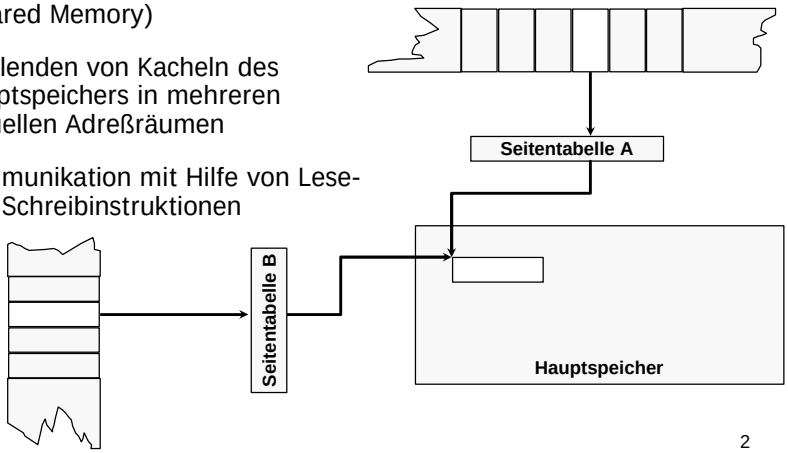
Systemsoftware I

7. Kommunikation

1

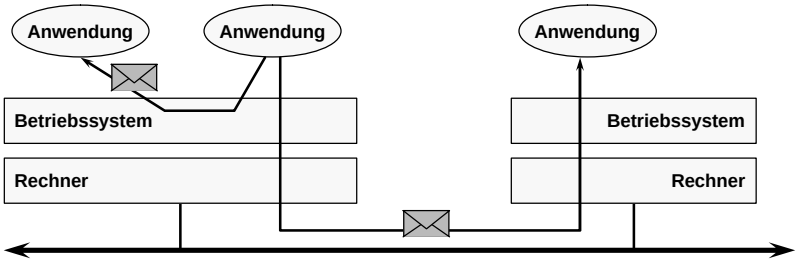
Speicherbasierte Kommunikation

- Mehrere Prozesse teilen sich einen bestimmten Speicher-bereich (Shared Memory)
- Einblenden von Kacheln des Hauptspeichers in mehreren virtuellen Adreßräumen
- Kommunikation mit Hilfe von Lese- und Schreibinstruktionen



2

Nachrichtenbasierte Kommunikation



- Nachrichten = Menge von Bytes
- Senden (Send) und Empfangen (Receive) von Nachrichten

3

Vergleich

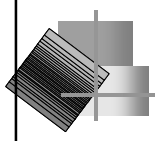
Speicherbasierte Kommunikation

- Lesen und Schreiben in einem gemeinsamen Speicher
- Vorteile
  - Geschwindigkeit
  - Transparenz
- Nachteile
  - Voraussetzung: gemeinsamer Speicher (Mono- und Multi-prozessorsysteme)
  - Transparenz
  - Synchronisation zur Wahrung der Datenkonsistenz

Nachrichtenbasierte Kommunikation

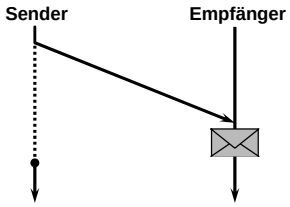
- Senden und Empfangen von Nachrichten
- Vorteile
  - Allgemeineres Modell: Anwendbar auf Mono- und Multiprozessorsysteme sowie auf Rechnernetze
  - Sichtbare Auswirkunge (Spezielle Funktionen)
- Nachteile
  - Effizienz
    - Sender und Empfänger auf einem Rechner
    - “Langsame” Netze
  - Sichtbare Auswirkungen (Spezielle Funktionen)

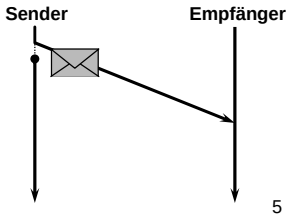
4



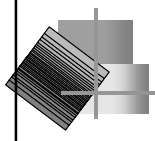
Synchronität

- Kopplungsgrad zwischen Sender und Empfänger
- Synchrone Kommunikation
  - Sender blockiert bis zum Empfang der Nachricht
  - Weniger Pufferplatz
  - Einschränkung in der Parallelität
  - Obere Schranken für Nachrichtenübertragung
  - “Blocking Send”
- Asynchrone Kommunikation
  - Sender “blockiert” bis Nachricht kopiert wurde
  - Viele Nachrichten von einem Sender kurz hintereinander (Congestion)
  - Pufferung von Nachrichten
  - Unbestimmte Übertragungszeit
  - “Non-blocking Send”



