Sprawozdanie 4

Józef Piechaczek

2019-05-26

1 Omówienie zadania

Celem zadania jest poznanie modelu TCP/IP. Mając cztery programy, symulujące przepływ danych pomiędzy dwoma użytkownikami sieci, należy je zmodyfikować w taki sposób, aby dane nie ulegały zgubieniu/duplikacji oraz były wyświetlane w porządku określonym przez numery sekwencyjne.

Do dyspozycji sa cztery przykładowe programy:

- **Z2Sender** - program przyjmuje dwa parametry wejściowe. Pierwszy parametr określa port na którym program nasłuchuje potwierdzeń otrzymania pakietów. Drugi parametr określa port, na który zostają wysłane pakiety, wygenerowane na podstawie danych otrzymanych na standardowym wejściu.
- **Z2Receiver** - program przyjmuje dwa parametry wejściowe. Pierwszy parametr określa port, na którym program nasłuchuje otrzymane pakiety. Pakiety te są następnie drukowane na standardowe wyjście. Drugi parametr określa port, na który przekazywane są potwierdzenia otrzyamania pakietów. (w podstawowej wersji potwierdzenie otrzymania pakietu oznacza odsyłanie każdego z odebranych pakietów)
- **Z2Forwarder** - program symuluje rzeczywsite połączenie w sieci. Podczas przekazywania pakietu z pierwszego portu na drugi program symuluje gubienie oraz duplikację pakietów.
- **Z2Packet** - klasa umożliwia wygodne wstawianie i odczytywanie czterobajtowych liczb całkowitych do tablicy bajtów przesyłanych w datagramie.

2 Wyniki przed modyfikacją

2.1 Przykład 1

Podane programy, przed modyfikacją, w przypadku następującego wywołania:

```
java Z2Receiver 6001 6000 & java Z2Sender 6000 6001 < plik.txt
```

zwracały poprawny wynik, ponieważ nie było zakłóceń pomiędzy użytkownikami

Z2Sender

```
S:0: A   S:9: t   S:17: n  S:26: t  S:34: m  S:43: t
S:1: l   S:10: a  S:18: i  S:27: a  S:35: a  S:44: y
S:2: a   S:11: .  S:19: e  S:28: .  S:36:    S:45: .
S:3:     S:12:    S:20:    S:29:    S:37: d  S:46:
S:4: m             S:21: m          S:38: w
S:5: a   S:13: O  S:22: a  S:30: E  S:39: a
S:6:     S:14: l  S:23:    S:31: l  S:40:
S:7: k   S:15: a  S:24: k  S:32: a  S:41: k
S:8: o   S:16:    S:25: o  S:33:    S:42: o
```

Z2Receiver

```
R:0: A   R:9: t   R:17: n  R:26: t  R:34: m  R:43: t
R:1: l   R:10: a  R:18: i  R:27: a  R:35: a  R:44: y
R:2: a   R:11: .  R:19: e  R:28: .  R:36:    R:45: .
R:3:     R:12:    R:20:    R:29:    R:37: d  R:46:
R:4: m             R:21: m          R:38: w
R:5: a   R:13: O  R:22: a  R:30: E  R:39: a
R:6:     R:14: l  R:23:    R:31: l  R:40:
R:7: k   R:15: a  R:24: k  R:32: a  R:41: k
R:8: o   R:16:    R:25: o  R:33:    R:42: o
```

2.2 Przykład 2

W przypadku następującego wywołania:

```
java Z2Receiver 6002 6003 & java Z2Forwarder 6001 6002 &
java Z2Forwarder 6003 6000 & java Z2Sender 6000 6001 < plik.txt
```

pierwszy Z2Forwarder przekazuje pakiety od Z2Sender do Z2Receiver, a drugi - w przeciwnym kierunku. Powoduje to gubienie oraz duplikację pakietów.

