4. Thread- und Netzwerk-Programmierung

4.1 Ziel dieses Kapitels

4.2 Prozeß versus Thread

4.3 Thread-Programmierung

4.4 TCP/IP Grundlagen

4.5 TCP Programmierung

4.1 Ziel dieses Kapitels

Die Entwicklung von **Internet-Applikationen** ist insbesondere in der Praxis von großer Bedeutung. Aus diesem Grund wurde für das Praktikum Programmiermethodik eine Problemstellung aus diesem Bereich gewählt.

Die **Parallelprogrammierun**g (Threads) ist für die Entwicklung größerer Java-Programme essentiell notwendig.

In diesem Kapitel können Sie

- Basiskenntnisse
  - der parallelen Programmierung und
  - des Internet-Protokolls
- sowie deren praktische Umsetzung in Java

erlernen.

Der Schwerpunkt liegt auf der praktischen Umsetzung!

4.2 Prozesse versus Threads

Ein ablaufendes Programm wird auch als Prozeß bezeichnet.

Definition Prozeß.

Ein Prozeß besteht aus:

- **Systemressourcen** (Dateisystem, etc.),
- einem **Adreßraum** (Speicherbereich für Objekte, die mit **`new`** angelegt werden sowie statische Attribute und Objekte) und
- mindestens einem **Thread** (Anweisungssequenz)

Bisher haben wir Programme entwickelt, die Anweisungen sequentiell ausgeführt haben (also nur einen Thread hatten). In einigen Fällen kann es jedoch notwendig/nützlich sein, Anweisungssequenzen parallel auszuführen.

Java unterstützt aus diesem Grund das Konzept des **Multi-Threadings**.

Threads

Definition Threads.

Threads bestehen aus:

- einer **Anweisungssequenz** (Programmzähler) und
- einem **Stackbereich** (für automatische Variablen und Rücksprungadressen)

Beispiel.

```
01 class MyClass {
02   static int j= 5;
03   static void main(String[] argv) {
04      Integer intObj= new Integer(j);
05      for (int i= 0; i < j; i++)
06        System.out.println("i = " + i);
07   }
08 }
```

4.3 Java Thread Programmierung (Teil 1)

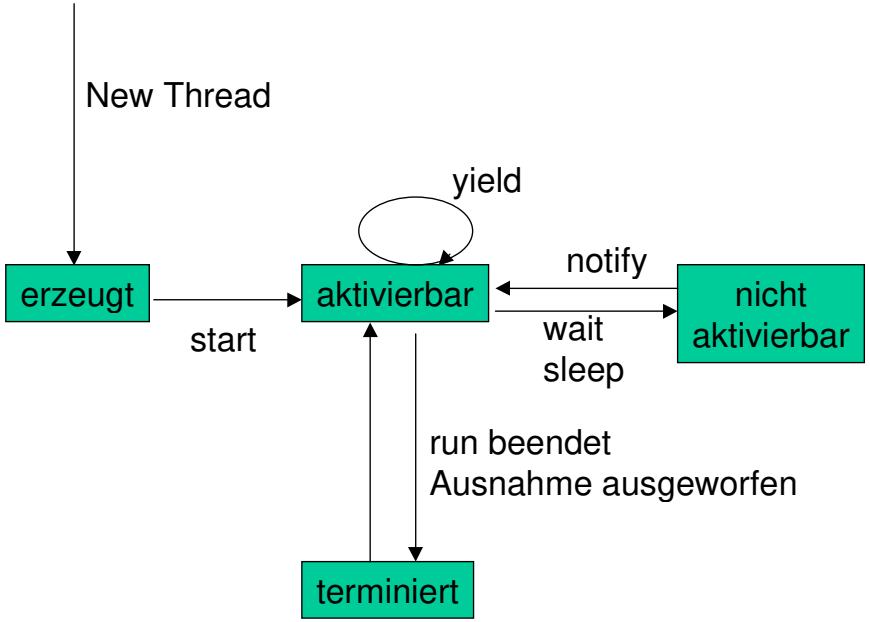
In Java werden Threads durch Objekte der Klasse Thread repräsentiert.

```
class Thread {
  Thread(Runnable target, String name);

  static Thread currentThread();
  static void   sleep(long millis);
  static void   yield();

  void          start();
  void          stop(); /* deprecated */
  void          run();
  String        getName() ;
  boolean       isAlive();
}
```