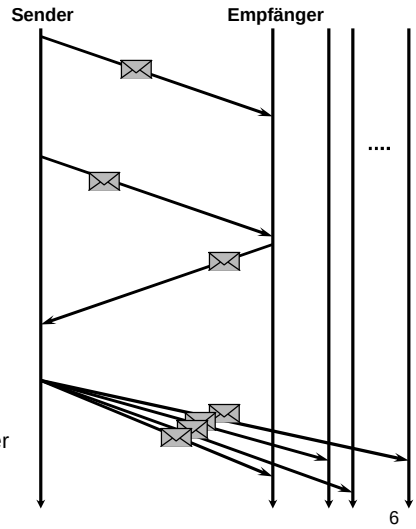


5



Kommunikationsmuster

- Struktur der “kleinsten” Kommunikationseinheit
  - Modellvorgabe
- Grundmuster
  - Mitteilung
    - Einzelne Nachricht vom Sender zum Empfänger
  - Auftrag
    - Auftrag zum Empfänger
    - Antwort zurück an den Absender
- Weitere Muster
  - Multicast
    - Eine Nachricht an mehrere Empfänger
  - Broadcast
    - Eine Nachricht an Alle



6

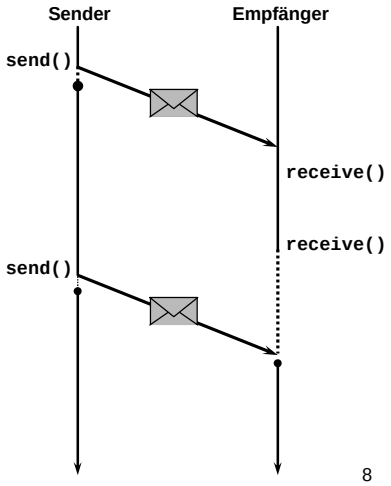
Elementare Kommunikationsformen

	Asynchron	Synchron
Mitteilungsorientiert	Datagramm	Rendezvous
Auftragsorientiert	Asynchroner Entfernter Dienstaufruf	Synchroner Entfernter Dienstaufruf

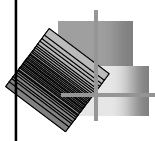
7

Datagramm

- Entkopplung von Sender und Empfänger
  - Parallelarbeit möglich
- Pufferung im Nachrichtensystem
  - Aufwendig (Überläufe, ...)
  - Blockade des Senders nur bei Speicherengpässen
- Beispiele
  - UDP (User Datagramm Protocol)
    - Teil von TCP/IP (Internet)
    - Best Effort
    - max. 64 Kbyte Daten
  - Signale
    - Software-Interrupts in UNIX
    - Keine Daten oder max. 4 Byte
- Sender weiß ...
  - Nichts

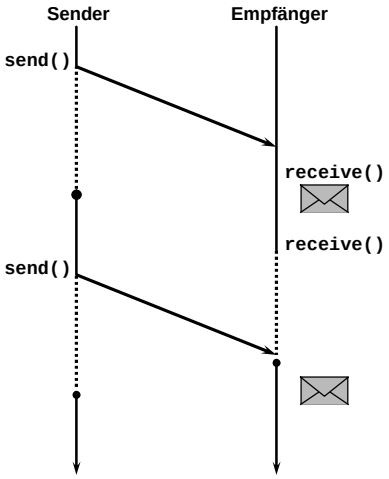


8

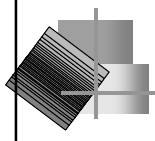


Rendezvous

- Sender und Empfänger sind für die Zeit der Nachrichtenüber-tragung gekoppelt
  - Eingeschränkte Parallelität
- Keine Pufferung
  - Direkte Datenübertragung
  - Keine Pufferung
  - Einfachere Implementierung
- Beispiele
  - Ada, QNX, ...
- Sender weiß ...
  - Nachricht angekommen

