Script **generated by TTT**

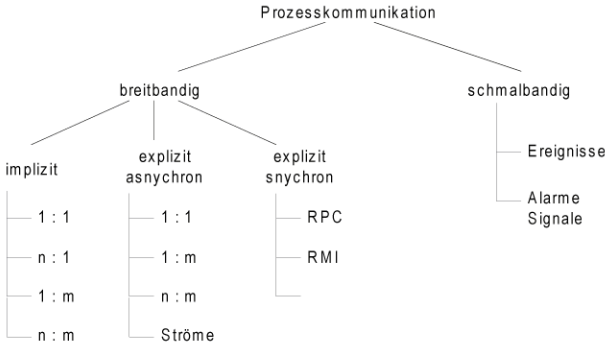
Title: Grundlagen_Betriebssysteme (14.12.2015)

Date: Mon Dec 14 13:45:20 CET 2015

Duration: 93:30 min

Pages: 35

Kommunikationsarten



Die Bandbreite des Kommunikationskanals bestimmt die Datenrate, in der Daten zwischen Prozessen ausgetauscht werden können.

Schmalbandige Kanäle

Implizite Kommunikation

Explizite Kommunikation

Generated by Targeteam

Implizite Kommunikation

Implizite Kommunikation ist eine breitbandige Kommunikationsform. Die Kommunikation erfolgt über einen gemeinsamen Speicher (Dateien, Register, Datenstrukturen).

Die Kommunikation findet ohne direkte Unterstützung und ohne Kenntnis des BS statt.

Vorteil: einfach und schnell (kein Kopieren zwischen Adressräumen).

Nachteil:

a) gemeinsame Bereiche sind nicht immer vorhanden: z.B. in physisch verteilten, vernetzten Systemen gibt es i.d.R. keinen gemeinsamen Speicher.

b) gegebenenfalls aufwendiges busy waiting ⇒ Mischform: Ereigniszustellung, d.h. schmalbandige Kommunikation, die das Vorhandensein von Daten signalisiert.

Implizite Kommunikationsformen

Generated by Targeteam

Implizite Kommunikationsformen

Verschiedene Formen der impliziten Kommunikation

1:1 ein Puffer pro Sender/Empfänger-Paar

n:1 n Sender senden Nachrichten an einen Empfänger, z.B.

Sender: Prozesse senden Druckaufträge

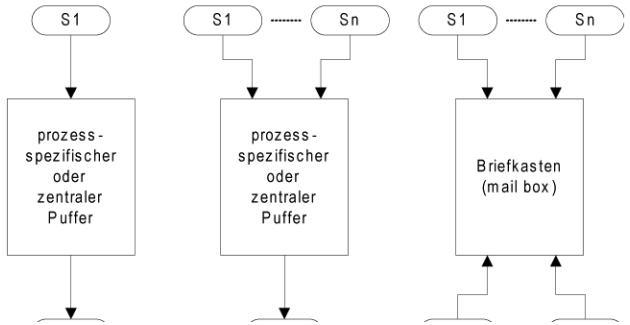
Empfänger: Drucker-Server

1:m Mitteilung an alle Prozesse (Broadcast, Multicast);

Broadcast : z.B. Erfragen, wo ein spezieller Dienst angeboten wird; Shutdown Message an alle Rechner.

Multicast : z.B. Nachricht an Gruppe gleichartiger Server.

n:m n Erzeuger schreiben in Puffer und m Verbraucher lesen aus Puffer.



Implizite Kommunikationsformen

Sender: Prozesse senden Druckaufträge

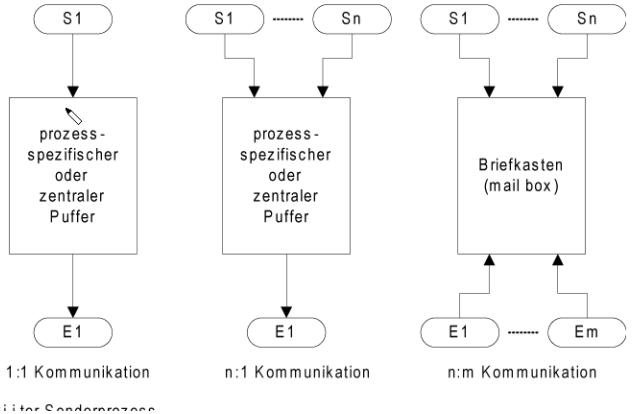
Empfänger: Drucker-Server

1:m Mitteilung an alle Prozesse (Broadcast, Multicast);

Broadcast : z.B. Erfragen, wo ein spezieller Dienst angeboten wird; Shutdown Message an alle Rechner.

