Verteiltheit & Kommunikation

1 Verteilte Programme

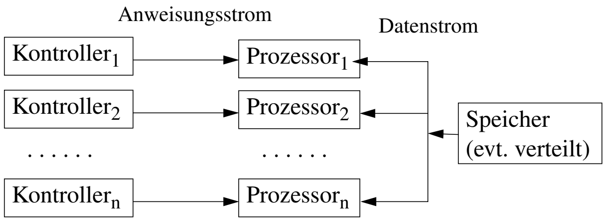
Ein Programm ist dann <u>verteilt</u> (*distributed*) wenn seine Daten- oder auch Funktionskomponenten auf mehrere zu einem Netz zusammengeschlossene Rechner verteilt sind.

1.1 Verteiltheit der Hardware

✎ Eine <u>Mehrprozessorsystem</u> verfügt über mehrere Prozessoren, von denen jeder einzelne als einziger Prozessor in einem Einprozessorsystem arbeiten kann.

✎ Ein <u>Mehrrechnersystem</u> (oder <u>Rechnernetz</u>) besteht aus mehreren Prozessoren, die lokale Speicher besitzen und miteinander über Kommunikationsleitungen zum Datenaustausch verbunden sind.

➪ MIMD — Multiple Instruction Multiple Data

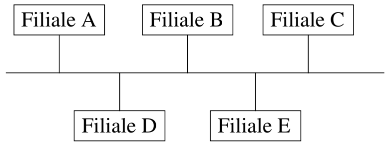


1.2 Verteiltheit der Software

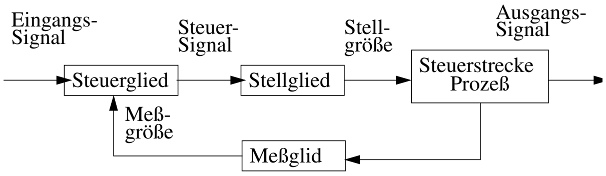
✎ Vorteile

- Dezentralisierung

➪ Datendezentralisierung



➪ Funktionsdezentralisierung



- Gemeinsame Nutzung von Betriebsmitteln

➪ Server (Druckdienst, Dateidienst)

- Erhöhen der Zuverlässigkeit

Ein <u>redundantes System</u> enthält mehrere Elemente, die einander gegenseitig ersetzen können. Falls ein Element ausfällt, kann ein anderes seine Aufgabe übernehmen.

✎ Probleme verteilter Systeme

- Unschärfen: Unvollständige oder überholte Zustandsinformation
  - nicht vernachlässigbare und nicht vorhersehbare Zeit für Austausch von Nachrichten
  - grobe Synchronisationszeitrahmen
  - unsichere Übertragungen
- Ungewißheit: keine globale Situationserfassung

2 Konfiguration

2.1 Physikalische Topographie

Die physikalische Totographie eines verteilten Systems beschreibt, welche Rechner dem verteilten System zugrunde liegen und wie sie miteinander verbunden sind.

➪ Vollständige Vernetzung: Jeder Rechner ist mit jedem anderen direkt verbunden.

➪ Bus: Alle Rechner an die gleiche Leitung angeschlossen werden.

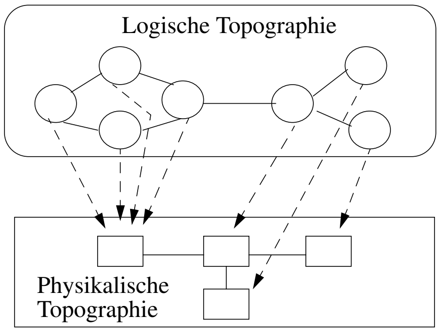
➪ Ring: Die Anordnung von Rechnern zu einem Ring bedingt, daß Nachrichten zwischen Rechnern weitergeleitet werden.

➪ ......

2.2 Logische Topographie

Prozesse eines verteiltes Systems werden zu <u>Verteilungseinheiten</u> geordnet. Innerhalb der Verteilungseinheit können Prozesse über gemeinsamen Speicher kommunizieren; über die Grenzen der Verteilungseinheit hinweg ist dies nur mit Nachrichten möglich.

➪ Abbilden der Topographien



✎ Benennung

- Ein <u>absoluter Name</u> ist kontextunabhängig und wird im ganzen System gleich interpretiert.
- Ein <u>relativer Name</u> ist immer auf einen bestimmten Kontext bezogen.

➩ im UNIX-System bei Bezeichnung von Dateien

➩ triton

➩ triton.informatik.uni-bremen.de

✎ Protokolle

• Ein <u>Protokoll</u> definiert Regeln, die ein Verständigen von Prozessen möglich machen.

• Aufgaben von Protokollen

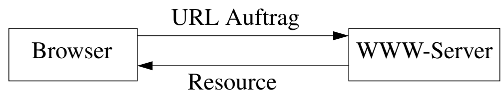
- Fehlererkennung (*error dedection*)
- Reihenfolgeerhaltung
- Flußkontrolle
- Nachrichtenbestätigung

➩ Protokolle für die Kommunikation über das Internet
IP - *Internet Protocol*
UDP - *Unreliable Datagram Protocol*
TCP - *Transmission Control Protocol*

2.3 Das Internet

✎ Beim <u>Client-Server-Modell</u> besteht eine Anwendung aus zwei Teilen:
- Client: Der Client fragt Dienste nach
- Server: Der Server bietet Dienste an

➩ WWW - *World Wide Web*



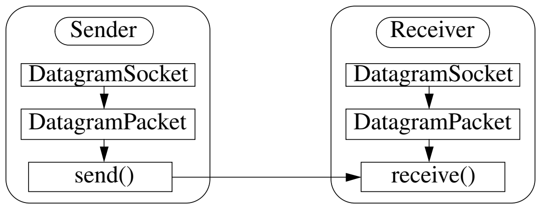
Ein <u>URL</u> - *Uniform Resource Locater* besteht aus der Bezeichnung für das Protokoll sowie der eigentlichen Adresse im Internet.