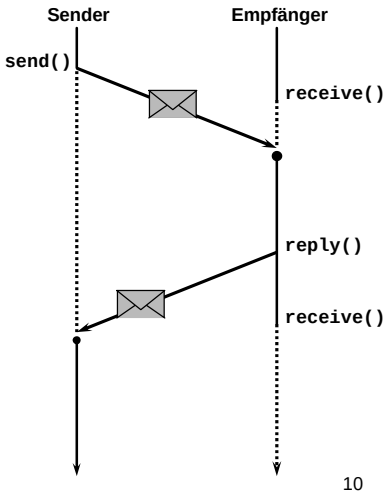


9



Synchroner entfernter Dienstaufruf

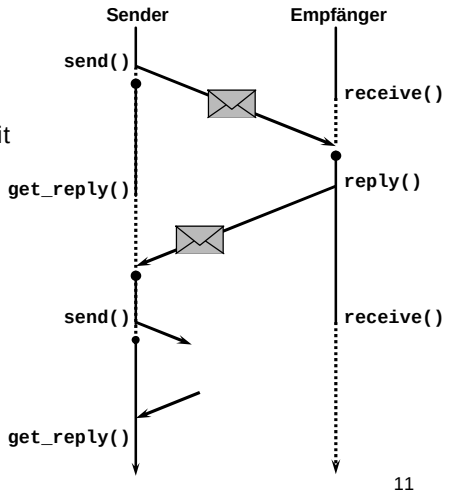
- Synchronous Remote Service Invocation (SRSI)
- Keine Parallelität
- Bidirektionale Kommunikation ohne Pufferung
- Beispiel
  - Remote Procedure Call (RPC)
    - DCE, Corba, COM+, Java RMI, ...
  - QNX
- Sender weiß ...
  - Auftrag angekommen
  - Auftrag bearbeitet
  - Resultat



10

Asynchroner entfernter Dienstaufruf

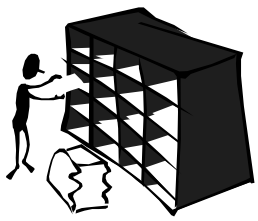
- Asynchronous Remote Service Invocation (ARSI)
  - Parallelität möglich
  - Bidirektionale Kommunikation mit Pufferung
  - Asynchroner RPC
- Beispiel
  - V-Kernel
- Sender weiß ...
  - Auftrag angekommen
  - Auftrag bearbeitet
  - Resultat



11

Klassifikationskriterien

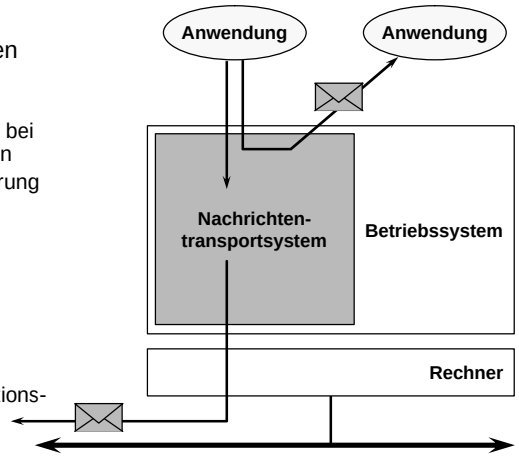
- Elementare Kommunikationsmuster
  - Synchron / Asynchron
  - Mitteilungs- und Auftragsorientiert
- Zusätzliche Kriterien
  - Direkte und indirekte Adressierung
  - Verbindungsorientiert / Verbindungslos
  - Pakete / Nachrichten / Ströme
  - Netztopologie
    - Bus
    - Einzelverbindungen (Graph)
  - Netzausdehnung (bzgl. Bandbreite, Fehlerrate, Latenz, ...)
    - Local Area Network (LAN)
    - Metropolitan Area Network (MAN)
    - Wide Area Network (WAN)
  - ...



12

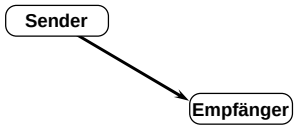
Nachrichtentransportsystem

- Umsetzung bestimmter Kommunikationseigenschaften
  - Verbindungsaufbau und -verwaltung
  - Nachrichtenwiederholungen bei zuverlässiger Kommunikation
  - Begrenzte Zwischenspeicherung von Nachrichten
  - Prüfsummen
  - Routing
  - …
- Realisierungsvarianten
  - Teil des Betriebssystems
  - Eigenständiger Kommunikations-Server
  - Laufzeitbibliothek
  - ↪ Betriebssystemarchitektur

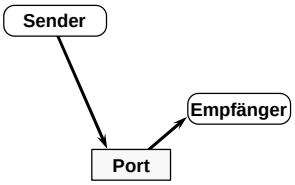


13

Direkte und indirekte Adressierung

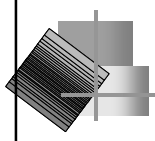


- Direkte Kommunikation
  - Angabe des empfangenden Prozesses
- Vorteile
  - Effizienz
- Nachteile
  - Wechsel des Empfängers schwierig



- Indirekte Kommunikation
  - Angabe eines zwischengeschalteten Nachrichtenspeichers (z.B. Port oder Mailbox)
- Vorteile
  - Flexibilität
    - Wechsel des Empfängers
    - Mehrere Empfänger
- Nachteile
  - zusätzliche Indirektionsstufe

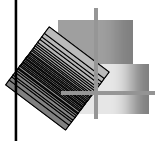
14



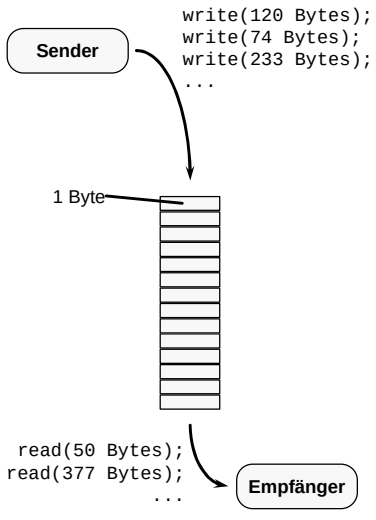
Pakete, Nachrichten ...