Z2Sender

S:1: l	S:9: t			S:31: l	S:46:
S:3:	S:17: n	S:34: m	S:25: o	S:37: d	
S:8: o	S:16:	S:22: a	S:26: t	S:37: d	S:44: y
S:2: a	S:15: a	S:32: a	S:31: l	S:34: m	
S:8: o	S:12:	S:27: a	S:36:	S:45: .	
S:14: l		S:20:	S:28: .	S:45: .	
S:5: a	S:10: a	S:24: k	S:42: o	S:43: t	
S:11: .	S:12:	S:29:	S:37: d	S:39: a	

Z2Receiver

R:3:	R:14: l	R:12:	R:24: k	R:27: a	R:39: a
R:5: a	R:16:		R:25: o	R:28: .	R:45: .
R:6:	R:9: t	R:15: a	R:26: t	R:42: o	R:44: y
R:7: k	R:10: a	R:20:	R:34: m	R:36:	R:46:
R:1: l	R:18: i	R:29:	R:32: a	R:43: t	
R:2: a	R:11: .		R:37: d	R:34: m	
R:8: o	R:17: n	R:22: a	R:31: l	R:45: .	

3 Modyfikacja programu

3.1 Z2Sender

3.1.1 SenderThread

Pierwszą modyfikacją dokonaną w programie Z2Sender jest wczywanie i utworzenie wszystkich pakietów przed rozpoczęciem wysyłania, oraz umieszczenie ich w *ArrayList*.

Z2Sender po wczytaniu wszystkich pakietów rozpoczyna wysyłanie. Wysyłanie polega na wysłaniu określonej przez bufor ilości pakietów, począwszy od ostatnio potwierdzonego. W danym przykładzie bufor wynosi 10.

Po wysłaniu pakietów Z2Sender oczekuje pewną określoną ilość czasu.

Kod wątku:

```
public void run() {
    readAllPackages(packetList);
    try {
        while(true) {
            int first;
            first = lastReceived;
            for (int i = 0; i < buffer && (first + i) < packetList.size(); i++)
                var p = packetList.get(first + i);
                socket.send(p);
                System.out.print("SENDER: Sent " + (first + i) + ": " + ((char)
            }
            sleep(millis);
        }
    } catch (InterruptedException | IOException e){
        System.out.println("Z2Sender.SenderThread.run: " + e);
    }
}
```

3.1.2 ReceiverThread

Po modyfikacji ReceiverThread otrzymuje informację, od jakiego datagramu powininen rozpocząć kolejne wysyłanie. W przypadku otrzymania wartości niższej niż aktualna wartość jest ignorowana.

Kod wątku:

```
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());
            var newLastReceived = p.getIntAt(0);
            if (newLastReceived > lastReceived) {
                lastReceived = newLastReceived;
                System.out.print("SENDER: Received " + newLastReceived + "\n");
           } else {
                System.out.print("SENDER(IGNORED): Received " + newLastReceived
           }
       }
    } catch (IOException e) {
        System.out.println("Z2Sender.ReceiverThread.run: " + e);
    }
}
```

3.2 Z2Receiver

3.2.1 SenderThread

W klasie Z2Receiver została stworzona klasa wewnętrzna SenderThread odpowiedzialna za wysyłanie informacji o oczekiwanym pakiecie co określony czas.

Oczekiwany pakiet, to taki pakiet dla którego otrzymaliśmy wszystkie poprzednie pakiety.

Kod wątku:

```
public void run() {
    while (true) {
        try {
            sleep(millis);
            if (lastRead > 0) {
                var z2p = new Z2Packet(4);
                z2p.setIntAt(lastRead + 1, 0);
                socket.send(new DatagramPacket(z2p.data, z2p.data.length, local
                System.out.print("RECEIVER: Sent " + z2p.getIntAt(0) + "\n");
            }
        } catch (IOException | InterruptedException e) {
            System.out.println("Z2Receiver.SenderThread.run: " + e);
        }
    }
}
```

3.2.2 ReceiverThread

Istniejący wątek ReciverThread został zmodyfikowany tak, aby po odczytaniu wiadomości umieścić ją na odpowiednim miejscu w liście. W przypadku, gdy numer sekwencyjny otrzymanego pakietu jest większy, niż numer sekwencyjny ostatnio odczytanego pakietu, oraz nie ma pomiędzy nimi nieotrzymanych pakietów następuje odczytanie otrzymanych wartości oraz zwiększenie numeru sekwencyjnego oczekiwanego pakietu.