✎ Programmierung von Sockets

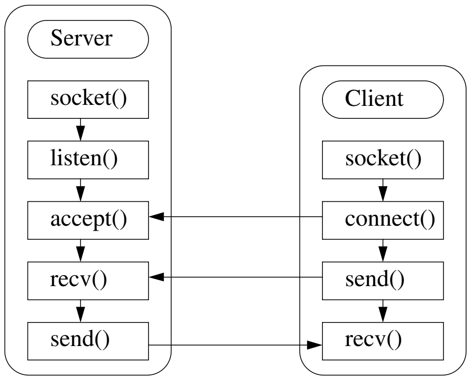
Ein <u>Socket</u> ist ein Endpunkt einer Kommunikation im Netzwerk. Um eine Verbindung zu ermöglichen, braucht man zwei solcher Enden.

• Verbindungsfreie Sockets basieren auf Datagrammen -- einzelnen Paketen von Nachrichten

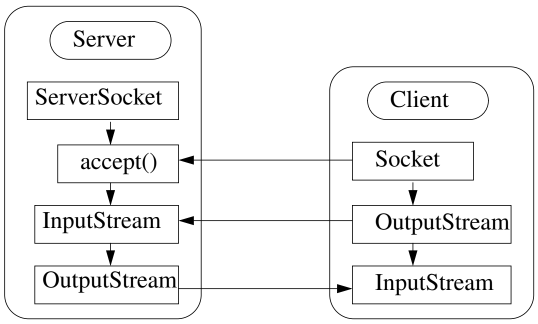
➩ Verbindungsfreie Sockets in Java



• Verbindungsorientierte Sockets ermöglichen sequentielle Bearbeitung in Streams



➩ Verbindungsorientierte Sockets in Java



3 Kommunikation

3.1 Nachrichten

✎ Definitionen

• Dateneinheiten, die Prozesse an andere Prozesse übergeben, werden <u>Nachrichten</u> (*messags*) genannt.

• Die auf dieser Übergabe beruhende Kommunikation wird als <u>Nachrichtenaustausch</u> (*message passing*) bezeichnet.

• An die Stelle des Schreibens und Lesens von Datenwerten tritt das <u>Senden</u> (*send*) und <u>Empfangen</u> (*receive*) von Nachrichten.

• An die Stelle der gemeinsamen Variablen, in der Datenwerte abgelegt werden, tritt ein <u>Kommunikationskanal</u> (*communication channel*), der Nachrichten aufnimmt und festlegt, wohin eine Nachricht gehen oder woher sie kommen soll. Es gibt zwei Aspekte zu behandeln:

- das Einrichten von Kommunikationskanälen und
- die Benutzung von Kommunikationskanälen

✎ Senden von Nachrichten

• Eine Sendeoperation, die einen Prozeß nur für die Dauer des Kopierens der Nachricht beansprucht, nennt man <u>nicht-blockierend</u> (*non-blocking*).

• Ein Sendeoperation, die einen Prozeß so lange blockieren wird, bis eine entsprechende Empfangsanweisung ausgeführt wird, nennt man <u>blockierend</u> (*blocking*).

• Rundsendung (*broadcast*): wenn eine Nachricht an alle denkbaren Empfänger geliefert wird.

• Mehrfachsendung (*multicast*): wenn zwar viele, aber nicht alle möglichen Empfänger die Nachricht erhalten sollen.

➩ send wert to kanal

➩ broadcast wert

➩ multicast wert to $\{kanal_1, kanal_2, kanal_3\}$

✎ Empfangen von Nachrichten

• Eine Nachricht wird von einer Empfangsoperation in eine Variable kopiert. Ein Empfangen wird konsumierend oder zerstörend (*destructive*), wenn die Nachricht nach dem Kopieren zerstört wird.

• Ein Empfangen kann auch konservierend (*non-destructive)* sein, wenn die Nachricht nach dem Empfang im Kommunikationskanal bleibt und erst durch eine spezielle Operation gelöscht wird.

• Üblichweise arbeitet eine Empfangsoperation blockierend. Sie versetzt den sie ausführenden Prozeß so lange in einen Wartezustand, bis für ihn eine Nachricht eingetroffen ist.

• Ein Prozeß erhält durch Ausführen der nichtblockierend Empfangsoperation entweder die Antwort, daß keine Nachricht vorliegt, und fährt dann in seiner Ausführung fort, oder er erhält die Nachricht selbst.

• Ein selektives Empfangen führt die Empfangsoperation aus, für die eine Nachricht vorliegt, etwa in der Art


```
select
    receive variable1 from kanal1 → ...
[]  receive variable2 from kanal2 → ...
[]  receive variable3 from kanal3 → ...
end select
```

➩ in Java

```
InputStream is;
OutputStream os;
is.write(...);          // nichtblockierend
os.read(...);           // blockierend
```

➩ ein MulticastSocket in Java ist ein DatagramSocket

```
// join a Multicast group and send the group salutations
byte[] msg = {'H', 'e', 'l', 'l', 'o'};
InetAddress group = InetAddress.getByName("228.5.6.7");
MulticastSocket s = new MulticastSocket(6789);
s.joinGroup(group);
DatagramPacket hi = new DatagramPacket(msg, msg.length,
    group, 6789);
s.send(hi);

// get their responses
byte[] buf = new byte[1000];
DatagramPacket recv= new DatagramPacket(buf,buf.length);
s.receive(recv);

...

// leave the group
s.leaveGroup(group);
```