- Maximale, auf einmal übertragbare Informationsmenge
- Pakete
  - “Kleine” obere Schranke
  - Größe wird häufig durch Kommunikationshardware bestimmt
    - z.B. 64 Byte ATM oder ca. 1500 Byte Ethernet
  - Anwendung muß größere Informationsmengen selbst paketisieren
- Nachrichten
  - Beliebig lange einzeln übertragbare Informationsmenge
  - Ausreichend größe obere Schranke hinreichend
  - Nachrichtentransportsystem paketisiert

15



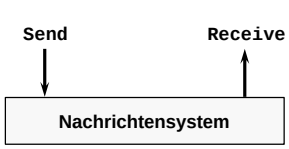
... und Ströme

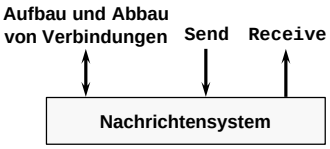


- Paket- und Nachrichtengrenzen für Sender und Empfänger unsichtbar
  - “unbeschränkte” sequentiell schreib- und lesbare Datei
  - Zugriff über Read und Write
  - Meist Verbindungsorientiert
- Beispiel
  - Pipes in UNIX
  - Internetprotokoll TCP

16

Verbindungen: Ja/Nein?





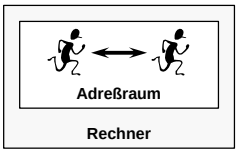
- Verbindungslos
  - Jede "Nachricht" ist autonom
  - Beschränkt auf Paket- und Nachrichten-basierte Verfahren
- Meist "unzuverlässig" (Best Effort)
  - Einzelne Nachrichten können sich überholen
  - Nachrichtenverluste

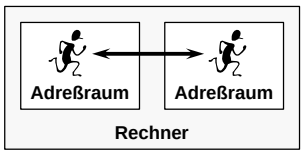
- Verbindungsorientiert
  - Aufbau einer Verbindung (für die Dauer der Kommunikation)
    - Reale Leitung (Circuit Switching)
    - Virtuelle Leitung (Virtual Circuit)
- Meist "zuverlässig" (Reliable)
  - Reihefolge der Daten bleibt erhalten (Keine Überholungen)
  - Keine Verluste
  - Flußkontrolle
  - Verbindungsaufbau kostet Zeit

17

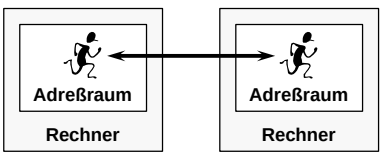
Kopplungsgrad

- Kommunikation im Adreßraum
  - Maximale Effizienz
  - Explizite speicherbasierte Datenaustausch und Synchronisation





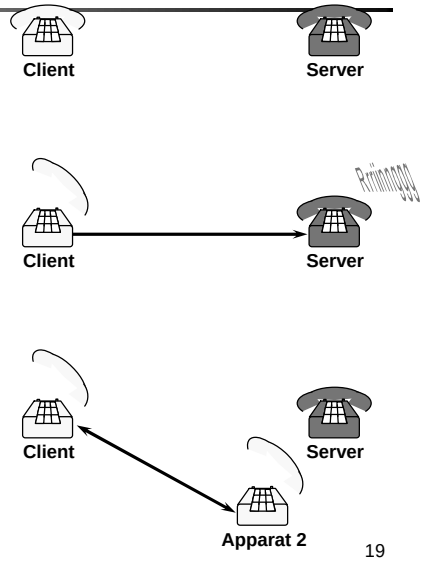
- Kommunikation über Rechnergrenzen
  - Beschränkte Bandbreite
  - Signifikante Fehlerrate
  - Große Latenzzeit



18

Exkurs: TCP

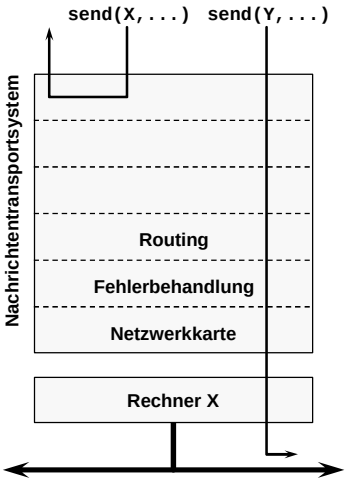
- Weltweit verbreitetes Kommunikationsprotokoll
- Reliable Stream
  - Verbindungsorientiert
  - Keine Verluste
  - Keine Wiederholungen
  - Keine Duplikate
  - Ende-zu-Ende Flußkontrolle
- Einfache Verwendung
  - Sockets (u.a. BSD UNIX)
  - Nach Verbindungsaufbau wie eine Datei ansprechbar
- Weltweit eindeutige Adresse für einen Kommunikationsendpunkt
  - IP-Adresse (32-Bit)
  - Portnummer