Thread Zustandsdiagramm



Interface Runnable

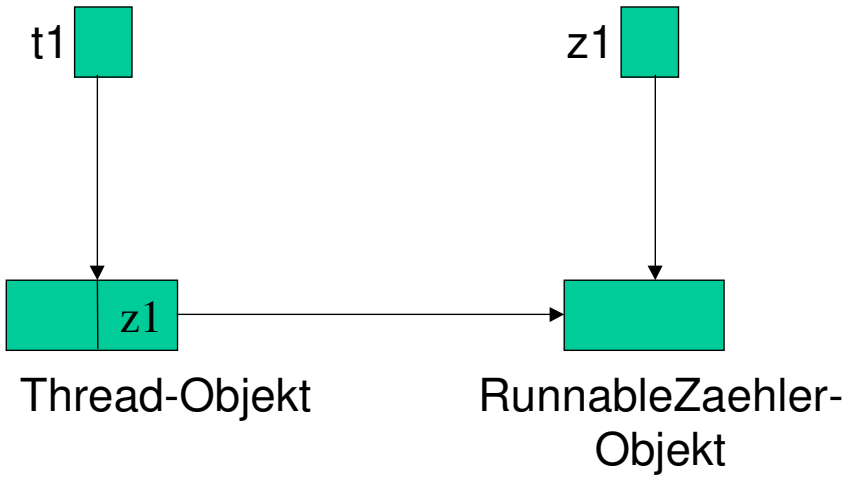
Threads können in eigenen Programmen durch Implementieren des Interfaces Runnable erzeugt werden:

```
interface Runnable {
   public void run();
}
```


Beispiel: RunnableZaehler

```
01 class RunnableZaehler implements Runnable {
02  public void run() {
03    for (int i= 0; i < 4; i++) {
04      System.out.println(
05        Thread.currentThread().getName() +
06        ": " + i);
07      try {
08        Thread.Sleep(1000);
09      } catch (InterruptedException e) {
10        System.err.println(e);
13  } } }
14  public static void main (String[] argv) {
15    Runnable z1= new RunnableZaehler();
16    Runnable z2= new RunnableZaehler();
17    Thread t1= new Thread(z1);
18    Thread t2= new Thread(z2);
19    t1.start();
20    t2.start();
22 } }
```

Objektmodell



4.4 Netzwerke, Client/Server-Programmierung (Teil 1)

Internet-Protokoll (IP).

Ziel des Internet-Protokolls ist die Bereitstellung der folgenden beiden Dienste:

- eindeutige **Identifikation** eines Rechners,
- **Transport** von Daten-Paketen von einem Rechner zu einem anderen Rechner.

Jedes Datenpaket enthält einen Programmkopf mit der Adresse des Absenders und des Empfängers. Jedes Paket wird unabhängig von vorhergehenden Paketen vom Sender zum Empfänger geleitet.

Größere Datenmengen müssen in Pakete zerlegt werden.

Internet Protokoll (IP)

IP sieht vor, daß jeder Rechner eine eindeutige Adresse zur Identifikation besitzt. Diese Adresse besteht aus einer 32-Bit Zahl.

Beispiel: Der Web Server des Lehrstuhls hat die IP-Adresse

2258317408

Typischerweise wird die IP-Adresse durch vier 8-Bit Werte dargestellt, die durch Punkte getrennt sind.

Beispiel: Lehrstuhl-Web Server

134.155.48.96

Da solche Zahlen nur schwer zu merken sind, können Rechner auch durch einen Namen beschrieben werden.

Beispiel: Lehrstuhl-Web Server

www.informatik.uni-mannheim.de

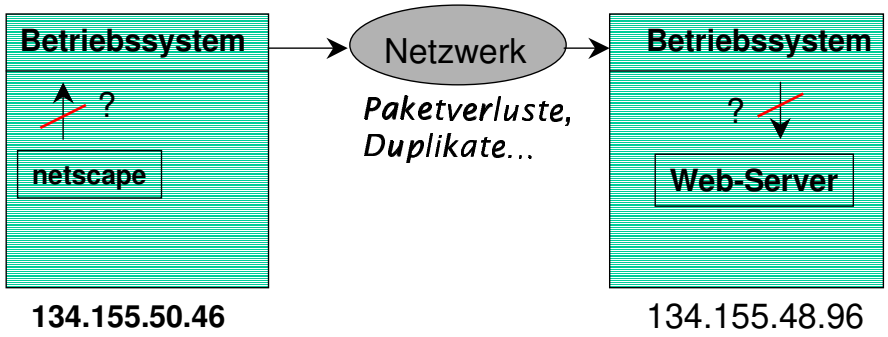
Die Übersetzung des Namens in die 32-Bit Adresse wird durch sogenannte Domain-Name-Services (DNS) übernommen.