Multicast : z.B. Nachricht an Gruppe gleichartiger Server.

n:m n Erzeuger schreiben in Puffer und m Verbraucher lesen aus Puffer.



Sinter Sonderprozess

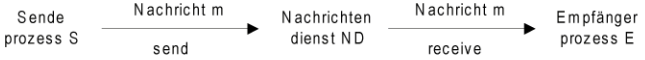
Explizite Kommunikation

Diese Kommunikationsart wird realisiert durch den Austausch von Nachrichten ("message passing") ⇒ ***nachrichtenbasierte Kommunikation*** .

Betriebssystem enthält einen ***Nachrichtendienst ND*** (das Kommunikationssystem), der den Austausch der Nachrichten zwischen Prozessen realisiert. ND unterstützt 2 Systemdienste:

```
send (E: process, m: message)
receive (S: process, m: message)
```

prinzipieller Ablauf



Aufbau einer Nachricht

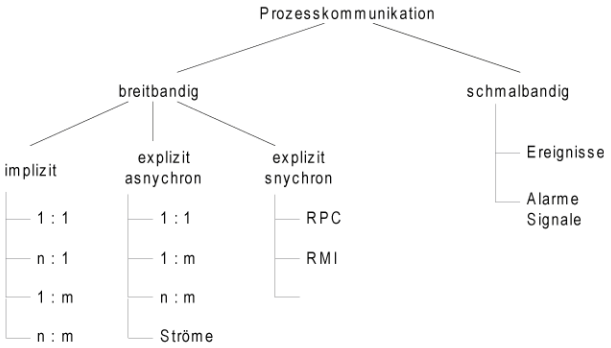
Eine Nachricht besteht aus 2 grundlegenden Komponenten:

Nachrichtenkopf: Verkehrsinformation, z.B. Sender- und Empfängeridentifikation

Nachrichteninhalt: Nutzlast (payload)

explizite Kommunikation ist besonders geeignet in verteilten, vernetzten Systemen.

Kommunikationsarten



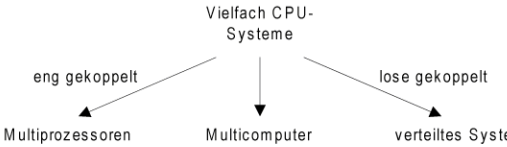
Die Bandbreite des Kommunikationskanals bestimmt die Datenrate, in der Daten zwischen Prozessen ausgetauscht werden können.

[Schmalbandige Kanäle]

[Implizite Kommunikation]

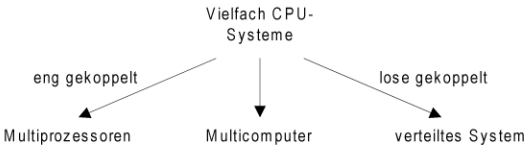
[Explizite Kommunikation]

Ansätze zur Kopplung von Recheneinheiten



	Multiprozessor	Multicomputer	verteiltes System
Konfiguration	nur CPU	CPU, Arbeitsspeicher, Netzschnittstelle	kompletter Rechner
Peripherie	gemeinsam	gemeinsam, außer Festplatte	getrennt
Ort	gemeinsames Gehäuse	gemeinsamer Raum	weltweit verteilt
Kommunikation	gemeinsamer Arbeitsspeicher	direkte Verbindung	Netzwerk
Betriebssystem	ein gemeinsames BS	getrennt, jedoch gleicher Typ	getrennt, vielfach unterschiedlich
Verwaltung	eine Organisation	eine Organisation	viele Organisationen

Ansätze zur Kopplung von Recheneinheiten



	Multiprozessor	Multicomputer	verteiltes System
Konfiguration	nur CPU	CPU, Arbeitsspeicher, Netzschnittstelle	kompletter Rechner
Peripherie	gemeinsam	gemeinsam, außer Festplatte	getrennt
Ort	gemeinsames Gehäuse	gemeinsamer Raum	weltweit verteilt
Kommunikation	gemeinsamer Arbeitsspeicher	direkte Verbindung	Netzwerk
Betriebssystem	ein gemeinsames BS	getrennt, jedoch gleicher Typ	getrennt, vielfach unterschiedlich
Verwaltung	eine Organisation	eine Organisation	viele Organisationen

Prozesskommunikation

Disjunkte Prozesse, d.h. Prozesse, die völlig isoliert voneinander ablaufen, stellen eher die Ausnahme dar. Häufig finden Wechselwirkungen zwischen den Prozessen statt ⇒ Prozesse interagieren. Die Unterstützung der Prozessinteraktion stellt einen unverzichtbaren Dienst dar.