19

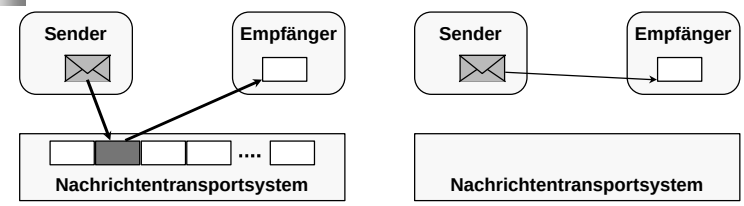
Abkürzungen

- Nutzung von Standards auch bei Prozeßkommunikation auf einem Rechner
  - Leichte Portierbarkeit
  - Ohne Änderung auf verteiltem System ausführbar
- Effizienzverlust
  - Balast vieler WAN-Protokolle
    - Routing
    - Flußkontrolle
    - ...
- Ziel
  - Möglichst frühzeitger Kurzschluß im Nachrichtentransportsystem = geringer Overhead



20

Kommunikation über realen Speicher



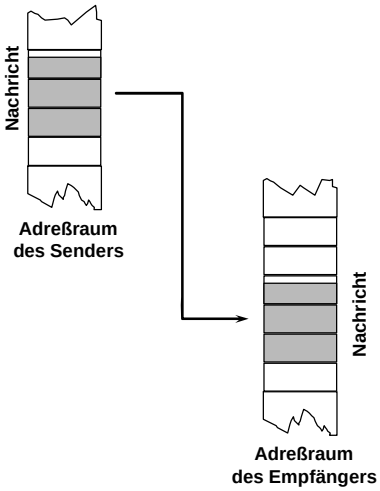
- Asynchrone Kommunikation
- Zwei Kopien
  - Übernahme in Zwischenpuffer
  - Übergabe an Empfänger
- Probleme
  - Pufferverwaltung
  - Puffer sind kritische Systemressourcen

- Synchrone Kommunikation
- Sender und Empfänger blockiert
  - Direkte Kopie vom Sender in den Puffer des Empfängers
  - 1 Kopie
- Probleme
  - Einschränkungen der Parallelität

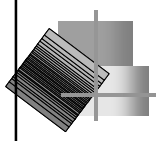
21

Kommunikation über virtuellen Speicher

- Einblenden der Nachricht in den Adreßraum des Empfängers
  - Copy-On-Write Markierung der betroffenen Seiten bei Sender und Empfänger
- Vorteil: u.U. kein Kopieren
- Probleme
  - Kopieren bei schreibendem Zugriff
  - Gleicher Versatz innerhalb der Seite
  - Weitergabe der Nachricht an Dritte
- Lösungsansatz
  - Transportsystem verwaltet Empfangspuffer
  - Empfangsoperation liefert Adresse der Nachricht zurück



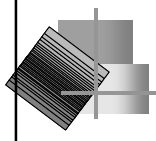
22



Beispiele

- Kommunikation zwischen zwei Adreßräumen
  - Signale
    - Asynchron, Mitteilung, Direkt, Paket
  - Pipes und Named Pipes
    - Synchron/Asynchron, Mitteilung, Direkt/Indirekt, Strom
  - Message Queues
    - Synchron/Asynchron, Mitteilung, Indirekt, Nachricht
- Client/Server-Modell
  - Remote Procedure Call (RPC)
    - Synchron/Asynchron, Auftrag, Direkt/Indirekt, Paket
  - RPC auf einem Rechner = Local Procedure Call (LPC)

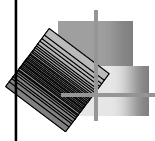
23



Signale

- Übermittlung asynchroner Ereignisse zwischen Prozessen (keine zusätzliche Übertragung von Daten möglich)
- Ursprung
  - Software-Interrupts in Fehlerfällen und für “Job Control”
    - Arithmetikfehler
    - Adreß- und Busfehler
    - Unerlaubter Zugriff
    - Timer
    - ...
- Reaktion auf Signal
  - Blockieren: Zeitliche Empfangsbeschränkung (vgl. Disable Interrupts)
  - Ignorieren
  - Verarbeiten: Anmeldung eines Signal Handlers
  - Prozeß terminieren