Kod wątku:

```
public void run() {
    try {
        while (true) {
            var data = new byte[datagramSize];
            var packet = new DatagramPacket(data, datagramSize);
            socket.receive(packet);
            var z2p = new Z2Packet(packet.getData());
            var seq = z2p.getIntAt(0);
            var message = (char) z2p.data[4];

            updateList(seq);
            if (decodedPacketList.get(seq) == null) {
                decodedPacketList.set(seq, message);
            }
            if (seq > lastRead && checkList(lastRead, seq)) {
                for (int i = lastRead + 1; i <= seq; i++) {
                    System.out.print("RECEIVER: Received " + i + ": " + decoded
                }
                lastRead = seq;
            }
        }
    } catch (IOException e) {
        System.out.println("Z2Receiver.ReceiverThread.run: " + e);
    }
}
```

4 Wyniki po modyfikacji

4.1 Przykład 1

W przykładzie pierwszym otrzymany wynik był taki sam jak w pierwszym wypadku, z tą różnicą, że datagramy były przesyłane w ”paczkach” po 10.

4.2 Przykład 2

Przez strzałkę w prawą stronę ozncznaono pakiety wysłane, a przez strzałkę w lewą stronę pakiety odebrane. Przez znak (I) oznaczono informacje, które zostały zignorowane.

Z2Receiver

```
R: <- 0: A
R: <- 1: l
R: <- 2: a
R: <- 3:
R: <- 4: m
R: <- 5: a
R: -> 6
R: <- 6:
R: -> 7
R: <- 7: k
R: <- 8: o
R: <- 9: t
R: -> 10
R: -> 10
R: <- 10: a
R: <- 11: .
R: <- 12:
R: -> 13
R: -> 13
R: <- 13: O
R: <- 14: l
R: <- 15: a
R: <- 16:
R: <- 17: n
R: -> 18
R: <- 18: i
R: <- 19: e
R: -> 20
R: <- 20:
R: -> 21
R: <- 21: m
R: <- 22: a
R: <- 23:
R: <- 24: k
R: <- 25: o
R: <- 26: t
R: <- 27: a
R: -> 28
R: <- 28: .
R: -> 29
R: <- 29:
R: <- 30: E
R: -> 31
R: -> 31
R: -> 31
R: <- 31: l
R: <- 32: a
R: <- 33:
R: <- 34: m
R: <- 35: a
R: <- 36:
R: <- 37: d
R: <- 38: w
R: -> 39
R: -> 39
R: <- 39: a
R: <- 40:
R: -> 41
R: -> 41
R: -> 41
R: <- 41: k
R: <- 42: o
R: -> 43
R: <- 43: t
R: -> 44
R: <- 44: y
R: <- 45: .
R: -> 46
R: -> 46
R: -> 46
R: -> 46
```

Jak można zauważyć, pakiety wydrukowane przez Z2Receiver znajdują się w odpowiedniej kolejności, mimo ich modyfikacji przez program Z2Forwarder. Można również dostrzec prośby o wysłanie kolejnych pakietów, wysyłane co 5 sekund.