Fragestellungen

Dieser Abschnitt beschäftigt sich mit den Mechanismen von Rechensystemen zum Austausch von Informationen zwischen Prozessen.

Kommunikationsarten.

nachrichtenbasierte Kommunikation, insbesondere Client-Server-Modell.

Netzwerkprogrammierung auf der Basis von Ports und Sockets.

Nachrichtenbasierte Kommunikation

Bei nachrichtenbasierter Prozessinteraktion tauschen Prozesse gezielt Informationen durch Verschicken und Empfangen von Nachrichten aus; ein Kommunikationssystem unterstützt an der Schnittstelle wenigstens die Funktionen `send` und `receive` .

Klassifikationsschema

Elementare Kommunikationsmuster sind Meldung ("signal") und Auftrag ("request").

Meldung : Einweg Nachricht vom Sender zum Empfänger (unidirektional).

Auftrag : Zweiweg Nachricht zwischen Sender und Empfänger (bidirektional). Sie beginnt mit dem Versenden eines Auftrags an den Empfänger und endet mit der Übergabe einer Erfolgsbestätigung über den durchgeführten Auftrag an den Sender.

Synchronität definiert den Kopplungsgrad zwischen Prozessen bei der Durchführung einer Nachrichtentransaktion:

asynchron : Entkopplung des Senders und Empfängers.

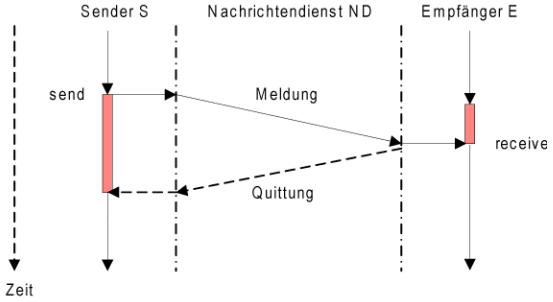
synchron : beide Prozesse werden zur Nachrichtenübertragung synchronisiert.

Klassifikationsschema für die Nachrichtenkommunikation anhand von 2 Dimensionen:

generelles Muster der Nachrichtenkommunikation.

zeitliche Kopplung der beteiligten Prozesse.

Sender und Empfänger von Meldungen sind zeitlich gekoppelt.



Rendezvous -Technik: Sender und Empfänger stellen vor Austausch der eigentlichen Meldung die Sende- und Empfangsbereitschaft her.

Klassifikationsschema für die Nachrichtenkommunikation anhand von 2 Dimensionen:

generelles Muster der Nachrichtenkommunikation.

zeitliche Kopplung der beteiligten Prozesse.

Klassifikationsschema

Meldung

- Asynchrone Meldung
- Synchrone Meldung

Auftrag

- Synchroner Auftrag
- Asynchroner Auftrag

Vorteile/Nachteile asynchrones Senden

Vorteile asynchrones Senden

- nützlich für Realzeitanwendungen.
- ermöglicht parallele Abarbeitung durch Sender und Empfänger.
- anwendbar zum Signalisieren von Ereignissen.

Nachteile asynchrones Senden

- Verwaltung des Nachrichtenpuffers durch BS erforderlich.
- Benachrichtigung des Senders S im Fehlerfall und Behandlung von Fehlern ist problematisch.
- Entwurf und Nachweis der Korrektheit des Systems ist schwierig.