24



Beispiel

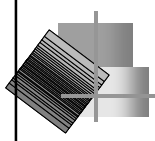
- Prozeß gibt Zustandsinformationen bei Erhalt eines Signals aus

```
int signal_arrived = 0;

void signal_handler ( int signo ) {
    signal_arrived = 1;
    }

int main () {
    signal(SIGUSR1,signal_handler);
    while (1) {
        // Tue etwas sinnvolles ...
        if (signal_arrived) {
            // Reagiere auf Signal ...
            signal_arrived = 0;
            }
        }
    }
```

- Absenden des Signals, z.B.

```
kill -USR1 pid
```

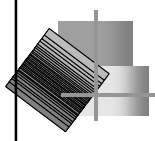
25



Signale (POSIX.4)

- Übermittlung einfacher Ereignisse in Echtzeitsystemen
- Unzulänglichkeiten einfacher Signale
  - Keine Zählung von Signalen
  - Keine Übermittlung zusätzlicher Daten
  - Nur 2 frei benutzbare Signale (SIGUSR1 und SIGUSR2)
  - Blockierendes Warten auf Signalantwort zeitaufwendig
  - Keine Ordnung auf den Signaltypen
- Erweiterungen in POSIX.4
  - Mindestens 8 benutzer-definierbare Signale
  - Übermittlung von maximal 32 Bit Information
  - Zählen von Signalen und Erkennen von Zählerüberläufen

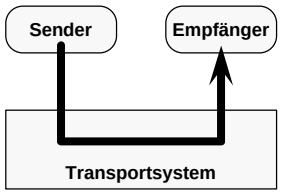
26



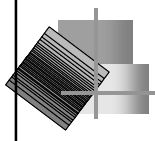
Pipes

- “Pipes” und “Named Pipes” (auch FIFO genannt) für Prozeßkommunikation auf einem Rechner
- Pipes zwischen Vater- und Kindprozessen:
  - Verbindung über vererbte Datei-Deskriptoren
- Beispiel

```
main () {
    int pipe_ends[2];

pipe(pipe_ends);
if (fork() != 0) {
    // Vaterprozeß
    write(pipe_ends[1],Daten,...);
    }
else {
    // Kindprozeß
    read(pipe_ends[0],Daten,...);
    }
}
```



27

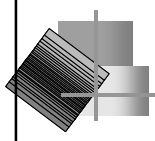


Exkurs: UNIX-Shell und Pipes

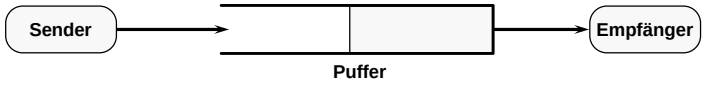
- Exemplarische Realisierung von: “a | b”

```
...
int pfd[2];

pipe(pfd);
if (fork()==0) {
    // Kind - Wird Prozeß a
    dup(pfd[1],1); /* stdout auf Pipe-Eingang */
    execve(“a”,...);
    }

if (fork()==0) {
    // Kind - Wird Prozeß b
    dup(pfd[0],0), /* stdin auf Pipe/Ausgang */
    execve(“b”,...);

if (Vordergrund-Bearbeitung) {
    wait(...);
    }

... zurück zur Eingabeschleife
```

28

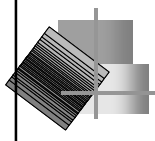


Realisierung



- Klassisches Erzeuger/Verbraucher-System
  - Puffer bestimmter Länge für jede Pipe
  - Empfänger wird blockiert, wenn keine Daten vorliegen
  - Sender wird blockiert, solange Puffer voll ist
- Synchrones Verfahren bei zu kleinem Puffer
  - Gefahr der Verklemmung?
- “Named Pipes”
  - Pipe-Deskriptoren nur über fork vererbbar
  - Pipe-Namen im Dateisystem aufbauen, damit unabhängige Prozesse miteinander kommunizieren können
  - Kommunizierte Daten werden nicht über Dateisystem geleitet!

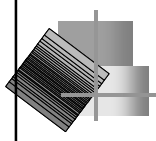
29



Message Queues (POSIX.4)

- Benannte Kommunikationsobjekte
  - Name wird im Dateisystem verankert
- Eigenschaften
  - Aufbau beliebiger Kommunikationstopologien (vgl. “named pipes”)
  - Paketgrenzen bleiben erhalten
  - Prioritätsbehaftete Nachrichten
  - Zustandsinformationen (z.B. Füllstand) abrufbar
- Asynchrone Nachrichtenverarbeitung möglich
- Untere Schranken:
  - Optional: Nicht jedes POSIX.4-konforme System muß Message Queues anbieten
  - Mindestens 8 Message Queues pro Prozeß verwendbar
  - Mindestens 32 verschiedene Prioritätswerte