Internet Protokoll (IP)

Was leistet IP **nicht**?

- IP realisiert keinerlei Zuverlässigkeit; d.h. Pakete können
  - verloren gehen,
  - in falscher Reihenfolge ankommen oder
  - dupliziert werden.

- IP realisiert keine Ende-zu-Ende Kommunikation:
  - es ist nicht möglich Daten zwischen zwei **Programmen** auszutauschen.
  - Es können keine Programme auf entfernten Rechnern adressiert werden.



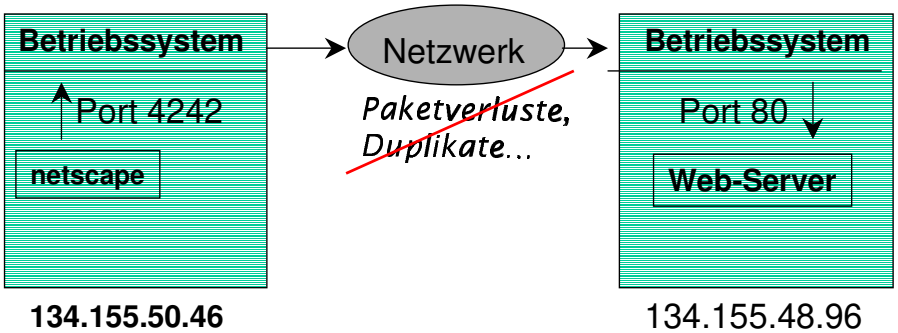
Transmission Control Protokoll (TCP)

TCP erweitert IP um die fehlende Funktionalität; d.h. es ermöglicht eine zuverlässige Ende-zu-Ende Kommunikation:

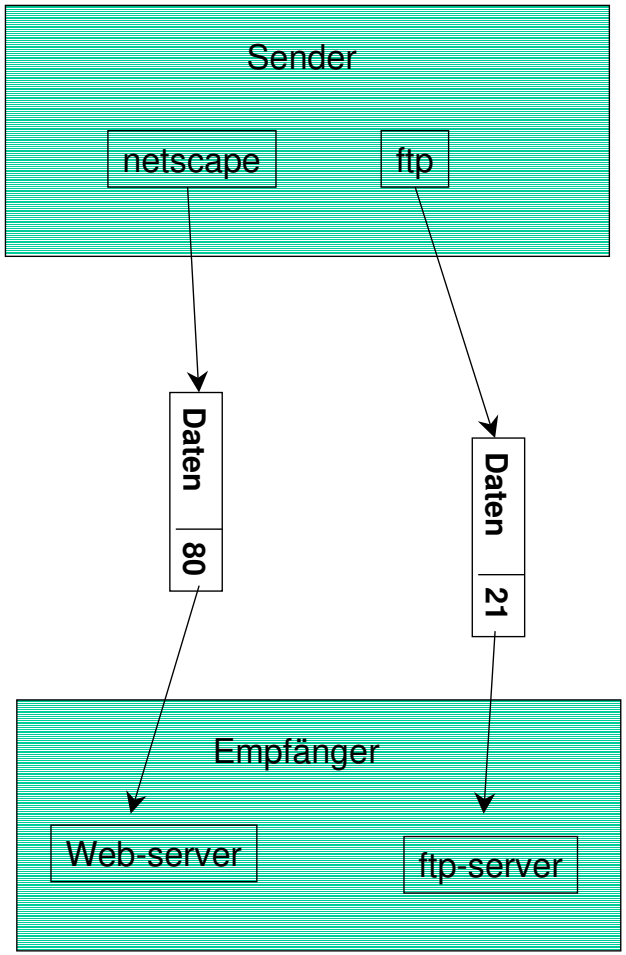
- Absicherung gegen Paketverluste,
- Korrektur der Paketreihenfolge und
- Entfernung der Duplikate.

Weiterhin:

- Adressierung von Programmen auf entfernten Rechnern durch Port-Adressen
- Port-Nummern sind im Paketkopf von TCP-Paketen enthalten.



Ende-zu-Ende Kommunikation



Netzwerke, Client/Server-Programmierung

Wir wollen uns hier im wesentlichen auf die Entwicklung von Client/Server-Architekturen per TCP beschränken.

Definition Server.

Mit **Server** wird ein Programm bezeichnet, das bestimmte Dienste (über das Internet) anbietet.

Definition Client.