Elementare Kommunikationsmodelle

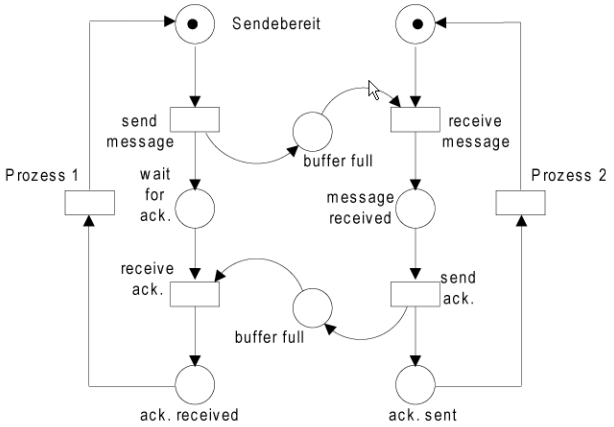
Klassifikationsschema für die Nachrichtenkommunikation anhand von 2 Dimensionen:

generelles Muster der Nachrichtenkommunikation.

zeitliche Kopplung der beteiligten Prozesse.

Modellierung durch ein Petrinetz

Petri-Netze dienen häufig zur Modellierung von Kommunikationsabläufen, sogenannten Kommunikationsprotokollen. Sie ermöglichen die Analyse der Protokolle, z.B. Erkennung von Verklemmungen. Modellierung einer synchronen Kommunikation:



Problem: unendliches Warten

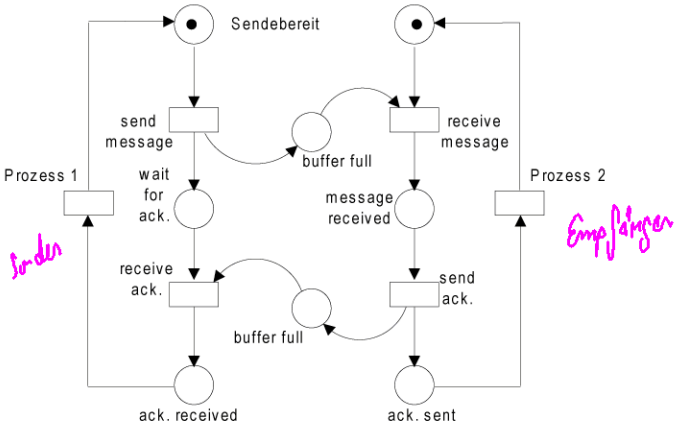
Pragmatische Lösung mit Hilfe von Timeouts

Sender bzw. Empfänger warten nur eine festgelegte Zeit, Sender: falls kein Acknowledgement (Quittung) eintrifft, z.B. erneutes Senden.

Probleme dabei? u.a. Duplikate müssen vom Empfänger erkannt werden; gesendete Nachrichten kommen zu spät an, sind veraltet etc.

Modellierung durch ein Petrinetz

ermöglichen die Analyse der Protokolle, z.B. Erkennung von Verklemmungen. Modellierung einer synchronen Kommunikation:



Problem: unendliches Warten

Pragmatische Lösung mit Hilfe von Timeouts

Sender bzw. Empfänger warten nur eine festgelegte Zeit, Sender: falls kein Acknowledgement (Quittung) eintrifft, z.B. erneutes Senden.

Probleme dabei? u.a. Duplikate müssen vom Empfänger erkannt werden; gesendete Nachrichten kommen zu spät an, sind veraltet etc.

Ports

Bisher bestand zwischen Sender und Empfänger eine feste Beziehung, die über Prozessidentifikatoren (z.B. Namen oder Nummer) hergestellt wurde. Nachteile

Prozessnummern ändern sich mit jedem Neustart.

Prozessnamen sind nicht eindeutig, z.B. falls Programm mehrmals gestartet wurde.

⇒ Deshalb Senden von Nachrichten an ***Ports*** . Sie stellen Endpunkte einer Kommunikation dar. Sie können bei Bedarf dynamisch eingerichtet und gelöscht werden. Dazu existieren folgende Funktionen:

```
portID = createPort();
deletePort(portID);
send(E.portID, message);
receive(portID, message);
```

ein Port ist mit dem Adressraum des zugehörigen Prozesses verbunden.

der Empfängerprozess kann sender-spezifische Ports einrichten.

ein Rechner mit einer IP-Adresse unterstützt mehrere tausend Ports.

der Name des Ports ist für einen Rechner eindeutig.

die Portnummern 1 - 1023 sind fest reserviert für bestimmte Protokolle (bzw. deren Applikationen).