30



Erzeugung

- Anlegen:

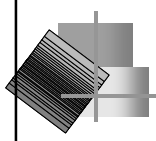
```
mqd_t mq_open (
  char *mq_name,
  int oflag,
  mode_t create_mode,
  struct mq_attr *attr
  )
```

  - oflag = O_RDONLY, O_WRONLY, O_RDWR
  - create_mode = Schutzbits

- Die Struktur **`mq_attr`**:

```
struct mq_attr {
  long mq_maxmsg;  // maximale Anzahl Nachrichten in Queue
  long mq_msgsize; // maximale Größe einer Nachricht
    ...
  }
```

31



Nutzung

- Senden

```
mq_send (
   mqd_t mq,
   char *message,
   size_t length,
   unsigned int priority)
```

- Nachrichtenlänge muß kleiner der Maximallänge sein
- Nachrichten werden gemäß ihrer Priorität in die Schlange eingereiht
- Bei gleicher Priorität FIFO-Ordnung

- Empfangen

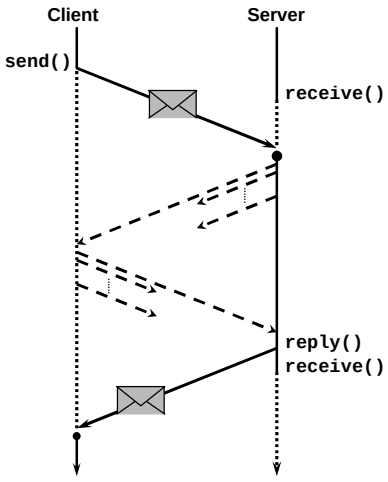
```
mq_receive (
   mqd_t mq,
   char *buffer,
   size_t buffer_size,
   unsigned int *priority)
```

- Puffer muß maximal große Nachricht fassen können
- Immer die erste Nachricht der Schlange wird entnommen
- Funktion liefert die Nachrichtenpriorität zurück

32

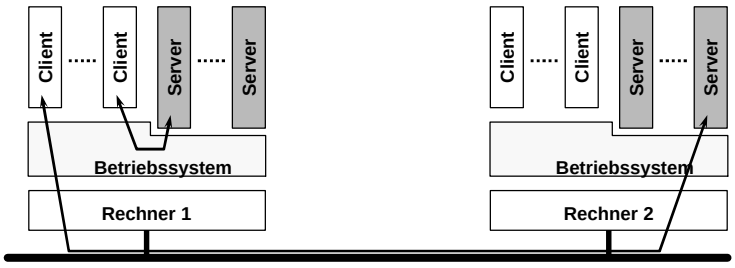
Client/Server-Systeme

- Basis SRSI (ab und zu auch ARSI)
- Offene, verteilte Systeme
  - Server bieten Dienste systemweit an
  - Clients nehmen Dienste in Anspruch
  - Meist RPC-basiert
  - Anwendbar auch in heterogenen Umgebungen
- Moderner Architekturansatz für Betriebssysteme
  - Minimaler Betriebssystemkern mit Grundfunktionen
  - Verlagerung von Systemdiensten in Servern
  - Verteilte Betriebssysteme
  - Effiziente Kommunikation zwischen Adreßräumen



