Wrocław, 04.06.2017

Laboratorium

Technologii Sieciowych

Temat:

Sprawozdanie 4

Autor: Bartosz Banasik
Informatyka, semestr: 4
Prowadzący: dr Łukasz Krzywiecki

Spis Treści:

1. Cel Ćwiczenia

Zadanie polega na takim wykorzystaniu potwierdzeń i numerów sekwencyjnych przez nadawcę i odbiorcę, aby odbiorca wydrukował wszystkie pakiety w kolejności ich numerów sekwencyjnych, nawet jeśli połączenie w obie strony odbywa się przez `Z2Frowarder`.

2. Realizacja

2.1 Opis Programu

2.1.1 Program `Z2Sender` 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.

2.1.2 Program `Z2Receiver` 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.

2.1.3 Klasa `Z2Packet` umożliwia wygodne wstawianie i odczytywanie czterobajtowych liczb całkowitych do tablicy bajtów przesyłanych w datagramie - metody: public void setIntAt(int value, int idx) oraz public int getIntAt(int idx). Wykorzystane jest to do wstawiania i odczytywania numerów sekwencyjnych pakietów.

2.1.4 Program `Z2Forwarder` symuluje działanie przesyłu danych w Internecie. 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.

2.1.5 Zmiany wprowadzone w celu realizacji zadania.

2.1.5.1 Zmiany w prorgamie Z2Sender

Do programu `Z2Sender` dodano wątek odpowiedzialny za sprawdzanie I ewentualne ponowne wysyłanie pakietów – `ResendThread,` listę elementów już wysłanych – `ArrayList<byte> orgData`, zmienne zarządzające czasem takie jak: `restransmissionWindow` – ilość pakietów wysłanych bez potwierdzenia oraz dwie zmienne synchronizujące ten wątek z `SenderThread`. Klasa `ResendThread` odczekuje określoną ilość wysłanych pakietów, po czym wstrzymuje działanie na określony czas. Następnie sprawdza jaki był ostatni najwyższy pakiet, który otrzymała klasa `ReceiverThread.` Analizując te dane oraz dane z `orgData`, klasa podejmuje decyzje, które pakiety ponownie przesłać, lub wznowić wysyłanie dalszych pakietów w klasie `SenderThread.`

```
130     class ResendThread extends Thread {
131         public void run(){
132             try{
133                 while(true){
134                     if(goNext)
135                         sleep(sleepTime * retsransmissionWindow);
136                     sleep(reSleep);
137                     System.out.println("Sprawdzam");
138                     Z2Packet pakiet;
139                     if(lastPacket != null){
140                         pakiet = new Z2Packet(lastPacket.getData());
141                     } else {
142                         pakiet = new Z2Packet(4 + 1);
143                         pakiet.setIntAt(0,0);
144                         pakiet.data[4] = (byte) 'X';
145                     }
146                     if( (pakiet.getIntAt(0)+1) >= packetSend) {
147                         latch.countDown();
148                         goNext = true;
149                         continue;
150                     } else {
151                         goNext = false;
152
```

```
152                         if(lastPacket != null){
153                               Z2Packet p = new
153                                     Z2Packet(lastPacket.getData());
154                             int id = p.getIntAt(0);
155                            for (int i = id + 1; i < orgData.size(); i++){
156                                 Z2Packet pa = new Z2Packet(4+1);
157                                 pa.setIntAt(i,0);
158                                 pa.data[4] = (byte) orgData.get(i);
159                                 DatagramPacket packet =
160                                 new DatagramPacket(pa.data,
160                                                 pa.data.length,
161                                         localHost, destinationPort);
162                                 socket.send(packet);
163                                 System.out.println("RS: "+ i);
164                                 sleep(sleepTime);
165                             }
166                          } else {
167                             for (int i = 0; i < orgData.size(); i++){
168                                 Z2Packet p = new Z2Packet(4+1);
169                                 p.setIntAt(i,0);
170                                 p.data[4] = (byte) orgData.get(i);
171                                 DatagramPacket packet =
172                                 new DatagramPacket(p.data, p.data.length,
173                                         localHost, destinationPort);
174                                 socket.send(packet);
175                                 System.out.println("RS: "+ i);
176                                 sleep(sleepTime);
177                             }
178                          }
179                     }
180                     latch.countDown();
181                 }
182             } catch (Exception e) {
183                 System.out.println("ResendThread");
184                 e.printStackTrace();
185             }
186         }
167   }
```