Übersicht: fest zugeordnete Ports

Protokoll	*Port*	*Beschreibung*
FTP	21	Kommandos für Dateitransfer (get, put)
Telnet	23	interaktive Terminal-Sitzung mit entferntem Rechner
SMTP	25	Senden von Email zwischen Rechnern
time	37	Time-Server liefert aktuelle Zeit
finger	79	liefert Informationen über einen Benutzer
HTTP	80	Protokoll des World Wide Web
POP3	110	Zugang zu Email durch einen sporadisch verbundenen Client
RMI	1099	Zugang zum Registrieren von entfernten Java Objekten.

Ports

Bisher bestand zwischen Sender und Empfänger eine feste Beziehung, die über Prozessidentifikatoren (z.B. Namen oder Nummer) hergestellt wurde. Nachteile

- Prozessnummern ändern sich mit jedem Neustart.
- Prozessnamen sind nicht eindeutig, z.B. falls Programm mehrmals gestartet wurde.

⇒ Deshalb Senden von Nachrichten an ***Ports*** . Sie stellen Endpunkte einer Kommunikation dar. Sie können bei Bedarf dynamisch eingerichtet und gelöscht werden. Dazu existieren folgende Funktionen:

```
portID = createPort();
deletePort(portID);
send(E.portID, message);
receive(portID, message);
```

- ein Port ist mit dem Adressraum des zugehörigen Prozesses verbunden.
- der Empfängerprozess kann sender-spezifische Ports einrichten.
- ein Rechner mit einer IP-Adresse unterstützt mehrere tausend Ports.
- der Name des Ports ist für einen Rechner eindeutig.
- die Portnummern 1 - 1023 sind fest reserviert für bestimmte Protokolle (bzw. deren Applikationen).

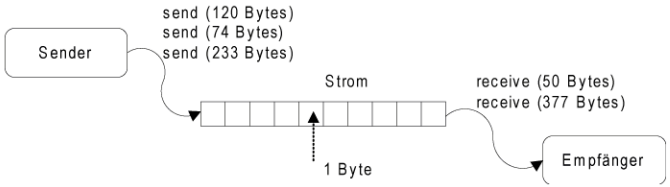
Übersicht: fest zugeordnete Ports

Nachrichtenbasierte Kommunikation

Bei nachrichtenbasierter Prozessinteraktion tauschen Prozesse gezielt Informationen durch Verschicken und Empfangen von Nachrichten aus; ein Kommunikationssystem unterstützt an der Schnittstelle wenigstens die Funktionen `send` und `receive` .

Ströme

Ströme (engl. streams) sind eine Abstraktion von Kanälen. Sie verdecken die tatsächlichen Nachrichtengrenzen.



BS-Dienste: Verbindungsauf- und -abbau, schreiben in Strom, lesen aus Strom.

Dienste für Dateizugriffe oder Zugriffe auf Geräte: spezielle Ausprägung der stromorientierten Kommunikation.

I/O in Java basiert auf Ströme.

- Klasse `java.io.OutputStream` zum Schreiben von Daten
- Klasse `java.io.InputStream` zum Lesen von Daten

Spezialisierungen z.B. durch FileOutputStream, BufferedOutputStream oder FileInputStream.

Pipes

Realisierung von Strömen über Pipe-Konzept (z.B. in Unix, Windows); Strom zwischen 2 Kommunikationspartnern

unidirektional

FIFO-artiger Datentransfer mit Operationen: open pipe, read, write.

gepuffert und zuverlässig.

Pipes können als Dateien ohne Plattenbeteiligung betrachtet werden.

Ordnung der Zeichen bleibt erhalten (Zeichenstrom)

Blockieren bei voller Pipe (write) und leerer Pipe (read)

Beispiele für Unix Pipes

ls -l | head

grep "name" datei.txt | sort | more

cat datei.txt | awk '{print $1}' | more

Beispielnutzung einer Pipe

Die Pipe wird im Vaterprozess angelegt, der anschließend 2 Kindprozesse startet, die über eine Pipe kommunizieren.