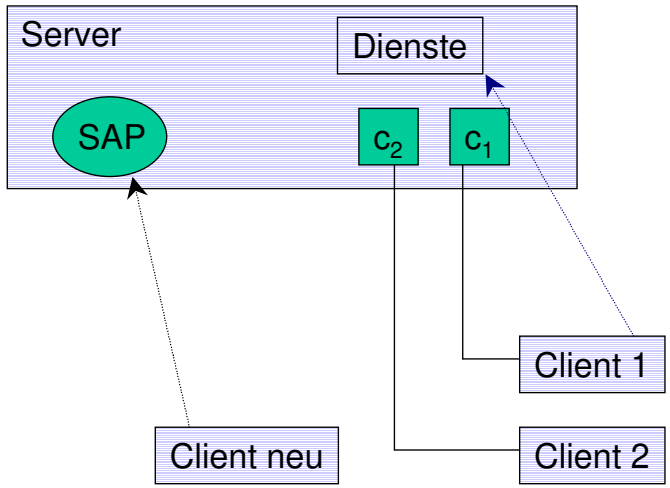
Mit **Client** wird ein Programm bezeichnet, das bestimmte Dienste bei einem Server (über das Internet) anfordert.

Beispiele für Client/Server-Architekturen:

- WWW,
- FTP,
- Telnet

Client/Server Architektur

Clients können eine **Verbindung** zum Server beantragen, indem sie dessen **Dienstzugangspunkt** kontaktieren. Akzeptiert der Server die Verbindungsanfrage, wird eine Verbindung hergestellt. Über diese Verbindung können **Dienste** angefordert werden. Dies geschieht, indem zwischen Client und Server Daten ausgetauscht werden.



SAP = Dienstzugangspunkt (Service Access Point)
$c_{1/2}$ = Verbindung zu Client 1 bzw. Client 2

Dienstzugangspunkte

Dienstzugangspunkte sind „öffentlich bekannte Eingangstore“ (**Ports**) eines Rechners. Das Internet-Protokoll erlaubt die eindeutige Identifikation eines Rechners. Auf einem Rechner (Server) laufen oftmals mehrere Server-Programme gleichzeitig, die unterschiedliche Dienste zur Verfügung stellen (z.B. FTP-Server und HTTP-Server). Um ein solches Server-Programm eindeutig ansprechen zu können, stellt das Betriebssystem sogenannte Ports zur Verfügung.

Dienstzugangspunkte

Definition Port.

Ports werden durch eine 16-Bit Zahl repräsentiert. Ein Server-Programm kann sich an einen Port binden und ist damit eindeutig ansprechbar.

Prinzipiell kann ein Server-Programm einen beliebigen freien Port binden, bestimmte standardisierte Dienste wie z.B. HTTP haben fest zugeordnete Ports. Selbst entwickelte Programme sollten deshalb Port-Nummern größer 10000 wählen.

Standardisierte Dienste (Auszug):

Anwendung	*Port*	*Anwendung*	*Port*
Echo	7	Daytime	13
FTP	21	Telnet	23
SMTP	25	Finger	79
HTTP	80	POP	110

4.5 Client/Server Programmierung in Java

Das Paket `java.net` enthält Klassen für den Zugriff auf andere Rechner per Internet.

Server.

Server-Programme **erzeugen** zunächst einen Dienstzugangspunkt. Dieser wird durch Objekte der Klasse `ServerSocket` repräsentiert. Der Dienstzugangspunkt wird an einen Port des Rechners **gebunden**. Im Anschluß **„lauscht“** der Server auf eingehende Verbindungswünsche von Clients. Geht am Dienstzugangspunkt ein Verbindungswunsch ein, kann der Server diesen Wunsch **akzeptieren**. Damit ist eine **Verbindung** hergestellt. Die Verbindung wird durch Objekte der Klasse `Socket` repräsentiert.