Text 1: listing1 ResenderThread

Wątek `SenderThread` dodatkowo po wysłaniu pakietu dodaje go do listy `orgData` oraz sprawdza, czy po tym pakiecie ma czekać na potwierdzenie.

```
72                      orgData.add((byte)x);
73                      packetSend++;
74                      if(packetInRow == retsransmissionWindow) {
75
76                          latch.await();
77                          latch = new CountDownLatch(1);
78                          packetInRow = 0;
79                      }
80                      packetInRow++;
81                      sleep(sleepTime);
82
83                      while(!goNext){
84                          sleep(sleepTime);
85                      }
```

Text 2: listing2 SenderThread

Klasa `ReceiverThread` dodatkowo aktualizuje zmienną `lastPacket`, jeśli potwierdzenie, które przyszło, wnosi postęp w wysyłaniu danych.

```
109                     Z2Packet p = new Z2Packet(packet.getData());
110                     if(lastPacket == null){
111                         lastPacket = packet;
112                     } else {
113                         Z2Packet l = new Z2Packet(lastPacket.getData());
114                         if(p.getIntAt(0)>l.getIntAt(0))
115                         lastPacket = packet;
116                     }
```

Text 3: listing3 ReceiverThread

2.1.5.2 Zmiany w programie Z2Receiver

Została dodana nowa klasa jako wątek `ResendThread`, która po określonym czasie ponownie ubiega się o otrzymanie nowego pakietu. Została ona przedstawiona na `listingu 5`. Dodatkowo klasa `ReceiverThread` kolekcjonuje pakiety, które przyszły za wcześnie w tablicy haszującej `Hashtable<Integer, DatagramPacket> queue`. Po otrzymaniu spodziewanego pakietu klasa wypisuje go na standardowe wyjście oraz sprawdza czy ma następne pakiety w kolejce, jeśli tak to również je wypisuje. Po tych czynnościach wysyła potwierdzenie.

```
33    class ReceiverThread extends Thread {
34         // Na poczatku oczekujemy id
35
36         Hashtable<Integer, DatagramPacket> queue;
37
38         public void run() {
39             queue = new Hashtable<Integer, DatagramPacket>();
40             try {
41                 while(true)
42                 {
43                     byte[] data = new byte[datagramSize];
44                     DatagramPacket packet = new DatagramPacket(data,
45                                                                datagramSize);
45                     socket.receive(packet);
46
47                     Z2Packet p = new Z2Packet(packet.getData());
48                     int id = p.getIntAt(0);
49
50                     if(id != expected) {
51
52                         if(!queue.containsKey(id) && id >= expected) {
53                             queue.put(id, packet);
54                             System.out.println("queue "+ id);
55                         }
56
57                     } else {
58                         // Pdebralismy pakiet na ktory czekamy
59                         System.out.println("R:" + p.getIntAt(0) + ": "
59                                                 + (char) p.data[4]);
60                         expected++;
61                         // sprawdzamy czy wczesniej nie odebralismy juz
61                                              nastepnych pakietow
62                         while(queue.containsKey(expected)) {
63                             packet = queue.get(expected);
64                             queue.remove(expected);
65                             p = new Z2Packet(packet.getData());
66                             System.out.println("R:" + p.getIntAt(0)
66                                            + ": " + (char) p.data[4]);
67                             expected++;
68                         }
69                         // wysylamy ostatni pakiet jaki mamy wyswietlony
70                         packet.setPort(destinationPort);
71                         socket.send(packet);
72                         System.out.println("next: "+ expected);
73                     }
74                 }
75             }
76             catch(Exception e)
77             {
78                 System.out.println("Z2Receiver.ReceiverThread.run: "+e);
79             }
80         }
81
82     }
```

