

LISTA 4

TECHNOLOGIE SIECIOWE



31 maja 2019

MIKOŁAJ DUKIEL

235908

**1. Cel**

Zadanie polega na przekształceniu podanych programów które służą symulowaniu przesyłania pakietu pomiędzy dwoma urządzeniami przy pomocy protokołu TCP/IP. Po przekształceniu programy, wykorzystując potwierdzenia dostarczenia oraz numery sekwencyjne przez nadawcę i odbiorcę, miały dawać gwarancję że odbiorca otrzyma ostatecznie pełen niezmieniony komunikat, który został pierwotnie nadany.

**2. Realizacja**

**2.1 Definicje pojęć**

**2.1.1 TCP** (Transmission Control Protocol)

Protokół służący do przesyłania danych między hostami. Gwarantuje on dostarczenie wszystkich pakietów w całości, z zachowaniem kolejności i bez duplikatów (protokół niezawodny). TCP działa w trybie klient – serwer. Serwer oczekuje na nawiązanie połączenia w określonym porcie, a klient inicjuje połączenie.  
 **2.1.2 TCP/IP**

Jest to rozszerzenie protokołu TCP, korzystające z protokołu IP, do przesyłania pakietów danych przez sieć.

**2.1.3 Datagram**

Podstawowa jednostka przekazu, powiązana z siecią komutacyjną pakietów. Zazwyczaj datagramy zbudowane są z sekcji nagłówka i ładunku. Dostarczają one możliwość bezpołączeniowej komunikacji w sieci komutacyjnej pakietów.

**2.2 Opis programów przed modyfikacją**

**2.2.1 Z2Sender**

Symuluje działanie hosta wysyłającego dane. Program wysyła w osobnych datagramach po jednym znaku wczytanym z wejścia do portu o numerze podanym jako drugi parametr wywołania programu. Jednocześnie drukuje na wyjściu informacje o pakietach otrzymanych w porcie podanym jako pierwszy parametr wywołania.

**2.2.2 Z2Packet**

Klasa, która imituje pakiet w datagramie.

**2.2.3 Z2Receiver**

Symuluje działanie hosta odbierającego dane. Program drukuje informacje o każdym pakiecie, który otrzymał w porcie o numerze podanym jako pierwszy parametr wywołania programu i odsyła go do portu podanego jako drugi parametr wywołania programu.

**2.2.4 Z2Forwarder**

Program symuluje połączenie internetowe w którym każdy pakiet przesyłany jest niezależnie i w miarę dostępnych możliwości. W związku z tym pakiety wysyłane przez nadawcę mogą być tracone, przybywać z różnymi opóźnieniami, w zmienionej kolejności, a nawet mogą być duplikowane.

**2.3 Opis programów po modyfikacji**

W czasie realizacji zadania modyfikacji zostały poddane programy: Z2Reciver i Z2Sender.

**2.3.1 Z2Sender**

Klasa SenderThread, w której dodajemy pakiet do mapy.

class SenderThread extends Thread  
{  
 public void **run**()  
 {  
 int i, x;  
 try  
 {  
 for(i=0; (x=System.*in*.read()) >= 0 ; i++)  
 {  
 Z2Packet p=new Z2Packet(4+1);  
 p.setIntAt(i,0);  
 p.data[4]= (byte) x;  
 DatagramPacket packet =  
 new DatagramPacket(p.data, p.data.length,  
 localHost, destinationPort);  
 sent.put(i,packet);  
 socket.send(packet);  
 *sleep*(*sleepTime*);  
 }  
 }  
 catch(Exception e)  
 {  
 System.*out*.println("Z2Sender.SenderThread.run: "+e);  
 }  
 }

Poniżej usuwamy element w klasie ReceiverThread, po odebraniu pakietu zwrotnego.