33

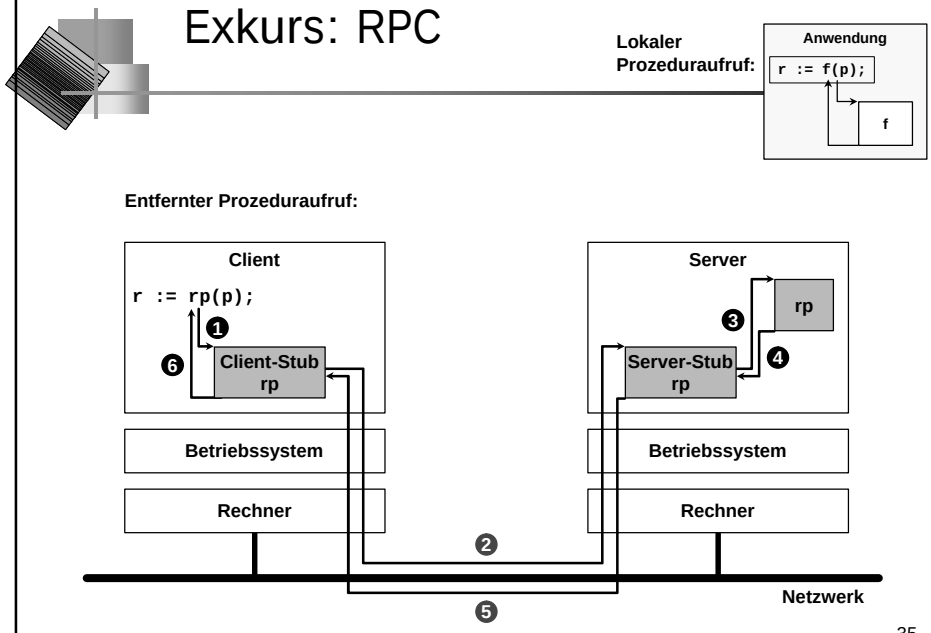
Client/Server-Architektur



- Lokale Server
  - Dateisystem
  - Ein- und Ausgabedienste (Graphik, Kommunikation, ...)
  - Prozeßverwaltung, Paging, ...
- Entfernte Server
  - Netzwerkdateisystem, z.B. NFS
  - Verteiltes Dateisystem
  - Rlogin, Telnet, ...
  - WWW-Server (httpd)
  - Uhrensynchronisation (ntp)

34

Exkurs: RPC



35

Stub-Prozeduren

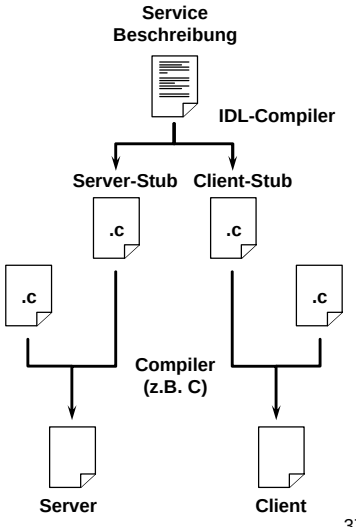
- Einpacken (Marshelling)
  - Linearisierung der Daten (Argumente, Ergebnisse)
    - Strukturen, Felder, …
    - Listen, Bäume, …
  - Umwandlung in einen allgemeinen Netzwerkstandard
    - z.B. xdr, ASN.1
- Auspacken
  - Umwandlung der Daten aus Netzwerk-standard in Rechnerstandard
  - Anlegen der ursprünglichen Struktur
- Ideal: RPC = LPC?
  - Probleme wegen Rechnerausfällen und Nachrichtenverlusten
  - Argumente vom Typ “Call by Reference”?
  - …

```
Client-Stub fc ( a ) : r {
    Puffer m;

    m := Einpacken(a);
    send(server,m);
    receive(m);
    r := Auspacken(m);
    return r;
    }

Server-Stub fs () {
    Adresse client;
    Puffer m;
    Argumente a;
    Resultat r;

    client := receive(m);
    a := Auspacken(m);
    r := f(a);
    m := Einpacken(r);
    send(client,m);
    }
```

36

RPC- oder IDL-Compiler

- Service = Menge von entfernt aufrufbaren Prozeduren
- Service-Beschreibung (Interface Description)
  - Angabe der Signatur jeder Prozedur
    - Anzahl, Typ und Reihenfolge der Argumente
    - Prozedurname
    - Typ des Rückgabewertes
  - Verwendete Sprache: "Interface Description Language" (IDL)
- RPC- oder IDL-Compiler generiert
  - Client-Stub-Prozeduren
  - Server-Stub-Prozeduren
  - Weitere Server-Komponenten
  - ...

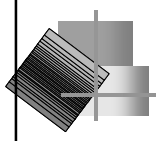


37

"RPC" auf einem Rechner

- Wunsch: RPC durchgängiges Kommunikationsmodell zwischen Prozessen (Client/Server)
- "Leichtgewichtige" RPC-Varianten zur Kommunikation zwischen Adreßräumen auf einem Rechner
  - Homogenes System, d.h. Marhsalling unnötig
  - Umwandlung in Nachricht unnötig
- Probleme
  - Adresse der Zielfunktion nur in einem anderen Adreßraum gültig
  - Stacks sind Adreßraum-gebunden
  - Schutz

38



Beispiel: LPC in Windows NT

- Verbindungsorientiert (Port-Objekte)
  - angelehnt an Mach-System
  - Aufbau ähnlich zu TCP (Verbindungs- und Kommunikationsports)
- 3 Kommunikationsvarianten
  - Austausch von Nachrichten (max. 256 Byte; vgl. Message Queues)
  - Client übermittelt Zeiger auf einen gemeinsam nutzbaren Speicherbereich
    - Client erzeugt und verwaltet gemeinsamen Speicherbereich
  - Client überträgt Daten in einen dedizierten gemeinsam nutzbaren Speicherbereich des Servers (Quick LPC)
    - Client sendet kurze Nachricht an Server
    - Server stellt Client einen 64 Kbyte großen gemeinsamen Speicherbereich zur Verfügung
    - Über ein Semaphor weckt Client den Server nach Beendigung der Datenübertragung
    - Server weckt Client über Semaphor nach Auftragsfertigstellung
    - Quick LPC wird nur intern eingesetzt

39