Text 4: listing 4 ReceiverThread

```
84     class ResendThread extends Thread {
85         int waiting = 0;
86         public void run() {
87             try {
88                 while(true){
89                     sleep(sleepTime);
90                     if(waiting == expected){
91                         Z2Packet p = new Z2Packet(4+1);
92                         p.setIntAt(waiting-1,0);
93                         p.data[4] = (byte) 'R';
94                         DatagramPacket packet = new DatagramPacket(p.data,
94                           p.data.length, localHost, destinationPort);
95                         socket.send(packet);
96                         System.out.println("Ponawiam prosbe "+ waiting);
97                     } else {
98                         waiting = expected;
99                     }
100                }
101             } catch (Exception e){
102                 e.printStackTrace();
103             }
104         }
105     }
```

Text 5: listing 5 ResendThread

2.2 Przykładowe wywołanie

W jednym oknie terminala uruchamiamy Program `Sender` wraz z dwoma programami `Z2Forwarder`. W drugim oknie uruchamiamy program `Receiver.`

```
Input1: java Z2Forwarder 6001 6002 & java Z2Forwarder 6003 6000 & java Sender 6000 6001

Input2: java Receiver 6002 6003
```

Listing 6 oraz 7 przedstawiają dane otrzymane podczas uruchomienia powyższych programów. W programie `Sender` oznaczenie `S: 0: A` mówi nam, że został wysłany pakiet o `id = 0,` z wiadomością `‘A’.` Analogicznie `RS: 0` mówi nam, że pakiet `0` został ponownie wysłany. Dodatkowo symbolem `r:` oznaczane są pakiety odebrane. Program `Receiver` wyświetla nam odebrane pakiety, żądanie ponownego wysłania pakietu oraz pakiety, które kolejkóje.

```
$ java Z2Forwarder 6001 6002 &
java Z2Forwarder 6003 6000 &
java Sender 6000 6001 < plik.txt
[1] 3635
[2] 3636
S: 0: A
S: 1: l
S: 2: a
S: 3:
S: 4: m
S: 5: a
S: 6:
Sprawdzam
RS: 0
RS: 1
RS: 2
RS: 3
RS: 4
RS: 5
RS: 6
r: -1: R
r: -1: R
r: 6:
Sprawdzam
S: 7: k
S: 8: o
S: 9: t
S: 10: a
S: 11:
```

Text 7: listing 6 Sender

```
$ java Receiver 6002 6003
queue 1
Ponawiam prosbe 0
queue 5
queue 4
queue 2
queue 6
queue 3
Ponawiam prosbe 0
R:0: A
R:1: l
R:2: a
R:3:
R:4: m
R:5: a
R:6:
next: 7
Ponawiam prosbe 7
queue 8
queue 9
queue 11
R:7: k
R:8: o
R:9: t
next: 10
R:10: a
```

Text 6: listing 7 Receiver

2.3 Analiza danych

Na listingu 7 widać, że program zadziałał poprawnie. Wyświetlone pakiety występują w kolejności rosnącej. Dodatkowo widać, że program kolejkóje pakiety “na zapas”. Natomiast program `Sender` ponawia wysyłanie pakietów, tak długo, aż nie przyjdzie potwierdzenie, jeśli takie już przyszło, wznawiane jest wysyłanie dalszych pakietów.

3. Wnioski

Programy `Z2Sender`, `Z2Reciver` wraz z programem `Z2Forwarder` w jasny i przejrzysty sposób ilustrują przesyłanie pakietów internetowych za pomocą protokołu TPC/IP. Programy przybliżają w przystępny sposób problem potwierdzania doręczania pakietów i kolejności ich dostarczania.