Systemaufruf `dup()` ermöglicht Umlenkung von Standard-Ein/-Ausgabe auf Pipe

```
main (int argc, char *argv[]) {
     int fhandle[2]; int rc;
     pipe(fhandle); /* erzeugt Pipe*/
     rc = fork();
     if (rc ==0) {
         close(0); /* Schließen Standard Eingabe */
         dup(fhandle[0]); /* Umlenken von fhandle[0] auf Standard Eingabe */
         close(fhandle[0]); /* Schließen der nicht mehr benötigten fhandle[0] */
             /* der nun gestartete Prozess liest bei Zugriff auf Standard Eingabe aus der Pipe
             */
         rc = execl("kind1", "Kind 1", "\0");
     } else {
         rc = fork(); /* Vaterprozess */
         if (rc == 0) {
             close(1); /* Schließen Standard Ausgabe */
             dup(fhandle[1]); /* Umlenken von fhandle[1] auf Standard Ausgabe */
             close(fhandle[1]); /* Schließen der nicht mehr benötigten fhandle[1] */
                 /* der nun gestartete Prozess schreibt bei Zugriff auf Standard Ausgabe in die
                 Pipe; zwischen den Kindern ist dadurch eine unidirektionale Kommunikation
                 mittels Pipes erzeugt worden */
             rc = execl("kind2", "Kind 2", "\0");
     }
     sleep(1);
     exit(0);
```

Beispielnutzung einer Pipe

Systemaufruf `dup()` ermöglicht Umlenkung von Standard-Ein/-Ausgabe auf Pipe

```
main (int argc, char *argv[]) {
     int fhandle[2]; int rc;
     pipe(fhandle); /* erzeugt Pipe*/
     rc = fork();
     if (rc ==0) {
         close(0); /* Schließen Standard Eingabe */
         dup(fhandle[0]); /* Umlenken von fhandle[0] auf Standard Eingabe */
         close(fhandle[0]); /* Schließen der nicht mehr benötigten fhandle[0] */
             /* der nun gestartete Prozess liest bei Zugriff auf Standard Eingabe aus der Pipe
             */
         rc = execl("kind1", "Kind 1", "\0");
     } else {
         rc = fork(); /* Vaterprozess */
         if (rc == 0) {
             close(1); /* Schließen Standard Ausgabe */
             dup(fhandle[1]); /* Umlenken von fhandle[1] auf Standard Ausgabe */
             close(fhandle[1]); /* Schließen der nicht mehr benötigten fhandle[1] */
                 /* der nun gestartete Prozess schreibt bei Zugriff auf Standard Ausgabe in die
                 Pipe; zwischen den Kindern ist dadurch eine unidirektionale Kommunikation
                 mittels Pipes erzeugt worden */
             rc = execl("kind2", "Kind 2", "\0");
     }
     sleep(1);
     exit(0);
```


Named Pipes

Problem: nur Prozesse, die über `fork()` eng miteinander verwandt sind, können im Beispielprogramm kommunizieren.

Einführung von Named Pipes, die mittels `mknod` erzeugt werden.

ermöglicht die Kommunikation zwischen Prozessen, die nicht miteinander verwandt sind.

```
main (int argc, char *argv[]) {
     int rc;
     mknod("myfifo", 0600, S_IFIFO);
     if (rc == 0) {
         int fd;
         fd = open("myfifo", O_WRONLY);
         write(fd, "Satz 1\n", 7);
         close(fd);
         exit(0)
     } else {
         int fd;
         char block[20];
         fd = open("myfifo", O_RDONLY);
         rc = read(fd, block, 7);
         write(1, ":", 1);
         if (rc > 0) { write(1, block, rc); }
         else printf("Fehler: %d\n", rc);
         close(fd);
         exit(0)
     }
}
```

Kinderprozesse

Kind 1 liest aus der Pipe, während Kind 2 in die Pipe schreibt.

```
/* Kind 1 */
main (int argc, char *argv[]) {
      char buffer[100];
      read(0, buffer, 28); /* Prozess liest von Pipe */
      write(1, buffer, 28); /* schreibt auf Standard Aus */
      exit(0);
}
```

```
/* Kind 2 */
main (int argc, char *argv[]) {
      write(1, "Dies erscheint in der Queue\n", 28);
      exit(0);
}
```


Named Pipes

Problem: nur Prozesse, die über `fork()` eng miteinander verwandt sind, können im Beispielprogramm kommunizieren.

