w naszym przypadku, wysłac ten pakiet na kolejny port). Teraż tworzymy wątek, w którym będziemy wysylali i otrzymywali pakiety do/od serwera, dopóki tego wątku nie przerwiemy. Zrobiłem tak, że klient po sprobie wysłania jakiegokolwiek pakietu na wszystkie porty z listy ma możliwość kontynuować swoją pracę, czyli wysyłac pakiety o innej treści na te porty, dopóki sam nie zdecyduje przerwac się. Zeby przerwać działanie wątku i skończyć pracę klient musi napisać polecenie "/end".

W wątku, dopóki jego nie przerwiemy, wywolujemy jedynie metodę **sendData(clientSocket),** która przyjmuje obiekt klasy **DatagramSocket**. Ta metoda będzie wywolywała metodę **receiveData(clientSocket)**, która czeka na odpowiedź od serwera. Dane odczytujemy z konsoli przy pomocy **BufferedReader** i w cyklu dla wszystkich portów na liście **UDP_Ports** tworzymy pakiety i wysyłamy ich do posczególnych portów. Sprawdzamy, czy przypadkiem klient nie napisał polecenie "/end", i jeżeli tak jest,

przerywamy wątek, wypisujemy odpowiedni komunikat i kończymy pracę. W przeciwnym przypadku wywolujemy metodę **receiveData(clientSocket)**, która czeka 1 sekundę na odpowiedź od serwera, i jeżeli ona nie nastąpi (powsztanie wyjątek **SocketTimeoutException**), wypisze odpowiedni komunikat i pakiet będzie wysłany do kolejnego portu. Ponieważ, serwer wyśle odpowiedź tylko wtedy, kiedy zostanie poproszony o stworzeniu portu TCP, więc, w przypadku wysłania do nas przez serwer portu TCP metoda **receiveData(clientSocket)** otrzyma te dane w mniej wiecej taki sposob, jak jest to realizowane w serwerze, zapisze ten numer portu w zmienną **int TCP_Port** i wywola metodę **connectTCP(int TCP_Port)**, która służy do nawiązania połączenia TCP na podanym w parametrze porcie. Ona tworzy Socket, który potem wywoluje metode **connect(InetSocketAddress)**, służącą do połączenia z serwerem TCP (tworzymy nowy **InetSocketAddress** podając w parametrach adres IP serwera i numer portu TCP). Dalej przy pomocy **Scanner** i **PrintWriter** zczytujemy z konsoli i wysyłamy do serwera to, co wpiszemy i czekamy na jego odpowiedź po czym połączenie się kończy. Po zakończeniu połączenia TCP, kontynujemy wysyłanie naszego pakietu do innych portów z listy.