Client-Server – UDP - Kodierung

UDP

- ✓ Ein Prozess lauscht auf Datenpakete
- ✓ Ein anderer Prozess schickt dem ersten ein Datenpaket
- ✓ Der empfangende Prozess erkennt am Datenpaket den Absender und kann somit Datenpakete an diesen adressieren
- ✓ Wird Fehlerkorrektur gewünscht, z.B. um verlorene Pakete zu erhalten, so müssen das die Prozesse selbst leisten.
- ✓ Im Gegensatz zu TCP, dass auf der Basis von IP eine verbindungssicher und fehlerfreie Verbindung realisiert, ist UDP ein verbindungsloses Protokoll, bei dem die Anwendung dafür sorgen muss, dass die Pakete überhaupt und in der richtigen Reihenfolge beim Empfänger ankommen.
- ✓ Der Vorteil von UDP ist die höhere Geschwindigkeit

UDP II

- ✓ Im Gegensatz zu TCP erzeugt UDP wesentlich weniger Netzbelastung
  - keine Verbindung, sondern einzelne Pakete
  - ungesichert
- ✓ UDP wird eingesetzt bei Kommunikation, die leicht von Client und Server selbst überwacht werden kann
  - falls Netzlast klein zu halten ist
  - wenn keine Gewähr für das Ankommen der Pakete übernommen werden muss
  - bei "Broadcast", d.h. einer sendet an viele

Sender-Kodierung I

1.
```
InetAddress addr = InetAddress.getByName(host);
DatagramPacket packet = new DatagramPacket(new byte[1], 1, addr, 1024);
```

Bestimmt den lokalen Hostnamen, erstellt einen neuen Socket mit Port 1024 und erstellt ein leeres Paket

2.
```
String text = "Text 2 Send";
byte[] data = text.getBytes();
packet.setData(data);
packet.setLength(text.length());
socket.send(packet);
```

- ✓ Liest die Bytes aus dem Text
- ✓ Speichert die Bytes im Datagram Packet Obj.
- ✓ Setzt die Länge
- ✓ Sendet den Text

3.
```
socket.close()
```

Schließt den DatagramPacket Socket

Receiver-Kodierung I

1. DatagramSocket socket = new DatagramSocket(1024);

   Erstellt einen neuen Socket mit Port 1024

2. byte[] buf = new byte[100];
   DatagramPacket packet = new DatagramPacket(buf, buf.length);

   Erstellt ein neues Packet für den Datenempfang für max. 100 Bytes in buf

3. socket.receive(packet);

   Wartet auf Daten zum Empfang

4. String text = new String(packet.getData(), 0, packet.getLength());

   Schreibt den empfangenen Text in die Text-Variable text

Receiver-Kodierung II

4. packet.setLength(buf.length);

Setzt wieder die Länge von buf auf 100, so dass max. 100 Zeichen empfangen werden können

Einführungsbeispiel Receiver-Sender: Receiver

```
try {
   DatagramSocket socket = new DatagramSocket(1024);

   InetAddress addr = InetAddress.getLocalHost();
   System.out.println(……..);
   byte[] buf = new byte[100];
   DatagramPacket packet = new DatagramPacket(buf, buf.length);
   while (true) {
     socket.receive(packet);
     String text = new String(packet.getData(), 0, packet.getLength());
     InetAddress a = packet.getAddress();
     System.out.println(text+……..);
     packet.setLength(buf.length);
   }
 }
```

Neues DatagramSocket Objekt

Ermittelt den Hostnamen und gibt diesen, IP und Port aus

Puffer von 100 Zeichen vorbereiten und neues DatagramPacket Objekt erstellen

Unendlich oft auf Nachrichten warten

Auf Nachrichten warten

Text ausgeben

Pufferlänge wieder auf Vorgabewert von 100 setzen

->Aufgabe 1

Einführungsbeispiel Receiver-Sender: Sender

```
try {
  DatagramSocket socket = new DatagramSocket();
  System.out.println("Socket benutzt Port " + socket.getLocalPort());
  InetAddress addr = InetAddress.getByName("localhost");
  // ein leeres Paket
  DatagramPacket packet = new DatagramPacket(new byte[1], 1, addr, 1024);
  String text = "Text 2 Send";
  byte[] data = text.getBytes();
  packet.setData(data);
  packet.setLength(text.length());
  socket.send(packet);
  socket.close();
 }
```

Neues DatagramSocket Objekt

Neues DatagramPacket Objekt

Text in Bytes konvertieren und dem Packet zuweisen

Länge setzen und Packet senden

Socket schließen

->Aufgabe 1