Client/Server-Programmierung

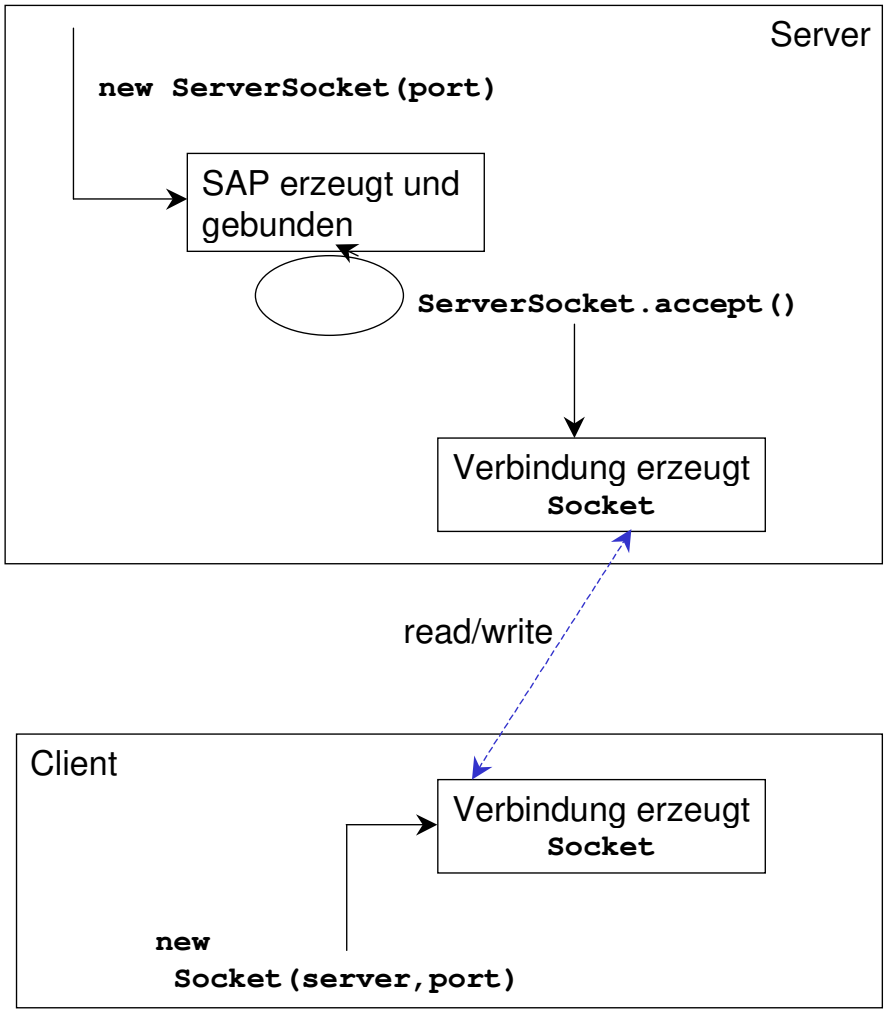
Client.

Client-Programme **erzeugen** eine Verbindung zum Dienstzugangspunkt des Servers. Diese Verbindung wird durch Objekte der Klasse `Socket` repräsentiert.

Sowohl Client als auch Server können Daten an den Gegenpart verschicken (full duplex).

Anmerkung. Der Begriff Socket stammt aus dem Bereich der Betriebssysteme. Hier stellen sie ebenfalls die (C-)Programmierschnittstelle zur TCP/IP Implementierung des jeweiligen Betriebssystems dar.

TCP Zustandsdiagramm



Die Klassen Socket und ServerSocket

```
class ServerSocket {
    ServerSocket(int port);
    Socket accept();
    void close();
}

class Socket {
    Socket(String host, int port);
    void close();
    InputStream getInputStream();
    OutputStream getOutputStream();
}
```

(Auszug)

TCP Client

```
01 class TCPIPClient {
02   TCPIPClient(String hostn, int port) {
03     try {
04       Socket sock= new Socket(hostn, port);
05       InputStream sockIn=
06          sock.getInputStream();
07       OutputStream sockOut=
08          sock.getOutputStream();
09       int inValue= sockIn.read();
10       int outVal= 42;
11       sockOut.write(outVal);
12       sock.close();
13     } catch (UnknownHostException eu) {
14       . . .
15     } catch (IOException ioe) {
16       . . .
17   } }
18   . . .
19 }
```


TCP Server

```
01 class DumpServer {
02   DumpServer(int port) {
03     try {
04       ServerSocket server=
05         new ServerSocket(port);
06       for (;;) {
07         Socket sock= server.accept();
08         OutputStream sockOut=
09           sock.getOutputStream();
10         int outVal= 42;
11         sockOut.write(outVal);
12         sock.close();
13       }
14     } catch (IOException ioe) {
15       . . .
16   } }
17   . . .
18 }
```

Komplexere Beispiele

Unser Server hat den Nachteil, daß er jeweils immer nur einen Client bedienen kann.

Dies liegt daran, daß nur ein Thread vorhanden ist.

Typischerweise erzeugen sowohl Client als auch Server zu jedem Socket eigene Threads, die Daten auf diesen Socket schreiben bzw. Daten von diesem Socket lesen.

Auf der Web-Seite des Lehrstuhls ist der Quellcode einiger Client/ Server-Applikationen abgelegt.