class ReceiverThread extends Thread  
{  
  
 public void **run**()  
 {  
 try  
 {  
 while(true)  
 {  
 byte[] data=new byte[*datagramSize*];  
 DatagramPacket packet=  
 new DatagramPacket(data, *datagramSize*);  
 socket.receive(packet);  
 Z2Packet p=new Z2Packet(packet.getData());  
 sent.remove(p.getIntAt(0));  
 System.*out*.println("S:"+p.getIntAt(0)+  
 ": "+(char) p.data[4]);  
 }  
 }  
 catch(Exception e)  
 {  
 System.*out*.println("Z2Sender.ReceiverThread.run: "+e);  
 }  
 }  
  
}

Następnie mamy klasę ResenderThread, której używamy do ponownego wysłania pakietu, w przypadku gdy nie dotrze do celu. Tworzymy nową **hashmapę**, która posłuży nam jako kopia danych (dane te są kopią wysyłanych pakietów), po czym wysyłamy pakiet.

class ResenderThread extends Thread  
{  
 public void run()  
 {  
 HashMap<Integer, DatagramPacket> copy = new HashMap();  
 try  
 {  
 while(true) {  
 if(sent.size() != 0) {  
 if(copy.size() != 0) {  
 for (HashMap.Entry<Integer, DatagramPacket> entry : copy.entrySet()) {  
 int key = entry.getKey();  
 DatagramPacket packet = sent.get(key);  
 if (packet != null) {  
 socket.send(packet);  
 }  
 }  
 }  
  
 copy.clear();  
 synchronized (sent) {  
 for (HashMap.Entry<Integer, DatagramPacket> entry : sent.entrySet()) {  
 copy.put(entry.getKey(),entry.getValue());  
 }  
 }  
 }  
 *sleep*(*sleepTime*\*6);  
 }  
 }  
 catch(Exception e)  
 {  
 System.*out*.println("Z2Sender.ReceiverThread.run: "+e);  
 }  
 }  
  
}

**2.3.2 Z2Receiver**

W tej klasie drukowane są odebrane pakiety, które są kolejkowane w **TreeMap**. Zmienna current wyznacza nam pakiet do odebrania. Gdy to nastąpi zostaje on pobrany, wydrukowany i usunięty z **TreeMap**.

class ReceiverThread extends Thread  
{  
  
 public void run()  
 {  
 int current = 0;  
 int code;  
 char character;  
 try  
 {  
 while(true)  
 {  
 byte[] data=new byte[*datagramSize*];  
 DatagramPacket packet=  
 new DatagramPacket(data, *datagramSize*);  
 socket.receive(packet);  
 Z2Packet p = new Z2Packet(packet.getData());  
 code = p.getIntAt(0);  
 character = (char) p.data[4];  
  
 if(code == current){  
 current++;  
 System.*out*.println("R:"+code +": "+character);  
  
 while (!received.isEmpty() && received.firstKey() == current) {  
 Map.Entry<Integer, Character> entry = received.pollFirstEntry();  
 System.*out*.println("R:" + entry.getKey() + ": " + entry.getValue());  
 current++;  
 }  
  
  
 }else if (code > current){  
 received.put(code,character);  
 }else{  
 continue;  
 }  
  
 packet.setPort(destinationPort);  
 socket.send(packet);  
 }  
 }  
 catch(IOException e)  
 {  
 System.*out*.println("Z2Receiver.ReceiverThread.run: "+e);  
 }  
 }  
  
}

**3. Wnioski**

Programy te udowodniają, że da się zabezpieczyć przesyłanie pakietów w sieci przed komplikacjami związanymi z drogą. Nawet zgubienie pojedynczego pakietu nie musi oznaczać niepowodzenia operacji, lecz tylko nieco ją przedłużyć. Programy pokazują również, że pakiety nie muszą być odbierane w ściśle określonej kolejności, gdyż przy dobrym skonstruowaniu datagramów treść można odtworzyć gdy dotrą one nie po kolei. Tak właśnie działa protokół TCP/IP.