Einführung von Named Pipes, die mittels `mknod` erzeugt werden.

ermöglicht die Kommunikation zwischen Prozessen, die nicht miteinander verwandt sind.

```
main (int argc, char *argv[]) {
      int rc;
      mknod("myfifo", 0600, S_IFIFO);
      if (rc == 0) {
          int fd;
          fd = open("myfifo", O_WRONLY);
          write(fd, "Satz 1\n", 7);
          close(fd);
          exit(0)
      } else {
          int fd;
          char block[20];
          fd = open("myfifo", O_RDONLY);
          rc = read(fd, block, 7);
          write(1, ":", 1);
          if (rc > 0) { write(1, block, rc); }
          else printf("Fehler: %d\n", rc);
          close(fd);
          exit(0)
      }
}
```

Pipes

Realisierung von Strömen über Pipe-Konzept (z.B. in Unix, Windows); Strom zwischen 2 Kommunikationspartnern

unidirektional

FIFO-artiger Datentransfer mit Operationen: open pipe, read, write.

gepuffert und zuverlässig.

Pipes können als Dateien ohne Plattenbeteiligung betrachtet werden.

Ordnung der Zeichen bleibt erhalten (Zeichenstrom)

Blockieren bei voller Pipe (write) und leerer Pipe (read)

Beispiele für Unix Pipes

ls -l | head

grep "name" datei.txt | sort | more

cat datei.txt | awk '{print $1}' | more

Prinzipieller Aufbau
Beispielnutzung einer Pipe
Named Pipes

Nachrichtenbasierte Kommunikation

Bei nachrichtenbasierter Prozessinteraktion tauschen Prozesse gezielt Informationen durch Verschicken und Empfangen von Nachrichten aus; ein Kommunikationssystem unterstützt an der Schnittstelle wenigstens die Funktionen `send` und `receive` .

Elementare Kommunikationsmodelle
Erzeuger-Verbraucher Problem
Modellierung durch ein Petrinetz
Ports
Kanäle
Ströme
Pipes

Definitionen

Definition: Client

Ein Client ist eine Anwendung, die auf einer Clientmaschine läuft und i.a. einen Auftrag initiiert, und die den geforderten Dienst von einem Server erhält.

Clients sind meist a-priori nicht bekannt.

Definition: Server

Ein Server ist ein Subsystem, das auf einer Servermaschine läuft und einen bestimmten Dienst für a-priori unbekannte Clients zur Verfügung stellt.

Server sind dedizierte Prozesse, die kontinuierlich folgende Schleife abarbeiten.

```
while (true) {
    receive (empfangsport, auftrag);
    führe Auftrag aus und erzeuge Antwortnachricht;
    bestimme sendeport für Antwortnachricht;
    send (sendeport, resultat);
}
```

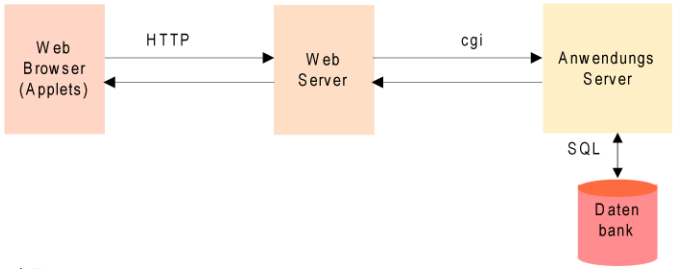
Client und Server kommunizieren über Nachrichten, wobei beide Systeme auf unterschiedlichen Rechnern ablaufen können und über ein Netz miteinander verbunden sind.

Generated by Targeteam

Multi-Tier

ein System kann sowohl Client als auch Server sein.

Beispiel



Generated by Targeteam

Peer-to-Peer Computing

Es findet keine Unterscheidung zwischen Client und Server statt

⇒ Systeme übernehmen beide Rollen, d.h. sie sind sowohl Client als auch Server (Peers).

Damit soll der Server-Flaschenhals eliminiert werden.

Teilnahme an einem P2P-System erfordert explizites Beitreten.

Feststellung, welche Dienste im P2P-Systeme verfügbar sind

ein Knoten registriert seine Dienste bei einem zentrale Directory-Service.

es existiert ein Discovery-Protokoll, mit Hilfe dessen ein Client-Knoten eine Liste möglicher Dienste im P2P-System erstellen kann.

Beispielsysteme sind Napster und Gnutella.

P2P-Systeme wurde in der Vergangenheit häufig im Zusammenhang mit Musik-Tauschbörsen eingesetzt ⇒ rechtliche Probleme.

Generated by Targeteam