Z2Sender

```
S: 46 read
S: -> 0: A
S: -> 1: l
S: -> 2: a
S: -> 3:
S: -> 4: m
S: -> 5: a
S: -> 6:
S: -> 7: k
S: -> 8: o
S: -> 9: t
S: -> 0: A
S: -> 1: l
S: -> 2: a
S: -> 3:
S: -> 4: m
S: -> 5: a
S: -> 6:
S: -> 7: k
S: -> 8: o
S: -> 9: t
S: -> 0: A
S: -> 1: l
S: -> 2: a
S: -> 3:
S: -> 4: m
S: -> 5: a
S: -> 6:
S: -> 7: k
S: -> 8: o
S: -> 9: t
S: -> 0: A
S: -> 1: l
S: -> 2: a
```

```
S: -> 3:
S: -> 4: m
S: -> 5: a
S: -> 6:
S: -> 7: k
S: -> 8: o
S: -> 9: t
S: <- 6
S: -> 6:
S: -> 7: k
S: -> 8: o
S: -> 9: t
S: -> 10: a
S: -> 11: .
S: -> 12:

S: -> 13: O
S: -> 14: l
S: -> 15: a
S: <- 7
S: -> 7: k
S: -> 8: o
S: -> 9: t
S: -> 10: a
S: -> 11: .
S: -> 12:

S: -> 13: O
S: -> 14: l
S: -> 15: a
S: -> 16:
S: <- 10
S: -> 10: a
S: -> 11: .
```

```
S: -> 12:

S: -> 13: O
S: -> 14: l
S: -> 15: a
S: -> 16:
S: -> 17: n
S: -> 18: i
S: -> 19: e
S: -> 10: a
S: -> 11: .
S: -> 12:

S: -> 13: O
S: -> 14: l
S: -> 15: a
S: -> 16:
S: -> 17: n
S: -> 18: i
S: -> 19: e
S: <- 13
S(I): <- 13
S: -> 13: O
S: -> 14: l
S: -> 15: a
S: -> 16:
S: -> 17: n
S: -> 18: i
S: -> 19: e
S: -> 20:
S: -> 21: m
S: -> 22: a
S: <- 18
S: -> 18: i
```

```
S: -> 19: e
S: -> 20:
S: -> 21: m
S: -> 22: a
S: -> 23:
S: -> 24: k
S: -> 25: o
S: -> 26: t
S: -> 27: a
S: -> 18: i
S: -> 19: e
S: -> 20:
S: -> 21: m
S: -> 22: a
S: -> 23:
S: -> 24: k
S: -> 25: o
S: -> 26: t
S: -> 27: a
S: <- 21
S: -> 21: m
S: -> 22: a
S: -> 23:
S: -> 24: k
S: -> 25: o
S: -> 26: t
S: -> 27: a
S: -> 28: .
S: -> 29:

S: -> 30: E
S: -> 21: m
S: -> 22: a
S: -> 23:
```

```
S: -> 24: k
S: -> 25: o
S: -> 26: t
S: -> 27: a
S: -> 28: .
S: -> 29:
S: -> 30: E
S: <- 28
S: <- 29
S: -> 29:
S: -> 30: E
S: -> 31: l
S: -> 32: a
S: -> 33:
S: -> 34: m
S: -> 35: a
S: -> 36:
S: -> 37: d
S: -> 38: w
S: -> 29:
S: -> 30: E
S: -> 31: l
S: -> 32: a
S: -> 33:
```

```
S: -> 34: m
S: -> 35: a
S: -> 36:
S: -> 37: d
S: -> 38: w
S(I): <- 29
S: <- 31
S: -> 31: l
S: -> 32: a
S: -> 33:
S: -> 34: m
S: -> 35: a
S: -> 36:
S: -> 37: d
S: -> 38: w
S: -> 39: a
S: -> 40:
S(I): <- 31
S: -> 31: l
S: -> 32: a
S: -> 33:
S: -> 34: m
S: -> 35: a
S: -> 36:
S: -> 37: d
S: -> 38: w
S: -> 39: a
```

```
S: -> 40:
S: -> 31: l
S: -> 32: a
S: -> 33:
S: -> 34: m
S: -> 35: a
S: -> 36:
S: -> 37: d
S: -> 38: w
S: -> 39: a
S: -> 40:
S: -> 31: l
S: -> 32: a
S: -> 33:
S: -> 34: m
S: -> 35: a
S: -> 36:
S: -> 37: d
S: -> 38: w
S: -> 39: a
S: -> 40:
S: <- 39
S: <- 41
S: -> 41: k
S: -> 42: o
S: -> 43: t
S: -> 44: y
```

```
S: -> 45: .
S: -> 41: k
S: -> 42: o
S: -> 43: t
S: -> 44: y
S: -> 45: .
S: -> 41: k
S: -> 42: o
S: -> 43: t
S: -> 44: y
S: -> 45: .
S(I): <- 41
S(I): <- 41
S: -> 41: k
S: -> 42: o
S: -> 43: t
S: -> 44: y
S: -> 45: .
S: <- 43
S(I): <- 43
S: <- 44
S: -> 44: y
S: -> 45: .
S: -> 44: y
S: -> 45: .
S: <- 46
```

Jak można zauważyć program Z2Sender musiał często wysyłać pakiety wielokrotnie. Dla 46 datagramów utworzonych na postawie pliku źródłowego Z2Sender wysłał pakiety 214 razy, co oznacza, że każdy z pakietów został wysłany średnio **4,65** razy.